

NXC

Version 1.2.1 r5

Generated by Doxygen 1.6.2

Mon Oct 17 09:11:03 2011

## Contents

|          |                                    |          |
|----------|------------------------------------|----------|
| <b>1</b> | <b>NXC Programmer's Guide</b>      | <b>1</b> |
| <b>2</b> | <b>Introduction</b>                | <b>1</b> |
| <b>3</b> | <b>The NXC Language</b>            | <b>2</b> |
| 3.1      | Lexical Rules . . . . .            | 2        |
| 3.1.1    | Comments . . . . .                 | 2        |
| 3.1.2    | Whitespace . . . . .               | 3        |
| 3.1.3    | Numerical Constants . . . . .      | 3        |
| 3.1.4    | String Constants . . . . .         | 4        |
| 3.1.5    | Character Constants . . . . .      | 4        |
| 3.1.6    | Identifiers and Keywords . . . . . | 4        |
| 3.2      | Program Structure . . . . .        | 8        |
| 3.2.1    | Code Order . . . . .               | 8        |
| 3.2.2    | Tasks . . . . .                    | 9        |
| 3.2.3    | Functions . . . . .                | 10       |
| 3.2.4    | Variables . . . . .                | 15       |
| 3.2.5    | Structures . . . . .               | 19       |
| 3.2.6    | Arrays . . . . .                   | 19       |
| 3.3      | Statements . . . . .               | 21       |
| 3.3.1    | Variable Declaration . . . . .     | 21       |
| 3.3.2    | Assignment . . . . .               | 22       |
| 3.3.3    | Control Structures . . . . .       | 23       |
| 3.3.4    | The asm statement . . . . .        | 28       |
| 3.3.5    | Other NXC Statements . . . . .     | 31       |
| 3.4      | Expressions . . . . .              | 33       |
| 3.4.1    | Conditions . . . . .               | 36       |
| 3.5      | The Preprocessor . . . . .         | 37       |
| 3.5.1    | #include . . . . .                 | 37       |
| 3.5.2    | #define . . . . .                  | 38       |

---

|          |                             |           |
|----------|-----------------------------|-----------|
| 3.5.3    | ## (Concatenation)          | 38        |
| 3.5.4    | Conditional Compilation     | 38        |
| 3.5.5    | #import                     | 39        |
| 3.5.6    | #download                   | 39        |
| <b>4</b> | <b>Todo List</b>            | <b>40</b> |
| <b>5</b> | <b>Deprecated List</b>      | <b>41</b> |
| <b>6</b> | <b>Module Documentation</b> | <b>43</b> |
| 6.1      | NXT Firmware Modules        | 43        |
| 6.1.1    | Detailed Description        | 44        |
| 6.2      | Input module                | 44        |
| 6.2.1    | Detailed Description        | 44        |
| 6.3      | Input module constants      | 45        |
| 6.3.1    | Detailed Description        | 46        |
| 6.3.2    | Define Documentation        | 46        |
| 6.4      | Sensor types and modes      | 46        |
| 6.4.1    | Detailed Description        | 47        |
| 6.5      | Output module               | 48        |
| 6.5.1    | Detailed Description        | 48        |
| 6.6      | Output module constants     | 49        |
| 6.6.1    | Detailed Description        | 50        |
| 6.7      | Command module              | 50        |
| 6.7.1    | Detailed Description        | 50        |
| 6.8      | Command module constants    | 50        |
| 6.8.1    | Detailed Description        | 51        |
| 6.8.2    | Define Documentation        | 51        |
| 6.9      | Comm module                 | 52        |
| 6.9.1    | Detailed Description        | 53        |
| 6.10     | Button module               | 53        |
| 6.10.1   | Detailed Description        | 54        |

---

|        |   |     |
|--------|---|-----|
| 6.11   | IOCtrl module . . . . .                         | 54  |
| 6.11.1 | Detailed Description . . . . .                  | 54  |
| 6.12   | Loader module . . . . .                         | 54  |
| 6.12.1 | Detailed Description . . . . .                  | 55  |
| 6.13   | Sound module . . . . .                          | 55  |
| 6.13.1 | Detailed Description . . . . .                  | 55  |
| 6.14   | Ui module . . . . .                             | 56  |
| 6.14.1 | Detailed Description . . . . .                  | 56  |
| 6.15   | Low Speed module . . . . .                      | 56  |
| 6.15.1 | Detailed Description . . . . .                  | 57  |
| 6.16   | Display module . . . . .                        | 58  |
| 6.16.1 | Detailed Description . . . . .                  | 58  |
| 6.17   | HiTechnic API Functions . . . . .               | 59  |
| 6.17.1 | Detailed Description . . . . .                  | 69  |
| 6.17.2 | Function Documentation . . . . .                | 69  |
| 6.18   | SuperPro analog output mode constants . . . . . | 122 |
| 6.18.1 | Detailed Description . . . . .                  | 122 |
| 6.18.2 | Define Documentation . . . . .                  | 123 |
| 6.19   | SuperPro LED control constants . . . . .        | 124 |
| 6.19.1 | Detailed Description . . . . .                  | 124 |
| 6.19.2 | Define Documentation . . . . .                  | 124 |
| 6.20   | SuperPro digital pin constants . . . . .        | 124 |
| 6.20.1 | Detailed Description . . . . .                  | 125 |
| 6.20.2 | Define Documentation . . . . .                  | 125 |
| 6.21   | SuperPro Strobe control constants . . . . .     | 126 |
| 6.21.1 | Detailed Description . . . . .                  | 126 |
| 6.21.2 | Define Documentation . . . . .                  | 127 |
| 6.22   | MindSensors API Functions . . . . .             | 127 |
| 6.22.1 | Detailed Description . . . . .                  | 143 |
| 6.22.2 | Function Documentation . . . . .                | 143 |
| 6.23   | Codatex API Functions . . . . .                 | 229 |

---

|  |     |
|--|-----|
| 6.23.1 Detailed Description . . . . .          | 230 |
| 6.23.2 Function Documentation . . . . .        | 230 |
| 6.24 Dexter Industries API Functions . . . . . | 233 |
| 6.24.1 Detailed Description . . . . .          | 235 |
| 6.24.2 Function Documentation . . . . .        | 236 |
| 6.25 Microinfinity API Functions . . . . .     | 247 |
| 6.25.1 Detailed Description . . . . .          | 247 |
| 6.26 RIC Macro Wrappers . . . . .              | 247 |
| 6.26.1 Detailed Description . . . . .          | 250 |
| 6.26.2 Define Documentation . . . . .          | 250 |
| 6.27 NXT firmware module names . . . . .       | 256 |
| 6.27.1 Detailed Description . . . . .          | 257 |
| 6.27.2 Define Documentation . . . . .          | 257 |
| 6.28 NXT firmware module IDs . . . . .         | 258 |
| 6.28.1 Detailed Description . . . . .          | 259 |
| 6.28.2 Define Documentation . . . . .          | 259 |
| 6.29 Miscellaneous NBC/NXC constants . . . . . | 260 |
| 6.29.1 Detailed Description . . . . .          | 261 |
| 6.29.2 Define Documentation . . . . .          | 261 |
| 6.30 Third-party NXT devices . . . . .         | 262 |
| 6.30.1 Detailed Description . . . . .          | 263 |
| 6.31 Standard-C API functions . . . . .        | 263 |
| 6.31.1 Detailed Description . . . . .          | 264 |
| 6.32 A simple 3D graphics library . . . . .    | 264 |
| 6.32.1 Detailed Description . . . . .          | 266 |
| 6.32.2 Function Documentation . . . . .        | 266 |
| 6.33 Type aliases . . . . .                    | 272 |
| 6.33.1 Detailed Description . . . . .          | 273 |
| 6.33.2 Define Documentation . . . . .          | 273 |
| 6.34 Input port constants . . . . .            | 273 |
| 6.34.1 Detailed Description . . . . .          | 274 |

---

|        |   |     |
|--------|---|-----|
| 6.34.2 | Define Documentation                    | 274 |
| 6.35   | Sensor type constants                   | 276 |
| 6.35.1 | Detailed Description                    | 277 |
| 6.35.2 | Define Documentation                    | 277 |
| 6.36   | Sensor mode constants                   | 279 |
| 6.36.1 | Detailed Description                    | 280 |
| 6.36.2 | Define Documentation                    | 280 |
| 6.37   | Combined sensor type and mode constants | 281 |
| 6.37.1 | Detailed Description                    | 282 |
| 6.37.2 | Define Documentation                    | 282 |
| 6.38   | Input module types                      | 284 |
| 6.38.1 | Detailed Description                    | 285 |
| 6.39   | Input module functions                  | 285 |
| 6.39.1 | Detailed Description                    | 289 |
| 6.39.2 | Function Documentation                  | 289 |
| 6.40   | Basic analog sensor value names         | 312 |
| 6.40.1 | Detailed Description                    | 312 |
| 6.40.2 | Define Documentation                    | 312 |
| 6.41   | Output module types                     | 313 |
| 6.41.1 | Detailed Description                    | 313 |
| 6.42   | Output module functions                 | 313 |
| 6.42.1 | Detailed Description                    | 318 |
| 6.42.2 | Function Documentation                  | 318 |
| 6.43   | Display module types                    | 347 |
| 6.43.1 | Detailed Description                    | 348 |
| 6.44   | Display module functions                | 348 |
| 6.44.1 | Detailed Description                    | 352 |
| 6.44.2 | Function Documentation                  | 352 |
| 6.45   | Sound module types                      | 374 |
| 6.45.1 | Detailed Description                    | 375 |
| 6.46   | Sound module functions                  | 375 |

---

|  |     |
|--|-----|
| 6.46.1 Detailed Description . . . . .                | 377 |
| 6.46.2 Function Documentation . . . . .              | 377 |
| 6.47 LowSpeed module types . . . . .                 | 387 |
| 6.47.1 Detailed Description . . . . .                | 387 |
| 6.48 LowSpeed module functions . . . . .             | 387 |
| 6.48.1 Detailed Description . . . . .                | 389 |
| 6.48.2 Function Documentation . . . . .              | 389 |
| 6.49 Low level LowSpeed module functions . . . . .   | 403 |
| 6.49.1 Detailed Description . . . . .                | 405 |
| 6.49.2 Function Documentation . . . . .              | 405 |
| 6.50 LowSpeed module system call functions . . . . . | 410 |
| 6.50.1 Detailed Description . . . . .                | 411 |
| 6.50.2 Function Documentation . . . . .              | 411 |
| 6.51 Command module types . . . . .                  | 412 |
| 6.51.1 Detailed Description . . . . .                | 414 |
| 6.52 Command module functions . . . . .              | 414 |
| 6.52.1 Detailed Description . . . . .                | 419 |
| 6.52.2 Function Documentation . . . . .              | 420 |
| 6.53 Comparison Constants . . . . .                  | 448 |
| 6.53.1 Detailed Description . . . . .                | 448 |
| 6.53.2 Define Documentation . . . . .                | 448 |
| 6.54 Array API functions . . . . .                   | 449 |
| 6.54.1 Detailed Description . . . . .                | 450 |
| 6.54.2 Function Documentation . . . . .              | 451 |
| 6.55 IOCtrl module types . . . . .                   | 459 |
| 6.56 IOCtrl module functions . . . . .               | 459 |
| 6.56.1 Detailed Description . . . . .                | 460 |
| 6.56.2 Function Documentation . . . . .              | 460 |
| 6.57 Comm module types . . . . .                     | 461 |
| 6.57.1 Detailed Description . . . . .                | 462 |
| 6.58 Comm module functions . . . . .                 | 462 |

---

|   |     |
|---|-----|
| 6.58.1 Detailed Description . . . . .   | 471 |
| 6.58.2 Function Documentation . . . . . | 471 |
| 6.59 Direct Command functions . . . . . | 517 |
| 6.59.1 Detailed Description . . . . .   | 519 |
| 6.59.2 Function Documentation . . . . . | 519 |
| 6.60 System Command functions . . . . . | 535 |
| 6.60.1 Detailed Description . . . . .   | 537 |
| 6.60.2 Function Documentation . . . . . | 537 |
| 6.61 Button module types . . . . .      | 551 |
| 6.61.1 Detailed Description . . . . .   | 551 |
| 6.62 Button module functions . . . . .  | 551 |
| 6.62.1 Detailed Description . . . . .   | 552 |
| 6.62.2 Function Documentation . . . . . | 553 |
| 6.63 Ui module types . . . . .          | 559 |
| 6.63.1 Detailed Description . . . . .   | 559 |
| 6.64 Ui module functions . . . . .      | 559 |
| 6.64.1 Detailed Description . . . . .   | 561 |
| 6.64.2 Function Documentation . . . . . | 561 |
| 6.65 Loader module types . . . . .      | 572 |
| 6.65.1 Detailed Description . . . . .   | 573 |
| 6.66 Loader module functions . . . . .  | 573 |
| 6.66.1 Detailed Description . . . . .   | 576 |
| 6.66.2 Function Documentation . . . . . | 576 |
| 6.67 Microinfinity types . . . . .      | 596 |
| 6.67.1 Detailed Description . . . . .   | 596 |
| 6.68 Microinfinity functions . . . . .  | 596 |
| 6.68.1 Detailed Description . . . . .   | 597 |
| 6.68.2 Function Documentation . . . . . | 597 |
| 6.69 cmath API . . . . .                | 599 |
| 6.69.1 Detailed Description . . . . .   | 604 |
| 6.69.2 Define Documentation . . . . .   | 604 |

---

|        |  |     |
|--------|--|-----|
| 6.69.3 | Function Documentation . . . . .         | 617 |
| 6.70   | cstdio API . . . . .                     | 633 |
| 6.70.1 | Detailed Description . . . . .           | 635 |
| 6.70.2 | Define Documentation . . . . .           | 635 |
| 6.70.3 | Function Documentation . . . . .         | 636 |
| 6.71   | fseek origin constants . . . . .         | 644 |
| 6.71.1 | Detailed Description . . . . .           | 644 |
| 6.71.2 | Define Documentation . . . . .           | 644 |
| 6.72   | cstdlib API . . . . .                    | 645 |
| 6.72.1 | Detailed Description . . . . .           | 646 |
| 6.72.2 | Function Documentation . . . . .         | 646 |
| 6.73   | cstdlib API types . . . . .              | 655 |
| 6.73.1 | Detailed Description . . . . .           | 655 |
| 6.74   | cstring API . . . . .                    | 655 |
| 6.74.1 | Detailed Description . . . . .           | 658 |
| 6.74.2 | Function Documentation . . . . .         | 658 |
| 6.75   | ctype API . . . . .                      | 673 |
| 6.75.1 | Detailed Description . . . . .           | 674 |
| 6.75.2 | Function Documentation . . . . .         | 674 |
| 6.76   | Property constants . . . . .             | 679 |
| 6.76.1 | Detailed Description . . . . .           | 680 |
| 6.76.2 | Define Documentation . . . . .           | 680 |
| 6.77   | Array operation constants . . . . .      | 680 |
| 6.77.1 | Detailed Description . . . . .           | 681 |
| 6.77.2 | Define Documentation . . . . .           | 681 |
| 6.78   | System Call function constants . . . . . | 682 |
| 6.78.1 | Detailed Description . . . . .           | 683 |
| 6.78.2 | Define Documentation . . . . .           | 684 |
| 6.79   | Line number constants . . . . .          | 691 |
| 6.79.1 | Detailed Description . . . . .           | 691 |
| 6.79.2 | Define Documentation . . . . .           | 692 |

---

|        |   |     |
|--------|---|-----|
| 6.80   | Time constants                          | 695 |
| 6.80.1 | Detailed Description                    | 696 |
| 6.80.2 | Define Documentation                    | 696 |
| 6.81   | Mailbox constants                       | 703 |
| 6.81.1 | Detailed Description                    | 703 |
| 6.81.2 | Define Documentation                    | 704 |
| 6.82   | VM state constants                      | 705 |
| 6.82.1 | Detailed Description                    | 705 |
| 6.82.2 | Define Documentation                    | 705 |
| 6.83   | Fatal errors                            | 706 |
| 6.83.1 | Detailed Description                    | 707 |
| 6.83.2 | Define Documentation                    | 707 |
| 6.84   | General errors                          | 708 |
| 6.84.1 | Detailed Description                    | 709 |
| 6.84.2 | Define Documentation                    | 709 |
| 6.85   | Communications specific errors          | 709 |
| 6.85.1 | Detailed Description                    | 710 |
| 6.85.2 | Define Documentation                    | 710 |
| 6.86   | Remote control (direct commands) errors | 710 |
| 6.86.1 | Detailed Description                    | 711 |
| 6.86.2 | Define Documentation                    | 711 |
| 6.87   | Program status constants                | 711 |
| 6.87.1 | Detailed Description                    | 711 |
| 6.87.2 | Define Documentation                    | 712 |
| 6.88   | Command module IOMAP offsets            | 712 |
| 6.88.1 | Detailed Description                    | 713 |
| 6.88.2 | Define Documentation                    | 713 |
| 6.89   | IOCtrl module constants                 | 714 |
| 6.89.1 | Detailed Description                    | 715 |
| 6.90   | PowerOn constants                       | 715 |
| 6.90.1 | Detailed Description                    | 715 |

---

|         |                                      |     |
|---------|--------------------------------------|-----|
| 6.90.2  | Define Documentation                 | 715 |
| 6.91    | IOCtrl module IOMAP offsets          | 716 |
| 6.91.1  | Detailed Description                 | 716 |
| 6.91.2  | Define Documentation                 | 716 |
| 6.92    | Loader module constants              | 716 |
| 6.92.1  | Detailed Description                 | 717 |
| 6.92.2  | Define Documentation                 | 717 |
| 6.93    | Loader module IOMAP offsets          | 717 |
| 6.93.1  | Detailed Description                 | 717 |
| 6.93.2  | Define Documentation                 | 717 |
| 6.94    | Loader module error codes            | 717 |
| 6.94.1  | Detailed Description                 | 718 |
| 6.94.2  | Define Documentation                 | 718 |
| 6.95    | Loader module function constants     | 722 |
| 6.95.1  | Detailed Description                 | 723 |
| 6.95.2  | Define Documentation                 | 723 |
| 6.96    | Sound module constants               | 726 |
| 6.96.1  | Detailed Description                 | 727 |
| 6.97    | SoundFlags constants                 | 727 |
| 6.97.1  | Detailed Description                 | 727 |
| 6.97.2  | Define Documentation                 | 728 |
| 6.98    | SoundState constants                 | 728 |
| 6.98.1  | Detailed Description                 | 728 |
| 6.98.2  | Define Documentation                 | 728 |
| 6.99    | SoundMode constants                  | 729 |
| 6.99.1  | Detailed Description                 | 729 |
| 6.99.2  | Define Documentation                 | 729 |
| 6.100   | Sound module IOMAP offsets           | 730 |
| 6.100.1 | Detailed Description                 | 730 |
| 6.100.2 | Define Documentation                 | 730 |
| 6.101   | Sound module miscellaneous constants | 731 |

---

|   |     |
|---|-----|
| 6.101.1 Detailed Description . . . . .      | 732 |
| 6.101.2 Define Documentation . . . . .      | 732 |
| 6.102 Tone constants . . . . .              | 732 |
| 6.102.1 Detailed Description . . . . .      | 734 |
| 6.102.2 Define Documentation . . . . .      | 734 |
| 6.103 Button module constants . . . . .     | 740 |
| 6.103.1 Detailed Description . . . . .      | 740 |
| 6.104 Button name constants . . . . .       | 740 |
| 6.104.1 Detailed Description . . . . .      | 741 |
| 6.104.2 Define Documentation . . . . .      | 741 |
| 6.105 ButtonState constants . . . . .       | 742 |
| 6.105.1 Detailed Description . . . . .      | 743 |
| 6.105.2 Define Documentation . . . . .      | 743 |
| 6.106 Button module IOMAP offsets . . . . . | 744 |
| 6.106.1 Detailed Description . . . . .      | 744 |
| 6.106.2 Define Documentation . . . . .      | 744 |
| 6.107 Ui module constants . . . . .         | 745 |
| 6.107.1 Detailed Description . . . . .      | 745 |
| 6.108 CommandFlags constants . . . . .      | 745 |
| 6.108.1 Detailed Description . . . . .      | 746 |
| 6.108.2 Define Documentation . . . . .      | 746 |
| 6.109 UIState constants . . . . .           | 747 |
| 6.109.1 Detailed Description . . . . .      | 747 |
| 6.109.2 Define Documentation . . . . .      | 748 |
| 6.110 UIButton constants . . . . .          | 749 |
| 6.110.1 Detailed Description . . . . .      | 750 |
| 6.110.2 Define Documentation . . . . .      | 750 |
| 6.111 BluetoothState constants . . . . .    | 751 |
| 6.111.1 Detailed Description . . . . .      | 751 |
| 6.111.2 Define Documentation . . . . .      | 751 |
| 6.112 VM run state constants . . . . .      | 752 |

---

|  |     |
|--|-----|
| 6.112.1 Detailed Description . . . . .           | 752 |
| 6.112.2 Define Documentation . . . . .           | 752 |
| 6.113Ui module IOMAP offsets . . . . .           | 753 |
| 6.113.1 Detailed Description . . . . .           | 754 |
| 6.113.2 Define Documentation . . . . .           | 754 |
| 6.114NBC Input port constants . . . . .          | 756 |
| 6.114.1 Detailed Description . . . . .           | 756 |
| 6.114.2 Define Documentation . . . . .           | 756 |
| 6.115NBC sensor type constants . . . . .         | 757 |
| 6.115.1 Detailed Description . . . . .           | 757 |
| 6.115.2 Define Documentation . . . . .           | 757 |
| 6.116NBC sensor mode constants . . . . .         | 759 |
| 6.116.1 Detailed Description . . . . .           | 760 |
| 6.116.2 Define Documentation . . . . .           | 760 |
| 6.117Input field constants . . . . .             | 761 |
| 6.117.1 Detailed Description . . . . .           | 762 |
| 6.117.2 Define Documentation . . . . .           | 762 |
| 6.118Input port digital pin constants . . . . .  | 762 |
| 6.118.1 Detailed Description . . . . .           | 763 |
| 6.118.2 Define Documentation . . . . .           | 763 |
| 6.119Color sensor array indices . . . . .        | 763 |
| 6.119.1 Detailed Description . . . . .           | 763 |
| 6.119.2 Define Documentation . . . . .           | 764 |
| 6.120Color values . . . . .                      | 764 |
| 6.120.1 Detailed Description . . . . .           | 765 |
| 6.120.2 Define Documentation . . . . .           | 765 |
| 6.121Color calibration state constants . . . . . | 765 |
| 6.121.1 Detailed Description . . . . .           | 766 |
| 6.121.2 Define Documentation . . . . .           | 766 |
| 6.122Color calibration constants . . . . .       | 766 |
| 6.122.1 Detailed Description . . . . .           | 767 |

---

|   |     |
|---|-----|
| 6.122.2 Define Documentation . . . . .                                | 767 |
| 6.123 Input module IOMAP offsets . . . . .                            | 767 |
| 6.123.1 Detailed Description . . . . .                                | 768 |
| 6.123.2 Define Documentation . . . . .                                | 768 |
| 6.124 Constants to use with the Input module's Pin function . . . . . | 770 |
| 6.124.1 Detailed Description . . . . .                                | 771 |
| 6.124.2 Define Documentation . . . . .                                | 771 |
| 6.125 Output port constants . . . . .                                 | 772 |
| 6.125.1 Detailed Description . . . . .                                | 773 |
| 6.125.2 Define Documentation . . . . .                                | 773 |
| 6.126 PID constants . . . . .   | 774 |
| 6.126.1 Detailed Description . . . . .                                | 775 |
| 6.126.2 Define Documentation . . . . .                                | 775 |
| 6.127 Output port update flag constants . . . . .                     | 776 |
| 6.127.1 Detailed Description . . . . .                                | 776 |
| 6.127.2 Define Documentation . . . . .                                | 776 |
| 6.128 Tachometer counter reset flags . . . . .                        | 777 |
| 6.128.1 Detailed Description . . . . .                                | 777 |
| 6.128.2 Define Documentation . . . . .                                | 778 |
| 6.129 Output port mode constants . . . . .                            | 778 |
| 6.129.1 Detailed Description . . . . .                                | 779 |
| 6.129.2 Define Documentation . . . . .                                | 779 |
| 6.130 Output port option constants . . . . .                          | 780 |
| 6.130.1 Detailed Description . . . . .                                | 780 |
| 6.130.2 Define Documentation . . . . .                                | 780 |
| 6.131 Output regulation option constants . . . . .                    | 780 |
| 6.131.1 Detailed Description . . . . .                                | 780 |
| 6.131.2 Define Documentation . . . . .                                | 781 |
| 6.132 Output port run state constants . . . . .                       | 781 |
| 6.132.1 Detailed Description . . . . .                                | 781 |
| 6.132.2 Define Documentation . . . . .                                | 781 |

---

|         |                                       |     |
|---------|---------------------------------------|-----|
| 6.133   | Output port regulation mode constants | 782 |
| 6.133.1 | Detailed Description                  | 782 |
| 6.133.2 | Define Documentation                  | 783 |
| 6.134   | Output field constants                | 783 |
| 6.134.1 | Detailed Description                  | 785 |
| 6.134.2 | Define Documentation                  | 785 |
| 6.135   | Output module IOMAP offsets           | 790 |
| 6.135.1 | Detailed Description                  | 790 |
| 6.135.2 | Define Documentation                  | 790 |
| 6.136   | LowSpeed module constants             | 793 |
| 6.136.1 | Detailed Description                  | 794 |
| 6.137   | LSState constants                     | 794 |
| 6.137.1 | Detailed Description                  | 794 |
| 6.137.2 | Define Documentation                  | 794 |
| 6.138   | LSChannelState constants              | 795 |
| 6.138.1 | Detailed Description                  | 795 |
| 6.138.2 | Define Documentation                  | 795 |
| 6.139   | LSMode constants                      | 796 |
| 6.139.1 | Detailed Description                  | 796 |
| 6.139.2 | Define Documentation                  | 797 |
| 6.140   | LSErrorType constants                 | 797 |
| 6.140.1 | Detailed Description                  | 797 |
| 6.140.2 | Define Documentation                  | 797 |
| 6.141   | Low speed module IOMAP offsets        | 798 |
| 6.141.1 | Detailed Description                  | 798 |
| 6.141.2 | Define Documentation                  | 798 |
| 6.142   | LSNoRestartOnRead constants           | 800 |
| 6.142.1 | Detailed Description                  | 800 |
| 6.142.2 | Define Documentation                  | 800 |
| 6.143   | Standard I2C constants                | 801 |
| 6.143.1 | Detailed Description                  | 801 |

---

|  |     |
|--|-----|
| 6.143.2 Define Documentation . . . . .           | 802 |
| 6.144LEGO I2C address constants . . . . .        | 803 |
| 6.144.1 Detailed Description . . . . .           | 803 |
| 6.144.2 Define Documentation . . . . .           | 803 |
| 6.145Ultrasonic sensor constants . . . . .       | 803 |
| 6.145.1 Detailed Description . . . . .           | 804 |
| 6.145.2 Define Documentation . . . . .           | 804 |
| 6.146LEGO temperature sensor constants . . . . . | 805 |
| 6.146.1 Detailed Description . . . . .           | 806 |
| 6.146.2 Define Documentation . . . . .           | 806 |
| 6.147E-Meter sensor constants . . . . .          | 808 |
| 6.147.1 Detailed Description . . . . .           | 809 |
| 6.147.2 Define Documentation . . . . .           | 809 |
| 6.148I2C option constants . . . . .              | 810 |
| 6.148.1 Detailed Description . . . . .           | 810 |
| 6.148.2 Define Documentation . . . . .           | 810 |
| 6.149Display module constants . . . . .          | 810 |
| 6.149.1 Detailed Description . . . . .           | 812 |
| 6.149.2 Define Documentation . . . . .           | 812 |
| 6.150DisplayExecuteFunction constants . . . . .  | 816 |
| 6.150.1 Detailed Description . . . . .           | 817 |
| 6.150.2 Define Documentation . . . . .           | 817 |
| 6.151Drawing option constants . . . . .          | 818 |
| 6.151.1 Detailed Description . . . . .           | 819 |
| 6.151.2 Define Documentation . . . . .           | 819 |
| 6.152Font drawing option constants . . . . .     | 821 |
| 6.152.1 Detailed Description . . . . .           | 822 |
| 6.152.2 Define Documentation . . . . .           | 822 |
| 6.153Display flags . . . . .                     | 823 |
| 6.153.1 Detailed Description . . . . .           | 824 |
| 6.153.2 Define Documentation . . . . .           | 824 |

---

|         |                                     |     |
|---------|-------------------------------------|-----|
| 6.154   | Display contrast constants          | 824 |
| 6.154.1 | Detailed Description                | 825 |
| 6.154.2 | Define Documentation                | 825 |
| 6.155   | Text line constants                 | 825 |
| 6.155.1 | Detailed Description                | 826 |
| 6.155.2 | Define Documentation                | 826 |
| 6.156   | Display module IOMAP offsets        | 827 |
| 6.156.1 | Detailed Description                | 828 |
| 6.156.2 | Define Documentation                | 828 |
| 6.157   | Comm module constants               | 830 |
| 6.157.1 | Detailed Description                | 831 |
| 6.158   | Miscellaneous Comm module constants | 831 |
| 6.158.1 | Detailed Description                | 831 |
| 6.158.2 | Define Documentation                | 832 |
| 6.159   | Bluetooth State constants           | 833 |
| 6.159.1 | Detailed Description                | 834 |
| 6.159.2 | Define Documentation                | 834 |
| 6.160   | Data mode constants                 | 834 |
| 6.160.1 | Detailed Description                | 834 |
| 6.160.2 | Define Documentation                | 834 |
| 6.161   | Bluetooth state status constants    | 835 |
| 6.161.1 | Detailed Description                | 835 |
| 6.161.2 | Define Documentation                | 836 |
| 6.162   | Remote connection constants         | 836 |
| 6.162.1 | Detailed Description                | 837 |
| 6.162.2 | Define Documentation                | 837 |
| 6.163   | Bluetooth hardware status constants | 839 |
| 6.163.1 | Detailed Description                | 839 |
| 6.163.2 | Define Documentation                | 839 |
| 6.164   | Hi-speed port constants             | 839 |
| 6.164.1 | Detailed Description                | 840 |

---

|         |   |     |
|---------|---|-----|
| 6.165   | Hi-speed port flags constants             | 840 |
| 6.165.1 | Detailed Description                      | 840 |
| 6.165.2 | Define Documentation                      | 840 |
| 6.166   | Hi-speed port state constants             | 841 |
| 6.166.1 | Detailed Description                      | 841 |
| 6.166.2 | Define Documentation                      | 841 |
| 6.167   | Hi-speed port SysCommHSCControl constants | 842 |
| 6.167.1 | Detailed Description                      | 842 |
| 6.167.2 | Define Documentation                      | 842 |
| 6.168   | Hi-speed port baud rate constants         | 843 |
| 6.168.1 | Detailed Description                      | 843 |
| 6.168.2 | Define Documentation                      | 843 |
| 6.169   | Hi-speed port UART mode constants         | 845 |
| 6.169.1 | Detailed Description                      | 846 |
| 6.169.2 | Define Documentation                      | 846 |
| 6.170   | Hi-speed port data bits constants         | 846 |
| 6.170.1 | Detailed Description                      | 847 |
| 6.170.2 | Define Documentation                      | 847 |
| 6.171   | Hi-speed port stop bits constants         | 847 |
| 6.171.1 | Detailed Description                      | 848 |
| 6.171.2 | Define Documentation                      | 848 |
| 6.172   | Hi-speed port parity constants            | 848 |
| 6.172.1 | Detailed Description                      | 848 |
| 6.172.2 | Define Documentation                      | 848 |
| 6.173   | Hi-speed port combined UART constants     | 849 |
| 6.173.1 | Detailed Description                      | 849 |
| 6.173.2 | Define Documentation                      | 849 |
| 6.174   | Hi-speed port address constants           | 850 |
| 6.174.1 | Detailed Description                      | 850 |
| 6.174.2 | Define Documentation                      | 850 |
| 6.175   | Device status constants                   | 851 |

---

|  |     |
|--|-----|
| 6.175.1 Detailed Description . . . . .                   | 851 |
| 6.175.2 Define Documentation . . . . .                   | 852 |
| 6.176 Comm module interface function constants . . . . . | 852 |
| 6.176.1 Detailed Description . . . . .                   | 853 |
| 6.176.2 Define Documentation . . . . .                   | 853 |
| 6.177 Comm module status code constants . . . . .        | 855 |
| 6.177.1 Detailed Description . . . . .                   | 855 |
| 6.177.2 Define Documentation . . . . .                   | 855 |
| 6.178 Comm module IOMAP offsets . . . . .                | 856 |
| 6.178.1 Detailed Description . . . . .                   | 858 |
| 6.178.2 Define Documentation . . . . .                   | 858 |
| 6.179 RCX constants . . . . .                            | 863 |
| 6.179.1 Detailed Description . . . . .                   | 864 |
| 6.180 RCX output constants . . . . .                     | 864 |
| 6.180.1 Detailed Description . . . . .                   | 864 |
| 6.180.2 Define Documentation . . . . .                   | 864 |
| 6.181 RCX output mode constants . . . . .                | 865 |
| 6.181.1 Detailed Description . . . . .                   | 866 |
| 6.181.2 Define Documentation . . . . .                   | 866 |
| 6.182 RCX output direction constants . . . . .           | 866 |
| 6.182.1 Detailed Description . . . . .                   | 866 |
| 6.182.2 Define Documentation . . . . .                   | 866 |
| 6.183 RCX output power constants . . . . .               | 867 |
| 6.183.1 Detailed Description . . . . .                   | 867 |
| 6.183.2 Define Documentation . . . . .                   | 867 |
| 6.184 RCX IR remote constants . . . . .                  | 868 |
| 6.184.1 Detailed Description . . . . .                   | 868 |
| 6.184.2 Define Documentation . . . . .                   | 868 |
| 6.185 RCX and Scout sound constants . . . . .            | 870 |
| 6.185.1 Detailed Description . . . . .                   | 870 |
| 6.185.2 Define Documentation . . . . .                   | 871 |

---

|   |     |
|---|-----|
| 6.186Scout constants . . . . .                | 872 |
| 6.186.1 Detailed Description . . . . .        | 872 |
| 6.187Scout light constants . . . . .          | 872 |
| 6.187.1 Detailed Description . . . . .        | 873 |
| 6.187.2 Define Documentation . . . . .        | 873 |
| 6.188Scout sound constants . . . . .          | 873 |
| 6.188.1 Detailed Description . . . . .        | 874 |
| 6.188.2 Define Documentation . . . . .        | 874 |
| 6.189Scout sound set constants . . . . .      | 876 |
| 6.189.1 Detailed Description . . . . .        | 876 |
| 6.189.2 Define Documentation . . . . .        | 877 |
| 6.190Scout mode constants . . . . .           | 877 |
| 6.190.1 Detailed Description . . . . .        | 877 |
| 6.190.2 Define Documentation . . . . .        | 878 |
| 6.191Scout motion rule constants . . . . .    | 878 |
| 6.191.1 Detailed Description . . . . .        | 878 |
| 6.191.2 Define Documentation . . . . .        | 878 |
| 6.192Scout touch rule constants . . . . .     | 879 |
| 6.192.1 Detailed Description . . . . .        | 880 |
| 6.192.2 Define Documentation . . . . .        | 880 |
| 6.193Scout light rule constants . . . . .     | 880 |
| 6.193.1 Detailed Description . . . . .        | 881 |
| 6.193.2 Define Documentation . . . . .        | 881 |
| 6.194Scout transmit rule constants . . . . .  | 881 |
| 6.194.1 Detailed Description . . . . .        | 882 |
| 6.194.2 Define Documentation . . . . .        | 882 |
| 6.195Scout special effect constants . . . . . | 882 |
| 6.195.1 Detailed Description . . . . .        | 883 |
| 6.195.2 Define Documentation . . . . .        | 883 |
| 6.196RCX and Scout source constants . . . . . | 883 |
| 6.196.1 Detailed Description . . . . .        | 884 |

---

|  |     |
|--|-----|
| 6.196.2 Define Documentation . . . . .                                 | 885 |
| 6.197RCX and Scout opcode constants . . . . .                          | 889 |
| 6.197.1 Detailed Description . . . . .                                 | 891 |
| 6.197.2 Define Documentation . . . . .                                 | 891 |
| 6.198HiTechnic/mindsensors Power Function/IR Train constants . . . . . | 899 |
| 6.198.1 Detailed Description . . . . .                                 | 900 |
| 6.199Power Function command constants . . . . .                        | 900 |
| 6.199.1 Detailed Description . . . . .                                 | 900 |
| 6.199.2 Define Documentation . . . . .                                 | 900 |
| 6.200Power Function channel constants . . . . .                        | 901 |
| 6.200.1 Detailed Description . . . . .                                 | 901 |
| 6.200.2 Define Documentation . . . . .                                 | 901 |
| 6.201Power Function mode constants . . . . .                           | 902 |
| 6.201.1 Detailed Description . . . . .                                 | 902 |
| 6.201.2 Define Documentation . . . . .                                 | 903 |
| 6.202PF/IR Train function constants . . . . .                          | 903 |
| 6.202.1 Detailed Description . . . . .                                 | 904 |
| 6.202.2 Define Documentation . . . . .                                 | 904 |
| 6.203IR Train channel constants . . . . .                              | 904 |
| 6.203.1 Detailed Description . . . . .                                 | 904 |
| 6.203.2 Define Documentation . . . . .                                 | 905 |
| 6.204Power Function output constants . . . . .                         | 905 |
| 6.204.1 Detailed Description . . . . .                                 | 905 |
| 6.204.2 Define Documentation . . . . .                                 | 905 |
| 6.205Power Function pin constants . . . . .                            | 906 |
| 6.205.1 Detailed Description . . . . .                                 | 906 |
| 6.205.2 Define Documentation . . . . .                                 | 906 |
| 6.206Power Function single pin function constants . . . . .            | 906 |
| 6.206.1 Detailed Description . . . . .                                 | 907 |
| 6.206.2 Define Documentation . . . . .                                 | 907 |
| 6.207Power Function CST options constants . . . . .                    | 907 |

---

|  |     |
|--|-----|
| 6.207.1 Detailed Description . . . . .                           | 908 |
| 6.207.2 Define Documentation . . . . .                           | 908 |
| 6.208 Power Function PWM option constants . . . . .              | 909 |
| 6.208.1 Detailed Description . . . . .                           | 909 |
| 6.208.2 Define Documentation . . . . .                           | 910 |
| 6.209 HiTechnic device constants . . . . .                       | 911 |
| 6.209.1 Detailed Description . . . . .                           | 912 |
| 6.209.2 Define Documentation . . . . .                           | 913 |
| 6.210 HiTechnic IRSeeker2 constants . . . . .                    | 914 |
| 6.210.1 Detailed Description . . . . .                           | 914 |
| 6.210.2 Define Documentation . . . . .                           | 915 |
| 6.211 HiTechnic IRReceiver constants . . . . .                   | 916 |
| 6.211.1 Detailed Description . . . . .                           | 917 |
| 6.211.2 Define Documentation . . . . .                           | 917 |
| 6.212 HiTechnic Color2 constants . . . . .                       | 918 |
| 6.212.1 Detailed Description . . . . .                           | 918 |
| 6.212.2 Define Documentation . . . . .                           | 918 |
| 6.213 HiTechnic Angle sensor constants . . . . .                 | 920 |
| 6.213.1 Detailed Description . . . . .                           | 920 |
| 6.213.2 Define Documentation . . . . .                           | 920 |
| 6.214 HiTechnic Barometric sensor constants . . . . .            | 921 |
| 6.214.1 Detailed Description . . . . .                           | 922 |
| 6.214.2 Define Documentation . . . . .                           | 922 |
| 6.215 HiTechnic Prototype board constants . . . . .              | 922 |
| 6.215.1 Detailed Description . . . . .                           | 923 |
| 6.215.2 Define Documentation . . . . .                           | 923 |
| 6.216 HiTechnic Prototype board analog input constants . . . . . | 924 |
| 6.216.1 Detailed Description . . . . .                           | 924 |
| 6.216.2 Define Documentation . . . . .                           | 924 |
| 6.217 HiTechnic SuperPro constants . . . . .                     | 925 |
| 6.217.1 Detailed Description . . . . .                           | 927 |

---

|   |     |
|---|-----|
| 6.217.2 Define Documentation . . . . .                          | 927 |
| 6.218HiTechnic SuperPro analog input index constants . . . . .  | 932 |
| 6.218.1 Detailed Description . . . . .                          | 932 |
| 6.218.2 Define Documentation . . . . .                          | 933 |
| 6.219HiTechnic SuperPro analog output index constants . . . . . | 933 |
| 6.219.1 Detailed Description . . . . .                          | 933 |
| 6.219.2 Define Documentation . . . . .                          | 933 |
| 6.220MindSensors device constants . . . . .                     | 934 |
| 6.220.1 Detailed Description . . . . .                          | 935 |
| 6.220.2 Define Documentation . . . . .                          | 935 |
| 6.221MindSensors DIST-Nx constants . . . . .                    | 939 |
| 6.221.1 Detailed Description . . . . .                          | 939 |
| 6.221.2 Define Documentation . . . . .                          | 939 |
| 6.222MindSensors PSP-Nx constants . . . . .                     | 941 |
| 6.222.1 Detailed Description . . . . .                          | 941 |
| 6.222.2 Define Documentation . . . . .                          | 941 |
| 6.223MindSensors PSP-Nx button set 1 constants . . . . .        | 942 |
| 6.223.1 Detailed Description . . . . .                          | 942 |
| 6.223.2 Define Documentation . . . . .                          | 943 |
| 6.224MindSensors PSP-Nx button set 2 constants . . . . .        | 944 |
| 6.224.1 Detailed Description . . . . .                          | 944 |
| 6.224.2 Define Documentation . . . . .                          | 944 |
| 6.225MindSensors nRLink constants . . . . .                     | 946 |
| 6.225.1 Detailed Description . . . . .                          | 946 |
| 6.225.2 Define Documentation . . . . .                          | 947 |
| 6.226MindSensors ACCL-Nx constants . . . . .                    | 948 |
| 6.226.1 Detailed Description . . . . .                          | 949 |
| 6.226.2 Define Documentation . . . . .                          | 949 |
| 6.227MindSensors ACCL-Nx sensitivity level constants . . . . .  | 951 |
| 6.227.1 Detailed Description . . . . .                          | 951 |
| 6.227.2 Define Documentation . . . . .                          | 951 |

---

|  |     |
|--|-----|
| 6.228MindSensors PFMate constants . . . . .                  | 952 |
| 6.228.1 Detailed Description . . . . .                       | 952 |
| 6.228.2 Define Documentation . . . . .                       | 953 |
| 6.229PFMate motor constants . . . . .                        | 954 |
| 6.229.1 Detailed Description . . . . .                       | 954 |
| 6.229.2 Define Documentation . . . . .                       | 954 |
| 6.230PFMate channel constants . . . . .                      | 954 |
| 6.230.1 Detailed Description . . . . .                       | 955 |
| 6.230.2 Define Documentation . . . . .                       | 955 |
| 6.231MindSensors NXTServo constants . . . . .                | 955 |
| 6.231.1 Detailed Description . . . . .                       | 956 |
| 6.232MindSensors NXTServo registers . . . . .                | 956 |
| 6.232.1 Detailed Description . . . . .                       | 957 |
| 6.232.2 Define Documentation . . . . .                       | 957 |
| 6.233MindSensors NXTServo position constants . . . . .       | 960 |
| 6.233.1 Detailed Description . . . . .                       | 960 |
| 6.233.2 Define Documentation . . . . .                       | 960 |
| 6.234MindSensors NXTServo quick position constants . . . . . | 961 |
| 6.234.1 Detailed Description . . . . .                       | 961 |
| 6.234.2 Define Documentation . . . . .                       | 961 |
| 6.235MindSensors NXTServo servo numbers . . . . .            | 961 |
| 6.235.1 Detailed Description . . . . .                       | 962 |
| 6.235.2 Define Documentation . . . . .                       | 962 |
| 6.236MindSensors NXTServo commands . . . . .                 | 963 |
| 6.236.1 Detailed Description . . . . .                       | 963 |
| 6.236.2 Define Documentation . . . . .                       | 963 |
| 6.237MindSensors NXTHID constants . . . . .                  | 964 |
| 6.237.1 Detailed Description . . . . .                       | 965 |
| 6.238MindSensors NXTHID registers . . . . .                  | 965 |
| 6.238.1 Detailed Description . . . . .                       | 965 |
| 6.238.2 Define Documentation . . . . .                       | 965 |

---

|  |     |
|--|-----|
| 6.239MindSensors NXTHID modifier keys . . . . .    | 966 |
| 6.239.1 Detailed Description . . . . .             | 966 |
| 6.239.2 Define Documentation . . . . .             | 966 |
| 6.240MindSensors NXTHID commands . . . . .         | 967 |
| 6.240.1 Detailed Description . . . . .             | 968 |
| 6.240.2 Define Documentation . . . . .             | 968 |
| 6.241MindSensors NXTPowerMeter constants . . . . . | 968 |
| 6.241.1 Detailed Description . . . . .             | 968 |
| 6.242MindSensors NXTPowerMeter registers . . . . . | 968 |
| 6.242.1 Detailed Description . . . . .             | 969 |
| 6.242.2 Define Documentation . . . . .             | 969 |
| 6.243MindSensors NXTPowerMeter commands . . . . .  | 971 |
| 6.243.1 Detailed Description . . . . .             | 971 |
| 6.243.2 Define Documentation . . . . .             | 971 |
| 6.244MindSensors NXTSumoEyes constants . . . . .   | 971 |
| 6.244.1 Detailed Description . . . . .             | 971 |
| 6.244.2 Define Documentation . . . . .             | 971 |
| 6.245MindSensors NXTLineLeader constants . . . . . | 972 |
| 6.245.1 Detailed Description . . . . .             | 972 |
| 6.246MindSensors NXTLineLeader registers . . . . . | 973 |
| 6.246.1 Detailed Description . . . . .             | 973 |
| 6.246.2 Define Documentation . . . . .             | 973 |
| 6.247MindSensors NXTLineLeader commands . . . . .  | 975 |
| 6.247.1 Detailed Description . . . . .             | 975 |
| 6.247.2 Define Documentation . . . . .             | 976 |
| 6.248Codatex device constants . . . . .            | 977 |
| 6.248.1 Detailed Description . . . . .             | 977 |
| 6.249Codatex RFID sensor constants . . . . .       | 977 |
| 6.249.1 Detailed Description . . . . .             | 977 |
| 6.249.2 Define Documentation . . . . .             | 978 |
| 6.250Codatex RFID sensor modes . . . . .           | 978 |

---

|   |     |
|---|-----|
| 6.250.1 Detailed Description . . . . .                                    | 978 |
| 6.250.2 Define Documentation . . . . .                                    | 978 |
| 6.251 Dexter Industries device constants . . . . .                        | 979 |
| 6.251.1 Detailed Description . . . . .                                    | 979 |
| 6.252 Dexter Industries GPS sensor constants . . . . .                    | 979 |
| 6.252.1 Detailed Description . . . . .                                    | 980 |
| 6.252.2 Define Documentation . . . . .                                    | 980 |
| 6.253 Dexter Industries IMU sensor constants . . . . .                    | 981 |
| 6.253.1 Detailed Description . . . . .                                    | 983 |
| 6.253.2 Define Documentation . . . . .                                    | 983 |
| 6.254 Dexter Industries IMU Gyro register constants . . . . .             | 983 |
| 6.254.1 Detailed Description . . . . .                                    | 984 |
| 6.254.2 Define Documentation . . . . .                                    | 984 |
| 6.255 Dexter Industries IMU Gyro control register 1 constants . . . . .   | 987 |
| 6.255.1 Detailed Description . . . . .                                    | 988 |
| 6.255.2 Define Documentation . . . . .                                    | 988 |
| 6.256 Dexter Industries IMU Gyro control register 2 constants . . . . .   | 989 |
| 6.256.1 Detailed Description . . . . .                                    | 990 |
| 6.256.2 Define Documentation . . . . .                                    | 990 |
| 6.257 Dexter Industries IMU Gyro control register 3 constants . . . . .   | 991 |
| 6.257.1 Detailed Description . . . . .                                    | 992 |
| 6.257.2 Define Documentation . . . . .                                    | 992 |
| 6.258 Dexter Industries IMU Gyro control register 4 constants . . . . .   | 993 |
| 6.258.1 Detailed Description . . . . .                                    | 993 |
| 6.258.2 Define Documentation . . . . .                                    | 993 |
| 6.259 Dexter Industries IMU Gyro control register 5 constants . . . . .   | 994 |
| 6.259.1 Detailed Description . . . . .                                    | 994 |
| 6.259.2 Define Documentation . . . . .                                    | 994 |
| 6.260 Dexter Industries IMU Gyro FIFO control register onstants . . . . . | 995 |
| 6.260.1 Detailed Description . . . . .                                    | 996 |
| 6.260.2 Define Documentation . . . . .                                    | 996 |

---

|  |      |
|--|------|
| 6.261 Dexter Industries IMU Gyro status register constants . . . . .               | 997  |
| 6.261.1 Detailed Description . . . . .   | 997  |
| 6.261.2 Define Documentation . . . . .   | 997  |
| 6.262 Dexter Industries IMU Accelerometer register constants . . . . .             | 998  |
| 6.262.1 Detailed Description . . . . .   | 999  |
| 6.262.2 Define Documentation . . . . .   | 999  |
| 6.263 Dexter Industries IMU Accelerometer status register constants . . . . .      | 1002 |
| 6.263.1 Detailed Description . . . . .   | 1002 |
| 6.263.2 Define Documentation . . . . .   | 1002 |
| 6.264 Dexter Industries IMU Accelerometer mode control register constants          | 1003 |
| 6.264.1 Detailed Description . . . . .   | 1003 |
| 6.264.2 Define Documentation . . . . .   | 1003 |
| 6.265 Dexter Industries IMU Accelerometer interrupt latch reset register constants | 1004 |
| 6.265.1 Detailed Description . . . . .   | 1004 |
| 6.265.2 Define Documentation . . . . .   | 1005 |
| 6.266 Dexter Industries IMU Accelerometer control register 1 constants . . . . .   | 1005 |
| 6.266.1 Detailed Description . . . . .   | 1005 |
| 6.266.2 Define Documentation . . . . .   | 1005 |
| 6.267 Dexter Industries IMU Accelerometer control register 2 constants . . . . .   | 1006 |
| 6.267.1 Detailed Description . . . . .   | 1007 |
| 6.267.2 Define Documentation . . . . .   | 1007 |
| 6.268 Microinfinity device constants . . . . .                                     | 1007 |
| 6.268.1 Detailed Description . . . . .   | 1007 |
| 6.269 Microinfinity CruizCore XG1300L sensor constants . . . . .                   | 1008 |
| 6.269.1 Detailed Description . . . . .   | 1008 |
| 6.269.2 Define Documentation . . . . .   | 1008 |
| 6.270 Microinfinity CruizCore XG1300L . . . . .                                    | 1009 |
| 6.270.1 Detailed Description . . . . .   | 1010 |
| 6.270.2 Define Documentation . . . . .   | 1010 |
| 6.271 Data type limits . . . . .   | 1010 |

|  |             |
|--|-------------|
| 6.271.1 Detailed Description . . . . .               | 1011        |
| 6.271.2 Define Documentation . . . . .               | 1011        |
| 6.272 Graphics library begin modes . . . . .         | 1013        |
| 6.272.1 Detailed Description . . . . .               | 1013        |
| 6.272.2 Define Documentation . . . . .               | 1013        |
| 6.273 Graphics library actions . . . . .             | 1014        |
| 6.273.1 Detailed Description . . . . .               | 1014        |
| 6.273.2 Define Documentation . . . . .               | 1014        |
| 6.274 Graphics library settings . . . . .            | 1016        |
| 6.274.1 Detailed Description . . . . .               | 1016        |
| 6.274.2 Define Documentation . . . . .               | 1016        |
| 6.275 Graphics library cull mode . . . . .           | 1017        |
| 6.275.1 Detailed Description . . . . .               | 1017        |
| 6.275.2 Define Documentation . . . . .               | 1017        |
| <b>7 Data Structure Documentation</b>                | <b>1018</b> |
| 7.1 ColorSensorReadType Struct Reference . . . . .   | 1018        |
| 7.1.1 Detailed Description . . . . .                 | 1018        |
| 7.1.2 Field Documentation . . . . .                  | 1018        |
| 7.2 CommBTCheckStatusType Struct Reference . . . . . | 1019        |
| 7.2.1 Detailed Description . . . . .                 | 1020        |
| 7.2.2 Field Documentation . . . . .                  | 1020        |
| 7.3 CommBTConnectionType Struct Reference . . . . .  | 1020        |
| 7.3.1 Detailed Description . . . . .                 | 1021        |
| 7.3.2 Field Documentation . . . . .                  | 1021        |
| 7.4 CommBTOnOffType Struct Reference . . . . .       | 1022        |
| 7.4.1 Detailed Description . . . . .                 | 1022        |
| 7.4.2 Field Documentation . . . . .                  | 1022        |
| 7.5 CommBTWriteType Struct Reference . . . . .       | 1023        |
| 7.5.1 Detailed Description . . . . .                 | 1023        |
| 7.5.2 Field Documentation . . . . .                  | 1023        |

|        |  |      |
|--------|--|------|
| 7.6    | CommExecuteFunctionType Struct Reference | 1024 |
| 7.6.1  | Detailed Description                     | 1024 |
| 7.6.2  | Field Documentation                      | 1025 |
| 7.7    | CommHSCheckStatusType Struct Reference   | 1026 |
| 7.7.1  | Detailed Description                     | 1027 |
| 7.7.2  | Field Documentation                      | 1027 |
| 7.8    | CommHSControlType Struct Reference       | 1027 |
| 7.8.1  | Detailed Description                     | 1028 |
| 7.8.2  | Field Documentation                      | 1028 |
| 7.9    | CommHSReadWriteType Struct Reference     | 1029 |
| 7.9.1  | Detailed Description                     | 1029 |
| 7.9.2  | Field Documentation                      | 1029 |
| 7.10   | CommLSCheckStatusType Struct Reference   | 1030 |
| 7.10.1 | Detailed Description                     | 1030 |
| 7.10.2 | Field Documentation                      | 1031 |
| 7.11   | CommLSReadType Struct Reference          | 1031 |
| 7.11.1 | Detailed Description                     | 1032 |
| 7.11.2 | Field Documentation                      | 1032 |
| 7.12   | CommLSWriteExType Struct Reference       | 1033 |
| 7.12.1 | Detailed Description                     | 1033 |
| 7.12.2 | Field Documentation                      | 1033 |
| 7.13   | CommLSWriteType Struct Reference         | 1034 |
| 7.13.1 | Detailed Description                     | 1035 |
| 7.13.2 | Field Documentation                      | 1035 |
| 7.14   | ComputeCalibValueType Struct Reference   | 1036 |
| 7.14.1 | Detailed Description                     | 1036 |
| 7.14.2 | Field Documentation                      | 1036 |
| 7.15   | DatalogGetTimesType Struct Reference     | 1037 |
| 7.15.1 | Detailed Description                     | 1038 |
| 7.15.2 | Field Documentation                      | 1038 |
| 7.16   | DatalogWriteType Struct Reference        | 1038 |

---

|  |      |
|--|------|
| 7.16.1 Detailed Description . . . . .                      | 1039 |
| 7.16.2 Field Documentation . . . . .                       | 1039 |
| 7.17 DisplayExecuteFunctionType Struct Reference . . . . . | 1039 |
| 7.17.1 Detailed Description . . . . .                      | 1040 |
| 7.17.2 Field Documentation . . . . .                       | 1040 |
| 7.18 div_t Struct Reference . . . . .                      | 1042 |
| 7.18.1 Detailed Description . . . . .                      | 1042 |
| 7.18.2 Field Documentation . . . . .                       | 1042 |
| 7.19 DrawCircleType Struct Reference . . . . .             | 1043 |
| 7.19.1 Detailed Description . . . . .                      | 1043 |
| 7.19.2 Field Documentation . . . . .                       | 1043 |
| 7.20 DrawEllipseType Struct Reference . . . . .            | 1044 |
| 7.20.1 Detailed Description . . . . .                      | 1044 |
| 7.20.2 Field Documentation . . . . .                       | 1045 |
| 7.21 DrawFontType Struct Reference . . . . .               | 1046 |
| 7.21.1 Detailed Description . . . . .                      | 1046 |
| 7.21.2 Field Documentation . . . . .                       | 1046 |
| 7.22 DrawGraphicArrayType Struct Reference . . . . .       | 1047 |
| 7.22.1 Detailed Description . . . . .                      | 1048 |
| 7.22.2 Field Documentation . . . . .                       | 1048 |
| 7.23 DrawGraphicType Struct Reference . . . . .            | 1049 |
| 7.23.1 Detailed Description . . . . .                      | 1049 |
| 7.23.2 Field Documentation . . . . .                       | 1049 |
| 7.24 DrawLineType Struct Reference . . . . .               | 1050 |
| 7.24.1 Detailed Description . . . . .                      | 1051 |
| 7.24.2 Field Documentation . . . . .                       | 1051 |
| 7.25 DrawPointType Struct Reference . . . . .              | 1052 |
| 7.25.1 Detailed Description . . . . .                      | 1052 |
| 7.25.2 Field Documentation . . . . .                       | 1052 |
| 7.26 DrawPolygonType Struct Reference . . . . .            | 1053 |
| 7.26.1 Detailed Description . . . . .                      | 1053 |

---

|        |  |      |
|--------|--|------|
| 7.26.2 | Field Documentation                    | 1054 |
| 7.27   | DrawRectType Struct Reference          | 1054 |
| 7.27.1 | Detailed Description                   | 1055 |
| 7.27.2 | Field Documentation                    | 1055 |
| 7.28   | DrawTextType Struct Reference          | 1056 |
| 7.28.1 | Detailed Description                   | 1056 |
| 7.28.2 | Field Documentation                    | 1056 |
| 7.29   | FileCloseType Struct Reference         | 1057 |
| 7.29.1 | Detailed Description                   | 1057 |
| 7.29.2 | Field Documentation                    | 1058 |
| 7.30   | FileDeleteType Struct Reference        | 1058 |
| 7.30.1 | Detailed Description                   | 1058 |
| 7.30.2 | Field Documentation                    | 1059 |
| 7.31   | FileFindType Struct Reference          | 1059 |
| 7.31.1 | Detailed Description                   | 1059 |
| 7.31.2 | Field Documentation                    | 1060 |
| 7.32   | FileOpenType Struct Reference          | 1060 |
| 7.32.1 | Detailed Description                   | 1061 |
| 7.32.2 | Field Documentation                    | 1061 |
| 7.33   | FileReadWriteType Struct Reference     | 1062 |
| 7.33.1 | Detailed Description                   | 1062 |
| 7.33.2 | Field Documentation                    | 1062 |
| 7.34   | FileRenameType Struct Reference        | 1063 |
| 7.34.1 | Detailed Description                   | 1064 |
| 7.34.2 | Field Documentation                    | 1064 |
| 7.35   | FileResizeType Struct Reference        | 1065 |
| 7.35.1 | Detailed Description                   | 1065 |
| 7.35.2 | Field Documentation                    | 1065 |
| 7.36   | FileResolveHandleType Struct Reference | 1066 |
| 7.36.1 | Detailed Description                   | 1066 |
| 7.36.2 | Field Documentation                    | 1066 |

---

|  |      |
|--|------|
| 7.37 FileSeekType Struct Reference . . . . .         | 1067 |
| 7.37.1 Detailed Description . . . . .                | 1068 |
| 7.37.2 Field Documentation . . . . .                 | 1068 |
| 7.38 FileTellType Struct Reference . . . . .         | 1069 |
| 7.38.1 Detailed Description . . . . .                | 1069 |
| 7.38.2 Field Documentation . . . . .                 | 1069 |
| 7.39 GetStartTickType Struct Reference . . . . .     | 1070 |
| 7.39.1 Detailed Description . . . . .                | 1070 |
| 7.39.2 Field Documentation . . . . .                 | 1070 |
| 7.40 InputPinFunctionType Struct Reference . . . . . | 1070 |
| 7.40.1 Detailed Description . . . . .                | 1071 |
| 7.40.2 Field Documentation . . . . .                 | 1071 |
| 7.41 InputValuesType Struct Reference . . . . .      | 1072 |
| 7.41.1 Detailed Description . . . . .                | 1073 |
| 7.41.2 Field Documentation . . . . .                 | 1073 |
| 7.42 IOMapReadByIDType Struct Reference . . . . .    | 1074 |
| 7.42.1 Detailed Description . . . . .                | 1074 |
| 7.42.2 Field Documentation . . . . .                 | 1075 |
| 7.43 IOMapReadType Struct Reference . . . . .        | 1076 |
| 7.43.1 Detailed Description . . . . .                | 1076 |
| 7.43.2 Field Documentation . . . . .                 | 1076 |
| 7.44 IOMapWriteByIDType Struct Reference . . . . .   | 1077 |
| 7.44.1 Detailed Description . . . . .                | 1078 |
| 7.44.2 Field Documentation . . . . .                 | 1078 |
| 7.45 IOMapWriteType Struct Reference . . . . .       | 1079 |
| 7.45.1 Detailed Description . . . . .                | 1079 |
| 7.45.2 Field Documentation . . . . .                 | 1079 |
| 7.46 JoystickMessageType Struct Reference . . . . .  | 1080 |
| 7.46.1 Detailed Description . . . . .                | 1081 |
| 7.46.2 Field Documentation . . . . .                 | 1081 |
| 7.47 KeepAliveType Struct Reference . . . . .        | 1082 |

---

|   |      |
|---|------|
| 7.47.1 Detailed Description . . . . .                     | 1082 |
| 7.47.2 Field Documentation . . . . .                      | 1083 |
| 7.48 Idiv_t Struct Reference . . . . .                    | 1083 |
| 7.48.1 Detailed Description . . . . .                     | 1083 |
| 7.48.2 Field Documentation . . . . .                      | 1084 |
| 7.49 ListFilesType Struct Reference . . . . .             | 1084 |
| 7.49.1 Detailed Description . . . . .                     | 1084 |
| 7.49.2 Field Documentation . . . . .                      | 1085 |
| 7.50 LoaderExecuteFunctionType Struct Reference . . . . . | 1085 |
| 7.50.1 Detailed Description . . . . .                     | 1086 |
| 7.50.2 Field Documentation . . . . .                      | 1088 |
| 7.51 LocationType Struct Reference . . . . .              | 1088 |
| 7.51.1 Detailed Description . . . . .                     | 1089 |
| 7.51.2 Field Documentation . . . . .                      | 1089 |
| 7.52 MemoryManagerType Struct Reference . . . . .         | 1090 |
| 7.52.1 Detailed Description . . . . .                     | 1090 |
| 7.52.2 Field Documentation . . . . .                      | 1090 |
| 7.53 MessageReadType Struct Reference . . . . .           | 1091 |
| 7.53.1 Detailed Description . . . . .                     | 1091 |
| 7.53.2 Field Documentation . . . . .                      | 1092 |
| 7.54 MessageWriteType Struct Reference . . . . .          | 1092 |
| 7.54.1 Detailed Description . . . . .                     | 1093 |
| 7.54.2 Field Documentation . . . . .                      | 1093 |
| 7.55 OutputStateType Struct Reference . . . . .           | 1094 |
| 7.55.1 Detailed Description . . . . .                     | 1094 |
| 7.55.2 Field Documentation . . . . .                      | 1094 |
| 7.56 RandomExType Struct Reference . . . . .              | 1095 |
| 7.56.1 Detailed Description . . . . .                     | 1096 |
| 7.56.2 Field Documentation . . . . .                      | 1096 |
| 7.57 RandomNumberType Struct Reference . . . . .          | 1096 |
| 7.57.1 Detailed Description . . . . .                     | 1096 |

---

|        |                                       |      |
|--------|---------------------------------------|------|
| 7.57.2 | Field Documentation                   | 1097 |
| 7.58   | ReadButtonType Struct Reference       | 1097 |
| 7.58.1 | Detailed Description                  | 1097 |
| 7.58.2 | Field Documentation                   | 1098 |
| 7.59   | ReadLastResponseType Struct Reference | 1098 |
| 7.59.1 | Detailed Description                  | 1099 |
| 7.59.2 | Field Documentation                   | 1099 |
| 7.60   | ReadSemDataType Struct Reference      | 1100 |
| 7.60.1 | Detailed Description                  | 1100 |
| 7.60.2 | Field Documentation                   | 1101 |
| 7.61   | SetScreenModeType Struct Reference    | 1101 |
| 7.61.1 | Detailed Description                  | 1101 |
| 7.61.2 | Field Documentation                   | 1102 |
| 7.62   | SetSleepTimeoutType Struct Reference  | 1102 |
| 7.62.1 | Detailed Description                  | 1102 |
| 7.62.2 | Field Documentation                   | 1103 |
| 7.63   | SizeType Struct Reference             | 1103 |
| 7.63.1 | Detailed Description                  | 1103 |
| 7.63.2 | Field Documentation                   | 1103 |
| 7.64   | SoundGetStateType Struct Reference    | 1104 |
| 7.64.1 | Detailed Description                  | 1104 |
| 7.64.2 | Field Documentation                   | 1104 |
| 7.65   | SoundPlayFileType Struct Reference    | 1105 |
| 7.65.1 | Detailed Description                  | 1105 |
| 7.65.2 | Field Documentation                   | 1105 |
| 7.66   | SoundPlayToneType Struct Reference    | 1106 |
| 7.66.1 | Detailed Description                  | 1107 |
| 7.66.2 | Field Documentation                   | 1107 |
| 7.67   | SoundSetStateType Struct Reference    | 1108 |
| 7.67.1 | Detailed Description                  | 1108 |
| 7.67.2 | Field Documentation                   | 1108 |

|          |   |             |
|----------|---|-------------|
| 7.68     | Tone Struct Reference . . . . .                     | 1109        |
| 7.68.1   | Detailed Description . . . . .                      | 1109        |
| 7.68.2   | Field Documentation . . . . .                       | 1109        |
| 7.69     | UpdateCalibCacheInfoType Struct Reference . . . . . | 1110        |
| 7.69.1   | Detailed Description . . . . .                      | 1110        |
| 7.69.2   | Field Documentation . . . . .                       | 1110        |
| 7.70     | VectorType Struct Reference . . . . .               | 1111        |
| 7.70.1   | Detailed Description . . . . .                      | 1112        |
| 7.70.2   | Field Documentation . . . . .                       | 1112        |
| 7.71     | WriteSemDataType Struct Reference . . . . .         | 1112        |
| 7.71.1   | Detailed Description . . . . .                      | 1113        |
| 7.71.2   | Field Documentation . . . . .                       | 1113        |
| 7.72     | XGPacketType Struct Reference . . . . .             | 1114        |
| 7.72.1   | Detailed Description . . . . .                      | 1114        |
| 7.72.2   | Field Documentation . . . . .                       | 1114        |
| <b>8</b> | <b>File Documentation</b>                           | <b>1115</b> |
| 8.1      | NBCCCommon.h File Reference . . . . .               | 1115        |
| 8.1.1    | Detailed Description . . . . .                      | 1162        |
| 8.1.2    | Define Documentation . . . . .                      | 1163        |
| 8.2      | NXCAPIDocs.h File Reference . . . . .               | 1388        |
| 8.2.1    | Detailed Description . . . . .                      | 1388        |
| 8.3      | NXCDefs.h File Reference . . . . .                  | 1389        |
| 8.3.1    | Detailed Description . . . . .                      | 1479        |
| 8.3.2    | Define Documentation . . . . .                      | 1480        |
| 8.3.3    | Function Documentation . . . . .                    | 1503        |
| <b>9</b> | <b>Example Documentation</b>                        | <b>1972</b> |
| 9.1      | alternating_tasks.nxc . . . . .                     | 1972        |
| 9.2      | ex_abort.nxc . . . . .                              | 1973        |
| 9.3      | ex_AbortFlag.nxc . . . . .                          | 1973        |
| 9.4      | ex_abs.nxc . . . . .                                | 1973        |

---

|      |                               |      |
|------|-------------------------------|------|
| 9.5  | ex_ACCLNxCalibrateX.nxc       | 1973 |
| 9.6  | ex_ACCLNxCalibrateXEnd.nxc    | 1974 |
| 9.7  | ex_ACCLNxCalibrateY.nxc       | 1974 |
| 9.8  | ex_ACCLNxCalibrateYEnd.nxc    | 1974 |
| 9.9  | ex_ACCLNxCalibrateZ.nxc       | 1974 |
| 9.10 | ex_ACCLNxCalibrateZEnd.nxc    | 1974 |
| 9.11 | ex_ACCLNxResetCalibration.nxc | 1974 |
| 9.12 | ex_ACCLNxSensitivity.nxc      | 1975 |
| 9.13 | ex_ACCLNxXOffset.nxc          | 1975 |
| 9.14 | ex_ACCLNxXRange.nxc           | 1975 |
| 9.15 | ex_ACCLNxYOffset.nxc          | 1975 |
| 9.16 | ex_ACCLNxYRange.nxc           | 1975 |
| 9.17 | ex_ACCLNxZOffset.nxc          | 1975 |
| 9.18 | ex_ACCLNxZRange.nxc           | 1975 |
| 9.19 | ex_acos.nxc                   | 1976 |
| 9.20 | ex_acosd.nxc                  | 1976 |
| 9.21 | ex_Acquire.nxc                | 1977 |
| 9.22 | ex_addressof.nxc              | 1977 |
| 9.23 | ex_addressofex.nxc            | 1978 |
| 9.24 | ex_ArrayBuild.nxc             | 1979 |
| 9.25 | ex_ArrayInit.nxc              | 1980 |
| 9.26 | ex_ArrayLen.nxc               | 1980 |
| 9.27 | ex_ArrayMax.nxc               | 1980 |
| 9.28 | ex_ArrayMean.nxc              | 1980 |
| 9.29 | ex_ArrayMin.nxc               | 1980 |
| 9.30 | ex_ArrayOp.nxc                | 1981 |
| 9.31 | ex_ArraySort.nxc              | 1981 |
| 9.32 | ex_ArrayStd.nxc               | 1982 |
| 9.33 | ex_ArraySubset.nxc            | 1982 |
| 9.34 | ex_ArraySum.nxc               | 1982 |
| 9.35 | ex_ArraySumSqr.nxc            | 1982 |

|      |                                  |      |
|------|----------------------------------|------|
| 9.36 | ex_asin.nxc                      | 1983 |
| 9.37 | ex_asind.nxc                     | 1983 |
| 9.38 | ex_atan.nxc                      | 1984 |
| 9.39 | ex_atan2.nxc                     | 1984 |
| 9.40 | ex_atan2d.nxc                    | 1985 |
| 9.41 | ex_atand.nxc                     | 1985 |
| 9.42 | ex_atof.nxc                      | 1986 |
| 9.43 | ex_atoi.nxc                      | 1986 |
| 9.44 | ex_atol.nxc                      | 1986 |
| 9.45 | ex_BatteryState.nxc              | 1987 |
| 9.46 | ex_bcd2dec.nxc                   | 1987 |
| 9.47 | ex_BluetoothState.nxc            | 1987 |
| 9.48 | ex_BluetoothStatus.nxc           | 1987 |
| 9.49 | ex_BluetoothWrite.nxc            | 1987 |
| 9.50 | ex_BrickDataBluecoreVersion.nxc  | 1987 |
| 9.51 | ex_BrickDataBtHardwareStatus.nxc | 1988 |
| 9.52 | ex_BrickDataBtStateStatus.nxc    | 1988 |
| 9.53 | ex_BrickDataName.nxc             | 1988 |
| 9.54 | ex_BrickDataTimeoutValue.nxc     | 1988 |
| 9.55 | ex_BTConnectionClass.nxc         | 1988 |
| 9.56 | ex_BTConnectionHandleNum.nxc     | 1988 |
| 9.57 | ex_BTConnectionLinkQuality.nxc   | 1989 |
| 9.58 | ex_BTConnectionName.nxc          | 1989 |
| 9.59 | ex_BTConnectionPinCode.nxc       | 1989 |
| 9.60 | ex_BTConnectionStreamStatus.nxc  | 1989 |
| 9.61 | ex_BTDeviceClass.nxc             | 1989 |
| 9.62 | ex_BTDeviceCount.nxc             | 1989 |
| 9.63 | ex_BTDeviceName.nxc              | 1990 |
| 9.64 | ex_BTDeviceNameCount.nxc         | 1990 |
| 9.65 | ex_BTDeviceStatus.nxc            | 1990 |
| 9.66 | ex_BTInputBufferInPtr.nxc        | 1990 |

|      |                                   |      |
|------|-----------------------------------|------|
| 9.67 | ex_BTInputBufferOutPtr.nxc        | 1990 |
| 9.68 | ex_BTOutputBufferInPtr.nxc        | 1990 |
| 9.69 | ex_BTOutputBufferOutPtr.nxc       | 1991 |
| 9.70 | ex_ButtonCount.nxc                | 1991 |
| 9.71 | ex_ButtonLongPressCount.nxc       | 1991 |
| 9.72 | ex_ButtonLongReleaseCount.nxc     | 1991 |
| 9.73 | ex_ButtonPressCount.nxc           | 1991 |
| 9.74 | ex_buttonpressed.nxc              | 1991 |
| 9.75 | ex_ButtonReleaseCount.nxc         | 1992 |
| 9.76 | ex_ButtonShortReleaseCount.nxc    | 1992 |
| 9.77 | ex_ButtonState.nxc                | 1992 |
| 9.78 | ex_ByteArrayToStr.nxc             | 1992 |
| 9.79 | ex_ByteArrayToStrEx.nxc           | 1992 |
| 9.80 | ex_ceil.nxc                       | 1993 |
| 9.81 | ex_CircleOut.nxc                  | 1993 |
| 9.82 | ex_clearline.nxc                  | 1993 |
| 9.83 | ex_ClearScreen.nxc                | 1993 |
| 9.84 | ex_ClearSensor.nxc                | 1994 |
| 9.85 | ex_CloseFile.nxc                  | 1994 |
| 9.86 | ex_coast.nxc                      | 1994 |
| 9.87 | ex_coastex.nxc                    | 1994 |
| 9.88 | ex_ColorADRaw.nxc                 | 1994 |
| 9.89 | ex_ColorBoolean.nxc               | 1994 |
| 9.90 | ex_ColorCalibration.nxc           | 1995 |
| 9.91 | ex_ColorCalibrationState.nxc      | 1995 |
| 9.92 | ex_ColorCalLimits.nxc             | 1995 |
| 9.93 | ex_ColorSensorRaw.nxc             | 1995 |
| 9.94 | ex_ColorSensorValue.nxc           | 1995 |
| 9.95 | ex_CommandFlags.nxc               | 1995 |
| 9.96 | ex_ConfigureTemperatureSensor.nxc | 1996 |
| 9.97 | ex_contrast.nxc                   | 1996 |

|  |      |
|--|------|
| 9.98 ex_copy.nxc . . . . .                         | 1996 |
| 9.99 ex_cosh.nxc . . . . .                         | 1996 |
| 9.100ex_CreateFile.nxc . . . . .                   | 1997 |
| 9.101ex_CreateFileLinear.nxc . . . . .             | 1997 |
| 9.102ex_CreateFileNonLinear.nxc . . . . .          | 1997 |
| 9.103ex_cstdio.nxc . . . . .                       | 1997 |
| 9.104ex_cstring.nxc . . . . .                      | 1998 |
| 9.105ex_ctype.nxc . . . . .                        | 1998 |
| 9.106ex_CurrentTick.nxc . . . . .                  | 1999 |
| 9.107ex_CustomSensorActiveStatus.nxc . . . . .     | 1999 |
| 9.108ex_CustomSensorPercentFullScale.nxc . . . . . | 1999 |
| 9.109ex_CustomSensorZeroOffset.nxc . . . . .       | 1999 |
| 9.110ex_DataMode.nxc . . . . .                     | 1999 |
| 9.111ex_delete_data_file.nxc . . . . .             | 2000 |
| 9.112ex_DeleteFile.nxc . . . . .                   | 2001 |
| 9.113ex_diaccl.nxc . . . . .                       | 2001 |
| 9.114ex_digps.nxc . . . . .                        | 2003 |
| 9.115ex_digyro.nxc . . . . .                       | 2003 |
| 9.116ex_dispfnout.nxc . . . . .                    | 2004 |
| 9.117ex_dispftout.nxc . . . . .                    | 2004 |
| 9.118ex_dispfunc.nxc . . . . .                     | 2005 |
| 9.119ex_dispgaout.nxc . . . . .                    | 2005 |
| 9.120ex_dispgaoutex.nxc . . . . .                  | 2007 |
| 9.121ex_dispgout.nxc . . . . .                     | 2012 |
| 9.122ex_dispgoutex.nxc . . . . .                   | 2012 |
| 9.123ex_DisplayDisplay.nxc . . . . .               | 2013 |
| 9.124ex_DisplayEraseMask.nxc . . . . .             | 2013 |
| 9.125ex_DisplayFlags.nxc . . . . .                 | 2013 |
| 9.126ex_displayfont.nxc . . . . .                  | 2013 |
| 9.127ex_DisplayTextLinesCenterFlags.nxc . . . . .  | 2014 |
| 9.128ex_DisplayUpdateMask.nxc . . . . .            | 2015 |

---

|   |      |
|---|------|
| 9.129ex_dispmisc.nxc . . . . .          | 2015 |
| 9.130ex_DISTNxDistance.nxc . . . . .    | 2015 |
| 9.131ex_DISTNxGP2D12.nxc . . . . .      | 2016 |
| 9.132ex_DISTNxGP2D120.nxc . . . . .     | 2016 |
| 9.133ex_DISTNxGP2YA02.nxc . . . . .     | 2016 |
| 9.134ex_DISTNxGP2YA21.nxc . . . . .     | 2016 |
| 9.135ex_DISTNxMaxDistance.nxc . . . . . | 2016 |
| 9.136ex_DISTNxMinDistance.nxc . . . . . | 2016 |
| 9.137ex_DISTNxModuleType.nxc . . . . .  | 2016 |
| 9.138ex_DISTNxNumPoints.nxc . . . . .   | 2017 |
| 9.139ex_DISTNxVoltage.nxc . . . . .     | 2017 |
| 9.140ex_div.nxc . . . . .               | 2017 |
| 9.141ex_EllipseOut.nxc . . . . .        | 2017 |
| 9.142ex_exp.nxc . . . . .               | 2017 |
| 9.143ex_fclose.nxc . . . . .            | 2018 |
| 9.144ex_feof.nxc . . . . .              | 2018 |
| 9.145ex_fflush.nxc . . . . .            | 2018 |
| 9.146ex_fgetc.nxc . . . . .             | 2018 |
| 9.147ex_fgets.nxc . . . . .             | 2018 |
| 9.148ex_file_system.nxc . . . . .       | 2018 |
| 9.149ex_findfirstfile.nxc . . . . .     | 2022 |
| 9.150ex_findnextfile.nxc . . . . .      | 2022 |
| 9.151ex_FirstTick.nxc . . . . .         | 2023 |
| 9.152ex_Flatten.nxc . . . . .           | 2023 |
| 9.153ex_FlattenVar.nxc . . . . .        | 2023 |
| 9.154ex_float.nxc . . . . .             | 2023 |
| 9.155ex_floor.nxc . . . . .             | 2024 |
| 9.156ex_Follows.nxc . . . . .           | 2024 |
| 9.157ex_fopen.nxc . . . . .             | 2024 |
| 9.158ex_ForceOff.nxc . . . . .          | 2024 |
| 9.159ex_FormatNum.nxc . . . . .         | 2024 |

|   |      |
|---|------|
| 9.160ex_fprintf.nxc . . . . .                 | 2024 |
| 9.161ex_fputc.nxc . . . . .                   | 2025 |
| 9.162ex_fputs.nxc . . . . .                   | 2025 |
| 9.163ex_frac.nxc . . . . .                    | 2025 |
| 9.164ex_FreeMemory.nxc . . . . .              | 2025 |
| 9.165ex_fseek.nxc . . . . .                   | 2025 |
| 9.166ex_ftell.nxc . . . . .                   | 2025 |
| 9.167ex_GetBrickDataAddress.nxc . . . . .     | 2026 |
| 9.168ex_GetBTCConnectionAddress.nxc . . . . . | 2026 |
| 9.169ex_GetBTDeviceAddress.nxc . . . . .      | 2026 |
| 9.170ex_GetBTInputBuffer.nxc . . . . .        | 2026 |
| 9.171ex_GetBTOutputBuffer.nxc . . . . .       | 2026 |
| 9.172ex_getc.nxc . . . . .                    | 2027 |
| 9.173ex_getchar.nxc . . . . .                 | 2027 |
| 9.174ex_GetDisplayNormal.nxc . . . . .        | 2027 |
| 9.175ex_GetDisplayPopup.nxc . . . . .         | 2027 |
| 9.176ex_GetHSInputBuffer.nxc . . . . .        | 2027 |
| 9.177ex_GetHSOutputBuffer.nxc . . . . .       | 2028 |
| 9.178ex_GetInput.nxc . . . . .                | 2028 |
| 9.179ex_GetLastResponseInfo.nxc . . . . .     | 2028 |
| 9.180ex_GetLSInputBuffer.nxc . . . . .        | 2028 |
| 9.181ex_GetLSOutputBuffer.nxc . . . . .       | 2028 |
| 9.182ex_getmemoryinfo.nxc . . . . .           | 2028 |
| 9.183ex_getoutput.nxc . . . . .               | 2030 |
| 9.184ex_GetUSBInputBuffer.nxc . . . . .       | 2030 |
| 9.185ex_GetUSBOutputBuffer.nxc . . . . .      | 2030 |
| 9.186ex_GetUSBPollBuffer.nxc . . . . .        | 2030 |
| 9.187ex_GraphicOut.nxc . . . . .              | 2030 |
| 9.188ex_GraphicOutEx.nxc . . . . .            | 2031 |
| 9.189ex_HSFlags.nxc . . . . .                 | 2031 |
| 9.190ex_HSInputBufferInPtr.nxc . . . . .      | 2031 |

---

|  |      |
|--|------|
| 9.191ex_HSInputBufferOutPtr.nxc . . . . .  | 2031 |
| 9.192ex_HSMMode.nxc . . . . .              | 2031 |
| 9.193ex_HSOutputBufferInPtr.nxc . . . . .  | 2031 |
| 9.194ex_HSOutputBufferOutPtr.nxc . . . . . | 2032 |
| 9.195ex_HSSpeed.nxc . . . . .              | 2032 |
| 9.196ex_HSState.nxc . . . . .              | 2032 |
| 9.197ex_HTGyroTest.nxc . . . . .           | 2032 |
| 9.198ex_HTIRTrain.nxc . . . . .            | 2033 |
| 9.199ex_HTPFComboDirect.nxc . . . . .      | 2034 |
| 9.200ex_HTPFComboPWM.nxc . . . . .         | 2034 |
| 9.201ex_HTPFRawOutput.nxc . . . . .        | 2034 |
| 9.202ex_HTPFRepeat.nxc . . . . .           | 2034 |
| 9.203ex_HTPFSingleOutputCST.nxc . . . . .  | 2034 |
| 9.204ex_HTPFSingleOutputPWM.nxc . . . . .  | 2034 |
| 9.205ex_HTPFSinglePin.nxc . . . . .        | 2034 |
| 9.206ex_HTPFTrain.nxc . . . . .            | 2035 |
| 9.207ex_HTRCXAddToDatalog.nxc . . . . .    | 2035 |
| 9.208ex_HTRCXBatteryLevel.nxc . . . . .    | 2035 |
| 9.209ex_HTRCXClearAllEvents.nxc . . . . .  | 2035 |
| 9.210ex_HTRCXClearCounter.nxc . . . . .    | 2035 |
| 9.211ex_HTRCXClearMsg.nxc . . . . .        | 2035 |
| 9.212ex_HTRCXClearSensor.nxc . . . . .     | 2036 |
| 9.213ex_HTRCXClearSound.nxc . . . . .      | 2036 |
| 9.214ex_HTRCXClearTimer.nxc . . . . .      | 2036 |
| 9.215ex_HTRCXCreateDatalog.nxc . . . . .   | 2036 |
| 9.216ex_HTRCXDecCounter.nxc . . . . .      | 2036 |
| 9.217ex_HTRCXDeleteSub.nxc . . . . .       | 2036 |
| 9.218ex_HTRCXDeleteSubs.nxc . . . . .      | 2037 |
| 9.219ex_HTRCXDeleteTask.nxc . . . . .      | 2037 |
| 9.220ex_HTRCXDeleteTasks.nxc . . . . .     | 2037 |
| 9.221ex_HTRCXDisableOutput.nxc . . . . .   | 2037 |

---

|   |      |
|---|------|
| 9.222ex_HTRCXEnableOutput.nxc . . . . .       | 2037 |
| 9.223ex_HTRCXEvent.nxc . . . . .              | 2037 |
| 9.224ex_HTRCXFloat.nxc . . . . .              | 2038 |
| 9.225ex_HTRCXFwd.nxc . . . . .                | 2038 |
| 9.226ex_HTRCXIncCounter.nxc . . . . .         | 2038 |
| 9.227ex_HTRCXInvertOutput.nxc . . . . .       | 2038 |
| 9.228ex_HTRCXMuteSound.nxc . . . . .          | 2038 |
| 9.229ex_HTRCXObvertOutput.nxc . . . . .       | 2038 |
| 9.230ex_HTRCXOff.nxc . . . . .                | 2039 |
| 9.231ex_HTRCXOn.nxc . . . . .                 | 2039 |
| 9.232ex_HTRCXOnFor.nxc . . . . .              | 2039 |
| 9.233ex_HTRCXOnFwd.nxc . . . . .              | 2039 |
| 9.234ex_HTRCXOnRev.nxc . . . . .              | 2039 |
| 9.235ex_HTRCXPBTurnOff.nxc . . . . .          | 2039 |
| 9.236ex_HTRCXPing.nxc . . . . .               | 2040 |
| 9.237ex_HTRCXPlaySound.nxc . . . . .          | 2040 |
| 9.238ex_HTRCXPlayTone.nxc . . . . .           | 2040 |
| 9.239ex_HTRCXPlayToneVar.nxc . . . . .        | 2040 |
| 9.240ex_HTRCXPoll.nxc . . . . .               | 2040 |
| 9.241ex_HTRCXPollMemory.nxc . . . . .         | 2040 |
| 9.242ex_HTRCXRemote.nxc . . . . .             | 2041 |
| 9.243ex_HTRCXRev.nxc . . . . .                | 2041 |
| 9.244ex_HTRCXSelectDisplay.nxc . . . . .      | 2041 |
| 9.245ex_HTRCXSelectProgram.nxc . . . . .      | 2041 |
| 9.246ex_HTRCXSendSerial.nxc . . . . .         | 2041 |
| 9.247ex_HTRCXSetDirection.nxc . . . . .       | 2041 |
| 9.248ex_HTRCXSetEvent.nxc . . . . .           | 2042 |
| 9.249ex_HTRCXSetGlobalDirection.nxc . . . . . | 2042 |
| 9.250ex_HTRCXSetGlobalOutput.nxc . . . . .    | 2042 |
| 9.251ex_HTRCXSetIRLinkPort.nxc . . . . .      | 2042 |
| 9.252ex_HTRCXSetMaxPower.nxc . . . . .        | 2042 |

---

|   |      |
|---|------|
| 9.253ex_HTRCXSetMessage.nxc . . . . .             | 2042 |
| 9.254ex_HTRCXSetOutput.nxc . . . . .              | 2043 |
| 9.255ex_HTRCXSetPower.nxc . . . . .               | 2043 |
| 9.256ex_HTRCXSetPriority.nxc . . . . .            | 2043 |
| 9.257ex_HTRCXSetSensorMode.nxc . . . . .          | 2043 |
| 9.258ex_HTRCXSetSensorType.nxc . . . . .          | 2043 |
| 9.259ex_HTRCXSetSleepTime.nxc . . . . .           | 2043 |
| 9.260ex_HTRCXSetTxPower.nxc . . . . .             | 2044 |
| 9.261ex_HTRCXSetWatch.nxc . . . . .               | 2044 |
| 9.262ex_HTRCXStartTask.nxc . . . . .              | 2044 |
| 9.263ex_HTRCXStopAllTasks.nxc . . . . .           | 2044 |
| 9.264ex_HTRCXStopTask.nxc . . . . .               | 2044 |
| 9.265ex_HTRCXToggle.nxc . . . . .                 | 2044 |
| 9.266ex_HTRCXUnmuteSound.nxc . . . . .            | 2045 |
| 9.267ex_HTSscoutCalibrateSensor.nxc . . . . .     | 2045 |
| 9.268ex_HTSscoutMuteSound.nxc . . . . .           | 2045 |
| 9.269ex_HTSscoutSelectSounds.nxc . . . . .        | 2045 |
| 9.270ex_HTSscoutSendVLL.nxc . . . . .             | 2045 |
| 9.271ex_HTSscoutSetEventFeedback.nxc . . . . .    | 2045 |
| 9.272ex_HTSscoutSetLight.nxc . . . . .            | 2046 |
| 9.273ex_HTSscoutSetScoutMode.nxc . . . . .        | 2046 |
| 9.274ex_HTSscoutSetSensorClickTime.nxc . . . . .  | 2046 |
| 9.275ex_HTSscoutSetSensorHysteresis.nxc . . . . . | 2046 |
| 9.276ex_HTSscoutSetSensorLowerLimit.nxc . . . . . | 2046 |
| 9.277ex_HTSscoutSetSensorUpperLimit.nxc . . . . . | 2046 |
| 9.278ex_HTSscoutUnmuteSound.nxc . . . . .         | 2047 |
| 9.279ex_I2CBytes.nxc . . . . .                    | 2047 |
| 9.280ex_I2CBytesReady.nxc . . . . .               | 2047 |
| 9.281ex_I2CCheckStatus.nxc . . . . .              | 2047 |
| 9.282ex_i2cdeviceid.nxc . . . . .                 | 2047 |
| 9.283ex_i2cdeviceinfo.nxc . . . . .               | 2048 |

---

|  |      |
|--|------|
| 9.284ex_I2CRead.nxc . . . . .                | 2048 |
| 9.285ex_I2CSendCommand.nxc . . . . .         | 2048 |
| 9.286ex_I2CStatus.nxc . . . . .              | 2048 |
| 9.287ex_i2cvendorid.nxc . . . . .            | 2048 |
| 9.288ex_i2cversion.nxc . . . . .             | 2049 |
| 9.289ex_I2CWrite.nxc . . . . .               | 2049 |
| 9.290ex_isalnum.nxc . . . . .                | 2049 |
| 9.291ex_isalpha.nxc . . . . .                | 2049 |
| 9.292ex_iscntrl.nxc . . . . .                | 2049 |
| 9.293ex_isdigit.nxc . . . . .                | 2050 |
| 9.294ex_isgraph.nxc . . . . .                | 2050 |
| 9.295ex_islower.nxc . . . . .                | 2050 |
| 9.296ex_isnan.nxc . . . . .                  | 2050 |
| 9.297ex_isprint.nxc . . . . .                | 2050 |
| 9.298ex_ispunct.nxc . . . . .                | 2051 |
| 9.299ex_isspace.nxc . . . . .                | 2051 |
| 9.300ex_isupper.nxc . . . . .                | 2051 |
| 9.301ex_isxdigit.nxc . . . . .               | 2051 |
| 9.302ex_joystickmsg.nxc . . . . .            | 2051 |
| 9.303ex_labs.nxc . . . . .                   | 2052 |
| 9.304ex_ldiv.nxc . . . . .                   | 2052 |
| 9.305ex_leftstr.nxc . . . . .                | 2053 |
| 9.306ex_LineOut.nxc . . . . .                | 2053 |
| 9.307ex_log.nxc . . . . .                    | 2053 |
| 9.308ex_log10.nxc . . . . .                  | 2053 |
| 9.309ex_LongAbort.nxc . . . . .              | 2053 |
| 9.310ex_LowLevelModuleRoutines.nxc . . . . . | 2054 |
| 9.311ex_LowspeedBytesReady.nxc . . . . .     | 2054 |
| 9.312ex_LowspeedCheckStatus.nxc . . . . .    | 2055 |
| 9.313ex_LowspeedRead.nxc . . . . .           | 2055 |
| 9.314ex_LowspeedStatus.nxc . . . . .         | 2055 |

---

|   |      |
|---|------|
| 9.315ex_LowspeedWrite.nxc . . . . .           | 2055 |
| 9.316ex_LSChannelState.nxc . . . . .          | 2055 |
| 9.317ex_LSErrorType.nxc . . . . .             | 2055 |
| 9.318ex_LSInputBufferBytesToRx.nxc . . . . .  | 2056 |
| 9.319ex_LSInputBufferInPtr.nxc . . . . .      | 2056 |
| 9.320ex_LSInputBufferOutPtr.nxc . . . . .     | 2056 |
| 9.321ex_LSMode.nxc . . . . .                  | 2056 |
| 9.322ex_LSNorestartOnRead.nxc . . . . .       | 2056 |
| 9.323ex_LSOutputBufferBytesToRx.nxc . . . . . | 2056 |
| 9.324ex_LSOutputBufferInPtr.nxc . . . . .     | 2057 |
| 9.325ex_LSOutputBufferOutPtr.nxc . . . . .    | 2057 |
| 9.326ex_LSSpeed.nxc . . . . .                 | 2057 |
| 9.327ex_LSState.nxc . . . . .                 | 2057 |
| 9.328ex_memcmp.nxc . . . . .                  | 2057 |
| 9.329ex_memcpy.nxc . . . . .                  | 2058 |
| 9.330ex_memmove.nxc . . . . .                 | 2058 |
| 9.331ex_midstr.nxc . . . . .                  | 2058 |
| 9.332ex_motoractualspeed.nxc . . . . .        | 2058 |
| 9.333ex_motorblocktachocount.nxc . . . . .    | 2058 |
| 9.334ex_motormode.nxc . . . . .               | 2058 |
| 9.335ex_motoroutputoptions.nxc . . . . .      | 2059 |
| 9.336ex_motoroverload.nxc . . . . .           | 2059 |
| 9.337ex_motorpower.nxc . . . . .              | 2059 |
| 9.338ex_motorpwnfreq.nxc . . . . .            | 2059 |
| 9.339ex_motorregdvalue.nxc . . . . .          | 2059 |
| 9.340ex_motorregivalue.nxc . . . . .          | 2059 |
| 9.341ex_motorregpvalue.nxc . . . . .          | 2060 |
| 9.342ex_motorregulation.nxc . . . . .         | 2060 |
| 9.343ex_motorrotationcount.nxc . . . . .      | 2060 |
| 9.344ex_motorrynstate.nxc . . . . .           | 2060 |
| 9.345ex_motortachocount.nxc . . . . .         | 2060 |

---

|   |      |
|---|------|
| 9.346ex_motortacholimit.nxc . . . . .     | 2060 |
| 9.347ex_motorturnratio.nxc . . . . .      | 2060 |
| 9.348ex_MSADPAOff.nxc . . . . .           | 2061 |
| 9.349ex_MSADPAOn.nxc . . . . .            | 2061 |
| 9.350ex_MSDeenergize.nxc . . . . .        | 2061 |
| 9.351ex_MSEnergize.nxc . . . . .          | 2061 |
| 9.352ex_MSIRTrain.nxc . . . . .           | 2061 |
| 9.353ex_MSPFComboDirect.nxc . . . . .     | 2061 |
| 9.354ex_MSPFComboPWM.nxc . . . . .        | 2062 |
| 9.355ex_MSPFRawOutput.nxc . . . . .       | 2062 |
| 9.356ex_MSPFRepeat.nxc . . . . .          | 2062 |
| 9.357ex_MSPFSingleOutputCST.nxc . . . . . | 2062 |
| 9.358ex_MSPFSingleOutputPWM.nxc . . . . . | 2062 |
| 9.359ex_MSPFSinglePin.nxc . . . . .       | 2062 |
| 9.360ex_MSPFTrain.nxc . . . . .           | 2063 |
| 9.361ex_MSRCXAbsVar.nxc . . . . .         | 2063 |
| 9.362ex_MSRCXAddToDatalog.nxc . . . . .   | 2063 |
| 9.363ex_MSRCXAndVar.nxc . . . . .         | 2063 |
| 9.364ex_MSRCXBatteryLevel.nxc . . . . .   | 2063 |
| 9.365ex_MSRCXBoot.nxc . . . . .           | 2063 |
| 9.366ex_MSRCXCalibrateEvent.nxc . . . . . | 2064 |
| 9.367ex_MSRCXClearAllEvents.nxc . . . . . | 2064 |
| 9.368ex_MSRCXClearCounter.nxc . . . . .   | 2064 |
| 9.369ex_MSRCXClearMsg.nxc . . . . .       | 2064 |
| 9.370ex_MSRCXClearSensor.nxc . . . . .    | 2064 |
| 9.371ex_MSRCXClearSound.nxc . . . . .     | 2064 |
| 9.372ex_MSRCXClearTimer.nxc . . . . .     | 2065 |
| 9.373ex_MSRCXCreateDatalog.nxc . . . . .  | 2065 |
| 9.374ex_MSRCXDecCounter.nxc . . . . .     | 2065 |
| 9.375ex_MSRCXDeleteSub.nxc . . . . .      | 2065 |
| 9.376ex_MSRCXDeleteSubs.nxc . . . . .     | 2065 |

---

|  |      |
|--|------|
| 9.377ex_MSRCXDeleteTask.nxc . . . . .    | 2065 |
| 9.378ex_MSRCXDeleteTasks.nxc . . . . .   | 2066 |
| 9.379ex_MSRCXDisableOutput.nxc . . . . . | 2066 |
| 9.380ex_MSRCXDivVar.nxc . . . . .        | 2066 |
| 9.381ex_MSRCXEnableOutput.nxc . . . . .  | 2066 |
| 9.382ex_MSRCXEvent.nxc . . . . .         | 2066 |
| 9.383ex_MSRCXFloat.nxc . . . . .         | 2066 |
| 9.384ex_MSRCXFwd.nxc . . . . .           | 2067 |
| 9.385ex_MSRCXIncCounter.nxc . . . . .    | 2067 |
| 9.386ex_MSRCXInvertOutput.nxc . . . . .  | 2067 |
| 9.387ex_MSRCXMulVar.nxc . . . . .        | 2067 |
| 9.388ex_MSRCXMuteSound.nxc . . . . .     | 2067 |
| 9.389ex_MSRCXObvertOutput.nxc . . . . .  | 2067 |
| 9.390ex_MSRCXOff.nxc . . . . .           | 2068 |
| 9.391ex_MSRCXOn.nxc . . . . .            | 2068 |
| 9.392ex_MSRCXOnFor.nxc . . . . .         | 2068 |
| 9.393ex_MSRCXOnFwd.nxc . . . . .         | 2068 |
| 9.394ex_MSRCXOnRev.nxc . . . . .         | 2068 |
| 9.395ex_MSRCXOrVar.nxc . . . . .         | 2068 |
| 9.396ex_MSRCXPBTurnOff.nxc . . . . .     | 2069 |
| 9.397ex_MSRCXPing.nxc . . . . .          | 2069 |
| 9.398ex_MSRCXPlaySound.nxc . . . . .     | 2069 |
| 9.399ex_MSRCXPlayTone.nxc . . . . .      | 2069 |
| 9.400ex_MSRCXPlayToneVar.nxc . . . . .   | 2069 |
| 9.401ex_MSRCXPoll.nxc . . . . .          | 2069 |
| 9.402ex_MSRCXPollMemory.nxc . . . . .    | 2070 |
| 9.403ex_MSRCXRemote.nxc . . . . .        | 2070 |
| 9.404ex_MSRCXReset.nxc . . . . .         | 2070 |
| 9.405ex_MSRCXRev.nxc . . . . .           | 2070 |
| 9.406ex_MSRCXSelectDisplay.nxc . . . . . | 2070 |
| 9.407ex_MSRCXSelectProgram.nxc . . . . . | 2070 |

|   |      |
|---|------|
| 9.408ex_MSRCXSendSerial.nxc . . . . .         | 2071 |
| 9.409ex_MSRCXSet.nxc . . . . .                | 2071 |
| 9.410ex_MSRCXSetDirection.nxc . . . . .       | 2071 |
| 9.411ex_MSRCXSetEvent.nxc . . . . .           | 2071 |
| 9.412ex_MSRCXSetGlobalDirection.nxc . . . . . | 2071 |
| 9.413ex_MSRCXSetGlobalOutput.nxc . . . . .    | 2071 |
| 9.414ex_MSRCXSetMaxPower.nxc . . . . .        | 2072 |
| 9.415ex_MSRCXSetMessage.nxc . . . . .         | 2072 |
| 9.416ex_MSRCXSetNRLinkPort.nxc . . . . .      | 2072 |
| 9.417ex_MSRCXSetOutput.nxc . . . . .          | 2072 |
| 9.418ex_MSRCXSetPower.nxc . . . . .           | 2072 |
| 9.419ex_MSRCXSetPriority.nxc . . . . .        | 2072 |
| 9.420ex_MSRCXSetSensorMode.nxc . . . . .      | 2073 |
| 9.421ex_MSRCXSetSensorType.nxc . . . . .      | 2073 |
| 9.422ex_MSRCXSetSleepTime.nxc . . . . .       | 2073 |
| 9.423ex_MSRCXSetTxPower.nxc . . . . .         | 2073 |
| 9.424ex_MSRCXSetUserDisplay.nxc . . . . .     | 2073 |
| 9.425ex_MSRCXSetVar.nxc . . . . .             | 2073 |
| 9.426ex_MSRCXSetWatch.nxc . . . . .           | 2074 |
| 9.427ex_MSRCXSignVar.nxc . . . . .            | 2074 |
| 9.428ex_MSRCXStartTask.nxc . . . . .          | 2074 |
| 9.429ex_MSRCXStopAllTasks.nxc . . . . .       | 2074 |
| 9.430ex_MSRCXStopTask.nxc . . . . .           | 2074 |
| 9.431ex_MSRCXSubVar.nxc . . . . .             | 2074 |
| 9.432ex_MSRCXSumVar.nxc . . . . .             | 2075 |
| 9.433ex_MSRCXToggle.nxc . . . . .             | 2075 |
| 9.434ex_MSRCXUnlock.nxc . . . . .             | 2075 |
| 9.435ex_MSRCXUnmuteSound.nxc . . . . .        | 2075 |
| 9.436ex_MSReadValue.nxc . . . . .             | 2075 |
| 9.437ex_MSScoutCalibrateSensor.nxc . . . . .  | 2075 |
| 9.438ex_MSScoutMuteSound.nxc . . . . .        | 2076 |

---

|  |      |
|--|------|
| 9.439ex_MSScoutSelectSounds.nxc . . . . .        | 2076 |
| 9.440ex_MSScoutSendVLL.nxc . . . . .             | 2076 |
| 9.441ex_MSScoutSetCounterLimit.nxc . . . . .     | 2076 |
| 9.442ex_MSScoutSetEventFeedback.nxc . . . . .    | 2076 |
| 9.443ex_MSScoutSetLight.nxc . . . . .            | 2076 |
| 9.444ex_MSScoutSetScoutMode.nxc . . . . .        | 2077 |
| 9.445ex_MSScoutSetScoutRules.nxc . . . . .       | 2077 |
| 9.446ex_MSScoutSetSensorClickTime.nxc . . . . .  | 2077 |
| 9.447ex_MSScoutSetSensorHysteresis.nxc . . . . . | 2077 |
| 9.448ex_MSScoutSetSensorLowerLimit.nxc . . . . . | 2077 |
| 9.449ex_MSScoutSetSensorUpperLimit.nxc . . . . . | 2077 |
| 9.450ex_MSScoutSetTimerLimit.nxc . . . . .       | 2078 |
| 9.451ex_MSScoutUnmuteSound.nxc . . . . .         | 2078 |
| 9.452ex_muldiv32.nxc . . . . .                   | 2078 |
| 9.453ex_nbcopt.nxc . . . . .                     | 2078 |
| 9.454ex_NRLink2400.nxc . . . . .                 | 2079 |
| 9.455ex_NRLink4800.nxc . . . . .                 | 2079 |
| 9.456ex_NRLinkFlush.nxc . . . . .                | 2079 |
| 9.457ex_NRLinkIRLong.nxc . . . . .               | 2079 |
| 9.458ex_NRLinkIRShort.nxc . . . . .              | 2079 |
| 9.459ex_NRLinkSetPF.nxc . . . . .                | 2079 |
| 9.460ex_NRLinkSetRCX.nxc . . . . .               | 2080 |
| 9.461ex_NRLinkSetTrain.nxc . . . . .             | 2080 |
| 9.462ex_NRLinkStatus.nxc . . . . .               | 2080 |
| 9.463ex_NRLinkTxRaw.nxc . . . . .                | 2080 |
| 9.464ex_NumOut.nxc . . . . .                     | 2080 |
| 9.465ex_NumToStr.nxc . . . . .                   | 2080 |
| 9.466ex_NXTHID.nxc . . . . .                     | 2081 |
| 9.467ex_NXTLineLeader.nxc . . . . .              | 2081 |
| 9.468ex_NXTPowerMeter.nxc . . . . .              | 2083 |
| 9.469ex_NXTServo.nxc . . . . .                   | 2083 |

---

|   |      |
|---|------|
| 9.470ex_NXTSumoEyes.nxc . . . . .           | 2085 |
| 9.471ex_off.nxc . . . . .                   | 2085 |
| 9.472ex_offex.nxc . . . . .                 | 2085 |
| 9.473ex_OnBrickProgramPointer.nxc . . . . . | 2085 |
| 9.474ex_onfwd.nxc . . . . .                 | 2086 |
| 9.475ex_onfwdex.nxc . . . . .               | 2086 |
| 9.476ex_onfwdreg.nxc . . . . .              | 2086 |
| 9.477ex_onfwdregex.nxc . . . . .            | 2086 |
| 9.478ex_onfwdregexpid.nxc . . . . .         | 2086 |
| 9.479ex_onfwdregpid.nxc . . . . .           | 2086 |
| 9.480ex_onfwdsync.nxc . . . . .             | 2087 |
| 9.481ex_onfwdsyncex.nxc . . . . .           | 2087 |
| 9.482ex_onfwdsyncexpid.nxc . . . . .        | 2087 |
| 9.483ex_onfwdsyncpid.nxc . . . . .          | 2087 |
| 9.484ex_onrev.nxc . . . . .                 | 2087 |
| 9.485ex_onrevex.nxc . . . . .               | 2087 |
| 9.486ex_onrevreg.nxc . . . . .              | 2088 |
| 9.487ex_onrevregex.nxc . . . . .            | 2088 |
| 9.488ex_onrevregexpid.nxc . . . . .         | 2088 |
| 9.489ex_onrevregpid.nxc . . . . .           | 2088 |
| 9.490ex_onrevsync.nxc . . . . .             | 2088 |
| 9.491ex_onrevsyncex.nxc . . . . .           | 2088 |
| 9.492ex_onrevsyncexpid.nxc . . . . .        | 2088 |
| 9.493ex_onrevsyncpid.nxc . . . . .          | 2089 |
| 9.494ex_OpenFileAppend.nxc . . . . .        | 2089 |
| 9.495ex_OpenFileRead.nxc . . . . .          | 2089 |
| 9.496ex_OpenFileReadLinear.nxc . . . . .    | 2089 |
| 9.497ex_PFMate.nxc . . . . .                | 2089 |
| 9.498ex_PlayFile.nxc . . . . .              | 2090 |
| 9.499ex_PlayFileEx.nxc . . . . .            | 2090 |
| 9.500ex_playsound.nxc . . . . .             | 2090 |

---

|  |      |
|--|------|
| 9.501ex_PlayTone.nxc . . . . .                 | 2090 |
| 9.502ex_PlayToneEx.nxc . . . . .               | 2090 |
| 9.503ex_playtones.nxc . . . . .                | 2091 |
| 9.504ex_PointOut.nxc . . . . .                 | 2091 |
| 9.505ex_PolyOut.nxc . . . . .                  | 2091 |
| 9.506ex_Pos.nxc . . . . .                      | 2092 |
| 9.507ex_PosReg.nxc . . . . .                   | 2092 |
| 9.508ex_pow.nxc . . . . .                      | 2092 |
| 9.509ex_PowerDown.nxc . . . . .                | 2092 |
| 9.510ex_Precedes.nxc . . . . .                 | 2093 |
| 9.511ex_printf.nxc . . . . .                   | 2093 |
| 9.512ex_proto.nxc . . . . .                    | 2093 |
| 9.513ex_PSPNxAnalog.nxc . . . . .              | 2093 |
| 9.514ex_PSPNxDigital.nxc . . . . .             | 2094 |
| 9.515ex_putc.nxc . . . . .                     | 2094 |
| 9.516ex_rand.nxc . . . . .                     | 2094 |
| 9.517ex_Random.nxc . . . . .                   | 2094 |
| 9.518ex_Read.nxc . . . . .                     | 2094 |
| 9.519ex_ReadButtonEx.nxc . . . . .             | 2094 |
| 9.520ex_ReadBytes.nxc . . . . .                | 2095 |
| 9.521ex_readi2cregister.nxc . . . . .          | 2095 |
| 9.522ex_ReadLn.nxc . . . . .                   | 2095 |
| 9.523ex_ReadNRLinkBytes.nxc . . . . .          | 2095 |
| 9.524ex_ReadSensorColorEx.nxc . . . . .        | 2095 |
| 9.525ex_ReadSensorColorRaw.nxc . . . . .       | 2095 |
| 9.526ex_ReadSensorEMeter.nxc . . . . .         | 2096 |
| 9.527ex_ReadSensorHTAccel.nxc . . . . .        | 2096 |
| 9.528ex_ReadSensorHTAngle.nxc . . . . .        | 2096 |
| 9.529ex_ReadSensorHTBarometric.nxc . . . . .   | 2096 |
| 9.530ex_ReadSensorHTColor.nxc . . . . .        | 2097 |
| 9.531ex_ReadSensorHTColor2Active.nxc . . . . . | 2097 |

|  |      |
|--|------|
| 9.532ex_ReadSensorHTIRReceiver.nxc . . . . .             | 2097 |
| 9.533ex_ReadSensorHTIRReceiverEx.nxc . . . . .           | 2097 |
| 9.534ex_ReadSensorHTIRSeeker.nxc . . . . .               | 2097 |
| 9.535ex_ReadSensorHTIRSeeker2AC.nxc . . . . .            | 2098 |
| 9.536ex_ReadSensorHTIRSeeker2DC.nxc . . . . .            | 2098 |
| 9.537ex_ReadSensorHTNormalizedColor.nxc . . . . .        | 2098 |
| 9.538ex_ReadSensorHTNormalizedColor2Active.nxc . . . . . | 2098 |
| 9.539ex_ReadSensorHTRawColor.nxc . . . . .               | 2098 |
| 9.540ex_ReadSensorHTRawColor2.nxc . . . . .              | 2098 |
| 9.541ex_ReadSensorHTTouchMultiplexer.nxc . . . . .       | 2099 |
| 9.542ex_ReadSensorMSAccel.nxc . . . . .                  | 2099 |
| 9.543ex_ReadSensorMSPlayStation.nxc . . . . .            | 2099 |
| 9.544ex_ReadSensorMSRTClock.nxc . . . . .                | 2100 |
| 9.545ex_ReadSensorMSTilt.nxc . . . . .                   | 2100 |
| 9.546ex_ReadSensorUSEx.nxc . . . . .                     | 2101 |
| 9.547ex_RebootInFirmwareMode.nxc . . . . .               | 2101 |
| 9.548ex_ReceiveMessage.nxc . . . . .                     | 2101 |
| 9.549ex_ReceiveRemoteBool.nxc . . . . .                  | 2101 |
| 9.550ex_ReceiveRemoteMessageEx.nxc . . . . .             | 2101 |
| 9.551ex_ReceiveRemoteNumber.nxc . . . . .                | 2101 |
| 9.552ex_ReceiveRemoteString.nxc . . . . .                | 2102 |
| 9.553ex_RechargeableBattery.nxc . . . . .                | 2102 |
| 9.554ex_RectOut.nxc . . . . .                            | 2102 |
| 9.555ex_reladdressof.nxc . . . . .                       | 2102 |
| 9.556ex_Release.nxc . . . . .                            | 2103 |
| 9.557ex_RemoteBluetoothFactoryReset.nxc . . . . .        | 2103 |
| 9.558ex_RemoteCloseFile.nxc . . . . .                    | 2103 |
| 9.559ex_RemoteConnectionIdle.nxc . . . . .               | 2103 |
| 9.560ex_RemoteConnectionWrite.nxc . . . . .              | 2103 |
| 9.561ex_RemoteDatalogRead.nxc . . . . .                  | 2104 |
| 9.562ex_RemoteDatalogSetTimes.nxc . . . . .              | 2104 |

|   |      |
|---|------|
| 9.563ex_RemoteDeleteFile.nxc . . . . .            | 2104 |
| 9.564ex_RemoteDeleteUserFlash.nxc . . . . .       | 2104 |
| 9.565ex_RemoteFindFirstFile.nxc . . . . .         | 2104 |
| 9.566ex_RemoteFindNextFile.nxc . . . . .          | 2104 |
| 9.567ex_RemoteGetBatteryLevel.nxc . . . . .       | 2105 |
| 9.568ex_RemoteGetBluetoothAddress.nxc . . . . .   | 2105 |
| 9.569ex_RemoteGetConnectionCount.nxc . . . . .    | 2105 |
| 9.570ex_RemoteGetConnectionName.nxc . . . . .     | 2105 |
| 9.571ex_RemoteGetContactCount.nxc . . . . .       | 2105 |
| 9.572ex_RemoteGetContactName.nxc . . . . .        | 2106 |
| 9.573ex_RemoteGetCurrentProgramName.nxc . . . . . | 2106 |
| 9.574ex_RemoteGetDeviceInfo.nxc . . . . .         | 2106 |
| 9.575ex_RemoteGetFirmwareVersion.nxc . . . . .    | 2106 |
| 9.576ex_RemoteGetInputValues.nxc . . . . .        | 2106 |
| 9.577ex_RemoteGetOutputState.nxc . . . . .        | 2107 |
| 9.578ex_RemoteGetProperty.nxc . . . . .           | 2107 |
| 9.579ex_RemoteIOMapRead.nxc . . . . .             | 2107 |
| 9.580ex_RemoteIOMapWriteBytes.nxc . . . . .       | 2107 |
| 9.581ex_RemoteIOMapWriteValue.nxc . . . . .       | 2107 |
| 9.582ex_RemoteKeepAlive.nxc . . . . .             | 2108 |
| 9.583ex_RemoteLowspeedGetStatus.nxc . . . . .     | 2108 |
| 9.584ex_RemoteLowspeedRead.nxc . . . . .          | 2108 |
| 9.585ex_RemoteLowspeedWrite.nxc . . . . .         | 2108 |
| 9.586ex_RemoteMessageRead.nxc . . . . .           | 2108 |
| 9.587ex_RemoteMessageWrite.nxc . . . . .          | 2109 |
| 9.588ex_RemoteOpenAppendData.nxc . . . . .        | 2109 |
| 9.589ex_RemoteOpenRead.nxc . . . . .              | 2109 |
| 9.590ex_RemoteOpenWrite.nxc . . . . .             | 2109 |
| 9.591ex_RemoteOpenWriteData.nxc . . . . .         | 2109 |
| 9.592ex_RemoteOpenWriteLinear.nxc . . . . .       | 2110 |
| 9.593ex_RemotePlaySoundFile.nxc . . . . .         | 2110 |

---

|  |      |
|--|------|
| 9.594ex_RemotePlayTone.nxc . . . . .           | 2110 |
| 9.595ex_RemotePollCommand.nxc . . . . .        | 2110 |
| 9.596ex_RemotePollCommandLength.nxc . . . . .  | 2110 |
| 9.597ex_RemoteRead.nxc . . . . .               | 2110 |
| 9.598ex_RemoteRenameFile.nxc . . . . .         | 2111 |
| 9.599ex_RemoteResetMotorPosition.nxc . . . . . | 2111 |
| 9.600ex_RemoteResetScaledValue.nxc . . . . .   | 2111 |
| 9.601ex_RemoteResetTachoCount.nxc . . . . .    | 2111 |
| 9.602ex_RemoteSetBrickName.nxc . . . . .       | 2111 |
| 9.603ex_RemoteSetInputMode.nxc . . . . .       | 2111 |
| 9.604ex_RemoteSetOutputState.nxc . . . . .     | 2112 |
| 9.605ex_RemoteSetProperty.nxc . . . . .        | 2112 |
| 9.606ex_RemoteStartProgram.nxc . . . . .       | 2112 |
| 9.607ex_RemoteStopProgram.nxc . . . . .        | 2112 |
| 9.608ex_RemoteStopSound.nxc . . . . .          | 2112 |
| 9.609ex_RemoteWrite.nxc . . . . .              | 2112 |
| 9.610ex_remove.nxc . . . . .                   | 2113 |
| 9.611ex_rename.nxc . . . . .                   | 2113 |
| 9.612ex_RenameFile.nxc . . . . .               | 2113 |
| 9.613ex_resetalltachocounts.nxc . . . . .      | 2113 |
| 9.614ex_resetblocktachocount.nxc . . . . .     | 2113 |
| 9.615ex_resetrotationcount.nxc . . . . .       | 2113 |
| 9.616ex_ResetScreen.nxc . . . . .              | 2114 |
| 9.617ex_ResetSensor.nxc . . . . .              | 2114 |
| 9.618ex_ResetSensorHTAngle.nxc . . . . .       | 2114 |
| 9.619ex_ResetSleepTimer.nxc . . . . .          | 2114 |
| 9.620ex_resettachocount.nxc . . . . .          | 2114 |
| 9.621ex_resizefile.nxc . . . . .               | 2114 |
| 9.622ex_ResolveHandle.nxc . . . . .            | 2115 |
| 9.623ex_rewind.nxc . . . . .                   | 2115 |
| 9.624ex_RFIDInit.nxc . . . . .                 | 2115 |

---

|                                       |      |
|---------------------------------------|------|
| 9.625ex_RFIDMode.nxc                  | 2115 |
| 9.626ex_RFIDRead.nxc                  | 2115 |
| 9.627ex_RFIDReadContinuous.nxc        | 2115 |
| 9.628ex_RFIDReadSingle.nxc            | 2116 |
| 9.629ex_RFIDStatus.nxc                | 2116 |
| 9.630ex_RFIDStop.nxc                  | 2116 |
| 9.631ex_rightstr.nxc                  | 2116 |
| 9.632ex_rotatemotor.nxc               | 2116 |
| 9.633ex_rotatemotorex.nxc             | 2116 |
| 9.634ex_rotatemotorexpid.nxc          | 2117 |
| 9.635ex_rotatemotorpid.nxc            | 2117 |
| 9.636ex_RS485Receive.nxc              | 2117 |
| 9.637ex_RS485Send.nxc                 | 2118 |
| 9.638ex_RunNRLinkMacro.nxc            | 2119 |
| 9.639ex_SendMessage.nxc               | 2119 |
| 9.640ex_SendRemoteBool.nxc            | 2119 |
| 9.641ex_SendRemoteNumber.nxc          | 2119 |
| 9.642ex_SendRemoteString.nxc          | 2120 |
| 9.643ex_SendResponseBool.nxc          | 2120 |
| 9.644ex_SendResponseNumber.nxc        | 2120 |
| 9.645ex_SendResponseString.nxc        | 2120 |
| 9.646ex_Sensor.nxc                    | 2120 |
| 9.647ex_SensorBoolean.nxc             | 2120 |
| 9.648ex_SensorDigiPinsDirection.nxc   | 2121 |
| 9.649ex_SensorDigiPinsOutputLevel.nxc | 2121 |
| 9.650ex_SensorDigiPinsStatus.nxc      | 2121 |
| 9.651ex_SensorHTColorNum.nxc          | 2121 |
| 9.652ex_SensorHTCompass.nxc           | 2121 |
| 9.653ex_SensorHTEOPD.nxc              | 2121 |
| 9.654ex_SensorHTGyro.nxc              | 2122 |
| 9.655ex_SensorHTIRSeeker2ACDir.nxc    | 2122 |

---

|   |      |
|---|------|
| 9.656ex_SensorHTIRSeeker2Addr.nxc . . . . .     | 2122 |
| 9.657ex_SensorHTIRSeeker2DCDir.nxc . . . . .    | 2122 |
| 9.658ex_SensorHTIRSeekerDir.nxc . . . . .       | 2122 |
| 9.659ex_SensorHTMagnet.nxc . . . . .            | 2122 |
| 9.660ex_SensorInvalid.nxc . . . . .             | 2123 |
| 9.661ex_SensorMode.nxc . . . . .                | 2123 |
| 9.662ex_SensorMSCompass.nxc . . . . .           | 2123 |
| 9.663ex_SensorMSDROD.nxc . . . . .              | 2123 |
| 9.664ex_SensorMSPressure.nxc . . . . .          | 2123 |
| 9.665ex_SensorMSPressureRaw.nxc . . . . .       | 2123 |
| 9.666ex_SensorNormalized.nxc . . . . .          | 2124 |
| 9.667ex_SensorRaw.nxc . . . . .                 | 2124 |
| 9.668ex_SensorScaled.nxc . . . . .              | 2124 |
| 9.669ex_SensorTemperature.nxc . . . . .         | 2124 |
| 9.670ex_SensorType.nxc . . . . .                | 2124 |
| 9.671ex_SensorUS.nxc . . . . .                  | 2124 |
| 9.672ex_SensorValue.nxc . . . . .               | 2125 |
| 9.673ex_SensorValueBool.nxc . . . . .           | 2125 |
| 9.674ex_SensorValueRaw.nxc . . . . .            | 2125 |
| 9.675ex_SetAbortFlag.nxc . . . . .              | 2125 |
| 9.676ex_SetACCLNxSensitivity.nxc . . . . .      | 2126 |
| 9.677ex_SetBatteryState.nxc . . . . .           | 2126 |
| 9.678ex_SetBluetoothState.nxc . . . . .         | 2126 |
| 9.679ex_SetBTInputBuffer.nxc . . . . .          | 2126 |
| 9.680ex_SetBTInputBufferInPtr.nxc . . . . .     | 2126 |
| 9.681ex_SetBTInputBufferOutPtr.nxc . . . . .    | 2127 |
| 9.682ex_SetBTOutputBuffer.nxc . . . . .         | 2127 |
| 9.683ex_SetBTOutputBufferInPtr.nxc . . . . .    | 2127 |
| 9.684ex_SetBTOutputBufferOutPtr.nxc . . . . .   | 2127 |
| 9.685ex_SetButtonLongPressCount.nxc . . . . .   | 2127 |
| 9.686ex_SetButtonLongReleaseCount.nxc . . . . . | 2127 |

|   |      |
|---|------|
| 9.687ex_SetButtonPressCount.nxc . . . . .             | 2128 |
| 9.688ex_SetButtonReleaseCount.nxc . . . . .           | 2128 |
| 9.689ex_SetButtonShortReleaseCount.nxc . . . . .      | 2128 |
| 9.690ex_SetButtonState.nxc . . . . .                  | 2128 |
| 9.691ex_SetCommandFlags.nxc . . . . .                 | 2128 |
| 9.692ex_SetCustomSensorActiveStatus.nxc . . . . .     | 2128 |
| 9.693ex_SetCustomSensorPercentFullScale.nxc . . . . . | 2129 |
| 9.694ex_SetCustomSensorZeroOffset.nxc . . . . .       | 2129 |
| 9.695ex_setdisplaycontrast.nxc . . . . .              | 2129 |
| 9.696ex_SetDisplayDisplay.nxc . . . . .               | 2129 |
| 9.697ex_SetDisplayEraseMask.nxc . . . . .             | 2129 |
| 9.698ex_SetDisplayFlags.nxc . . . . .                 | 2129 |
| 9.699ex_setdisplayfont.nxc . . . . .                  | 2130 |
| 9.700ex_SetDisplayNormal.nxc . . . . .                | 2131 |
| 9.701ex_SetDisplayPopup.nxc . . . . .                 | 2131 |
| 9.702ex_SetDisplayTextLinesCenterFlags.nxc . . . . .  | 2131 |
| 9.703ex_SetDisplayUpdateMask.nxc . . . . .            | 2131 |
| 9.704ex_SetHSFlags.nxc . . . . .                      | 2131 |
| 9.705ex_SetHSInputBuffer.nxc . . . . .                | 2132 |
| 9.706ex_SetHSInputBufferInPtr.nxc . . . . .           | 2132 |
| 9.707ex_SetHSInputBufferOutPtr.nxc . . . . .          | 2132 |
| 9.708ex_sethsmode.nxc . . . . .                       | 2132 |
| 9.709ex_SetHSOutputBuffer.nxc . . . . .               | 2132 |
| 9.710ex_SetHSOutputBufferInPtr.nxc . . . . .          | 2132 |
| 9.711ex_SetHSOutputBufferOutPtr.nxc . . . . .         | 2133 |
| 9.712ex_SetHSSpeed.nxc . . . . .                      | 2133 |
| 9.713ex_SetHSState.nxc . . . . .                      | 2133 |
| 9.714ex_sethtcolor2mode.nxc . . . . .                 | 2133 |
| 9.715ex_sethtirseeker2mode.nxc . . . . .              | 2133 |
| 9.716ex_SetInput.nxc . . . . .                        | 2133 |
| 9.717ex_SetLongAbort.nxc . . . . .                    | 2134 |

---

|  |      |
|--|------|
| 9.718ex_SetMotorPwnFreq.nxc              | 2134 |
| 9.719ex_SetOnBrickProgramPointer.nxc     | 2134 |
| 9.720ex_setoutput.nxc                    | 2135 |
| 9.721ex_SetSensor.nxc                    | 2135 |
| 9.722ex_setsensorboolean.nxc             | 2135 |
| 9.723ex_setsensorcolorblue.nxc           | 2135 |
| 9.724ex_setsensorcolorfull.nxc           | 2135 |
| 9.725ex_setsensorcolorgreen.nxc          | 2135 |
| 9.726ex_setsensorcolornone.nxc           | 2135 |
| 9.727ex_setsensorcolorred.nxc            | 2136 |
| 9.728ex_SetSensorDigiPinsDirection.nxc   | 2136 |
| 9.729ex_SetSensorDigiPinsOutputLevel.nxc | 2136 |
| 9.730ex_SetSensorDigiPinsStatus.nxc      | 2136 |
| 9.731ex_SetSensorEMeter.nxc              | 2136 |
| 9.732ex_setsensorhteopd.nxc              | 2136 |
| 9.733ex_SetSensorHTGyro.nxc              | 2137 |
| 9.734ex_SetSensorHTMagnet.nxc            | 2137 |
| 9.735ex_SetSensorLight.nxc               | 2137 |
| 9.736ex_SetSensorLowspeed.nxc            | 2137 |
| 9.737ex_SetSensorMode.nxc                | 2137 |
| 9.738ex_setsensormsdrod.nxc              | 2137 |
| 9.739ex_setsensormspressure.nxc          | 2138 |
| 9.740ex_SetSensorSound.nxc               | 2138 |
| 9.741ex_SetSensorTemperature.nxc         | 2138 |
| 9.742ex_SetSensorTouch.nxc               | 2138 |
| 9.743ex_SetSensorType.nxc                | 2138 |
| 9.744ex_SetSensorUltrasonic.nxc          | 2138 |
| 9.745ex_setsleeptime.nxc                 | 2139 |
| 9.746ex_SetSleepTimeout.nxc              | 2139 |
| 9.747ex_SetSleepTimer.nxc                | 2139 |
| 9.748ex_SetSoundDuration.nxc             | 2139 |

---

|  |      |
|--|------|
| 9.749ex_SetSoundFlags.nxc . . . . .            | 2139 |
| 9.750ex_SetSoundFrequency.nxc . . . . .        | 2139 |
| 9.751ex_SetSoundMode.nxc . . . . .             | 2140 |
| 9.752ex_SetSoundModuleState.nxc . . . . .      | 2140 |
| 9.753ex_SetSoundSampleRate.nxc . . . . .       | 2140 |
| 9.754ex_SetSoundVolume.nxc . . . . .           | 2140 |
| 9.755ex_SetUIButton.nxc . . . . .              | 2140 |
| 9.756ex_SetUIState.nxc . . . . .               | 2140 |
| 9.757ex_SetUSBInputBuffer.nxc . . . . .        | 2141 |
| 9.758ex_SetUSBInputBufferInPtr.nxc . . . . .   | 2141 |
| 9.759ex_SetUSBInputBufferOutPtr.nxc . . . . .  | 2141 |
| 9.760ex_SetUSBOutputBuffer.nxc . . . . .       | 2141 |
| 9.761ex_SetUSBOutputBufferInPtr.nxc . . . . .  | 2141 |
| 9.762ex_SetUSBOutputBufferOutPtr.nxc . . . . . | 2141 |
| 9.763ex_SetUSBPollBuffer.nxc . . . . .         | 2142 |
| 9.764ex_SetUSBPollBufferInPtr.nxc . . . . .    | 2142 |
| 9.765ex_SetUSBPollBufferOutPtr.nxc . . . . .   | 2142 |
| 9.766ex_SetUsbState.nxc . . . . .              | 2142 |
| 9.767ex_SetVMRunState.nxc . . . . .            | 2142 |
| 9.768ex_SetVolume.nxc . . . . .                | 2142 |
| 9.769ex_sign.nxc . . . . .                     | 2143 |
| 9.770ex_sin_cos.nxc . . . . .                  | 2143 |
| 9.771ex_sind_cosd.nxc . . . . .                | 2143 |
| 9.772ex_sinh.nxc . . . . .                     | 2144 |
| 9.773ex_SizeOf.nxc . . . . .                   | 2144 |
| 9.774ex_SleepNow.nxc . . . . .                 | 2145 |
| 9.775ex_sleeptime.nxc . . . . .                | 2145 |
| 9.776ex_SleepTimeout.nxc . . . . .             | 2145 |
| 9.777ex_SleepTimer.nxc . . . . .               | 2145 |
| 9.778ex_SoundDuration.nxc . . . . .            | 2145 |
| 9.779ex_SoundFlags.nxc . . . . .               | 2145 |

---

|                             |      |
|-----------------------------|------|
| 9.780ex_SoundFrequency.nxc  | 2146 |
| 9.781ex_SoundMode.nxc       | 2146 |
| 9.782ex_SoundSampleRate.nxc | 2146 |
| 9.783ex_SoundState.nxc      | 2146 |
| 9.784ex_SoundVolume.nxc     | 2146 |
| 9.785ex_sprintf.nxc         | 2146 |
| 9.786ex_sqrt.nxc            | 2147 |
| 9.787ex_srand.nxc           | 2147 |
| 9.788ex_StartTask.nxc       | 2147 |
| 9.789ex_Stop.nxc            | 2147 |
| 9.790ex_StopAllTasks.nxc    | 2147 |
| 9.791ex_StopSound.nxc       | 2147 |
| 9.792ex_StopTask.nxc        | 2148 |
| 9.793ex_StrCat.nxc          | 2148 |
| 9.794ex_StrCatOld.nxc       | 2148 |
| 9.795ex_strcmp.nxc          | 2148 |
| 9.796ex_strcpy.nxc          | 2148 |
| 9.797ex_StrIndex.nxc        | 2149 |
| 9.798ex_string.nxc          | 2149 |
| 9.799ex_StrLen.nxc          | 2150 |
| 9.800ex_StrLenOld.nxc       | 2150 |
| 9.801ex_strncat.nxc         | 2150 |
| 9.802ex_strncmp.nxc         | 2150 |
| 9.803ex_strncpy.nxc         | 2150 |
| 9.804ex_StrReplace.nxc      | 2151 |
| 9.805ex_StrToByteArray.nxc  | 2151 |
| 9.806ex_strtod.nxc          | 2151 |
| 9.807ex_strtol.nxc          | 2151 |
| 9.808ex_StrToNum.nxc        | 2152 |
| 9.809ex_strtoul.nxc         | 2152 |
| 9.810ex_SubStr.nxc          | 2152 |

---

|   |      |
|---|------|
| 9.811ex_superpro.nxc . . . . .                  | 2152 |
| 9.812ex_syscall.nxc . . . . .                   | 2153 |
| 9.813ex_SysColorSensorRead.nxc . . . . .        | 2153 |
| 9.814ex_syscommbtcheckstatus.nxc . . . . .      | 2154 |
| 9.815ex_syscommbtconnection.nxc . . . . .       | 2154 |
| 9.816ex_SysCommBTONOff.nxc . . . . .            | 2155 |
| 9.817ex_syscommbtwrite.nxc . . . . .            | 2155 |
| 9.818ex_syscommexecutefunction.nxc . . . . .    | 2155 |
| 9.819ex_SysCommHSCheckStatus.nxc . . . . .      | 2156 |
| 9.820ex_SysCommHSControl.nxc . . . . .          | 2156 |
| 9.821ex_SysCommHSRead.nxc . . . . .             | 2156 |
| 9.822ex_SysCommHSWrite.nxc . . . . .            | 2156 |
| 9.823ex_syscommmlscheckstatus.nxc . . . . .     | 2157 |
| 9.824ex_syscommmlsread.nxc . . . . .            | 2157 |
| 9.825ex_syscommmlswrite.nxc . . . . .           | 2157 |
| 9.826ex_syscommmlswriteex.nxc . . . . .         | 2158 |
| 9.827ex_SysComputeCalibValue.nxc . . . . .      | 2158 |
| 9.828ex_sysdataloggettimes.nxc . . . . .        | 2158 |
| 9.829ex_SysDatalogWrite.nxc . . . . .           | 2159 |
| 9.830ex_sysdisplayexecutefunction.nxc . . . . . | 2159 |
| 9.831ex_sysdrawcircle.nxc . . . . .             | 2159 |
| 9.832ex_SysDrawEllipse.nxc . . . . .            | 2160 |
| 9.833ex_sysdrawfont.nxc . . . . .               | 2160 |
| 9.834ex_sysdrawgraphic.nxc . . . . .            | 2160 |
| 9.835ex_sysdrawgraphicarray.nxc . . . . .       | 2161 |
| 9.836ex_sysdrawline.nxc . . . . .               | 2162 |
| 9.837ex_sysdrawpoint.nxc . . . . .              | 2162 |
| 9.838ex_sysdrawpolygon.nxc . . . . .            | 2162 |
| 9.839ex_sysdrawrect.nxc . . . . .               | 2163 |
| 9.840ex_sysdrawtext.nxc . . . . .               | 2163 |
| 9.841ex_sysfileclose.nxc . . . . .              | 2163 |

---

|   |      |
|---|------|
| 9.842ex_sysfiledelete.nxc . . . . .             | 2164 |
| 9.843ex_sysfilefindfirst.nxc . . . . .          | 2164 |
| 9.844ex_sysfilefindnext.nxc . . . . .           | 2164 |
| 9.845ex_sysfileopenappend.nxc . . . . .         | 2165 |
| 9.846ex_sysfileopenread.nxc . . . . .           | 2165 |
| 9.847ex_sysfileopenreadlinear.nxc . . . . .     | 2165 |
| 9.848ex_sysfileopenwrite.nxc . . . . .          | 2166 |
| 9.849ex_sysfileopenwritelinear.nxc . . . . .    | 2166 |
| 9.850ex_sysfileopenwritenonlinear.nxc . . . . . | 2166 |
| 9.851ex_sysfileread.nxc . . . . .               | 2167 |
| 9.852ex_sysfilerename.nxc . . . . .             | 2167 |
| 9.853ex_sysfileresize.nxc . . . . .             | 2167 |
| 9.854ex_sysfileresolvehandle.nxc . . . . .      | 2168 |
| 9.855ex_sysfileseek.nxc . . . . .               | 2168 |
| 9.856ex_sysfilewrite.nxc . . . . .              | 2168 |
| 9.857ex_sysgetstarttick.nxc . . . . .           | 2169 |
| 9.858ex_sysinputpinfunction.nxc . . . . .       | 2169 |
| 9.859ex_sysiomapread.nxc . . . . .              | 2170 |
| 9.860ex_sysiomapreadbyid.nxc . . . . .          | 2171 |
| 9.861ex_sysiomapwrite.nxc . . . . .             | 2171 |
| 9.862ex_sysiomapwritebyid.nxc . . . . .         | 2171 |
| 9.863ex_syskeepalive.nxc . . . . .              | 2172 |
| 9.864ex_syslistfiles.nxc . . . . .              | 2172 |
| 9.865ex_sysloaderexecutefunction.nxc . . . . .  | 2172 |
| 9.866ex_sysmemorymanager.nxc . . . . .          | 2172 |
| 9.867ex_sysmessageread.nxc . . . . .            | 2173 |
| 9.868ex_sysmessagewrite.nxc . . . . .           | 2173 |
| 9.869ex_sysrandomex.nxc . . . . .               | 2174 |
| 9.870ex_sysrandomnumber.nxc . . . . .           | 2174 |
| 9.871ex_sysreadbutton.nxc . . . . .             | 2174 |
| 9.872ex_SysReadLastResponse.nxc . . . . .       | 2175 |

|   |      |
|---|------|
| 9.873ex_SysReadSemData.nxc . . . . .          | 2175 |
| 9.874ex_syssetscreenmode.nxc . . . . .        | 2175 |
| 9.875ex_SysSetSleepTimeout.nxc . . . . .      | 2175 |
| 9.876ex_syssoundgetstate.nxc . . . . .        | 2176 |
| 9.877ex_syssoundplayfile.nxc . . . . .        | 2176 |
| 9.878ex_syssoundplaytone.nxc . . . . .        | 2176 |
| 9.879ex_syssoundsetstate.nxc . . . . .        | 2176 |
| 9.880ex_SysUpdateCalibCacheInfo.nxc . . . . . | 2177 |
| 9.881ex_SysWriteSemData.nxc . . . . .         | 2177 |
| 9.882ex_tan.nxc . . . . .                     | 2177 |
| 9.883ex_tand.nxc . . . . .                    | 2178 |
| 9.884ex_tanh.nxc . . . . .                    | 2179 |
| 9.885ex_TextOut.nxc . . . . .                 | 2179 |
| 9.886ex_tolower.nxc . . . . .                 | 2179 |
| 9.887ex_toupper.nxc . . . . .                 | 2179 |
| 9.888ex_trunc.nxc . . . . .                   | 2179 |
| 9.889ex_UIButton.nxc . . . . .                | 2180 |
| 9.890ex_UIState.nxc . . . . .                 | 2180 |
| 9.891ex_UiUsbState.nxc . . . . .              | 2180 |
| 9.892ex_UnflattenVar.nxc . . . . .            | 2180 |
| 9.893ex_USBInputBufferInPtr.nxc . . . . .     | 2181 |
| 9.894ex_USBInputBufferOutPtr.nxc . . . . .    | 2181 |
| 9.895ex_USBOutputBufferInPtr.nxc . . . . .    | 2181 |
| 9.896ex_USBOutputBufferOutPtr.nxc . . . . .   | 2181 |
| 9.897ex_USBPollBufferInPtr.nxc . . . . .      | 2181 |
| 9.898ex_USBPollBufferOutPtr.nxc . . . . .     | 2181 |
| 9.899ex_UsbState.nxc . . . . .                | 2182 |
| 9.900ex_VMRunState.nxc . . . . .              | 2182 |
| 9.901ex_Volume.nxc . . . . .                  | 2182 |
| 9.902ex_wait.nxc . . . . .                    | 2182 |
| 9.903ex_Write.nxc . . . . .                   | 2182 |

|  |      |
|--|------|
| <a href="#">9.904ex_WriteBytes.nxc</a>       | 2182 |
| <a href="#">9.905ex_WriteBytesEx.nxc</a>     | 2183 |
| <a href="#">9.906ex_writei2cregister.nxc</a> | 2183 |
| <a href="#">9.907ex_WriteLn.nxc</a>          | 2183 |
| <a href="#">9.908ex_WriteLnString.nxc</a>    | 2183 |
| <a href="#">9.909ex_writenrlinkbytes.nxc</a> | 2183 |
| <a href="#">9.910ex_WriteString.nxc</a>      | 2183 |
| <a href="#">9.911ex_xgl300.nxc</a>           | 2184 |
| <a href="#">9.912ex_yield.nxc</a>            | 2185 |
| <a href="#">9.913glBoxDemo.nxc</a>           | 2185 |
| <a href="#">9.914glCircleDemo.nxc</a>        | 2186 |
| <a href="#">9.915glRotateDemo.nxc</a>        | 2187 |
| <a href="#">9.916glScaleDemo.nxc</a>         | 2188 |
| <a href="#">9.917glTranslateDemo.nxc</a>     | 2188 |
| <a href="#">9.918util_battery_1.nxc</a>      | 2190 |
| <a href="#">9.919util_battery_2.nxc</a>      | 2190 |
| <a href="#">9.920util_rpm.nxc</a>            | 2191 |

## **1 NXC Programmer's Guide**

**October 10, 2011**

**by John Hansen**

- [Introduction](#)
- [The NXC Language](#)

## **2 Introduction**

NXC stands for Not eXactly C.

It is a simple language for programming the LEGO MINDSTORMS NXT product. The NXT has a bytecode interpreter (provided by LEGO), which can be used to execute programs. The NXC compiler translates a source program into NXT bytecodes, which can then be executed on the target itself. Although the preprocessor and control structures of NXC are very similar to C, NXC is not a general-purpose programming language - there are many restrictions that stem from limitations of the NXT bytecode interpreter.

Logically, NXC is defined as two separate pieces. The NXC language describes the syntax to be used in writing programs. The NXC Application Programming Interface (API) describes the system functions, constants, and macros that can be used by programs. This API is defined in a special file known as a "header file" which is, by default, automatically included when compiling a program.

This document describes both the NXC language and the NXC API. In short, it provides the information needed to write NXC programs. Since there are different interfaces for NXC, this document does not describe how to use any specific NXC implementation (such as the command-line compiler or Brick Command Center). Refer to the documentation provided with the NXC tool, such as the NXC User Manual, for information specific to that implementation.

For up-to-date information and documentation for NXC, visit the NXC website at <http://bricxcc.sourceforge.net/nxc/>.

## 3 The NXC Language

This section describes the NXC language.

This includes the lexical rules used by the compiler, the structure of programs, statements and expressions, and the operation of the preprocessor.

NXC is a case-sensitive language, just like C and C++, which means the identifier "xYz" is not the same identifier as "Xyz". Similarly, the "if" statement begins with the keyword "if" but "iF", "If", or "IF" are all just valid identifiers - not keywords.

- [Lexical Rules](#)
- [Program Structure](#)
- [Statements](#)
- [Expressions](#)
- [The Preprocessor](#)

### 3.1 Lexical Rules

The lexical rules describe how NXC breaks a source file into individual tokens.

This includes the way comments are written, the handling of whitespace, and valid characters for identifiers.

- [Comments](#)
- [Whitespace](#)
- [Numerical Constants](#)
- [String Constants](#)
- [Character Constants](#)
- [Identifiers and Keywords](#)

### 3.1.1 Comments

Two forms of comments are supported in NXC.

The first are traditional C comments. They begin with `/*` and end with `*/`. These comments are allowed to span multiple lines, but they cannot be nested.

```
/* this is a comment */

/* this is a two
   line comment */

/* another comment...
   /* trying to nest...
      ending the inner comment...*/
   this text is no longer a comment! */
```

The second form of comments supported in NXC begins with `//` and continues to the end of the current line. These are sometimes known as C++ style comments.

```
// a single line comment
```

As you might guess, the compiler ignores comments. Their only purpose is to allow the programmer to document the source code.

### 3.1.2 Whitespace

Whitespace consists of all spaces, tabs, and newlines.

It is used to separate tokens and to make a program more readable. As long as the tokens are distinguishable, adding or subtracting whitespace has no effect on the meaning of a program. For example, the following lines of code both have the same meaning:

```
x=2;
x = 2 ;
```

Some of the C++ operators consist of multiple characters. In order to preserve these tokens, whitespace cannot appear within them. In the example below, the first line uses a right shift operator ('>>'), but in the second line the added space causes the '>' symbols to be interpreted as two separate tokens and thus results in a compiler error.

```
x = 1 >> 4; // set x to 1 right shifted by 4 bits
x = 1 > > 4; // error
```

### 3.1.3 Numerical Constants

Numerical constants may be written in either decimal or hexadecimal form.

Decimal constants consist of one or more decimal digits. Decimal constants may optionally include a decimal point along with one or more decimal digits following the decimal point. Hexadecimal constants start with 0x or 0X followed by one or more hexadecimal digits.

```
x = 10; // set x to 10
x = 0x10; // set x to 16 (10 hex)
f = 10.5; // set f to 10.5
```

### 3.1.4 String Constants

String constants in NXC, just as in C, are delimited with double quote characters.

NXC has a string data type that makes strings easier to use than in C. Behind the scenes, a string is automatically converted into an array of bytes, with the last byte in the array being a zero. The final zero byte is generally referred to as the null terminator.

```
TextOut(0, LCD_LINE1, "testing");
```

### 3.1.5 Character Constants

Character constants in NXC are delimited with single quote characters and may contain a single ASCII character.

The value of a character constant is the numeric ASCII value of the character.

```
char ch = 'a'; // ch == 97
```

### 3.1.6 Identifiers and Keywords

Identifiers are used for variable, task, function, and subroutine names.

The first character of an identifier must be an upper or lower case letter or the underscore ('\_'). Remaining characters may be letters, numbers, and underscores.

A number of tokens are reserved for use in the NXC language itself. These are called keywords and may not be used as identifiers. A complete list of keywords appears below:

- 
- The asm statement
  - bool
  - The break statement
  - byte
  - The case label
  - char
  - const
  - The continue statement
  - The default label
  - The do statement
  - The if-else statement
  - enum
  - The false condition
  - float
  - The for statement
  - The goto statement
  - The if statement
  - The inline keyword
  - int
  - long
  - mutex
  - The priority statement
  - The repeat statement
  - The return statement
  - The safecall keyword
  - short
  - The start statement
  - static

- [The stop statement](#)
- [string](#)
- [Structures](#)
- [The sub keyword](#)
- [The switch statement](#)
- [Tasks](#)
- [The true condition](#)
- [typedef](#)
- [unsigned](#)
- [The until statement](#)
- [The void keyword](#)
- [The while statement](#)

#### 3.1.6.1 const

The `const` keyword is used to alter a variable declaration so that the variable cannot have its value changed after it is initialized.

The initialization must occur at the point of the variable declaration.

```
const int myConst = 23; // declare and initialize constant integer
task main() {
    int x = myConst; // this works fine
    myConst++; // compiler error - you cannot modify a constant's value
}
```

#### 3.1.6.2 enum

The `enum` keyword is used to create an enumerated type named `name`.

The syntax is show below.

```
enum [name] {name-list} var-list;
```

The enumerated type consists of the elements in `name-list`. The `var-list` argument is optional, and can be used to create instances of the type along with the declaration. For example, the following code creates an enumerated type for colors:

```
enum ColorT {red, orange, yellow, green, blue, indigo, violet};
```

In the above example, the effect of the enumeration is to introduce several new constants named red, orange, yellow, etc. By default, these constants are assigned consecutive integer values starting at zero. You can change the values of those constants, as shown by the next example:

```
enum ColorT { red = 10, blue = 15, green };
```

In the above example, green has a value of 16. Once you have defined an enumerated type you can use it to declare variables just like you use any native type. Here are a few examples of using the enum keyword:

```
// values start from 0 and increment upward by 1
enum { ONE, TWO, THREE };
// optional equal sign with constant expression for the value
enum { SMALL=10, MEDIUM=100, LARGE=1000 };
// names without equal sign increment by one from last name's value
enum { FRED=1, WILMA, BARNEY, BETTY };
// optional named type (like a typedef)
enum TheSeasons { SPRING, SUMMER, FALL, WINTER };
// optional variable at end
enum Days {
    saturday,           // saturday = 0 by default
    sunday = 0x0,       // sunday = 0 as well
    monday,             // monday = 1
    tuesday,            // tuesday = 2
    wednesday,         // etc.
    thursday,
    friday
} today;               // Variable today has type Days

Days tomorrow;

task main()
{
    TheSeasons test = FALL;
    today = monday;
    tomorrow = today+1;
    NumOut(0, LCD_LINE1, THREE);
    NumOut(0, LCD_LINE2, MEDIUM);
    NumOut(0, LCD_LINE3, FRED);
    NumOut(0, LCD_LINE4, SPRING);
    NumOut(0, LCD_LINE5, friday);
    NumOut(0, LCD_LINE6, today);
    NumOut(0, LCD_LINE7, test);
    NumOut(0, LCD_LINE8, tomorrow);
    Wait(SEC_5);
}
```

### 3.1.6.3 static

The static keyword is used to alter a variable declaration so that the variable is allocated statically - the lifetime of the variable extends across the entire run of the program - while having the same scope as variables declared without the static keyword.

Note that the initialization of automatic and static variables is quite different. Automatic variables (local variables are automatic by default, unless you explicitly use static keyword) are initialized during the run-time, so the initialization will be executed whenever it is encountered in the program. Static (and global) variables are initialized during the compile-time, so the initial values will simply be embedded in the executable file itself.

```
void func() {
    static int x = 0; // x is initialized only once across three calls of func()
    NumOut(0, LCD_LINE1, x); // outputs the value of x
    x = x + 1;
}

task main() {
    func(); // prints 0
    func(); // prints 1
    func(); // prints 2
}
```

#### 3.1.6.4 typedef

A typedef declaration introduces a name that, within its scope, becomes a synonym for the type given by the type-declaration portion of the declaration.

```
typedef type-declaration synonym;
```

You can use typedef declarations to construct shorter or more meaningful names for types already defined by the language or for types that you have declared. Typedef names allow you to encapsulate implementation details that may change.

A typedef declaration does not introduce a new type - it introduces a new name for an existing type. Here are a few examples of how to use the typedef keyword:

```
typedef char FlagType;
const FlagType x;
typedef char CHAR;           // Character type.
CHAR ch;
typedef unsigned long ulong;
ulong ul;                   // Equivalent to "unsigned long ul;"
```

## 3.2 Program Structure

An NXC program is composed of code blocks and variables.

There are two distinct types of code blocks: tasks and functions. Each type of code block has its own unique features, but they share a common structure. The maximum number of code blocks of both tasks and functions combined is 256.

- [Code Order](#)

- [Tasks](#)
- [Functions](#)
- [Variables](#)
- [Structures](#)
- [Arrays](#)

### 3.2.1 Code Order

Code order has two aspects: the order in which the code appears in the source code file and the order in which it is executed at runtime.

The first will be referred to as the lexical order and the second as the runtime order.

The lexical order is important to the NXC compiler, but not to the NXT brick. This means that the order in which you write your task and function definitions has no effect on the runtime order. The rules controlling runtime order are:

1. There must be a task called main and this task will always run first.
2. The time at which any other task will run is determined by the API functions documented in [Command module functions](#) section.
3. A function will run whenever it is called from another block of code.

This last rule may seem trivial, but it has important consequences when multiple tasks are running. If a task calls a function that is already in the midst of running because it was called first by another task, unpredictable behavior and results may ensue. Tasks can share functions by treating them as shared resources and using mutexes to prevent one task from calling the function while another task is using it. The [The safecall keyword](#) keyword (see [Functions](#)) may be used to simplify the coding.

The rules for lexical ordering are:

1. Any identifier naming a task or function must be known to the compiler before it is used in a code block.
2. A task or function definition makes its naming identifier known to the compiler.
3. A task or function declaration also makes a naming identifier known to the compiler.
4. Once a task or function is defined it cannot be redefined or declared.
5. Once a task or function is declared it cannot be redeclared.

Sometimes you will run into situations where it is impossible or inconvenient to order the task and function definitions so the compiler knows every task or function name before it sees that name used in a code block. You can work around this by inserting task or function declarations of the form

```
task name();  
  
return_type name(argument_list);
```

before the code block where the first usage occurs. The *argument\_list* must match the list of formal arguments given later in the function's actual definition.

### 3.2.2 Tasks

Since the NXT supports multi-threading, a task in NXC directly corresponds to an NXT thread.

Tasks are defined using the task keyword with the syntax shown in the code sample below.

```
task name()  
{  
    // the task's code is placed here  
}
```

The name of the task may be any legal identifier. A program must always have at least one task - named "main" - which is started whenever the program is run. The body of a task consists of a list of statements.

You can start and stop tasks with the start and stop statements, which are discussed below. However, the primary mechanism for starting dependant tasks is scheduling them with either the [Precedes](#) or the [Follows](#) API function.

The [StopAllTasks](#) API function stops all currently running tasks. You can also stop all tasks using the [Stop](#) function. A task can stop itself via the [ExitTo](#) function. Finally, a task will stop itself simply by reaching the end of its body.

In the code sample below, the main task schedules a music task, a movement task, and a controller task before exiting and allowing these three tasks to start executing concurrently. The controller task waits ten seconds before stopping the music task, and then waits another five seconds before stopping all tasks to end the program.

```
task music() {  
    while (true) {  
        PlayTone(TONE_A4, MS_500);  
        Wait(MS_600);  
    }  
}  
  
task movement() {  
    while (true) {  
        OnFwd(OUT_A, Random(100));  
        Wait(Random(SEC_1));  
    }  
}
```

```
    }  
  }  
  
  task controller() {  
    Wait(SEC_10);  
    stop music;  
    Wait(SEC_5);  
    StopAllTasks();  
  }  
  
  task main() {  
    Precedes(music, movement, controller);  
  }
```

### 3.2.3 Functions

It is often helpful to group a set of statements together into a single function, which your code can then call as needed.

NXC supports functions with arguments and return values. Functions are defined using the syntax below.

```
[safecall] [inline] return_type name(argument_list)  
{  
    // body of the function  
}
```

The return type is the type of data returned. In the C programming language, functions must specify the type of data they return. Functions that do not return data simply return void.

Additional details about the keywords safecall, inline, and void can be found below.

- [The safecall keyword](#)
- [The inline keyword](#)
- [The void keyword](#)

The argument list of a function may be empty, or may contain one or more argument definitions. An argument is defined by a type followed by a name. Commas separate multiple arguments. All values are represented as bool, char, byte, int, short, long, unsigned int, unsigned long, float, string, struct types, or arrays of any type.

NXC supports specifying a default value for function arguments that are not struct or array types. Simply add an equal sign followed by the default value. Specifying a default value makes the argument optional when you call the function. All optional arguments must be at the end of the argument list.

```
int foo(int x, int y = 20)  
{  
    return x*y;  
}
```

```
}  
  
task main()  
{  
    NumOut(0, LCD_LINE1, foo(10)); outputs 200  
    NumOut(0, LCD_LINE2, foo(10, 5)); outputs 50  
    Wait(SEC_10); // wait 10 seconds  
}
```

NXC also supports passing arguments by value, by constant value, by reference, and by constant reference. These four modes for passing parameters into a function are discussed below.

When arguments are passed by value from the calling function or task to the called function the compiler must allocate a temporary variable to hold the argument. There are no restrictions on the type of value that may be used. However, since the function is working with a copy of the actual argument, the caller will not see any changes the called function makes to the value. In the example below, the function `foo` attempts to set the value of its argument to 2. This is perfectly legal, but since `foo` is working on a copy of the original argument, the variable `y` from the main task remains unchanged.

```
void foo(int x)  
{  
    x = 2;  
}  
  
task main()  
{  
    int y = 1; // y is now equal to 1  
    foo(y); // y is still equal to 1!  
}
```

The second type of argument, `const arg_type`, is also passed by value. If the function is an inline function then arguments of this kind can sometimes be treated by the compiler as true constant values and can be evaluated at compile-time. If the function is not inline then the compiler treats the argument as if it were a constant reference, allowing you to pass either constants or variables. Being able to fully evaluate function arguments at compile-time can be important since some NXC API functions only work with true constant arguments.

```
void foo(const int x)  
{  
    PlayTone(x, MS_500);  
    x = 1; // error - cannot modify argument  
    Wait(SEC_1);  
}  
  
task main()  
{  
    int x = TONE_A4;  
    foo(TONE_A5); // ok  
    foo(4*TONE_A3); // expression is still constant  
    foo(x); // x is not a constant but is okay  
}
```

The third type, `arg_type &`, passes arguments by reference rather than by value. This allows the called function to modify the value and have those changes be available in

the calling function after the called function returns. However, only variables may be used when calling a function using `arg_type &` arguments:

```
void foo(int &x)
{
    x = 2;
}

task main()
{
    int y = 1;    // y is equal to 1

    foo(y); // y is now equal to 2
    foo(2); // error - only variables allowed
}
```

The fourth type, `const arg_type &`, is interesting. It is also passed by reference, but with the restriction that the called function is not allowed to modify the value. Because of this restriction, the compiler is able to pass anything, not just variables, to functions using this type of argument. Due to NXT firmware restrictions, passing an argument by reference in NXC is not as optimal as it is in C. A copy of the argument is still made but the compiler will enforce the restriction that the value may not be modified inside the called function.

Functions must be invoked with the correct number and type of arguments. The code example below shows several different legal and illegal calls to function `foo`.

```
void foo(int bar, const int baz)
{
    // do something here...
}

task main()
{
    int x; // declare variable x
    foo(1, 2); // ok
    foo(x, 2); // ok
    foo(2); // error - wrong number of arguments!
}
```

### 3.2.3.1 The safecall keyword

An optional keyword that can be specified prior to the return type of a function is the `safecall` keyword.

If a function is marked as `safecall` then the compiler will synchronize the execution of this function across multiple threads by wrapping each call to the function in `Acquire` and `Release` calls. If a second thread tries to call a `safecall` function while another thread is executing it the second thread will have to wait until the function returns to the first thread.

The code example below shows how you can use the `safecall` keyword to make a function synchronize its execution when it is shared between multiple threads.

```
safeCALL void foo(unsigned int frequency)
{
    PlayTone(frequency, SEC_1);
    Wait(SEC_1);
}

task task1()
{
    while(true) {
        foo(TONE_A4);
        Yield();
    }
}

task task2()
{
    while(true) {
        foo(TONE_A5);
        Yield();
    }
}

task main()
{
    Precedes(task1, task2);
}
```

### 3.2.3.2 The inline keyword

You can optionally mark NXC functions as inline functions.

This means that each call to the function will create another copy of the function's code. Unless used judiciously, inline functions can lead to excessive code size.

If a function is not marked as inline then an actual NXT subroutine is created and the call to the function in NXC code will result in a subroutine call to the NXT subroutine. The total number of non-inline functions (aka subroutines) and tasks must not exceed 256.

The code example below shows how you can use the inline keyword to make a function emit its code at the point where it is called rather than requiring a subroutine call.

```
inline void foo(unsigned int frequency)
{
    PlayTone(frequency, SEC_1);
    Wait(SEC_1);
}

task main()
{
    foo(TONE_A4);
    foo(TONE_B4);
    foo(TONE_C5);
    foo(TONE_D5);
}
```

In this case task main will contain 4 PlayTone calls and 4 Wait calls rather than 4 calls to the foo subroutine since it was expanded inline.

### 3.2.3.3 The void keyword

The void keyword allows you to define a function that returns no data.

Functions that do not return any value are sometimes referred to as procedures or sub-routines. The sub keyword is an alias for void. Both of these keywords can only be used when declaring or defining a function. Unlike C you cannot use void when declaring a variable type.

In NQC the void keyword was used to declare inline functions that could have arguments but could not return a value. In NXC void functions are not automatically inline as they were in NQC. To make a function inline you have to use the inline keyword prior to the function return type as described in the [Functions](#) section above.

- [The sub keyword](#)

**3.2.3.3.1 The sub keyword** The sub keyword allows you to define a function that returns no data.

Functions that do not return any value are sometimes referred to as procedures or sub-routines. The sub keyword is an alias for void. Both of these keywords can only be used when declaring or defining a function.

In NQC you used this keyword to define a true subroutine which could have no arguments and return no value. For the sake of C compatibility it is preferable to use the void keyword if you want to define a function that does not return a value.

## 3.2.4 Variables

All variables in NXC are defined using one of the types listed below:

- [bool](#)
- [byte](#)
- [char](#)
- [int](#)
- [short](#)
- [long](#)
- [unsigned](#)

- [float](#)
- [mutex](#)
- [string](#)
- [Structures](#)
- [Arrays](#)

Variables are declared using the keyword(s) for the desired type, followed by a comma-separated list of variable names and terminated by a semicolon (;). Optionally, an initial value for each variable may be specified using an equals sign (=) after the variable name. Several examples appear below:

```
int x;           // declare x
bool y,z;       // declare y and z
long a=1,b;     // declare a and b, initialize a to 1
float f=1.15, g; // declare f and g, initialize f
int data[10];  // an array of 10 zeros in data
bool flags[] = {true, true, false, false};
string msg = "hello world";
```

Global variables are declared at the program scope (outside of any code block). Once declared, they may be used within all tasks, functions, and subroutines. Their scope begins at declaration and ends at the end of the program.

Local variables may be declared within tasks and functions. Such variables are only accessible within the code block in which they are defined. Specifically, their scope begins with their declaration and ends at the end of their code block. In the case of local variables, a compound statement (a group of statements bracketed by '{' and '}') is considered a block:

```
int x; // x is global

task main()
{
    int y; // y is local to task main
    x = y; // ok
    {
        // begin compound statement
        int z; // local z declared
        y = z; // ok
    }
    y = z; // error - z no longer in scope
}

task foo()
{
    x = 1; // ok
    y = 2; // error - y is not global
}
```

### 3.2.4.1 bool

In NXC the bool type is an unsigned 8-bit value.

Normally you would only store a zero or one in a variable of this type but it can store values from zero to [UCHAR\\_MAX](#).

```
bool flag=true;
```

### 3.2.4.2 byte

In NXC the byte type is an unsigned 8-bit value.

This type can store values from zero to [UCHAR\\_MAX](#). You can also define an unsigned 8-bit variable using the [unsigned](#) keyword followed by the [char](#) type.

```
byte x=12;
unsigned char b = 0xE2;
```

### 3.2.4.3 char

In NXC the char type is a signed 8-bit value.

This type can store values from [SCHAR\\_MIN](#) to [SCHAR\\_MAX](#). The char type is often used to store the ASCII value of a single character. Use [Character Constants](#) page has more details about this usage.

```
char ch=12;
char test = 'A';
```

### 3.2.4.4 int

In NXC the int type is a signed 16-bit value.

This type can store values from [INT\\_MIN](#) to [INT\\_MAX](#). To declare an unsigned 16-bit value you have to use the [unsigned](#) keyword followed by the int type. The range of values that can be stored in an unsigned int variable is from zero to [UINT\\_MAX](#).

```
int x = 0xffff;
int y = -23;
unsigned int z = 62043;
```

### 3.2.4.5 short

In NXC the short type is a signed 16-bit value.

This type can store values from [SHRT\\_MIN](#) to [SHRT\\_MAX](#). This is an alias for the int type.

```
short x = 0xffff;
short y = -23;
```

### 3.2.4.6 long

In NXC the long type is a signed 32-bit value.

This type can store values from [LONG\\_MIN](#) to [LONG\\_MAX](#). To declare an unsigned 32-bit value you have to use the [unsigned](#) keyword followed by the long type. The range of values that can be stored in an unsigned long variable is from zero to [ULONG\\_MAX](#).

```
long x = 2147000000;
long y = -88235;
unsigned long b = 0xdeadbeef;
```

### 3.2.4.7 unsigned

The unsigned keyword is used to modify the char, int, and long types in order to define unsigned versions of these types.

The unsigned types can store the full 8-, 16-, and 32-bits of data without requiring that one of the bits be used to represent the sign of the value. This doubles the range of positive values that can be stored in each of these variable types.

```
unsigned char uc = 0xff;
unsigned int ui = 0xffff;
unsigned long ul = 0xffffffff;
```

### 3.2.4.8 float

In NXC the float type is a 32-bit IEEE 754 single precision floating point representation.

This is a binary format that occupies 32 bits (4 bytes) and its significand has a precision of 24 bits (about 7 decimal digits).

Floating point arithmetic will be slower than integer operations but if you need to easily store decimal values the float type is your best option. The standard NXT firmware provides the `sqrt` function which benefits from the ability to use the float type. In the enhanced NBC/NXC firmware there are many more native opcodes from the standard C math library which are designed to work with floats.

```
float pi = 3.14159;
float e = 2.71828;
float s2 = 1.4142;
```

### 3.2.4.9 mutex

In NXC the mutex type is a 32-bit value that is used to synchronize access to resources shared across multiple threads.

For this reason there is never a reason to declare a mutex variable inside a task or a function. It is designed for global variables that all tasks or functions can [Acquire](#) or

[Release](#) in order to obtain exclusive access to a resource that other tasks or functions are also trying to use.

```
mutex motorMutex;
task t1()
{
    while (true) {
        Acquire(motorMutex);
        // use the motor(s) protected by this mutex.
        Release(motorMutex);
        Wait (MS_500);
    }
}
task t2()
{
    while (true) {
        Acquire(motorMutex);
        // use the motor(s) protected by this mutex.
        Release(motorMutex);
        Wait (MS_200);
    }
}
task main()
{
    Precedes(t1, t2);
}
```

#### 3.2.4.10 string

In NXC the string type is provided for easily defining and manipulating strings which consist of an array of byte with a 0 or null value at the end of the array.

You can write strings to the NXC mailboxes, to files, and to the LCD, for example. You can initialize string variables using constant strings. See [String Constants](#) for additional details.

```
string msg = "Testing";
string ff = "Fred Flintstone";
```

#### 3.2.5 Structures

NXC supports user-defined aggregate types known as structs.

These are declared very much like you declare structs in a C program.

```
struct car
{
    string car_type;
    int manu_year;
};

struct person
{
```

```
    string name;
    int age;
    car vehicle;
};

person myPerson;
```

After you have defined the structure type you can use the new type to declare a variable or nested within another structure type declaration. Members (or fields) within the struct are accessed using a dot notation.

```
myPerson.age = 40;
anotherPerson = myPerson;
fooBar.car_type = "honda";
fooBar.manu_year = anotherPerson.age;
```

You can assign structs of the same type but the compiler will complain if the types do not match.

### 3.2.6 Arrays

NXC also support arrays.

Arrays are declared the same way as ordinary variables, but with an open and close bracket following the variable name.

```
int my_array[]; // declare an array with 0 elements
```

To declare arrays with more than one dimension simply add more pairs of square brackets. The maximum number of dimensions supported in NXC is 4.

```
bool my_array[][]; // declare a 2-dimensional array
```

Arrays of up to two dimensions may be initialized at the point of declaration using the following syntax:

```
int X[] = {1, 2, 3, 4}, Y[]={10, 10}; // 2 arrays
int matrix[][] = {{1, 2, 3}, {4, 5, 6}};
string cars[] = {"honda", "ford", "chevy"};
```

The elements of an array are identified by their position within the array (called an index). The first element has an index of 0, the second has index 1, and so on. For example:

```
my_array[0] = 123; // set first element to 123
my_array[1] = my_array[2]; // copy third into second
```

You may also initialize local arrays or arrays with multiple dimensions using the `ArrayInit` function. The following example shows how to initialize a two-dimensional array using `ArrayInit`. It also demonstrates some of the supported array API functions and expressions.

```
task main()
{
    int myArray[][];
    int myVector[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    byte fooArray[][][];

    ArrayInit(myArray, myVector, 10); // 10 vectors
    ArrayInit(fooArray, myArray, 2); // 2 myArrays

    fooArray[1] = myArray;
    myArray[1][4] = 34;

    int ax[], ay[];
    ArrayBuild(ax, 5, 7);
    ArrayBuild(ay, 2, 10, 6, 43);
    int axlen = ArrayLen(ax);
    ArraySubset(ax, ay, 1, 2); // ax = {10, 6}
    if (ax == ay) {
        // compare two arrays
        NumOut(0, LCD_LINE1, myArray[1][4]);
    }
}
```

NXC also supports specifying an initial size for both global and local arrays. The compiler automatically generates the required code to correctly initialize the array to zeros. If an array declaration includes both a size and a set of initial values the size is ignored in favor of the specified values.

```
task main()
{
    int myArray[10][10];
    int myVector[10];

    //ArrayInit(myVector, 0, 10); // 10 zeros in myVector
    //ArrayInit(myArray, myVector, 10); // 10 vectors myArray
}
```

The calls to `ArrayInit` are not required since we specified the initial sizes in the preceding array declarations, which means the arrays were already initialized to all zeros. In fact, the `myVector` array declaration is not needed unless we have a use for `myVector` other than initializing `myArray`.

### 3.3 Statements

The body of a code block (task or function) is composed of statements.

Statements are terminated with a semi-colon (;), as you have seen in the example code above.

- [Variable Declaration](#)
- [Assignment](#)

- [Control Structures](#)
- [The asm statement](#)
- [Other NXC Statements](#)

### 3.3.1 Variable Declaration

Variable declaration, which has already been discussed, is one type of statement.

Its purpose is to declare a local variable (with optional initialization) for use within the code block. The syntax for a variable declaration is shown below.

```
arg_type variables;
```

Here `arg_type` must be one of the types supported by NXC. Following the type are variable names, which must be a comma-separated list of identifiers with optional initial values as shown in the code fragment below.

```
name[=expression]
```

Arrays of variables may also be declared:

```
int array[n][=initializer];
```

You can also define variables using user-defined aggregate structure types.

```
struct TPerson {  
    int age;  
    string name;  
};  
TPerson bob; // cannot be initialized at declaration
```

### 3.3.2 Assignment

Once declared, variables may be assigned the value of an expression using the syntax shown in the code sample below.

```
variable assign_operator expression;
```

There are thirteen different assignment operators. The most basic operator, '=', simply assigns the value of the expression to the variable. The other operators modify the variable's value in some other way as shown in the table below.

| Operator | Action   |
|----------|--|
| =        | Set variable to expression                             |
| +=       | Add expression to variable                             |
| -=       | Subtract expression from variable                      |
| *=       | Multiple variable by expression                        |
| /=       | Divide variable by expression                          |
| %=       | Set variable to remainder after dividing by expression |
| &=       | Bitwise AND expression into variable                   |
| =        | Bitwise OR expression into variable                    |
| ^=       | Bitwise exclusive OR into variable                     |
| =        | Set variable to absolute value of expression           |
| +-=      | Set variable to sign (-1,+1,0) of expression           |
| >>=      | Right shift variable by expression                     |
| <<=      | Left shift variable by expression                      |

Table 3. Operators

The code sample below shows a few of the different types of operators that you can use in NXC expressions.

```
x = 2; // set x to 2
y = 7; // set y to 7
x += y; // x is 9, y is still 7
```

### 3.3.3 Control Structures

An NXC task or function usually contains a collection of nested control structures.

There are several types described below.

- [The compound statement](#)
- [The if statement](#)
- [The if-else statement](#)
- [The while statement](#)
- [The do statement](#)
- [The for statement](#)
- [The repeat statement](#)
- [The switch statement](#)
- [The goto statement](#)
- [The until statement](#)

### 3.3.3.1 The compound statement

The simplest control structure is a compound statement.

This is a list of statements enclosed within curly braces ('{' and '}'):

```
{
    x = 1;
    y = 2;
}
```

Although this may not seem very significant, it plays a crucial role in building more complicated control structures. Many control structures expect a single statement as their body. By using a compound statement, the same control structure can be used to control multiple statements.

### 3.3.3.2 The if statement

The if statement evaluates a condition.

If the condition is true, it executes one statement (the consequence). The value of a condition is considered to be false only when it evaluates to zero. If it evaluates to any non-zero value, it is true. The syntax for an if statement is shown below.

```
if (condition) consequence
```

The condition of an if-statement must be enclosed in parentheses, as shown in the code sample below. The compound statement in the last example allows two statements to execute as a consequence of the condition being true.

```
if (x==1) y = 2;
if (x==1) { y = 1; z = 2; }
```

### 3.3.3.3 The if-else statement

The if-else statement evaluates a condition.

If the condition is true, it executes one statement (the consequence). A second statement (the alternative), preceded by the keyword else, is executed if the condition is false. The value of a condition is considered to be false only when it evaluates to zero. If it evaluates to any non-zero value, it is true. The syntax for an if-else statement is shown below.

```
if (condition) consequence else alternative
```

The condition of an if-statement must be enclosed in parentheses, as shown in the code sample below. The compound statement in the last example allows two statements to execute as a consequence of the condition being true as well as two which execute when the condition is false.

```
if (x==1)
    y = 3;
else
    y = 4;
if (x==1) {
    y = 1;
    z = 2;
}
else {
    y = 3;
    z = 5;
}
```

### 3.3.3.4 The while statement

The while statement is used to construct a conditional loop.

The condition is evaluated, and if true the body of the loop is executed, then the condition is tested again. This process continues until the condition becomes false (or a break statement is executed). The syntax for a while loop appears in the code fragment below.

```
while (condition) body
```

Because the body of a while statement must be a single statement, it is very common to use a compound statement as the body. The sample below illustrates this usage pattern.

```
while(x < 10)
{
    x = x+1;
    y = y*2;
}
```

### 3.3.3.5 The do statement

A variant of the while loop is the do-while loop.

The syntax for this control structure is shown below.

```
do body while (condition)
```

The difference between a while loop and a do-while loop is that the do-while loop always executes the body at least once, whereas the while loop may not execute it at all.

```
do
{
    x = x+1;
    y = y*2;
} while(x < 10);
```

### 3.3.3.6 The for statement

Another kind of loop is the for loop.

This type of loop allows automatic initialization and incrementation of a counter variable. It uses the syntax shown below.

```
for(statement1 ; condition ; statement2) body
```

A for loop always executes statement1, and then it repeatedly checks the condition. While the condition remains true, it executes the body followed by statement2. The for loop is equivalent to the code shown below.

```
statement1;
while(condition)
{
    body
    statement2;
}
```

Frequently, statement1 sets a loop counter variable to its starting value. The condition is generally a relational statement that checks the counter variable against a termination value, and statement2 increments or decrements the counter value.

Here is an example of how to use the for loop:

```
for (int i=0; i<8; i++)
{
    NumOut (0, LCD_LINE1-i*8, i);
}
```

### 3.3.3.7 The repeat statement

The repeat statement executes a loop a specified number of times.

This control structure is not included in the set of Standard C looping constructs. NXC inherits this statement from NQC. The syntax is shown below.

```
repeat (expression) body
```

The expression determines how many times the body will be executed. Note: the expression following the repeat keyword is evaluated a single time and then the body is repeated that number of times. This is different from both the while and do-while loops which evaluate their condition each time through the loop.

Here is an example of how to use the repeat loop:

```
int i=0;
repeat (8)
{
    NumOut (0, LCD_LINE1-i*8, i++);
}
```

### 3.3.3.8 The switch statement

A switch statement executes one of several different code sections depending on the value of an expression.

One or more case labels precede each code section. Each case must be a constant and unique within the switch statement. The switch statement evaluates the expression, and then looks for a matching case label. It will execute any statements following the matching case until either a break statement or the end of the switch is reached. A single default label may also be used - it will match any value not already appearing in a case label. A switch statement uses the syntax shown below.

```
switch (expression) body
```

Additional information about the case and default labels and the break statement can be found below.

- [The case label](#)
- [The default label](#)
- [The break statement](#)

A typical switch statement might look like this:

```
switch(x)
{
    case 1:
        // do something when x is 1
        break;
    case 2:
    case 3:
        // do something else when x is 2 or 3
        break;
    default:
        // do this when x is not 1, 2, or 3
        break;
}
```

NXC also supports using string types in the switch expression and constant strings in case labels.

**3.3.3.8.1 The case label** The case label in a switch statement is not a statement in itself.

It is a label that precedes a list of statements. Multiple case labels can precede the same statement. The case label has the syntax shown below.

```
case constant_expression :
```

[The switch statement](#) page contains an example of how to use the case label.

**3.3.3.8.2 The default label** The default label in a switch statement is not a statement in itself.

It is a label that precedes a list of statements. There can be only one default label within a switch statement. The default label has the syntax shown below.

```
default :
```

The [switch statement](#) page contains an example of how to use the default label.

### 3.3.3.9 The goto statement

The goto statement forces a program to jump to the specified location.

Statements in a program can be labeled by preceding them with an identifier and a colon. A goto statement then specifies the label that the program should jump to. You can only branch to a label within the current function or task, not from one function or task to another.

Here is an example of an infinite loop that increments a variable:

```
my_loop:
    x++;
    goto my_loop;
```

The goto statement should be used sparingly and cautiously. In almost every case, control structures such as if, while, and switch make a program much more readable and maintainable than using goto.

### 3.3.3.10 The until statement

NXC also defines an until macro for compatibility with NQC.

This construct provides a convenient alternative to the while loop. The actual definition of until is shown below.

```
#define until(c)      while(!(c))
```

In other words, until will continue looping until the condition becomes true. It is most often used in conjunction with an empty body statement or a body which simply yields to other tasks:

```
until(EVENT_OCCURS); // wait for some event to occur
```

## 3.3.4 The asm statement

The asm statement is used to define many of the NXC API calls.

The syntax of the statement is shown below.

```
asm {
  one or more lines of NBC assembly language
}
```

The statement simply emits the body of the statement as NeXT Byte Codes (NBC) code and passes it directly to the NBC compiler's backend. The asm statement can often be used to optimize code so that it executes as fast as possible on the NXT firmware. The following example shows an asm block containing variable declarations, labels, and basic NBC statements as well as comments.

```
asm {
//      jmp __lb100D5
  dseg segment
    s10000 long
    s10005 long
    bGTTrue byte
  dseg ends
  mov     s10000, 0x0
  mov     s10005, s10000
  mov     s10000, 0x1
  cmp     GT, bGTTrue, s10005, s10000
  set bGTTrue, FALSE
  brtst  EQ, __lb100D5, bGTTrue
__lb100D5:
}
```

A few NXC keywords have meaning only within an asm statement. These keywords provide a means for returning string or scalar values from asm statements and for using temporary variables of byte, word, long, and float types.

| ASM Keyword  | Meaning  |
|--|--|
| <code>__RETURN__</code> , <code>__RETURNS__</code> | Used to return a signed value other than <code>__RETVAL__</code> or <code>__STRRETVAL__</code> |
| <code>__RETURNU__</code>                           | Used to return an unsigned value.  |
| <code>__RETURNF__</code>                           | Used to return a floating point value.   |
| <code>__RETVAL__</code>                            | Writing to this 4-byte signed value returns it to the calling program                          |
| <code>__GENRETVAL__</code>                         | Writing to this generic value returns it to the calling program                                |
| <code>__URETVAL__</code>                           | Writing to this 4-byte unsigned value returns it to the calling program                        |
| <code>__STRRETVAL__</code>                         | Writing to this string value returns it to the calling program                                 |
| <code>__FLTRETVAL__</code>                         | Writing to this 4-byte floating point value returns it to the calling program                  |
| <code>__STRBUFFER__</code>                         | This is primary string buffer which can be used to store intermediate string values.           |
| <code>__STRTMPBUFFER__</code>                      | This is a secondary string buffer.   |
| <code>__TMPBYTE__</code>                           | Use this temporary variable to write and return single byte signed values                      |
| <code>__TMPWORD__</code>                           | Use this temporary variable to write and return 2-byte signed values                           |
| <code>__TMPLONG__</code>                           | Use this temporary variable to write and return 4-byte signed values                           |
| <code>__TMPULONG__</code>                          | Use this temporary variable to write and return 4-byte unsigned values                         |
| <code>__TMPFLOAT__</code>                          | Use this temporary variable to write and return 4-byte floating point values                   |
| <code>__I__</code>                                 | A local counter variable   |
| <code>__J__</code>                                 | A second local counter variable  |
| <code>__IncI__</code>                              | Increment the local counter variable named I   |
| <code>__IncJ__</code>                              | Increment the local counter variable named J   |
| <code>__DecI__</code>                              | Decrement the local counter variable named I   |
| <code>__DecJ__</code>                              | Decrement the local counter variable named J   |
| <code>__ResetI__</code>                            | Reset the local counter variable named I to zero   |
| <code>__ResetJ__</code>                            | Reset the local counter variable named J to zero   |
| <code>__THREADNAME__</code>                        | The current thread name  |
| <code>__LINE__</code>                              | The current line number  |
| <code>__FILE__</code>                              | The current file name  |
| <code>__VER__</code>                               | The product version number   |

Table 4. ASM Keywords

The asm block statement and these special ASM keywords are used throughout the NXC API. You can have a look at the [NXCDefs.h](#) header file for several examples of how they are used. To keep the main NXC code as "C-like" as possible and for the sake of better readability NXC asm block statements can be wrapped in preprocessor macros and placed in custom header files which are included using `#include`. The following example demonstrates using a macro wrapper around an asm block.

```
#define SetMotorSpeed(port, cc, thresh, fast, slow) \  
asm { \  
    set theSpeed, fast \  
    brcmp cc, EndIfOut__I__, SV, thresh \  
    set theSpeed, slow \  
EndIfOut__I__: \  
    OnFwd(port, theSpeed) \  
    __IncI__ \  
}
```

### 3.3.5 Other NXC Statements

NXC supports a few other statement types.

The other NXC statements are described below.

- [The function call statement](#)
- [The start statement](#)
- [The stop statement](#)
- [The priority statement](#)
- [The break statement](#)
- [The continue statement](#)
- [The return statement](#)

Many expressions are not legal statements. A notable exception are expressions using increment (++) or decrement (--) operators.

```
x++;
```

The empty statement (just a bare semicolon) is also a legal statement.

#### 3.3.5.1 The function call statement

A function call can also be a statement of the following form:

```
name (arguments) ;
```

The arguments list is a comma-separated list of expressions. The number and type of arguments supplied must match the definition of the function itself. Optionally, the return value may be assigned to a variable.

### 3.3.5.2 The start statement

You can start a task with the start statement.

This statement can be used with both the standard and enhanced NBC/NXC firmwares. The resulting operation is a native opcode in the enhanced firmware but it requires special compiler-generated subroutines in order to work with the standard firmware.

```
start task_name;
```

### 3.3.5.3 The stop statement

You can stop a task with the stop statement.

The stop statement is only supported if you are running the enhanced NBC/NXC firmware on your NXT.

```
stop task_name;
```

### 3.3.5.4 The priority statement

You can adjust the priority of a task using the priority statement.

Setting task priorities also requires the enhanced NBC/NXC firmware. A task's priority is simply the number of operations it will try to execute before yielding to another task. This usually is 20 operations.

```
priority task_name, new_priority;
```

### 3.3.5.5 The break statement

Within loops (such as a while loop) you can use the break statement to exit the loop immediately.

It only exits out of the innermost loop

```
break;
```

The break statement is also a critical component of most switch statements. It prevents code in subsequent code sections from being executed, which is usually a programmer's intent, by immediately exiting the switch statement. Missing break statements in a switch are a frequent source of hard-to-find bugs.

Here is an example of how to use the break statement:

```
while (x<100) {
  x = get_new_x();
  if (button_pressed())
    break;
  process(x);
}
```

#### 3.3.5.6 The continue statement

Within loops you can use the continue statement to skip to the top of the next iteration of the loop without executing any of the code in the loop that follows the continue statement.

```
continue;
```

Here is an example of how to use the continue statement:

```
while (x<100) {
  ch = get_char();
  if (ch != 's')
    continue;
  process(ch);
}
```

#### 3.3.5.7 The return statement

If you want a function to return a value or to return before it reaches the end of its code, use a return statement.

An expression may optionally follow the return keyword and, when present, is the value returned by the function. The type of the expression must be compatible with the return type of the function.

```
return [expression];
```

## 3.4 Expressions

Values are the most primitive type of expressions.

More complicated expressions are formed from values using various operators.

Numerical constants in the NXT are represented as integers or floating point values. The type depends on the value of the constant. NXC internally uses 32 bit floating point math for constant expression evaluation. Numeric constants are written as either decimal (e.g. 123, 3.14) or hexadecimal (e.g. 0xABC). Presently, there is very

little range checking on constants, so using a value larger than expected may produce unusual results.

Two special values are predefined: true and false. The value of false is zero (0), while the value of true is one (1). The same values hold for relational operators (e.g. <): when the relation is false the value is 0, otherwise the value is 1.

Values may be combined using operators. NXC operators are listed here in order of precedence from highest to lowest.

| Operator                              | Description                       | Associativity | Restriction    | Example                     |
|---------------------------------------|-----------------------------------|---------------|----------------|-----------------------------|
| <code>abs()</code>                    | Absolute value                    | n/a           |                | <code>abs(x)</code>         |
| <code>sign()</code>                   | Sign of operand                   | n/a           |                | <code>sign(x)</code>        |
| <code>++, --</code>                   | Postfix increment/decrement       | left          | variables only | <code>x++</code>            |
| <code>++, --</code>                   | Prefix increment/decrement        | right         | variables only | <code>++x</code>            |
| <code>-</code>                        | Unary minus                       | right         |                | <code>-x</code>             |
| <code>~</code>                        | Bitwise negation (unary)          | right         |                | <code>~123</code>           |
| <code>!</code>                        | Logical negation                  | right         |                | <code>!x</code>             |
| <code>*, /, %</code>                  | Multiplication, division, modulus | left          |                | <code>x * y</code>          |
| <code>+, -</code>                     | Addition, subtraction             | left          |                | <code>x + y</code>          |
| <code>&lt;&lt;, &gt;&gt;</code>       | Bitwise shift left and right      | left          |                | <code>x &lt;&lt; 4</code>   |
| <code>&lt;, &gt;, &lt;=, &gt;=</code> | relational operators              | left          |                | <code>x &lt; y</code>       |
| <code>==, !=</code>                   | equal to, not equal to            | left          |                | <code>x == 1</code>         |
| <code>&amp;</code>                    | Bitwise AND                       | left          |                | <code>x &amp; y</code>      |
| <code>^</code>                        | Bitwise exclusive OR              | left          |                | <code>x ^ y</code>          |
| <code> </code>                        | Bitwise inclusive OR              | left          |                | <code>x   y</code>          |
| <code>&amp;&amp;</code>               | Logical AND                       | left          |                | <code>x &amp;&amp; y</code> |
| <code>  </code>                       | Logical OR                        | left          |                | <code>x    y</code>         |
| <code>?:</code>                       | Ternary conditional value         | right         |                | <code>x==1 ? y : z</code>   |

Table 5. Expression Operators

Where needed, parentheses are used to change the order of evaluation:

```
x = 2 + 3 * 4; // set x to 14
y = (2 + 3) * 4; // set y to 20
```

- [Conditions](#)

### 3.4.1 Conditions

Comparing two expressions forms a condition.

A condition may be negated with the logical negation operator, or two conditions combined with the logical AND and logical OR operators. Like most modern computer languages, NXC supports something called "short-circuit" evaluation of conditions. This means that if the entire value of the conditional can be logically determined by only evaluating the left hand term of the condition, then the right hand term will not be evaluated.

The table below summarizes the different types of conditions.

| Condition      | Meaning  |
|----------------|--|
| Expr           | true if expr is not equal to 0   |
| Expr1 == expr2 | true if expr1 equals expr2   |
| Expr1 != expr2 | true if expr1 is not equal to expr2  |
| Expr1 < expr2  | true if one expr1 is less than expr2   |
| Expr1 <= expr2 | true if expr1 is less than or equal to expr2   |
| Expr1 > expr2  | true if expr1 is greater than expr2  |
| Expr1 >= expr2 | true if expr1 is greater than or equal to expr2  |
| ! condition    | logical negation of a condition - true if condition is false                               |
| Cond1 && cond2 | logical AND of two conditions (true if and only if both conditions are true)               |
| Cond1    cond2 | logical OR of two conditions (true if and only if at least one of the conditions are true) |

Table 6. Conditions

There are also two special constant conditions which can be used anywhere that the above conditions are allowed. They are listed below.

- [The true condition](#)
- [The false condition](#)

You can use conditions in NXC control structures, such as the if-statement and the while or until statements, to specify exactly how you want your program to behave.

#### 3.4.1.1 The true condition

The keyword true has a value of one.

It represents a condition that is always true.

### 3.4.1.2 The false condition

The keyword `false` has a value of zero.

It represents a condition that is always false.

## 3.5 The Preprocessor

NXC also includes a preprocessor that is modeled after the Standard C preprocessor.

The C preprocessor processes a source code file before the compiler does. It handles such tasks as including code from other files, conditionally including or excluding blocks of code, stripping comments, defining simple and parameterized macros, and expanding macros wherever they are encountered in the source code.

The NXC preprocessor implements the following standard preprocessor directives: `#include`, `#define`, `#ifdef`, `#ifndef`, `#endif`, `#if`, `#elif`, `#undef`, `##`, `#line`, `#error`, and `#pragma`. It also supports two non-standard directives: `#download` and `#import`. Its implementation is close to a standard C preprocessor's, so most preprocessor directives should work as C programmers expect in NXC. Any significant deviations are explained below.

- [include](#)
- [define](#)
- [## \(Concatenation\)](#)
- [Conditional Compilation](#)
- [import](#)
- [download](#)

### 3.5.1 #include

The `#include` command works as in Standard C, with the caveat that the filename must be enclosed in double quotes.

There is no notion of a system include path, so enclosing a filename in angle brackets is forbidden.

```
#include "foo.h" // ok
#include <foo.h> // error!
```

NXC programs can begin with `#include "NXCDefs.h"` but they don't need to. This standard header file includes many important constants and macros, which form the core NXC API. NXC no longer require that you manually include the [NXCDefs.h](#) header file. Unless you specifically tell the compiler to ignore the standard system files, this header file is included automatically.

### 3.5.2 #define

The `#define` command is used for macro substitution.

Redefinition of a macro will result in a compiler warning. Macros are normally restricted to one line because the newline character at the end of the line acts as a terminator. However, you can write multiline macros by instructing the preprocessor to ignore the newline character. This is accomplished by escaping the newline character with a backslash (`'\'`). The backslash character must be the very last character in the line or it will not extend the macro definition to the next line. The code sample below shows how to write a multi-line preprocessor macro.

```
#define foo(x) do { bar(x); \  
                baz(x); } while(false)
```

The `#undef` directive may be used to remove a macro's definition.

### 3.5.3 ## (Concatenation)

The `##` directive works similar to the C preprocessor.

It is replaced by nothing, which causes tokens on either side to be concatenated together. Because it acts as a separator initially, it can be used within macro functions to produce identifiers via combination with parameter values.

```
#define ELEMENT_OUT(n) \  
    NumOut(0, LCD_LINE##n, b##n)  
  
bool b1 = false;  
bool b2 = true;  
  
task main()  
{  
    ELEMENT_OUT(1);  
    ELEMENT_OUT(2);  
    Wait(SEC_2);  
}
```

This is the same as writing

```
bool b1 = false;  
bool b2 = true;  
  
task main()  
{  
    NumOut(0, LCD_LINE1, b1);  
    NumOut(0, LCD_LINE2, b2);  
    Wait(SEC_2);  
}
```

### 3.5.4 Conditional Compilation

Conditional compilation works similar to the C preprocessor's conditional compilation.

The following preprocessor directives may be used:

| Directive                   | Meaning  |
|-----------------------------|--|
| <code>#ifdef symbol</code>  | If symbol is defined then compile the following code               |
| <code>#ifndef symbol</code> | If symbol is not defined then compile the following code           |
| <code>#else</code>          | Switch from compiling to not compiling and vice versa              |
| <code>#endif</code>         | Return to previous compiling state                                 |
| <code>#if condition</code>  | If the condition evaluates to true then compile the following code |
| <code>#elif</code>          | Same as <code>#else</code> but used with <code>#if</code>          |

Table 7. Conditional compilation directives

See the `NXTDefs.h` and `NXCDefs.h` header files for many examples of how to use conditional compilation.

### 3.5.5 `#import`

The `#import` directive lets you define a global byte array variable in your NXC program that contains the contents of the imported file.

Like `#include`, this directive is followed by a filename enclosed in double quote characters. Following the filename you may optionally include a format string for constructing the name of the variable you want to define using this directive.

```
#import "myfile.txt" data
```

By default, the format string is `s` which means that the name of the file without any file extension will be the name of the variable. For instance, if the format string `"data"` were not specified in the example above, then the name of the byte array variable would be `"myfile"`. In this case the name of the byte array variable will be `"data"`.

The `#import` directive is often used in conjunction with the [GraphicArrayOut](#) and [GraphicArrayOutEx](#) API functions.

### 3.5.6 `#download`

The `#download` directive works in conjunction with the compiler's built-in download capability.

It lets you tell the compiler to download a specified auxiliary file in addition to the `.rx` file produced from your source code. If the file extension matches a type of source code that the compiler knows how to compile (such as `.rs` or `.nbc`) then the compiler will first compile the source before downloading the resulting binary. The name of the file to

download (and optionally compile) is enclosed in double quote characters immediately following this directive. If the compiler is only told to compile the original source code then the #download directive is ignored.

```
#download "myfile.rs"  
#download "mypicture.pic"
```

## 4 Todo List

Global `<globalScope>::StopSound()` ?.

Global `<globalScope>::SysComputeCalibValue(ComputeCalibValueType &args)`  
figure out what this function is intended for

Global `<globalScope>::SysDatalogGetTimes(DatalogGetTimesType &args)`  
figure out what this function is intended for

Global `<globalScope>::SysDatalogWrite(DatalogWriteType &args)` figure out  
what this function is intended for

Global `<globalScope>::SysUpdateCalibCacheInfo(UpdateCalibCacheInfoType &args)`  
figure out what this function is intended for

Global `CommHSControlType::Result` values?

Global `ComputeCalibValueType::Name` ?.

Global `ComputeCalibValueType::RawVal` ?.

Global `ComputeCalibValueType::Result` ?.

Global `UpdateCalibCacheInfoType::Name` ?.

Global `UpdateCalibCacheInfoType::Result` ?.

## 5 Deprecated List

Global `<globalScope>::Acos(_X)` Use `acos()` instead.

Global `<globalScope>::AcosD(_X)` Use `acosd()` instead.

Global `<globalScope>::Asin(_X)` Use `asin()` instead.

Global `<globalScope>::AsinD(_X)` Use `asind()` instead.

Global `<globalScope>::Atan(_X)` Use `atan()` instead.

Global `<globalScope>::Atan2(_Y, _X)` Use `atan2()` instead.

Global `<globalScope>::Atan2D(_Y, _X)` Use `atan2d()` instead.

Global `<globalScope>::AtanD(_X)` Use `atand()` instead.

Global `<globalScope>::Ceil(_X)` Use `ceil()` instead.

Global `<globalScope>::Cos(_X)` Use `cos()` instead.

Global `<globalScope>::CosD(_X)` Use `cosd()` instead.

Global `<globalScope>::Cosh(_X)` Use `cosh()` instead.

Global `<globalScope>::CoshD(_X)` Use `coshd()` instead.

Global `<globalScope>::Exp(_X)` Use `exp()` instead.

Global `<globalScope>::Floor(_X)` Use `floor()` instead.

Global `<globalScope>::Frac(_X)` Use `frac()` instead.

Global `<globalScope>::Log(_X)` Use `log()` instead.

Global `<globalScope>::Log10(_X)` Use `log10()` instead.

Global `<globalScope>::MulDiv32(_A, _B, _C)` Use `muldiv32()` instead.

Global `<globalScope>::Pow(_Base, _Exponent)` Use `pow()` instead.

Global `<globalScope>::Sin(_X)` Use `sin()` instead.

Global `<globalScope>::SinD(_X)` Use `sind()` instead.

Global `<globalScope>::Sinh(_X)` Use `sinh()` instead.

Global `<globalScope>::SinhD(_X)` Use `sinhd()` instead.

Global `<globalScope>::Sqrt(_X)` Use `sqrt()` instead.

Global `<globalScope>::Tan(_X)` Use `tan()` instead.

Global `<globalScope>::TanD(_X)` Use `tand()` instead.

Global `<globalScope>::Tanh(_X)` Use `tanh()` instead.

Global `<globalScope>::TanhD(_X)` Use `tanhd()` instead.

Global `<globalScope>::Trunc(_X)` Use `trunc()` instead.

## 6 Module Documentation

### 6.1 NXT Firmware Modules

Documentation common to all NXT firmware modules.

#### Modules

- [Input module](#)  
*Constants and functions related to the Input module.*
- [Output module](#)  
*Constants and functions related to the Output module.*
- [Display module](#)  
*Constants and functions related to the Display module.*
- [Sound module](#)  
*Constants and functions related to the Sound module.*
- [Low Speed module](#)  
*Constants and functions related to the Low Speed module.*
- [Command module](#)  
*Constants and functions related to the Command module.*
- [IOCtrl module](#)  
*Constants and functions related to the IOCtrl module.*
- [Comm module](#)  
*Constants and functions related to the Comm module.*
- [Button module](#)  
*Constants and functions related to the Button module.*
- [Ui module](#)  
*Constants and functions related to the Ui module.*
- [Loader module](#)  
*Constants and functions related to the Loader module.*
- [NXT firmware module names](#)

*Constant string names for all the NXT firmware modules.*

- [NXT firmware module IDs](#)

*Constant numeric IDs for all the NXT firmware modules.*

### 6.1.1 Detailed Description

Documentation common to all NXT firmware modules.

## 6.2 Input module

Constants and functions related to the Input module.

### Modules

- [Input module types](#)

*Types used by various input module functions.*

- [Input module functions](#)

*Functions for accessing and modifying input module features.*

- [Input module constants](#)

*Constants that are part of the NXT firmware's Input module.*

### 6.2.1 Detailed Description

Constants and functions related to the Input module. The NXT input module encompasses all sensor inputs except for digital I2C (LowSpeed) sensors.

There are four sensors, which internally are numbered 0, 1, 2, and 3. This is potentially confusing since they are externally labeled on the NXT as sensors 1, 2, 3, and 4. To help mitigate this confusion, the sensor port names [S1](#), [S2](#), [S3](#), and [S4](#) have been defined. See [Input port constants](#). These sensor names may be used in any function that requires a sensor port as an argument. Alternatively, the NBC port name constants [IN\\_1](#), [IN\\_2](#), [IN\\_3](#), and [IN\\_4](#) may also be used when a sensor port is required, although this is not recommended. See [NBC Input port constants](#). Sensor value names [SENSOR\\_1](#), [SENSOR\\_2](#), [SENSOR\\_3](#), and [SENSOR\\_4](#) have also been defined. These names may also be used whenever a program wishes to read the current value of the analog sensor:

```
x = SENSOR_1; // read sensor and store value in x
```

## 6.3 Input module constants

Constants that are part of the NXT firmware's Input module.

### Modules

- [Input port constants](#)  
*Input port constants are used when calling NXC sensor control API functions.*
- [NBC Input port constants](#)  
*Input port constants are used when calling sensor control API functions.*
- [Input field constants](#)  
*Constants for use with `SetInput()` and `GetInput()`.*
- [Input port digital pin constants](#)  
*Constants for use when directly controlling or reading a port's digital pin state.*
- [Color sensor array indices](#)  
*Constants for use with color sensor value arrays to index RGB and blank return values.*
- [Color values](#)  
*Constants for use with the `ColorValue` returned by the color sensor in full color mode.*
- [Color calibration state constants](#)  
*Constants for use with the color calibration state function.*
- [Color calibration constants](#)  
*Constants for use with the color calibration functions.*
- [Input module IOMAP offsets](#)  
*Constant offsets into the Input module IOMAP structure.*
- [Constants to use with the Input module's Pin function](#)  
*Constants for use with the Input module's Pin function.*
- [Sensor types and modes](#)  
*Constants that are used for defining sensor types and modes.*

### Defines

- #define [INPUT\\_CUSTOMINACTIVE](#) 0x00
- #define [INPUT\\_CUSTOM9V](#) 0x01
- #define [INPUT\\_CUSTOMACTIVE](#) 0x02
- #define [INPUT\\_INVALID\\_DATA](#) 0x01

#### 6.3.1 Detailed Description

Constants that are part of the NXT firmware's Input module.

#### 6.3.2 Define Documentation

##### 6.3.2.1 #define INPUT\_CUSTOM9V 0x01

Custom sensor 9V

##### 6.3.2.2 #define INPUT\_CUSTOMACTIVE 0x02

Custom sensor active

##### 6.3.2.3 #define INPUT\_CUSTOMINACTIVE 0x00

Custom sensor inactive

##### 6.3.2.4 #define INPUT\_INVALID\_DATA 0x01

Invalid data flag

## 6.4 Sensor types and modes

Constants that are used for defining sensor types and modes.

### Modules

- [Sensor type constants](#)  
*Use sensor type constants to configure an input port for a specific type of sensor.*
- [Sensor mode constants](#)  
*Use sensor mode constants to configure an input port for the desired sensor mode.*

- [Combined sensor type and mode constants](#)

*Use the combined sensor type and mode constants to configure both the sensor mode and type in a single function call.*

- [NBC sensor type constants](#)

*Use sensor type constants to configure an input port for a specific type of sensor.*

- [NBC sensor mode constants](#)

*Use sensor mode constants to configure an input port for the desired sensor mode.*

### 6.4.1 Detailed Description

Constants that are used for defining sensor types and modes. The sensor ports on the NXT are capable of interfacing to a variety of different sensors. It is up to the program to tell the NXT what kind of sensor is attached to each port. Calling [SetSensorType](#) configures a sensor's type. There are 16 sensor types, each corresponding to a specific type of LEGO RCX or NXT sensor. Two of these types are for NXT I2C digital sensors, either 9V powered or unpowered, and a third is used to configure port S4 as a high-speed RS-485 serial port. A seventeenth type ([SENSOR\\_TYPE\\_CUSTOM](#)) is for use with custom analog sensors. And an eighteenth type ([SENSOR\\_TYPE\\_NONE](#)) is used to indicate that no sensor has been configured, effectively turning off the specified port.

In general, a program should configure the type to match the actual sensor. If a sensor port is configured as the wrong type, the NXT may not be able to read it accurately. Use either the [Sensor type constants](#) or the [NBC sensor type constants](#).

The NXT allows a sensor to be configured in different modes. The sensor mode determines how a sensor's raw value is processed. Some modes only make sense for certain types of sensors, for example [SENSOR\\_MODE\\_ROTATION](#) is useful only with rotation sensors. Call [SetSensorMode](#) to set the sensor mode. The possible modes are shown below. Use either the [Sensor mode constants](#) or the [NBC sensor mode constants](#).

When using the NXT, it is common to set both the type and mode at the same time. The [SetSensor](#) function makes this process a little easier by providing a single function to call and a set of standard type/mode combinations. Use the [Combined sensor type and mode constants](#).

The NXT provides a boolean conversion for all sensors - not just touch sensors. This boolean conversion is normally based on preset thresholds for the raw value. A "low" value (less than 460) is a boolean value of 1. A high value (greater than 562) is a boolean value of 0. This conversion can be modified: a slope value between 0 and 31 may be added to a sensor's mode when calling [SetSensorMode](#). If the sensor's value changes more than the slope value during a certain time (3ms), then the sensor's

boolean state will change. This allows the boolean state to reflect rapid changes in the raw value. A rapid increase will result in a boolean value of 0, a rapid decrease is a boolean value of 1.

Even when a sensor is configured for some other mode (i.e. [SENSOR\\_MODE\\_PERCENT](#)), the boolean conversion will still be carried out.

## 6.5 Output module

Constants and functions related to the Output module.

### Modules

- [Output module types](#)  
*Types used by various output module functions.*
- [Output module functions](#)  
*Functions for accessing and modifying output module features.*
- [Output module constants](#)  
*Constants that are part of the NXT firmware's Output module.*

### 6.5.1 Detailed Description

Constants and functions related to the Output module. The NXT output module encompasses all the motor outputs.

Nearly all of the NXC API functions dealing with outputs take either a single output or a set of outputs as their first argument. Depending on the function call, the output or set of outputs may be a constant or a variable containing an appropriate output port value. The constants [OUT\\_A](#), [OUT\\_B](#), and [OUT\\_C](#) are used to identify the three outputs. Unlike NQC, adding individual outputs together does not combine multiple outputs. Instead, the NXC API provides predefined combinations of outputs: [OUT\\_AB](#), [OUT\\_AC](#), [OUT\\_BC](#), and [OUT\\_ABC](#). Manually combining outputs involves creating an array and adding two or more of the three individual output constants to the array.

Output power levels can range 0 (lowest) to 100 (highest). Negative power levels reverse the direction of rotation (i.e., forward at a power level of -100 actually means reverse at a power level of 100).

The outputs each have several fields that define the current state of the output port. These fields are defined in the [Output field constants](#) section.

## 6.6 Output module constants

Constants that are part of the NXT firmware's Output module.

### Modules

- [Output port constants](#)  
*Output port constants are used when calling motor control API functions.*
- [PID constants](#)  
*PID constants are for adjusting the Proportional, Integral, and Derivative motor controller parameters.*
- [Output port update flag constants](#)  
*Use these constants to specify which motor values need to be updated.*
- [Tachometer counter reset flags](#)  
*Use these constants to specify which of the three tachometer counters should be reset.*
- [Output port mode constants](#)  
*Use these constants to configure the desired mode for the specified motor(s): coast, motoron, brake, or regulated.*
- [Output port option constants](#)  
*Use these constants to configure the desired options for the specified motor(s): hold at limit and ramp down to limit.*
- [Output regulation option constants](#)  
*Use these constants to configure the desired options for position regulation.*
- [Output port run state constants](#)  
*Use these constants to configure the desired run state for the specified motor(s): idle, rampup, running, rampdown, or hold.*
- [Output port regulation mode constants](#)  
*Use these constants to configure the desired regulation mode for the specified motor(s): none, speed regulation, multi-motor synchronization, or position regulation (requires the enhanced NBC/NXC firmware version 1.31+).*
- [Output field constants](#)  
*Constants for use with `SetOutput()` and `GetOutput()`.*
- [Output module IOMAP offsets](#)  
*Constant offsets into the Output module IOMAP structure.*

### 6.6.1 Detailed Description

Constants that are part of the NXT firmware's Output module.

## 6.7 Command module

Constants and functions related to the Command module.

### Modules

- [Command module types](#)  
*Types used by various Command module functions.*
- [Command module functions](#)  
*Functions for accessing and modifying Command module features.*
- [Command module constants](#)  
*Constants that are part of the NXT firmware's Command module.*

### 6.7.1 Detailed Description

Constants and functions related to the Command module. The NXT command module encompasses support for the execution of user programs via the NXT virtual machine. It also implements the direct command protocol support that enables the NXT to respond to USB or Bluetooth requests from other devices such as a PC or another NXT brick.

## 6.8 Command module constants

Constants that are part of the NXT firmware's Command module.

### Modules

- [Array operation constants](#)  
*Constants for use with the NXC ArrayOp function and the NBC arrop statement.*
- [System Call function constants](#)  
*Constants for use in the SysCall() function or NBC syscall statement.*
- [Time constants](#)

Constants for use with the `Wait()` function.

- **VM state constants**

Constants defining possible VM states.

- **Fatal errors**

Constants defining various fatal error conditions.

- **General errors**

Constants defining general error conditions.

- **Communications specific errors**

Constants defining communication error conditions.

- **Remote control (direct commands) errors**

Constants defining errors that can occur during remote control (RC) direct command operations.

- **Program status constants**

Constants defining various states of the command module virtual machine.

- **Command module IOMAP offsets**

Constant offsets into the Command module IOMAP structure.

## Defines

- `#define STAT_MSG_EMPTY_MAILBOX 64`
- `#define STAT_COMM_PENDING 32`
- `#define POOL_MAX_SIZE 32768`
- `#define NO_ERR 0`

### 6.8.1 Detailed Description

Constants that are part of the NXT firmware's Command module.

### 6.8.2 Define Documentation

#### 6.8.2.1 `#define NO_ERR 0`

Successful execution of the specified command

**Examples:**

[ex\\_joystickmsg.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommBTOnOff.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_SysCommHSWrite.nxc](#), [ex\\_syscommhwwrite.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_SysDatalogWrite.nxc](#), [ex\\_sysfileopenappend.nxc](#), [ex\\_sysfileopenread.nxc](#), [ex\\_sysfileopenreadlinear.nxc](#), [ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), [ex\\_sysfileopenwritenonlinear.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfileresize.nxc](#), [ex\\_sysfileseek.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_sysiomapread.nxc](#), [ex\\_sysiomapreadbyid.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_sysmessageread.nxc](#), and [ex\\_SysReadLastResponse.nxc](#).

**6.8.2.2 #define POOL\_MAX\_SIZE 32768**

Maximum size of memory pool, in bytes

**6.8.2.3 #define STAT\_COMM\_PENDING 32**

Pending setup operation in progress

**6.8.2.4 #define STAT\_MSG\_EMPTY\_MAILBOX 64**

Specified mailbox contains no new messages

**6.9 Comm module**

Constants and functions related to the Comm module.

**Modules**

- [Comm module types](#)  
*Types used by various Comm module functions.*
- [Comm module functions](#)  
*Functions for accessing and modifying Comm module features.*
- [Comm module constants](#)  
*Constants that are part of the NXT firmware's Comm module.*

### 6.9.1 Detailed Description

Constants and functions related to the Comm module. The NXT comm module encompasses support for all forms of Bluetooth, USB, and HiSpeed communication.

You can use the Bluetooth communication methods to send information to other devices connected to the NXT brick. The NXT firmware also implements a message queuing or mailbox system which you can access using these methods.

Communication via Bluetooth uses a master/slave connection system. One device must be designated as the master device before you run a program using Bluetooth. If the NXT is the master device then you can configure up to three slave devices using connection 1, 2, and 3 on the NXT brick. If your NXT is a slave device then connection 0 on the brick must be reserved for the master device.

Programs running on the master NXT brick can send packets of data to any connected slave devices using the BluetoothWrite method. Slave devices write response packets to the message queuing system where they wait for the master device to poll for the response.

Using the direct command protocol, a master device can send messages to slave NXT bricks in the form of text strings addressed to a particular mailbox. Each mailbox on the slave NXT brick is a circular message queue holding up to five messages. Each message can be up to 58 bytes long.

To send messages from a master NXT brick to a slave brick, use [BluetoothWrite](#) on the master brick to send a MessageWrite direct command packet to the slave. Then, you can use [ReceiveMessage](#) on the slave brick to read the message. The slave NXT brick must be running a program when an incoming message packet is received. Otherwise, the slave NXT brick ignores the message and the message is dropped.

## 6.10 Button module

Constants and functions related to the Button module.

### Modules

- [Button module types](#)  
*Types used by various Button module functions.*
- [Button module functions](#)  
*Functions for accessing and modifying Button module features.*
- [Button module constants](#)  
*Constants that are part of the NXT firmware's Button module.*

### 6.10.1 Detailed Description

Constants and functions related to the Button module. The NXT button module encompasses support for the 4 buttons on the NXT brick.

## 6.11 IOCtrl module

Constants and functions related to the IOCtrl module.

### Modules

- [IOCtrl module types](#)  
*Types used by various IOCtrl module functions.*
- [IOCtrl module functions](#)  
*Functions for accessing and modifying IOCtrl module features.*
- [IOCtrl module constants](#)  
*Constants that are part of the NXT firmware's IOCtrl module.*

### 6.11.1 Detailed Description

Constants and functions related to the IOCtrl module. The NXT ioctrl module encompasses low-level communication between the two processors that control the NXT. The NXC API exposes two functions that are part of this module.

## 6.12 Loader module

Constants and functions related to the Loader module.

### Modules

- [Loader module types](#)  
*Types used by various Loader module functions.*
- [Loader module functions](#)  
*Functions for accessing and modifying Loader module features.*
- [Loader module constants](#)  
*Constants that are part of the NXT firmware's Loader module.*

### 6.12.1 Detailed Description

Constants and functions related to the Loader module. The NXT loader module encompasses support for the NXT file system. The NXT supports creating files, opening existing files, reading, writing, renaming, and deleting files.

Files in the NXT file system must adhere to the 15.3 naming convention for a maximum filename length of 19 characters. While multiple files can be opened simultaneously, a maximum of 4 files can be open for writing at any given time.

When accessing files on the NXT, errors can occur. The NXC API defines several constants that define possible result codes. They are listed in the [Loader module error codes](#) section.

## 6.13 Sound module

Constants and functions related to the Sound module.

### Modules

- [Sound module types](#)

*Types used by various sound module functions.*

- [Sound module functions](#)

*Functions for accessing and modifying sound module features.*

- [Sound module constants](#)

*Constants that are part of the NXT firmware's Sound module.*

### 6.13.1 Detailed Description

Constants and functions related to the Sound module. The NXT sound module encompasses all sound output features. The NXT provides support for playing basic tones as well as two different types of files.

Sound files (.rso) are like .wav files. They contain thousands of sound samples that digitally represent an analog waveform. With sounds files the NXT can speak or play music or make just about any sound imaginable.

Melody files are like MIDI files. They contain multiple tones with each tone being defined by a frequency and duration pair. When played on the NXT a melody file sounds like a pure sine-wave tone generator playing back a series of notes. While not as fancy as sound files, melody files are usually much smaller than sound files.

When a sound or a file is played on the NXT, execution of the program does not wait for the previous playback to complete. To play multiple tones or files sequentially it is necessary to wait for the previous tone or file playback to complete first. This can be done via the Wait API function or by using the sound state value within a while loop.

The NXC API defines frequency and duration constants which may be used in calls to [PlayTone](#) or [PlayToneEx](#). Frequency constants start with [TONE\\_A3](#) (the 'A' pitch in octave 3) and go to [TONE\\_B7](#) (the 'B' pitch in octave 7). Duration constants start with [MS\\_1](#) (1 millisecond) and go up to [MIN\\_1](#) (60000 milliseconds) with several constants in between. See [NBCCCommon.h](#) for the complete list.

## 6.14 Ui module

Constants and functions related to the Ui module.

### Modules

- [Ui module types](#)  
*Types used by various Ui module functions.*
- [Ui module functions](#)  
*Functions for accessing and modifying Ui module features.*
- [Ui module constants](#)  
*Constants that are part of the NXT firmware's Ui module.*

### 6.14.1 Detailed Description

Constants and functions related to the Ui module. The NXT UI module encompasses support for various aspects of the user interface for the NXT brick.

## 6.15 Low Speed module

Constants and functions related to the Low Speed module.

### Modules

- [LowSpeed module types](#)  
*Types used by various low speed module functions.*
- [LowSpeed module functions](#)

*Functions for accessing and modifying low speed module features.*

- [LowSpeed module constants](#)

*Constants that are part of the NXT firmware's LowSpeed module.*

### 6.15.1 Detailed Description

Constants and functions related to the Low Speed module. The NXT low speed module encompasses support for digital I2C sensor communication.

Use the `lowspeed` (aka I2C) communication methods to access devices that use the I2C protocol on the NXT brick's four input ports.

You must set the input port's `Type` property to `SENSOR_TYPE_LOWSPEED` or `SENSOR_TYPE_LOWSPEED_9V` on a given port before using an I2C device on that port. Use `SENSOR_TYPE_LOWSPEED_9V` if your device requires 9V power from the NXT brick. Remember that you also need to set the input port's `InvalidDataField` property to true after setting `TypeField` to a new value, and then wait in a loop for the NXT firmware to set `InvalidDataField` back to false. This process ensures that the firmware has time to properly initialize the port, including the 9V power lines, if applicable. Some digital devices might need additional time to initialize after power up.

The `SetSensorLowspeed` API function sets the specified port to `SENSOR_TYPE_LOWSPEED_9V` and calls `ResetSensor` to perform the `InvalidDataField` reset loop described above.

When communicating with I2C devices, the NXT firmware uses a master/slave setup in which the NXT brick is always the master device. This means that the firmware is responsible for controlling the write and read operations. The NXT firmware maintains write and read buffers for each port, and the three main `Lowspeed` (I2C) methods described below enable you to access these buffers.

A call to `LowspeedWrite` starts an asynchronous transaction between the NXT brick and a digital I2C device. The program continues to run while the firmware manages sending bytes from the write buffer and reading the response bytes from the device. Because the NXT is the master device, you must also specify the number of bytes to expect from the device in response to each write operation. You can exchange up to 16 bytes in each direction per transaction.

After you start a write transaction with `LowspeedWrite`, use `LowspeedStatus` in a loop to check the status of the port. If `LowspeedStatus` returns a status code of 0 and a count of bytes available in the read buffer, the system is ready for you to use `LowspeedRead` to copy the data from the read buffer into the buffer you provide.

Note that any of these calls might return various status codes at any time. A status code of 0 means the port is idle and the last transaction (if any) did not result in any errors.

Negative status codes and the positive status code 32 indicate errors. There are a few possible errors per call.

Valid low speed return values include [NO\\_ERR](#) as well as the error codes listed in the [Communications specific errors](#) section.

## 6.16 Display module

Constants and functions related to the Display module.

### Modules

- [Display module types](#)  
*Types used by various display module functions.*
- [Display module functions](#)  
*Functions for accessing and modifying display module features.*
- [Display module constants](#)  
*Constants that are part of the NXT firmware's Display module.*

### 6.16.1 Detailed Description

Constants and functions related to the Display module. The NXT display module encompasses support for drawing to the NXT LCD. The NXT supports drawing points, lines, rectangles, and circles on the LCD. It supports drawing graphic icon files on the screen as well as text and numbers. With the enhanced NBC/NXC firmware you can also draw ellipses and polygons as well as text and numbers using custom RIC-based font files. Also, all of the drawing operations have several drawing options for how the shapes are drawn to the LCD.

The LCD screen has its origin (0, 0) at the bottom left-hand corner of the screen with the positive Y-axis extending upward and the positive X-axis extending toward the right. The NXC API provides constants for use in the [NumOut](#) and [TextOut](#) functions which make it possible to specify LCD line numbers between 1 and 8 with line 1 being at the top of the screen and line 8 being at the bottom of the screen. These constants ([LCD\\_LINE1](#), [LCD\\_LINE2](#), [LCD\\_LINE3](#), [LCD\\_LINE4](#), [LCD\\_LINE5](#), [LCD\\_LINE6](#), [LCD\\_LINE7](#), [LCD\\_LINE8](#)) should be used as the Y coordinate in NumOut and TextOut calls. Values of Y other than these constants will be adjusted so that text and numbers are on one of 8 fixed line positions.

## 6.17 HiTechnic API Functions

Functions for accessing and modifying HiTechnic devices.

### Modules

- [HiTechnic device constants](#)  
*Constants that are for use with HiTechnic devices.*

### Functions

- int [SensorHTGyro](#) (const byte &port, int offset=0)  
*Read HiTechnic Gyro sensor.*
- int [SensorHTMagnet](#) (const byte &port, int offset=0)  
*Read HiTechnic Magnet sensor.*
- int [SensorHTEOPD](#) (const byte &port)  
*Read HiTechnic EOPD sensor.*
- void [SetSensorHTEOPD](#) (const byte &port, bool bStandard)  
*Set sensor as HiTechnic EOPD.*
- void [SetSensorHTGyro](#) (const byte &port)  
*Set sensor as HiTechnic Gyro.*
- void [SetSensorHTMagnet](#) (const byte &port)  
*Set sensor as HiTechnic Magnet.*
- int [SensorHTColorNum](#) (const byte &port)  
*Read HiTechnic color sensor color number.*
- int [SensorHTCompass](#) (const byte &port)  
*Read HiTechnic compass.*
- int [SensorHTIRSeekerDir](#) (const byte &port)  
*Read HiTechnic IRSeeker direction.*
- int [SensorHTIRSeeker2Addr](#) (const byte &port, const byte reg)  
*Read HiTechnic IRSeeker2 register.*

- int [SensorHTIRSeeker2DCDir](#) (const byte &port)  
*Read HiTechnic IRSeeker2 DC direction.*
- int [SensorHTIRSeeker2ACDir](#) (const byte &port)  
*Read HiTechnic IRSeeker2 AC direction.*
- char [SetHTColor2Mode](#) (const byte &port, byte mode)  
*Set HiTechnic Color2 mode.*
- char [SetHTIRSeeker2Mode](#) (const byte &port, const byte mode)  
*Set HiTechnic IRSeeker2 mode.*
- bool [ReadSensorHTAccel](#) (const byte port, int &x, int &y, int &z)  
*Read HiTechnic acceleration values.*
- bool [ReadSensorHTColor](#) (const byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color values.*
- bool [ReadSensorHTIRSeeker](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)  
*Read HiTechnic IRSeeker values.*
- bool [ReadSensorHTNormalizedColor](#) (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color normalized values.*
- bool [ReadSensorHTRawColor](#) (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue)  
*Read HiTechnic Color raw values.*
- bool [ReadSensorHTColor2Active](#) (byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue, byte &White)  
*Read HiTechnic Color2 active values.*
- bool [ReadSensorHTNormalizedColor2Active](#) (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color2 normalized active values.*
- bool [ReadSensorHTRawColor2](#) (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue, unsigned int &White)  
*Read HiTechnic Color2 raw values.*

- bool [ReadSensorHTIRReceiver](#) (const byte port, char &pfdata[ ])
  - Read HiTechnic IRReceiver Power Function bytes.*
- bool [ReadSensorHTIRReceiverEx](#) (const byte port, const byte offset, char &pfchar)
  - Read HiTechnic IRReceiver Power Function value.*
- bool [ReadSensorHTIRSeeker2AC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)
  - Read HiTechnic IRSeeker2 AC values.*
- bool [ReadSensorHTIRSeeker2DC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9, byte &avg)
  - Read HiTechnic IRSeeker2 DC values.*
- char [ResetSensorHTAngle](#) (const byte port, const byte mode)
  - Reset HiTechnic Angle sensor.*
- bool [ReadSensorHTAngle](#) (const byte port, int &Angle, long &AccAngle, int &RPM)
  - Read HiTechnic Angle sensor values.*
- bool [ResetHTBarometricCalibration](#) (byte port)
  - Reset HiTechnic Barometric sensor calibration.*
- bool [SetHTBarometricCalibration](#) (byte port, unsigned int cal)
  - Set HiTechnic Barometric sensor calibration.*
- bool [ReadSensorHTBarometric](#) (const byte port, int &temp, unsigned int &press)
  - Read HiTechnic Barometric sensor values.*
- int [SensorHTProtoAnalog](#) (const byte port, const byte input)
  - Read HiTechnic Prototype board analog input value.*
- bool [ReadSensorHTProtoAllAnalog](#) (const byte port, int &a0, int &a1, int &a2, int &a3, int &a4)
  - Read all HiTechnic Prototype board analog input values.*
- bool [SetSensorHTProtoDigitalControl](#) (const byte port, byte value)
  - Control HiTechnic Prototype board digital pin direction.*
- byte [SensorHTProtoDigitalControl](#) (const byte port)

*Read HiTechnic Prototype board digital control values.*

- bool [SetSensorHTProtoDigital](#) (const byte port, byte value)  
*Set HiTechnic Prototype board digital output values.*
- byte [SensorHTProtoDigital](#) (const byte port)  
*Read HiTechnic Prototype board digital input values.*
- int [SensorHTSuperProAnalog](#) (const byte port, const byte input)  
*Read HiTechnic SuperPro board analog input value.*
- bool [ReadSensorHTSuperProAllAnalog](#) (const byte port, int &a0, int &a1, int &a2, int &a3)  
*Read all HiTechnic SuperPro board analog input values.*
- bool [SetSensorHTSuperProDigitalControl](#) (const byte port, byte value)  
*Control HiTechnic SuperPro board digital pin direction.*
- byte [SensorHTSuperProDigitalControl](#) (const byte port)  
*Read HiTechnic SuperPro board digital control values.*
- bool [SetSensorHTSuperProDigital](#) (const byte port, byte value)  
*Set HiTechnic SuperPro board digital output values.*
- byte [SensorHTSuperProDigital](#) (const byte port)  
*Read HiTechnic SuperPro board digital input values.*
- bool [SetSensorHTSuperProLED](#) (const byte port, byte value)  
*Set HiTechnic SuperPro LED value.*
- byte [SensorHTSuperProLED](#) (const byte port)  
*Read HiTechnic SuperPro LED value.*
- bool [SetSensorHTSuperProStrobe](#) (const byte port, byte value)  
*Set HiTechnic SuperPro strobe value.*
- byte [SensorHTSuperProStrobe](#) (const byte port)  
*Read HiTechnic SuperPro strobe value.*
- bool [SetSensorHTSuperProProgramControl](#) (const byte port, byte value)  
*Set HiTechnic SuperPro program control value.*
- byte [SensorHTSuperProProgramControl](#) (const byte port)

*Read HiTechnic SuperPro program control value.*

- bool [SetSensorHTSuperProAnalogOut](#) (const byte port, const byte dac, byte mode, int freq, int volt)

*Set HiTechnic SuperPro board analog output parameters.*

- bool [ReadSensorHTSuperProAnalogOut](#) (const byte port, const byte dac, byte &mode, int &freq, int &volt)

*Read HiTechnic SuperPro board analog output parameters.*

- void [ReadSensorHTTouchMultiplexer](#) (const byte port, byte &t1, byte &t2, byte &t3, byte &t4)

*Read HiTechnic touch multiplexer.*

- char [HTIRTrain](#) (const byte port, const byte channel, const byte func)

*HTIRTrain function.*

- char [HTPFComboDirect](#) (const byte port, const byte channel, const byte outa, const byte outb)

*HTPFComboDirect function.*

- char [HTPFComboPWM](#) (const byte port, const byte channel, const byte outa, const byte outb)

*HTPFComboPWM function.*

- char [HTPFRawOutput](#) (const byte port, const byte nibble0, const byte nibble1, const byte nibble2)

*HTPFRawOutput function.*

- char [HTPFRepeat](#) (const byte port, const byte count, const unsigned int delay)

*HTPFRepeat function.*

- char [HTPFSingleOutputCST](#) (const byte port, const byte channel, const byte out, const byte func)

*HTPFSingleOutputCST function.*

- char [HTPFSingleOutputPWM](#) (const byte port, const byte channel, const byte out, const byte func)

*HTPFSingleOutputPWM function.*

- char [HTPFSinglePin](#) (const byte port, const byte channel, const byte out, const byte pin, const byte func, bool cont)

*HTPFSinglePin function.*

- char [HTPFTrain](#) (const byte port, const byte channel, const byte func)  
*HTPFTrain function.*
- void [HTRCXSetIRLinkPort](#) (const byte port)  
*HTRCXSetIRLinkPort function.*
- int [HTRCXBatteryLevel](#) (void)  
*HTRCXBatteryLevel function.*
- int [HTRCXPoll](#) (const byte src, const byte value)  
*HTRCXPoll function Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.*
- int [HTRCXPollMemory](#) (const unsigned int address)  
*HTRCXPollMemory function.*
- void [HTRCXAddToDatalog](#) (const byte src, const unsigned int value)  
*HTRCXAddToDatalog function.*
- void [HTRCXCLEARAllEvents](#) (void)  
*HTRCXCLEARAllEvents function.*
- void [HTRCXCLEARCounter](#) (const byte counter)  
*HTRCXCLEARCounter function.*
- void [HTRCXCLEARMsg](#) (void)  
*HTRCXCLEARMsg function.*
- void [HTRCXCLEARSensor](#) (const byte port)  
*HTRCXCLEARSensor function.*
- void [HTRCXCLEARSound](#) (void)  
*HTRCXCLEARSound function.*
- void [HTRCXCLEARTimer](#) (const byte timer)  
*HTRCXCLEARTimer function.*
- void [HTRCXCreateDatalog](#) (const unsigned int size)  
*HTRCXCreateDatalog function.*
- void [HTRCXDecCounter](#) (const byte counter)

*HTRCXDecCounter function.*

- void [HTRCXDeleteSub](#) (const byte s)  
*HTRCXDeleteSub function.*
- void [HTRCXDeleteSubs](#) (void)  
*HTRCXDeleteSubs function.*
- void [HTRCXDeleteTask](#) (const byte t)  
*HTRCXDeleteTask function.*
- void [HTRCXDeleteTasks](#) (void)  
*HTRCXDeleteTasks function.*
- void [HTRCXDisableOutput](#) (const byte outputs)  
*HTRCXDisableOutput function.*
- void [HTRCXEnableOutput](#) (const byte outputs)  
*HTRCXEnableOutput function.*
- void [HTRCXEvent](#) (const byte src, const unsigned int value)  
*HTRCXEvent function.*
- void [HTRCXFloat](#) (const byte outputs)  
*HTRCXFloat function.*
- void [HTRCXFwd](#) (const byte outputs)  
*HTRCXFwd function.*
- void [HTRCXIncCounter](#) (const byte counter)  
*HTRCXIncCounter function.*
- void [HTRCXInvertOutput](#) (const byte outputs)  
*HTRCXInvertOutput function.*
- void [HTRCXMuteSound](#) (void)  
*HTRCXMuteSound function.*
- void [HTRCXObvertOutput](#) (const byte outputs)  
*HTRCXObvertOutput function.*
- void [HTRCXOff](#) (const byte outputs)

*HTRCXOff function.*

- void **HTRCXOn** (const byte outputs)  
*HTRCXOn function.*
- void **HTRCXOnFor** (const byte outputs, const unsigned int ms)  
*HTRCXOnFor function.*
- void **HTRCXOnFwd** (const byte outputs)  
*HTRCXOnFwd function.*
- void **HTRCXOnRev** (const byte outputs)  
*HTRCXOnRev function.*
- void **HTRCXPBTurnOff** (void)  
*HTRCXPBTurnOff function.*
- void **HTRCXPing** (void)  
*HTRCXPing function.*
- void **HTRCXPlaySound** (const byte snd)  
*HTRCXPlaySound function.*
- void **HTRCXPlayTone** (const unsigned int freq, const byte duration)  
*HTRCXPlayTone function.*
- void **HTRCXPlayToneVar** (const byte varnum, const byte duration)  
*HTRCXPlayToneVar function.*
- void **HTRCXRemote** (unsigned int cmd)  
*HTRCXRemote function.*
- void **HTRCXRev** (const byte outputs)  
*HTRCXRev function.*
- void **HTRCXSelectDisplay** (const byte src, const unsigned int value)  
*HTRCXSelectDisplay function.*
- void **HTRCXSelectProgram** (const byte prog)  
*HTRCXSelectProgram function.*
- void **HTRCXSendSerial** (const byte first, const byte count)

*HTRCXSendSerial function.*

- void [HTRCXSetDirection](#) (const byte outputs, const byte dir)  
*HTRCXSetDirection function.*
- void [HTRCXSetEvent](#) (const byte evt, const byte src, const byte type)  
*HTRCXSetEvent function.*
- void [HTRCXSetGlobalDirection](#) (const byte outputs, const byte dir)  
*HTRCXSetGlobalDirection function.*
- void [HTRCXSetGlobalOutput](#) (const byte outputs, const byte mode)  
*HTRCXSetGlobalOutput function.*
- void [HTRCXSetMaxPower](#) (const byte outputs, const byte pwrsrc, const byte pwrval)  
*HTRCXSetMaxPower function.*
- void [HTRCXSetMessage](#) (const byte msg)  
*HTRCXSetMessage function.*
- void [HTRCXSetOutput](#) (const byte outputs, const byte mode)  
*HTRCXSetOutput function.*
- void [HTRCXSetPower](#) (const byte outputs, const byte pwrsrc, const byte pwrval)  
*HTRCXSetPower function.*
- void [HTRCXSetPriority](#) (const byte p)  
*HTRCXSetPriority function.*
- void [HTRCXSetSensorMode](#) (const byte port, const byte mode)  
*HTRCXSetSensorMode function.*
- void [HTRCXSetSensorType](#) (const byte port, const byte type)  
*HTRCXSetSensorType function.*
- void [HTRCXSetSleepTime](#) (const byte t)  
*HTRCXSetSleepTime function.*
- void [HTRCXSetTxPower](#) (const byte pwr)  
*HTRCXSetTxPower function.*

- void [HTRCXSetWatch](#) (const byte hours, const byte minutes)  
*HTRCXSetWatch function.*
- void [HTRCXStartTask](#) (const byte t)  
*HTRCXStartTask function.*
- void [HTRCXStopAllTasks](#) (void)  
*HTRCXStopAllTasks function.*
- void [HTRCXStopTask](#) (const byte t)  
*HTRCXStopTask function.*
- void [HTRCXToggle](#) (const byte outputs)  
*HTRCXToggle function.*
- void [HTRCXUnmuteSound](#) (void)  
*HTRCXUnmuteSound function.*
- void [HTScoutCalibrateSensor](#) (void)  
*HTScoutCalibrateSensor function.*
- void [HTScoutMuteSound](#) (void)  
*HTScoutMuteSound function.*
- void [HTScoutSelectSounds](#) (const byte grp)  
*HTScoutSelectSounds function.*
- void [HTScoutSendVLL](#) (const byte src, const unsigned int value)  
*HTScoutSendVLL function.*
- void [HTScoutSetEventFeedback](#) (const byte src, const unsigned int value)  
*HTScoutSetEventFeedback function.*
- void [HTScoutSetLight](#) (const byte x)  
*HTScoutSetLight function.*
- void [HTScoutSetScoutMode](#) (const byte mode)  
*HTScoutSetScoutMode function.*
- void [HTScoutSetSensorClickTime](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorClickTime function.*

- void [HTScoutSetSensorHysteresis](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorHysteresis function.*
- void [HTScoutSetSensorLowerLimit](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorLowerLimit function.*
- void [HTScoutSetSensorUpperLimit](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorUpperLimit function.*
- void [HTScoutUnmuteSound](#) (void)  
*HTScoutUnmuteSound function.*

### 6.17.1 Detailed Description

Functions for accessing and modifying HiTechnic devices.

### 6.17.2 Function Documentation

#### 6.17.2.1 char HTIRTrain (const byte *port*, const byte *channel*, const byte *func*) [inline]

HTIRTrain function. Control an IR Train receiver set to the specified channel using the HiTechnic iRLink device. Valid func values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channel values are [TRAIN\\_CHANNEL\\_1](#) through [TRAIN\\_CHANNEL\\_3](#) and [TRAIN\\_CHANNEL\\_ALL](#). The port must be configured as a Low-speed port before using this function.

#### Parameters:

- port* The sensor port. See [Input port constants](#).
- channel* The IR Train channel. See [IR Train channel constants](#).
- func* The IR Train function. See [PF/IR Train function constants](#)

#### Returns:

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

#### Examples:

[ex\\_HTIRTrain.nxc](#).

### 6.17.2.2 char HTPFComboDirect (const byte *port*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

HTPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the HiTechnic iRLink device. Commands for *outa* and *outb* are [PF\\_CMD\\_STOP](#), [PF\\_CMD\\_REV](#), [PF\\_CMD\\_FWD](#), and [PF\\_CMD\\_BRAKE](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*outa* The Power Function command for output A. See [Power Function command constants](#).

*outb* The Power Function command for output B. See [Power Function command constants](#).

#### Returns:

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

#### Examples:

[ex\\_HTPFComboDirect.nxc](#).

### 6.17.2.3 char HTPFComboPWM (const byte *port*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

HTPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Valid output values are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#), [PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*outa* The Power Function PWM command for output A. See [Power Function PWM option constants](#).

*outb* The Power Function PWM command for output B. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFComboPWM.nxc](#).

#### 6.17.2.4 char HTPFRawOutput (const byte *port*, const byte *nibble0*, const byte *nibble1*, const byte *nibble2*) [[inline](#)]

HTPFRawOutput function. Control a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Build the raw data stream using the 3 nibbles (4 bit values). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*nibble0* The first raw data nibble.

*nibble1* The second raw data nibble.

*nibble2* The third raw data nibble.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFRawOutput.nxc](#).

#### 6.17.2.5 char HTPFRepeat (const byte *port*, const byte *count*, const unsigned int *delay*) [[inline](#)]

HTPFRepeat function. Repeat sending the last Power Function command using the HiTechnic iRLink device. Specify the number of times to repeat the command and the number of milliseconds of delay between each repetition. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- count* The number of times to repeat the command.
- delay* The number of milliseconds to delay between each repetition.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFRepeat.nxc](#).

**6.17.2.6 char HTPFSingleOutputCST (const byte *port*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

HTPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_CST\\_CLEAR1\\_CLEAR2](#), [PF\\_CST\\_SET1\\_CLEAR2](#), [PF\\_CST\\_CLEAR1\\_SET2](#), [PF\\_CST\\_SET1\\_SET2](#), [PF\\_CST\\_INCREMENT\\_PWM](#), [PF\\_CST\\_DECREMENT\\_PWM](#), [PF\\_CST\\_FULL\\_FWD](#), [PF\\_CST\\_FULL\\_REV](#), and [PF\\_CST\\_TOGGLE\\_DIR](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- func* The Power Function CST function. See [Power Function CST options constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#).

### 6.17.2.7 `char HTPFSingleOutputPWM (const byte port, const byte channel, const byte out, const byte func) [inline]`

HTPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using `PF_OUT_A` or `PF_OUT_B`. Valid functions are `PF_PWM_FLOAT`, `PF_PWM_FWD1`, `PF_PWM_FWD2`, `PF_PWM_FWD3`, `PF_PWM_FWD4`, `PF_PWM_FWD5`, `PF_PWM_FWD6`, `PF_PWM_FWD7`, `PF_PWM_BRAKE`, `PF_PWM_REV7`, `PF_PWM_REV6`, `PF_PWM_REV5`, `PF_PWM_REV4`, `PF_PWM_REV3`, `PF_PWM_REV2`, and `PF_PWM_REV1`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Low-speed port before using this function.

#### Parameters:

- port* The sensor port. See [Input port constants](#).
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- func* The Power Function PWM function. See [Power Function PWM option constants](#).

#### Returns:

The function call result. `NO_ERR` or [Communications specific errors](#).

#### Examples:

[ex\\_HTPFSingleOutputPWM.nxc](#).

### 6.17.2.8 `char HTPFSinglePin (const byte port, const byte channel, const byte out, const byte pin, const byte func, bool cont) [inline]`

HTPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using `PF_OUT_A` or `PF_OUT_B`. Select the desired pin using `PF_PIN_C1` or `PF_PIN_C2`. Valid functions are `PF_FUNC_NOCHANGE`, `PF_FUNC_CLEAR`, `PF_FUNC_SET`, and `PF_FUNC_TOGGLE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. Specify whether the mode by passing true (continuous) or false (time-out) as the final parameter. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*out* The Power Function output. See [Power Function output constants](#).

*pin* The Power Function pin. See [Power Function pin constants](#).

*func* The Power Function single pin function. See [Power Function single pin function constants](#).

*cont* Control whether the mode is continuous or timeout.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFSinglePin.nxc](#).

**6.17.2.9 char HTPFTrain (const byte *port*, const byte *channel*, const byte *func*)  
[inline]**

HTPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device as if it were an IR Train receiver. Valid function values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*func* The Power Function train function. See [PF/IR Train function constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFTrain.nxc](#).

**6.17.2.10 void HTRCXAddToDatalog (const byte *src*, const unsigned int *value*) [inline]**

HTRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#).

**6.17.2.11 int HTRCXBatteryLevel (void) [inline]**

HTRCXBatteryLevel function. Send the BatteryLevel command to an RCX to read the current battery level.

**Returns:**

The RCX battery level.

**Examples:**

[ex\\_HTRCXBatteryLevel.nxc](#).

**6.17.2.12 void HTRCXCLEARAllEvents (void) [inline]**

HTRCXCLEARAllEvents function. Send the CLEARAllEvents command to an RCX.

**Examples:**

[ex\\_HTRCXCLEARAllEvents.nxc](#).

**6.17.2.13 void HTRCXCLEARCounter (const byte *counter*) [inline]**

HTRCXCLEARCounter function. Send the CLEARCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

[ex\\_HTRCXCclearCounter.nxc](#).

**6.17.2.14 void HTRCXCclearMsg (void) [inline]**

HTRCXCclearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

[ex\\_HTRCXCclearMsg.nxc](#).

**6.17.2.15 void HTRCXCclearSensor (const byte port) [inline]**

HTRCXCclearSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

[ex\\_HTRCXCclearSensor.nxc](#).

**6.17.2.16 void HTRCXCclearSound (void) [inline]**

HTRCXCclearSound function. Send the ClearSound command to an RCX.

**Examples:**

[ex\\_HTRCXCclearSound.nxc](#).

**6.17.2.17 void HTRCXCclearTimer (const byte *timer*) [inline]**

HTRCXCclearTimer function. Send the ClearTimer command to an RCX.

**Parameters:**

*timer* The timer to clear.

**Examples:**

[ex\\_HTRCXCclearTimer.nxc](#).

**6.17.2.18 void HTRCXCreateDatalog (const unsigned int *size*) [inline]**

HTRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

**Parameters:**

*size* The new datalog size.

**Examples:**

[ex\\_HTRCXCreateDatalog.nxc](#).

**6.17.2.19 void HTRCXDecCounter (const byte *counter*) [inline]**

HTRCXDecCounter function. Send the DecCounter command to an RCX.

**Parameters:**

*counter* The counter to decrement.

**Examples:**

[ex\\_HTRCXDecCounter.nxc](#).

**6.17.2.20 void HTRCXDeleteSub (const byte *s*) [inline]**

HTRCXDeleteSub function. Send the DeleteSub command to an RCX.

**Parameters:**

*s* The subroutine number to delete.

**Examples:**

[ex\\_HTRCXDeleteSub.nxc](#).

**6.17.2.21 void HTRCXDeleteSubs (void) [inline]**

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

[ex\\_HTRCXDeleteSubs.nxc](#).

**6.17.2.22 void HTRCXDeleteTask (const byte *t*) [inline]**

HTRCXDeleteTask function. Send the DeleteTask command to an RCX.

**Parameters:**

*t* The task number to delete.

**Examples:**

[ex\\_HTRCXDeleteTask.nxc](#).

**6.17.2.23 void HTRCXDeleteTasks (void) [inline]**

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

[ex\\_HTRCXDeleteTasks.nxc](#).

**6.17.2.24 void HTRCXDisableOutput (const byte *outputs*) [inline]**

HTRCXDisableOutput function. Send the DisableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to disable. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXDisableOutput.nxc](#).

**6.17.2.25 void HTRCXEnableOutput (const byte *outputs*) [inline]**

HTRCXEnableOutput function. Send the EnableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to enable. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXEnableOutput.nxc](#).

**6.17.2.26 void HTRCXEvent (const byte *src*, const unsigned int *value*) [inline]**

HTRCXEvent function. Send the Event command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXEvent.nxc](#).

**6.17.2.27 void HTRCXFloat (const byte *outputs*) [inline]**

HTRCXFloat function. Send commands to an RCX to float the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to float. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXFloat.nxc](#).

**6.17.2.28 void HTRCXFwd (const byte *outputs*) [inline]**

HTRCXFwd function. Send commands to an RCX to set the specified outputs to the forward direction.

**Parameters:**

*outputs* The RCX output(s) to set forward. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXFwd.nxc](#).

**6.17.2.29 void HTRCXIncCounter (const byte *counter*) [inline]**

HTRCXIncCounter function. Send the IncCounter command to an RCX.

**Parameters:**

*counter* The counter to increment.

**Examples:**

[ex\\_HTRCXIncCounter.nxc](#).

**6.17.2.30 void HTRCXInvertOutput (const byte *outputs*) [inline]**

HTRCXInvertOutput function. Send the InvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to invert. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXInvertOutput.nxc](#).

**6.17.2.31 void HTRCXMuteSound (void) [inline]**

HTRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

[ex\\_HTRCXMuteSound.nxc](#).

**6.17.2.32 void HTRCXObvertOutput (const byte *outputs*) [inline]**

HTRCXObvertOutput function. Send the ObvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to obvert. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXObvertOutput.nxc](#).

**6.17.2.33 void HTRCXOff (const byte *outputs*) [inline]**

HTRCXOff function. Send commands to an RCX to turn off the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn off. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOff.nxc](#).

**6.17.2.34 void HTRCXOn (const byte *outputs*) [inline]**

HTRCXOn function. Send commands to an RCX to turn on the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOn.nxc](#).

**6.17.2.35 void HTRCXOnFor (const byte *outputs*, const unsigned int *ms*) [inline]**

HTRCXOnFor function. Send commands to an RCX to turn on the specified outputs in the forward direction for the specified duration.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

*ms* The number of milliseconds to leave the outputs on

**Examples:**

[ex\\_HTRCXOnFor.nxc](#).

**6.17.2.36 void HTRCXOnFwd (const byte *outputs*) [inline]**

HTRCXOnFwd function. Send commands to an RCX to turn on the specified outputs in the forward direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the forward direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOnFwd.nxc](#).

**6.17.2.37 void HTRCXOnRev (const byte *outputs*) [inline]**

HTRCXOnRev function. Send commands to an RCX to turn on the specified outputs in the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOnRev.nxc](#).

**6.17.2.38 void HTRCXPBTurnOff (void) [inline]**

HTRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

[ex\\_HTRCXPBTurnOff.nxc](#).

**6.17.2.39 void HTRCXPing (void) [inline]**

HTRCXPing function. Send the Ping command to an RCX.

**Examples:**

[ex\\_HTRCXPing.nxc](#).

**6.17.2.40 void HTRCXPlaySound (const byte *snd*) [inline]**

HTRCXPlaySound function. Send the PlaySound command to an RCX.

**Parameters:**

*snd* The sound number to play.

**Examples:**

[ex\\_HTRCXPlaySound.nxc](#).

**6.17.2.41 void HTRCXPlayTone (const unsigned int *freq*, const byte *duration*)  
[inline]**

HTRCXPlayTone function. Send the PlayTone command to an RCX.

**Parameters:**

*freq* The frequency of the tone to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_HTRCXPlayTone.nxc](#).

**6.17.2.42 void HTRCXPlayToneVar (const byte *varnum*, const byte *duration*)  
[inline]**

HTRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

**Parameters:**

*varnum* The variable containing the tone frequency to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_HTRCXPlayToneVar.nxc](#).

**6.17.2.43 int HTRCXPoll (const byte *src*, const byte *value*) [inline]**

HTRCXPoll function Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Returns:**

The value read from the specified port and value.

**Examples:**

[ex\\_HTRCXPoll.nxc](#).

**6.17.2.44 int HTRCXPollMemory (const unsigned int *address*) [inline]**

HTRCXPollMemory function. Send the PollMemory command to an RCX.

**Parameters:**

*address* The RCX memory address.

**Returns:**

The value read from the specified address.

**Examples:**

[ex\\_HTRCXPollMemory.nxc](#).

**6.17.2.45 void HTRCXRemote (unsigned int *cmd*) [inline]**

HTRCXRemote function. Send the Remote command to an RCX.

**Parameters:**

*cmd* The RCX IR remote command to send. See [RCX IR remote constants](#).

**Examples:**

[ex\\_HTRCXRemote.nxc](#).

**6.17.2.46 void HTRCXRev (const byte *outputs*) [inline]**

HTRCXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXRev.nxc](#).

**6.17.2.47 void HTRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]**

HTRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXSelectDisplay.nxc](#).

**6.17.2.48 void HTRCXSelectProgram (const byte *prog*) [inline]**

HTRCXSelectProgram function. Send the SelectProgram command to an RCX.

**Parameters:**

*prog* The program number to select.

**Examples:**

[ex\\_HTRCXSelectProgram.nxc](#).

**6.17.2.49 void HTRCXSendSerial (const byte *first*, const byte *count*)  
[inline]**

HTRCXSendSerial function. Send the SendSerial command to an RCX.

**Parameters:**

- first* The first byte address.
- count* The number of bytes to send.

**Examples:**

[ex\\_HTRCXSendSerial.nxc](#).

**6.17.2.50 void HTRCXSetDirection (const byte *outputs*, const byte *dir*)  
[inline]**

HTRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

**Parameters:**

- outputs* The RCX output(s) to set direction. See [RCX output constants](#).
- dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_HTRCXSetDirection.nxc](#).

**6.17.2.51 void HTRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)  
[inline]**

HTRCXSetEvent function. Send the SetEvent command to an RCX.

**Parameters:**

- evt* The event number to set.
- src* The RCX source. See [RCX and Scout source constants](#).
- type* The event type.

**Examples:**

[ex\\_HTRCXSetEvent.nxc](#).

**6.17.2.52 void HTRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)  
[inline]**

HTRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_HTRCXSetGlobalDirection.nxc](#).

**6.17.2.53 void HTRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)  
[inline]**

HTRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_HTRCXSetGlobalOutput.nxc](#).

**6.17.2.54 void HTRCXSetIRLinkPort (const byte *port*) [inline]**

HTRCXSetIRLinkPort function. Set the global port in advance of using the HTRCX\* and HTScout\* API functions for sending RCX and Scout messages over the HiTechnic iRLink device. The port must be configured as a Low-speed port before using any of the HiTechnic RCX and Scout iRLink functions.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**6.17.2.55 void HTRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]**

HTRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set max power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_HTRCXSetMaxPower.nxc](#).

**6.17.2.56 void HTRCXSetMessage (const byte *msg*) [inline]**

HTRCXSetMessage function. Send the SetMessage command to an RCX.

**Parameters:**

*msg* The numeric message to send.

**Examples:**

[ex\\_HTRCXSetMessage.nxc](#).

**6.17.2.57 void HTRCXSetOutput (const byte *outputs*, const byte *mode*) [inline]**

HTRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

**Parameters:**

*outputs* The RCX output(s) to set mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_HTRCXSetOutput.nxc](#).

**6.17.2.58 void HTRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]**

HTRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_HTRCXSetPower.nxc](#).

**6.17.2.59 void HTRCXSetPriority (const byte *p*) [inline]**

HTRCXSetPriority function. Send the SetPriority command to an RCX.

**Parameters:**

*p* The new task priority.

**Examples:**

[ex\\_HTRCXSetPriority.nxc](#).

**6.17.2.60 void HTRCXSetSensorMode (const byte *port*, const byte *mode*) [inline]**

HTRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

**Parameters:**

- port* The RCX sensor port.
- mode* The RCX sensor mode.

**Examples:**

[ex\\_HTRCXSetSensorMode.nxc](#).

**6.17.2.61 void HTRCXSetSensorType (const byte *port*, const byte *type*)  
[inline]**

HTRCXSetSensorType function. Send the SetSensorType command to an RCX.

**Parameters:**

- port* The RCX sensor port.
- type* The RCX sensor type.

**Examples:**

[ex\\_HTRCXSetSensorType.nxc](#).

**6.17.2.62 void HTRCXSetSleepTime (const byte *t*) [inline]**

HTRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

**Parameters:**

- t* The new sleep time value.

**Examples:**

[ex\\_HTRCXSetSleepTime.nxc](#).

**6.17.2.63 void HTRCXSetTxPower (const byte *pwr*) [inline]**

HTRCXSetTxPower function. Send the SetTxPower command to an RCX.

**Parameters:**

*pwr* The IR transmit power level.

**Examples:**

[ex\\_HTRCXSetTxPower.nxc](#).

**6.17.2.64 void HTRCXSetWatch (const byte hours, const byte minutes)  
[inline]**

HTRCXSetWatch function. Send the SetWatch command to an RCX.

**Parameters:**

*hours* The new watch time hours value.

*minutes* The new watch time minutes value.

**Examples:**

[ex\\_HTRCXSetWatch.nxc](#).

**6.17.2.65 void HTRCXStartTask (const byte t) [inline]**

HTRCXStartTask function. Send the StartTask command to an RCX.

**Parameters:**

*t* The task number to start.

**Examples:**

[ex\\_HTRCXStartTask.nxc](#).

**6.17.2.66 void HTRCXStopAllTasks (void) [inline]**

HTRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

[ex\\_HTRCXStopAllTasks.nxc](#).

**6.17.2.67 void HTRCXStopTask (const byte *t*) [inline]**

HTRCXStopTask function. Send the StopTask command to an RCX.

**Parameters:**

*t* The task number to stop.

**Examples:**

[ex\\_HTRCXStopTask.nxc](#).

**6.17.2.68 void HTRCXToggle (const byte *outputs*) [inline]**

HTRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to toggle. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXToggle.nxc](#).

**6.17.2.69 void HTRCXUnmuteSound (void) [inline]**

HTRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

[ex\\_HTRCXUnmuteSound.nxc](#).

**6.17.2.70 void HTScoutCalibrateSensor (void) [inline]**

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

[ex\\_HTScoutCalibrateSensor.nxc](#).

**6.17.2.71 void HTScoutMuteSound (void) [inline]**

HTScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

[ex\\_HTScoutMuteSound.nxc](#).

**6.17.2.72 void HTScoutSelectSounds (const byte grp) [inline]**

HTScoutSelectSounds function. Send the SelectSounds command to a Scout.

**Parameters:**

*grp* The Scout sound group to select.

**Examples:**

[ex\\_HTScoutSelectSounds.nxc](#).

**6.17.2.73 void HTScoutSendVLL (const byte src, const unsigned int value) [inline]**

HTScoutSendVLL function. Send the SendVLL command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSendVLL.nxc](#).

**6.17.2.74 void HTScoutSetEventFeedback (const byte src, const unsigned int value) [inline]**

HTScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetEventFeedback.nxc](#).

**6.17.2.75 void HTScoutSetLight (const byte *x*) [inline]**

HTScoutSetLight function. Send the SetLight command to a Scout.

**Parameters:**

*x* Set the light on or off using this value. See [Scout light constants](#).

**Examples:**

[ex\\_HTScoutSetLight.nxc](#).

**6.17.2.76 void HTScoutSetScoutMode (const byte *mode*) [inline]**

HTScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

**Parameters:**

*mode* Set the scout mode. See [Scout mode constants](#).

**Examples:**

[ex\\_HTScoutSetScoutMode.nxc](#).

**6.17.2.77 void HTScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorClickTime.nxc](#).

**6.17.2.78 void HTScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorHysteresis.nxc](#).

**6.17.2.79 void HTScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorLowerLimit.nxc](#).

**6.17.2.80 void HTScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorUpperLimit.nxc](#).

**6.17.2.81 void HTScoutUnmuteSound (void) [inline]**

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

[ex\\_HTScoutUnmuteSound.nxc](#).

**6.17.2.82 bool ReadSensorHTAccel (const byte *port*, int & *x*, int & *y*, int & *z*) [inline]**

Read HiTechnic acceleration values. Read X, Y, and Z axis acceleration values from the HiTechnic Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*x* The output x-axis acceleration.

*y* The output y-axis acceleration.

*z* The output z-axis acceleration.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTAccel.nxc](#).

**6.17.2.83 bool ReadSensorHTAngle (const byte *port*, int & *Angle*, long & *AccAngle*, int & *RPM*) [inline]**

Read HiTechnic Angle sensor values. Read values from the HiTechnic Angle sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Angle* Current angle in degrees (0-359).

*AccAngle* Accumulated angle in degrees (-2147483648 to 2147483647).

*RPM* rotations per minute (-1000 to 1000).

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTAngle.nxc](#).

**6.17.2.84 bool ReadSensorHTBarometric (const byte *port*, int & *temp*, unsigned int & *press*) [inline]**

Read HiTechnic Barometric sensor values. Read values from the HiTechnic Barometric sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*temp* Current temperature in 1/10ths of degrees Celcius.

*press* Current barometric pressure in 1/1000 inches of mercury.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTBarometric.nxc](#).

**6.17.2.85 bool ReadSensorHTColor (const byte *port*, byte & *ColorNum*, byte & *Red*, byte & *Green*, byte & *Blue*) [inline]**

Read HiTechnic Color values. Read color number, red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*ColorNum* The output color number.

*Red* The red color value.

*Green* The green color value.

*Blue* The blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTColor.nxc](#).

**6.17.2.86 bool ReadSensorHTColor2Active (byte *port*, byte & *ColorNum*, byte & *Red*, byte & *Green*, byte & *Blue*, byte & *White*) [inline]**

Read HiTechnic Color2 active values. Read color number, red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*ColorNum* The output color number.

*Red* The red color value.

*Green* The green color value.

*Blue* The blue color value.

*White* The white color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTColor2Active.nxc](#).

**6.17.2.87 bool ReadSensorHTIRReceiver (const byte port, char & pfdata[ ])  
[inline]**

Read HiTechnic IRReceiver Power Function bytes. Read Power Function bytes from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*pfdata* Eight bytes of power function remote IR data.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRReceiver.nxc](#).

**6.17.2.88 bool ReadSensorHTIRReceiverEx (const byte port, const byte offset,  
char & pfchar) [inline]**

Read HiTechnic IRReceiver Power Function value. Read a Power Function byte from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The power function data offset. See [HiTechnic IRReceiver constants](#).

*pfchar* A single byte of power function remote IR data.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRReceiverEx.nxc](#).

**6.17.2.89 bool ReadSensorHTIRSeeker (const byte *port*, byte & *dir*, byte & *s1*, byte & *s3*, byte & *s5*, byte & *s7*, byte & *s9*) [inline]**

Read HiTechnic IRSeeker values. Read direction, and five signal strength values from the HiTechnic IRSeeker sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

*s3* The signal strength from sensor 3.

*s5* The signal strength from sensor 5.

*s7* The signal strength from sensor 7.

*s9* The signal strength from sensor 9.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker.nxc](#).

**6.17.2.90** `bool ReadSensorHTIRSeeker2AC (const byte port, byte & dir, byte & s1, byte & s3, byte & s5, byte & s7, byte & s9) [inline]`

Read HiTechnic IRSeeker2 AC values. Read direction, and five signal strength values from the HiTechnic IRSeeker2 sensor in AC mode. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

*s3* The signal strength from sensor 3.

*s5* The signal strength from sensor 5.

*s7* The signal strength from sensor 7.

*s9* The signal strength from sensor 9.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker2AC.nxc](#).

**6.17.2.91** `bool ReadSensorHTIRSeeker2DC (const byte port, byte & dir, byte & s1, byte & s3, byte & s5, byte & s7, byte & s9, byte & avg) [inline]`

Read HiTechnic IRSeeker2 DC values. Read direction, five signal strength, and average strength values from the HiTechnic IRSeeker2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

- s3* The signal strength from sensor 3.
- s5* The signal strength from sensor 5.
- s7* The signal strength from sensor 7.
- s9* The signal strength from sensor 9.
- avg* The average signal strength.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker2DC.nxc](#).

**6.17.2.92 bool ReadSensorHTNormalizedColor (const byte *port*, byte & *ColorIdx*, byte & *Red*, byte & *Green*, byte & *Blue*) [inline]**

Read HiTechnic Color normalized values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- ColorIdx* The output color index.
- Red* The normalized red color value.
- Green* The normalized green color value.
- Blue* The normalized blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTNormalizedColor.nxc](#).

**6.17.2.93** `bool ReadSensorHTNormalizedColor2Active (const byte port, byte & ColorIdx, byte & Red, byte & Green, byte & Blue) [inline]`

Read HiTechnic Color2 normalized active values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*ColorIdx* The output color index.

*Red* The normalized red color value.

*Green* The normalized green color value.

*Blue* The normalized blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTNormalizedColor2Active.nxc](#).

**6.17.2.94** `bool ReadSensorHTProtoAllAnalog (const byte port, int & a0, int & a1, int & a2, int & a3, int & a4) [inline]`

Read all HiTechnic Prototype board analog input values. Read all 5 analog input values from the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*a0* The A0 analog input value.

*a1* The A1 analog input value.

*a2* The A2 analog input value.

*a3* The A3 analog input value.

*a4* The A4 analog input value.

**Returns:**

The function call result.

**Examples:**

[ex\\_proto.nxc](#).

**6.17.2.95 bool ReadSensorHTRawColor (const byte *port*, unsigned int & *Red*, unsigned int & *Green*, unsigned int & *Blue*) [inline]**

Read HiTechnic Color raw values. Read the raw red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Red* The raw red color value.

*Green* The raw green color value.

*Blue* The raw blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTRawColor.nxc](#).

**6.17.2.96 bool ReadSensorHTRawColor2 (const byte *port*, unsigned int & *Red*, unsigned int & *Green*, unsigned int & *Blue*, unsigned int & *White*) [inline]**

Read HiTechnic Color2 raw values. Read the raw red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Red* The raw red color value.

*Green* The raw green color value.

*Blue* The raw blue color value.

*White* The raw white color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTRawColor2.nxc](#).

**6.17.2.97** `bool ReadSensorHTSuperProAllAnalog (const byte port, int & a0, int & a1, int & a2, int & a3) [inline]`

Read all HiTechnic SuperPro board analog input values. Read all 4 analog input values from the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*a0* The A0 analog input value.

*a1* The A1 analog input value.

*a2* The A2 analog input value.

*a3* The A3 analog input value.

**Returns:**

The function call result.

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.98** `bool ReadSensorHTSuperProAnalogOut (const byte port, const byte dac, byte & mode, int & freq, int & volt) [inline]`

Read HiTechnic SuperPro board analog output parameters. Read the analog output parameters on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- dac* The analog output index. See [HiTechnic SuperPro analog output index constants](#).
- mode* The analog output mode. See [SuperPro analog output mode constants](#).
- freq* The analog output frequency. Between 1 and 8191.
- volt* The analog output voltage level. A 10 bit value (0..1023).

**Returns:**

The function call result.

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.99 void ReadSensorHTTouchMultiplexer (const byte port, byte & t1, byte & t2, byte & t3, byte & t4) [inline]**

Read HiTechnic touch multiplexer. Read touch sensor values from the HiTechnic touch multiplexer device.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- t1* The value of touch sensor 1.
- t2* The value of touch sensor 2.
- t3* The value of touch sensor 3.
- t4* The value of touch sensor 4.

**Examples:**

[ex\\_ReadSensorHTTouchMultiplexer.nxc](#).

**6.17.2.100 bool ResetHTBarometricCalibration (byte *port*) [inline]**

Reset HiTechnic Barometric sensor calibration. Reset the HiTechnic Barometric sensor to its factory calibration. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The function call result.

**6.17.2.101 char ResetSensorHTAngle (const byte *port*, const byte *mode*) [inline]**

Reset HiTechnic Angle sensor. Reset the HiTechnic Angle sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The Angle reset mode. See [HiTechnic Angle sensor constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_ResetSensorHTAngle.nxc](#).

**6.17.2.102 int SensorHTColorNum (const byte & *port*) [inline]**

Read HiTechnic color sensor color number. Read the color number from the HiTechnic Color sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The color number.

**Examples:**

[ex\\_SensorHTColorNum.nxc](#).

**6.17.2.103 int SensorHTCompass (const byte & port) [inline]**

Read HiTechnic compass. Read the compass heading value of the HiTechnic Compass sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The compass heading.

**Examples:**

[ex\\_SensorHTCompass.nxc](#).

**6.17.2.104 int SensorHTEOPD (const byte & port) [inline]**

Read HiTechnic EOPD sensor. Read the HiTechnic EOPD sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The EOPD sensor reading.

**Examples:**

[ex\\_SensorHTEOPD.nxc](#).

**6.17.2.105 int SensorHTGyro (const byte & port, int offset = 0) [inline]**

Read HiTechnic Gyro sensor. Read the HiTechnic Gyro sensor on the specified port. The offset value should be calculated by averaging several readings with an offset of zero while the sensor is perfectly still.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The zero offset.

**Returns:**

The Gyro sensor reading.

**Examples:**

[ex\\_HTGyroTest.nxc](#), and [ex\\_SensorHTGyro.nxc](#).

**6.17.2.106 int SensorHTIRSeeker2ACDir (const byte & port) [inline]**

Read HiTechnic IRSeeker2 AC direction. Read the AC direction value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker2 AC direction.

**Examples:**

[ex\\_SensorHTIRSeeker2ACDir.nxc](#).

**6.17.2.107 int SensorHTIRSeeker2Addr (const byte & port, const byte reg) [inline]**

Read HiTechnic IRSeeker2 register. Read a register value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*reg* The register address. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The IRSeeker2 register value.

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**6.17.2.108 int SensorHTIRSeeker2DCDir (const byte & port) [inline]**

Read HiTechnic IRSeeker2 DC direction. Read the DC direction value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker2 DC direction.

**Examples:**

[ex\\_SensorHTIRSeeker2DCDir.nxc](#).

**6.17.2.109 int SensorHTIRSeekerDir (const byte & port) [inline]**

Read HiTechnic IRSeeker direction. Read the direction value of the HiTechnic IR Seeker on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker direction.

**Examples:**

[ex\\_SensorHTIRSeekerDir.nxc](#).

**6.17.2.110 int SensorHTMagnet (const byte & port, int offset = 0) [inline]**

Read HiTechnic Magnet sensor. Read the HiTechnic Magnet sensor on the specified port. The offset value should be calculated by averaging several readings with an offset of zero while the sensor is perfectly still.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The zero offset.

**Returns:**

The Magnet sensor reading.

**Examples:**

[ex\\_SensorHTMagnet.nxc](#).

**6.17.2.111 int SensorHTProtoAnalog (const byte port, const byte input) [inline]**

Read HiTechnic Prototype board analog input value. Read an analog input value from the HiTechnic prototype board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*input* The analog input. See [HiTechnic Prototype board analog input constants](#).

**Returns:**

The analog input value.

**Examples:**

[ex\\_proto.nxc](#).

**6.17.2.112 byte SensorHTProtoDigital (const byte *port*) [inline]**

Read HiTechnic Prototype board digital input values. Read digital input values from the HiTechnic prototype board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_proto.nxc](#).

**6.17.2.113 byte SensorHTProtoDigitalControl (const byte *port*) [inline]**

Read HiTechnic Prototype board digital control values. Read digital control values from the HiTechnic prototype board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital control values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_proto.nxc](#).

**6.17.2.114 int SensorHTSuperProAnalog (const byte *port*, const byte *input*)  
[inline]**

Read HiTechnic SuperPro board analog input value. Read an analog input value from the HiTechnic SuperPro board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*input* The analog input. See [HiTechnic SuperPro analog input index constants](#).

**Returns:**

The analog input value.

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.115 byte SensorHTSuperProDigital (const byte *port*) [inline]**

Read HiTechnic SuperPro board digital input values. Read digital input values from the HiTechnic SuperPro board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.116 byte SensorHTSuperProDigitalControl (const byte *port*)  
[inline]**

Read HiTechnic SuperPro board digital control values. Read digital control values from the HiTechnic SuperPro board. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.117** byte `SensorHTSuperProLED` (const byte *port*) [`inline`]

Read HiTechnic SuperPro LED value. Read the HiTechnic SuperPro LED value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The LED value. See [SuperPro LED control constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.118** byte `SensorHTSuperProProgramControl` (const byte *port*) [`inline`]

Read HiTechnic SuperPro program control value. Read the HiTechnic SuperPro program control value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The program control value.

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.119 byte SensorHTSuperProStrobe (const byte *port*) [inline]**

Read HiTechnic SuperPro strobe value. Read the HiTechnic SuperPro strobe value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The strobe value. See [SuperPro Strobe control constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**6.17.2.120 bool SetHTBarometricCalibration (byte *port*, unsigned int *cal*) [inline]**

Set HiTechnic Barometric sensor calibration. Set the HiTechnic Barometric sensor pressure calibration value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*cal* The new pressure calibration value.

**Returns:**

The function call result.

**6.17.2.121 char SetHTColor2Mode (const byte & port, byte mode) [inline]**

Set HiTechnic Color2 mode. Set the mode of the HiTechnic Color2 sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The Color2 mode. See [HiTechnic Color2 constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_sethtcolor2mode.nxc](#).

**6.17.2.122 char SetHTIRSeeker2Mode (const byte & port, const byte mode) [inline]**

Set HiTechnic IRSeeker2 mode. Set the mode of the HiTechnic IRSeeker2 sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The IRSeeker2 mode. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_sethtirseeker2mode.nxc](#), and [ex\\_setsensorboolean.nxc](#).

**6.17.2.123 void SetSensorHTEOPD (const byte & port, bool bStandard) [inline]**

Set sensor as HiTechnic EOPD. Configure the sensor on the specified port as a HiTechnic EOPD sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*bStandard* Configure in standard or long-range mode.

**Examples:**

[ex\\_setsensorhteopd.nxc](#).

**6.17.2.124 void SetSensorHTGyro (const byte & port) [inline]**

Set sensor as HiTechnic Gyro. Configure the sensor on the specified port as a HiTechnic Gyro sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Examples:**

[ex\\_HTGyroTest.nxc](#), [ex\\_SensorHTGyro.nxc](#), and [ex\\_SetSensorHTGyro.nxc](#).

**6.17.2.125 void SetSensorHTMagnet (const byte & port) [inline]**

Set sensor as HiTechnic Magnet. Configure the sensor on the specified port as a HiTechnic Magnet sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorHTMagnet.nxc](#).

**6.17.2.126 bool SetSensorHTProtoDigital (const byte port, byte value) [inline]**

Set HiTechnic Prototype board digital output values. Set the digital pin output values on the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The digital pin output values. See [SuperPro digital pin constants](#).

**Returns:**

The function call result.

**6.17.2.127 bool SetSensorHTProtoDigitalControl (const byte *port*, byte *value*)  
[inline]**

Control HiTechnic Prototype board digital pin direction. Control the direction of the six digital pins on the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The digital pin control value. See [SuperPro digital pin constants](#). OR into this value the pins that you want to be output pins. The pins not included in the value will be input pins.

**Returns:**

The function call result.

**6.17.2.128 bool SetSensorHTSuperProAnalogOut (const byte *port*, const byte *dac*, byte *mode*, int *freq*, int *volt*) [inline]**

Set HiTechnic SuperPro board analog output parameters. Set the analog output parameters on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- dac* The analog output index. See [HiTechnic SuperPro analog output index constants](#).
- mode* The analog output mode. See [SuperPro analog output mode constants](#).
- freq* The analog output frequency. Between 1 and 8191.
- volt* The analog output voltage level. A 10 bit value (0..1023).

**Returns:**

The function call result.

**6.17.2.129 bool SetSensorHTSuperProDigital (const byte *port*, byte *value*) [inline]**

Set HiTechnic SuperPro board digital output values. Set the digital pin output values on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- value* The digital pin output values. See [SuperPro digital pin constants](#).

**Returns:**

The function call result.

**6.17.2.130 bool SetSensorHTSuperProDigitalControl (const byte *port*, byte *value*) [inline]**

Control HiTechnic SuperPro board digital pin direction. Control the direction of the eight digital pins on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).

*value* The digital pin control value. See [SuperPro digital pin constants](#). OR into this value the pins that you want to be output pins. The pins not included in the value will be input pins.

**Returns:**

The function call result.

**6.17.2.131 bool SetSensorHTSuperProLED (const byte *port*, byte *value*) [inline]**

Set HiTechnic SuperPro LED value. Set the HiTechnic SuperPro LED value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The LED value. See [SuperPro LED control constants](#).

**Returns:**

The function call result.

**6.17.2.132 bool SetSensorHTSuperProProgramControl (const byte *port*, byte *value*) [inline]**

Set HiTechnic SuperPro program control value. Set the HiTechnic SuperPro program control value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The program control value.

**Returns:**

The function call result.

**6.17.2.133 bool SetSensorHTSuperProStrobe (const byte *port*, byte *value*)  
[inline]**

Set HiTechnic SuperPro strobe value. Set the HiTechnic SuperPro strobe value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- value* The strobe value. See [SuperPro Strobe control constants](#).

**Returns:**

The function call result.

**6.18 SuperPro analog output mode constants**

Constants for controlling the 2 analog output modes.

**Defines**

- #define [DAC\\_MODE\\_DCOUT](#) 0
- #define [DAC\\_MODE\\_SINEWAVE](#) 1
- #define [DAC\\_MODE\\_SQUAREWAVE](#) 2
- #define [DAC\\_MODE\\_SAWPOSWAVE](#) 3
- #define [DAC\\_MODE\\_SAWNEGWAVE](#) 4
- #define [DAC\\_MODE\\_TRIANGLEWAVE](#) 5
- #define [DAC\\_MODE\\_PWMVOLTAGE](#) 6

**6.18.1 Detailed Description**

Constants for controlling the 2 analog output modes. Two analog outputs, which can span 0 to 3.3 volts, can be programmed to output a steady voltage or can be programmed to output a selection of waveforms over a range of frequencies.

In the DC output mode, the DAC0/DAC1 voltage fields control the voltage on the two analog outputs in increments of  $\sim 3.2\text{mV}$  from 0 - 1023 giving 0 - 3.3v.

In waveform modes, the channel outputs will center on 1.65 volts when generating waveforms. The DAC0/DAC1 voltage fields control the signal levels of the waveforms by adjusting the peak to peak signal levels from 0 - 3.3v.

In PWM voltage mode, the channel outputs will create a variable mark:space ratio square wave at 3.3v signal level. The average output voltage is set by the O0/O1 voltage fields.

### 6.18.2 Define Documentation

#### 6.18.2.1 #define DAC\_MODE\_DCOUT 0

Steady (DC) voltage output.

#### 6.18.2.2 #define DAC\_MODE\_PWMVOLTAGE 6

PWM square wave output.

#### 6.18.2.3 #define DAC\_MODE\_SAWNEGWAVE 4

Negative going sawtooth output.

#### 6.18.2.4 #define DAC\_MODE\_SAWPOSWAVE 3

Positive going sawtooth output.

#### 6.18.2.5 #define DAC\_MODE\_SINEWAVE 1

Sine wave output.

#### Examples:

[ex\\_superpro.nxc](#).

#### 6.18.2.6 #define DAC\_MODE\_SQUAREWAVE 2

Square wave output.

#### 6.18.2.7 #define DAC\_MODE\_TRIANGLEWAVE 5

Triangle wave output.

## 6.19 SuperPro LED control constants

Constants for controlling the 2 onboard LEDs.

### Defines

- #define `LED_BLUE` 0x02
- #define `LED_RED` 0x01
- #define `LED_NONE` 0x00

### 6.19.1 Detailed Description

Constants for controlling the 2 onboard LEDs.

### 6.19.2 Define Documentation

#### 6.19.2.1 #define `LED_BLUE` 0x02

Turn on the blue onboard LED.

#### Examples:

```
ex_superpro.nxc.
```

#### 6.19.2.2 #define `LED_NONE` 0x00

Turn off the onboard LEDs.

#### 6.19.2.3 #define `LED_RED` 0x01

Turn on the red onboard LED.

## 6.20 SuperPro digital pin constants

Constants for controlling the 8 digital pins.

### Defines

- #define `DIGI_PIN0` 0x01
- #define `DIGI_PIN1` 0x02

- #define DIGI\_PIN2 0x04
- #define DIGI\_PIN3 0x08
- #define DIGI\_PIN4 0x10
- #define DIGI\_PIN5 0x20
- #define DIGI\_PIN6 0x40
- #define DIGI\_PIN7 0x80

### 6.20.1 Detailed Description

Constants for controlling the 8 digital pins. The eight digital inputs are returned as a byte representing the state of the eight inputs. The eight digital outputs are controlled by two bytes, the first of which sets the state of any of the signals which have been defined as outputs and the second of which controls the input/output state of each signal.

### 6.20.2 Define Documentation

#### 6.20.2.1 #define DIGI\_PIN0 0x01

Access digital pin 0 (B0)

#### Examples:

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

#### 6.20.2.2 #define DIGI\_PIN1 0x02

Access digital pin 1 (B1)

#### Examples:

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

#### 6.20.2.3 #define DIGI\_PIN2 0x04

Access digital pin 2 (B2)

#### Examples:

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

**6.20.2.4 #define DIGI\_PIN3 0x08**

Access digital pin 3 (B3)

**6.20.2.5 #define DIGI\_PIN4 0x10**

Access digital pin 4 (B4)

**6.20.2.6 #define DIGI\_PIN5 0x20**

Access digital pin 5 (B5)

**6.20.2.7 #define DIGI\_PIN6 0x40**

Access digital pin 6 (B6)

**6.20.2.8 #define DIGI\_PIN7 0x80**

Access digital pin 7 (B7)

**6.21 SuperPro Strobe control constants**

Constants for manipulating the six digital strobe outputs.

**Defines**

- #define [STROBE\\_S0](#) 0x01
- #define [STROBE\\_S1](#) 0x02
- #define [STROBE\\_S2](#) 0x04
- #define [STROBE\\_S3](#) 0x08
- #define [STROBE\\_READ](#) 0x10
- #define [STROBE\\_WRITE](#) 0x20

**6.21.1 Detailed Description**

Constants for manipulating the six digital strobe outputs. Six digital strobe outputs are available. One is pre-configured as a read strobe, another is pre-configured as a write strobe while the other four can be set to a high or low logic level. These strobe lines enable external devices to synchronize with the digital data port and multiplex the eight digital input/output bits to wider bit widths.

The RD and WR bits set the inactive state of the read and write strobe outputs. Thus, if these bits are set to 0, the strobe outputs will pulse high.

### 6.21.2 Define Documentation

#### 6.21.2.1 #define STROBE\_READ 0x10

Access read pin (RD)

#### 6.21.2.2 #define STROBE\_S0 0x01

Access strobe 0 pin (S0)

#### Examples:

[ex\\_superpro.nxc](#).

#### 6.21.2.3 #define STROBE\_S1 0x02

Access strobe 1 pin (S1)

#### 6.21.2.4 #define STROBE\_S2 0x04

Access strobe 2 pin (S2)

#### 6.21.2.5 #define STROBE\_S3 0x08

Access strobe 3 pin (S3)

#### 6.21.2.6 #define STROBE\_WRITE 0x20

Access write pin (WR)

## 6.22 MindSensors API Functions

Functions for accessing and modifying MindSensors devices.

## Modules

- [MindSensors device constants](#)

*Constants that are for use with MindSensors devices.*

## Functions

- void [SetSensorMSPressure](#) (const byte &port)  
*Configure a mindsensors pressure sensor.*
- void [SetSensorMSDROD](#) (const byte &port, bool bActive)  
*Configure a mindsensors DROD sensor.*
- void [SetSensorNXTSumoEyes](#) (const byte &port, bool bLong)  
*Configure a mindsensors SumoEyes sensor.*
- int [SensorMSPressure](#) (const byte &port)  
*Read mindsensors pressure sensor.*
- char [SensorNXTSumoEyes](#) (const byte &port)  
*Read mindsensors NXTSumoEyes obstacle zone.*
- int [SensorMSCompass](#) (const byte &port, const byte i2caddr)  
*Read mindsensors compass value.*
- int [SensorMSDROD](#) (const byte &port)  
*Read mindsensors DROD value.*
- int [SensorNXTSumoEyesRaw](#) (const byte &port)  
*Read mindsensors NXTSumoEyes raw value.*
- int [SensorMSPressureRaw](#) (const byte &port)  
*Read mindsensors raw pressure value.*
- bool [ReadSensorMSAccel](#) (const byte port, const byte i2caddr, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSPlayStation](#) (const byte port, const byte i2caddr, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)  
*Read mindsensors playstation controller values.*

- bool [ReadSensorMSRTClock](#) (const byte port, byte &sec, byte &min, byte &hrs, byte &dow, byte &date, byte &month, byte &year)  
*Read mindsensors RTClock values.*
- bool [ReadSensorMSTilt](#) (const byte &port, const byte &i2caddr, byte &x, byte &y, byte &z)  
*Read mindsensors tilt values.*
- bool [PFMateSend](#) (const byte &port, const byte &i2caddr, const byte &channel, const byte &motors, const byte &cmdA, const byte &spdA, const byte &cmdB, const byte &spdB)  
*Send PFMate command.*
- bool [PFMateSendRaw](#) (const byte &port, const byte &i2caddr, const byte &channel, const byte &b1, const byte &b2)  
*Send raw PFMate command.*
- int [MSReadValue](#) (const byte port, const byte i2caddr, const byte reg, const byte numbytes)  
*Read a mindsensors device value.*
- char [MSEnergize](#) (const byte port, const byte i2caddr)  
*Turn on power to device.*
- char [MSDeenergize](#) (const byte port, const byte i2caddr)  
*Turn off power to device.*
- char [MSADPAOn](#) (const byte port, const byte i2caddr)  
*Turn on mindsensors ADPA mode.*
- char [MSADPAOff](#) (const byte port, const byte i2caddr)  
*Turn off mindsensors ADPA mode.*
- char [DISTNxGP2D12](#) (const byte port, const byte i2caddr)  
*Configure DISTNx as GP2D12.*
- char [DISTNxGP2D120](#) (const byte port, const byte i2caddr)  
*Configure DISTNx as GP2D120.*
- char [DISTNxGP2YA02](#) (const byte port, const byte i2caddr)  
*Configure DISTNx as GP2YA02.*
- char [DISTNxGP2YA21](#) (const byte port, const byte i2caddr)

*Configure DISTNx as GP2YA21.*

- int **DISTNxDistance** (const byte port, const byte i2caddr)  
*Read DISTNx distance value.*
- int **DISTNxMaxDistance** (const byte port, const byte i2caddr)  
*Read DISTNx maximum distance value.*
- int **DISTNxMinDistance** (const byte port, const byte i2caddr)  
*Read DISTNx minimum distance value.*
- byte **DISTNxModuleType** (const byte port, const byte i2caddr)  
*Read DISTNx module type value.*
- byte **DISTNxNumPoints** (const byte port, const byte i2caddr)  
*Read DISTNx num points value.*
- int **DISTNxVoltage** (const byte port, const byte i2caddr)  
*Read DISTNx voltage value.*
- char **ACCLNxCalibrateX** (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx X-axis.*
- char **ACCLNxCalibrateXEnd** (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx X-axis.*
- char **ACCLNxCalibrateY** (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx Y-axis.*
- char **ACCLNxCalibrateYEnd** (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx Y-axis.*
- char **ACCLNxCalibrateZ** (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx Z-axis.*
- char **ACCLNxCalibrateZEnd** (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx Z-axis.*
- char **ACCLNxResetCalibration** (const byte port, const byte i2caddr)  
*Reset ACCL-Nx calibration.*
- char **SetACCLNxSensitivity** (const byte port, const byte i2caddr, byte slevel)

*Set ACCL-Nx sensitivity.*

- byte **ACCLNxSensitivity** (const byte port, const byte i2caddr)  
*Read ACCL-Nx sensitivity value.*
- int **ACCLNxXOffset** (const byte port, const byte i2caddr)  
*Read ACCL-Nx X offset value.*
- int **ACCLNxXRange** (const byte port, const byte i2caddr)  
*Read ACCL-Nx X range value.*
- int **ACCLNxYOffset** (const byte port, const byte i2caddr)  
*Read ACCL-Nx Y offset value.*
- int **ACCLNxYRange** (const byte port, const byte i2caddr)  
*Read ACCL-Nx Y range value.*
- int **ACCLNxZOffset** (const byte port, const byte i2caddr)  
*Read ACCL-Nx Z offset value.*
- int **ACCLNxZRange** (const byte port, const byte i2caddr)  
*Read ACCL-Nx Z range value.*
- char **PSPNxDigital** (const byte &port, const byte &i2caddr)  
*Configure PSPNx in digital mode.*
- char **PSPNxAnalog** (const byte &port, const byte &i2caddr)  
*Configure PSPNx in analog mode.*
- unsigned int **NXTServoPosition** (const byte &port, const byte &i2caddr, const byte servo)  
*Read NXTServo servo position value.*
- byte **NXTServoSpeed** (const byte &port, const byte &i2caddr, const byte servo)  
*Read NXTServo servo speed value.*
- byte **NXTServoBatteryVoltage** (const byte &port, const byte &i2caddr)  
*Read NXTServo battery voltage value.*
- char **SetNXTServoSpeed** (const byte &port, const byte &i2caddr, const byte servo, const byte &speed)  
*Set NXTServo servo motor speed.*

- char [SetNXTServoQuickPosition](#) (const byte &port, const byte &i2caddr, const byte servo, const byte &qpos)  
*Set NXTServo servo motor quick position.*
- char [SetNXTServoPosition](#) (const byte &port, const byte &i2caddr, const byte servo, const byte &pos)  
*Set NXTServo servo motor position.*
- char [NXTServoReset](#) (const byte &port, const byte &i2caddr)  
*Reset NXTServo properties.*
- char [NXTServoHaltMacro](#) (const byte &port, const byte &i2caddr)  
*Halt NXTServo macro.*
- char [NXTServoResumeMacro](#) (const byte &port, const byte &i2caddr)  
*Resume NXTServo macro.*
- char [NXTServoPauseMacro](#) (const byte &port, const byte &i2caddr)  
*Pause NXTServo macro.*
- char [NXTServoInit](#) (const byte &port, const byte &i2caddr, const byte servo)  
*Initialize NXTServo servo properties.*
- char [NXTServoGotoMacroAddress](#) (const byte &port, const byte &i2caddr, const byte &macro)  
*Goto NXTServo macro address.*
- char [NXTServoEditMacro](#) (const byte &port, const byte &i2caddr)  
*Edit NXTServo macro.*
- char [NXTServoQuitEdit](#) (const byte &port)  
*Quit NXTServo macro edit mode.*
- char [NXTHIDAsciiMode](#) (const byte &port, const byte &i2caddr)  
*Set NXTHID into ASCII data mode.*
- char [NXTHIDDirectMode](#) (const byte &port, const byte &i2caddr)  
*Set NXTHID into direct data mode.*
- char [NXTHIDTransmit](#) (const byte &port, const byte &i2caddr)  
*Transmit NXTHID character.*

- char [NXTHIDLoadCharacter](#) (const byte &port, const byte &i2caddr, const byte &modifier, const byte &character)  
*Load NXTHID character.*
- char [NXTPowerMeterResetCounters](#) (const byte &port, const byte &i2caddr)  
*Reset NXTPowerMeter counters.*
- int [NXTPowerMeterPresentCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present current.*
- int [NXTPowerMeterPresentVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present voltage.*
- int [NXTPowerMeterCapacityUsed](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter capacity used.*
- int [NXTPowerMeterPresentPower](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present power.*
- long [NXTPowerMeterTotalPowerConsumed](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter total power consumed.*
- int [NXTPowerMeterMaxCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter maximum current.*
- int [NXTPowerMeterMinCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter minimum current.*
- int [NXTPowerMeterMaxVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter maximum voltage.*
- int [NXTPowerMeterMinVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter minimum voltage.*
- long [NXTPowerMeterElapsedTime](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter elapsed time.*
- int [NXTPowerMeterErrorCount](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter error count.*
- char [NXTLLineLeaderPowerDown](#) (const byte &port, const byte &i2caddr)  
*Powerdown NXTLLineLeader device.*

- char **NXTLineLeaderPowerUp** (const byte &port, const byte &i2caddr)  
*Powerup NXTLineLeader device.*
- char **NXTLineLeaderInvert** (const byte &port, const byte &i2caddr)  
*Invert NXTLineLeader colors.*
- char **NXTLineLeaderReset** (const byte &port, const byte &i2caddr)  
*Reset NXTLineLeader color inversion.*
- char **NXTLineLeaderSnapshot** (const byte &port, const byte &i2caddr)  
*Take NXTLineLeader line snapshot.*
- char **NXTLineLeaderCalibrateWhite** (const byte &port, const byte &i2caddr)  
*Calibrate NXTLineLeader white color.*
- char **NXTLineLeaderCalibrateBlack** (const byte &port, const byte &i2caddr)  
*Calibrate NXTLineLeader black color.*
- char **NXTLineLeaderSteering** (const byte &port, const byte &i2caddr)  
*Read NXTLineLeader steering.*
- char **NXTLineLeaderAverage** (const byte &port, const byte &i2caddr)  
*Read NXTLineLeader average.*
- byte **NXTLineLeaderResult** (const byte &port, const byte &i2caddr)  
*Read NXTLineLeader result.*
- char **SetNXTLineLeaderSetpoint** (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader setpoint.*
- char **SetNXTLineLeaderKpValue** (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kp value.*
- char **SetNXTLineLeaderKiValue** (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Ki value.*
- char **SetNXTLineLeaderKdValue** (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kd value.*

- char [SetNXTLineLeaderKpFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kp factor.*
- char [SetNXTLineLeaderKiFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Ki factor.*
- char [SetNXTLineLeaderKdFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kd factor.*
- char [NRLink2400](#) (const byte port, const byte i2caddr)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink4800](#) (const byte port, const byte i2caddr)  
*Configure NRLink in 4800 baud mode.*
- char [NRLinkFlush](#) (const byte port, const byte i2caddr)  
*Flush NRLink buffers.*
- char [NRLinkIRLong](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRShort](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR short mode.*
- char [NRLinkSetPF](#) (const byte port, const byte i2caddr)  
*Configure NRLink in power function mode.*
- char [NRLinkSetRCX](#) (const byte port, const byte i2caddr)  
*Configure NRLink in RCX mode.*
- char [NRLinkSetTrain](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR train mode.*
- char [NRLinkTxRaw](#) (const byte port, const byte i2caddr)  
*Configure NRLink in raw IR transmit mode.*
- byte [NRLinkStatus](#) (const byte port, const byte i2caddr)  
*Read NRLink status.*

- char [RunNRLinkMacro](#) (const byte port, const byte i2caddr, const byte macro)  
*Run NRLink macro.*
- char [WriteNRLinkBytes](#) (const byte port, const byte i2caddr, const byte data[ ])  
*Write data to NRLink.*
- bool [ReadNRLinkBytes](#) (const byte port, const byte i2caddr, byte &data[ ])  
*Read data from NRLink.*
- char [MSIRTrain](#) (const byte port, const byte i2caddr, const byte channel, const byte func)  
*MSIRTrain function.*
- char [MSPFComboDirect](#) (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirect function.*
- char [MSPFComboPWM](#) (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWM function.*
- char [MSPFRawOutput](#) (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutput function.*
- char [MSPFRepeat](#) (const byte port, const byte i2caddr, const byte count, const unsigned int delay)  
*MSPFRepeat function.*
- char [MSPFSingleOutputCST](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCST function.*
- char [MSPFSingleOutputPWM](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWM function.*
- char [MSPFSinglePin](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePin function.*
- char [MSPFTrain](#) (const byte port, const byte i2caddr, const byte channel, const byte func)

*MSPFTrain function.*

- void [MSRCXSetNRLinkPort](#) (const byte port, const byte i2caddr)  
*MSRCXSetIRLinkPort function.*
- int [MSRCXBatteryLevel](#) (void)  
*MSRCXBatteryLevel function.*
- int [MSRCXPoll](#) (const byte src, const byte value)  
*MSRCXPoll function.*
- int [MSRCXPollMemory](#) (const unsigned int address)  
*MSRCXPollMemory function.*
- void [MSRCXAbsVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXAbsVar function.*
- void [MSRCXAddToDatalog](#) (const byte src, const unsigned int value)  
*MSRCXAddToDatalog function.*
- void [MSRCXAndVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXAndVar function.*
- void [MSRCXBoot](#) (void)  
*MSRCXBoot function.*
- void [MSRCXCalibrateEvent](#) (const byte evt, const byte low, const byte hi, const byte hyst)  
*MSRCXCalibrateEvent function.*
- void [MSRCXClearAllEvents](#) (void)  
*MSRCXClearAllEvents function.*
- void [MSRCXClearCounter](#) (const byte counter)  
*MSRCXClearCounter function.*
- void [MSRCXClearMsg](#) (void)  
*MSRCXClearMsg function.*
- void [MSRCXClearSensor](#) (const byte port)  
*MSRCXClearSensor function.*

- void [MSRCXClearSound](#) (void)  
*MSRCXClearSound function.*
- void [MSRCXClearTimer](#) (const byte timer)  
*MSRCXClearTimer function.*
- void [MSRCXCreateDatalog](#) (const unsigned int size)  
*MSRCXCreateDatalog function.*
- void [MSRCXDecCounter](#) (const byte counter)  
*MSRCXDecCounter function.*
- void [MSRCXDeleteSub](#) (const byte s)  
*MSRCXDeleteSub function.*
- void [MSRCXDeleteSubs](#) (void)  
*MSRCXDeleteSubs function.*
- void [MSRCXDeleteTask](#) (const byte t)  
*MSRCXDeleteTask function.*
- void [MSRCXDeleteTasks](#) (void)  
*MSRCXDeleteTasks function.*
- void [MSRCXDisableOutput](#) (const byte outputs)  
*MSRCXDisableOutput function.*
- void [MSRCXDivVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXDivVar function.*
- void [MSRCXEnableOutput](#) (const byte outputs)  
*MSRCXEnableOutput function.*
- void [MSRCXEvent](#) (const byte src, const unsigned int value)  
*MSRCXEvent function.*
- void [MSRCXFloat](#) (const byte outputs)  
*MSRCXFloat function.*
- void [MSRCXFwd](#) (const byte outputs)

*MSRCXFwd function.*

- void [MSRCXIncCounter](#) (const byte counter)  
*MSRCXIncCounter function.*
- void [MSRCXInvertOutput](#) (const byte outputs)  
*MSRCXInvertOutput function.*
- void [MSRCXMulVar](#) (const byte varnum, const byte src, unsigned int value)  
*MSRCXMulVar function.*
- void [MSRCXMuteSound](#) (void)  
*MSRCXMuteSound function.*
- void [MSRCXObvertOutput](#) (const byte outputs)  
*MSRCXObvertOutput function.*
- void [MSRCXOff](#) (const byte outputs)  
*MSRCXOff function.*
- void [MSRCXOn](#) (const byte outputs)  
*MSRCXOn function.*
- void [MSRCXOnFor](#) (const byte outputs, const unsigned int ms)  
*MSRCXOnFor function.*
- void [MSRCXOnFwd](#) (const byte outputs)  
*MSRCXOnFwd function.*
- void [MSRCXOnRev](#) (const byte outputs)  
*MSRCXOnRev function.*
- void [MSRCXOrVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXOrVar function.*
- void [MSRCXPBTurnOff](#) (void)  
*MSRCXPBTurnOff function.*
- void [MSRCXPing](#) (void)  
*MSRCXPing function.*
- void [MSRCXPlaySound](#) (const byte snd)

*MSRCXPlaySound function.*

- void **MSRCXPlayTone** (const unsigned int freq, const byte duration)  
*MSRCXPlayTone function.*
- void **MSRCXPlayToneVar** (const byte varnum, const byte duration)  
*MSRCXPlayToneVar function.*
- void **MSRCXRemote** (unsigned int cmd)  
*MSRCXRemote function.*
- void **MSRCXReset** (void)  
*MSRCXReset function.*
- void **MSRCXRev** (const byte outputs)  
*MSRCXRev function.*
- void **MSRCXSelectDisplay** (const byte src, const unsigned int value)  
*MSRCXSelectDisplay function.*
- void **MSRCXSelectProgram** (const byte prog)  
*MSRCXSelectProgram function.*
- void **MSRCXSendSerial** (const byte first, const byte count)  
*MSRCXSendSerial function.*
- void **MSRCXSet** (const byte dstsrc, const byte dstval, const byte src, unsigned int value)  
*MSRCXSet function.*
- void **MSRCXSetDirection** (const byte outputs, const byte dir)  
*MSRCXSetDirection function.*
- void **MSRCXSetEvent** (const byte evt, const byte src, const byte type)  
*MSRCXSetEvent function.*
- void **MSRCXSetGlobalDirection** (const byte outputs, const byte dir)  
*MSRCXSetGlobalDirection function.*
- void **MSRCXSetGlobalOutput** (const byte outputs, const byte mode)  
*MSRCXSetGlobalOutput function.*

- void [MSRCXSetMaxPower](#) (const byte outputs, const byte pwsrc, const byte pwrval)  
*MSRCXSetMaxPower function.*
- void [MSRCXSetMessage](#) (const byte msg)  
*MSRCXSetMessage function.*
- void [MSRCXSetOutput](#) (const byte outputs, const byte mode)  
*MSRCXSetOutput function.*
- void [MSRCXSetPower](#) (const byte outputs, const byte pwsrc, const byte pwrval)  
*MSRCXSetPower function.*
- void [MSRCXSetPriority](#) (const byte p)  
*MSRCXSetPriority function.*
- void [MSRCXSetSensorMode](#) (const byte port, const byte mode)  
*MSRCXSetSensorMode function.*
- void [MSRCXSetSensorType](#) (const byte port, const byte type)  
*MSRCXSetSensorType function.*
- void [MSRCXSetSleepTime](#) (const byte t)  
*MSRCXSetSleepTime function.*
- void [MSRCXSetTxPower](#) (const byte pwr)  
*MSRCXSetTxPower function.*
- void [MSRCXSetUserDisplay](#) (const byte src, const unsigned int value, const byte precision)  
*MSRCXSetUserDisplay function.*
- void [MSRCXSetVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSetVar function.*
- void [MSRCXSetWatch](#) (const byte hours, const byte minutes)  
*MSRCXSetWatch function.*
- void [MSRCXSgnVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSgnVar function.*

- void [MSRCXStartTask](#) (const byte t)  
*MSRCXStartTask function.*
- void [MSRCXStopAllTasks](#) (void)  
*MSRCXStopAllTasks function.*
- void [MSRCXStopTask](#) (const byte t)  
*MSRCXStopTask function.*
- void [MSRCXSubVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSubVar function.*
- void [MSRCXSumVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSumVar function.*
- void [MSRCXToggle](#) (const byte outputs)  
*MSRCXToggle function.*
- void [MSRCXUnlock](#) (void)  
*MSRCXUnlock function.*
- void [MSRCXUnmuteSound](#) (void)  
*MSRCXUnmuteSound function.*
- void [MSScoutCalibrateSensor](#) (void)  
*MSScoutCalibrateSensor function.*
- void [MSScoutMuteSound](#) (void)  
*MSScoutMuteSound function.*
- void [MSScoutSelectSounds](#) (const byte grp)  
*MSScoutSelectSounds function.*
- void [MSScoutSendVLL](#) (const byte src, const unsigned int value)  
*MSScoutSendVLL function.*
- void [MSScoutSetCounterLimit](#) (const byte ctr, const byte src, const unsigned int value)  
*MSScoutSetCounterLimit function.*

- void [MSScoutSetEventFeedback](#) (const byte src, const unsigned int value)  
*MSScoutSetEventFeedback function.*
- void [MSScoutSetLight](#) (const byte x)  
*MSScoutSetLight function.*
- void [MSScoutSetScoutMode](#) (const byte mode)  
*MSScoutSetScoutMode function.*
- void [MSScoutSetScoutRules](#) (const byte m, const byte t, const byte l, const byte tm, const byte fx)  
*MSScoutSetScoutRules function.*
- void [MSScoutSetSensorClickTime](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorClickTime function.*
- void [MSScoutSetSensorHysteresis](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorHysteresis function.*
- void [MSScoutSetSensorLowerLimit](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorLowerLimit function.*
- void [MSScoutSetSensorUpperLimit](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorUpperLimit function.*
- void [MSScoutSetTimerLimit](#) (const byte tmr, const byte src, const unsigned int value)  
*MSScoutSetTimerLimit function.*
- void [MSScoutUnmuteSound](#) (void)  
*MSScoutUnmuteSound function.*

### 6.22.1 Detailed Description

Functions for accessing and modifying MindSensors devices.

### 6.22.2 Function Documentation

#### 6.22.2.1 char ACCLNxCalibrateX (const byte port, const byte i2caddr) [inline]

Calibrate ACCL-Nx X-axis. Calibrate the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateX.nxc](#).

**6.22.2.2 char ACCLNxCalibrateXEnd (const byte *port*, const byte *i2caddr*)  
[inline]**

Stop calibrating ACCL-Nx X-axis. Stop calibrating the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateXEnd.nxc](#).

**6.22.2.3 char ACCLNxCalibrateY (const byte *port*, const byte *i2caddr*)  
[inline]**

Calibrate ACCL-Nx Y-axis. Calibrate the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateY.nxc](#).

**6.22.2.4 char ACCLNxCalibrateYEnd (const byte port, const byte i2caddr) [inline]**

Stop calibrating ACCL-Nx Y-axis. Stop calibrating the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateYEnd.nxc](#).

**6.22.2.5 char ACCLNxCalibrateZ (const byte port, const byte i2caddr) [inline]**

Calibrate ACCL-Nx Z-axis. Calibrate the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateZ.nxc](#).

**6.22.2.6 char ACCLNxCalibrateZEnd (const byte *port*, const byte *i2caddr*)  
[inline]**

Stop calibrating ACCL-Nx Z-axis. Stop calibrating the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateZEnd.nxc](#).

**6.22.2.7 char ACCLNxResetCalibration (const byte *port*, const byte *i2caddr*)  
[inline]**

Reset ACCL-Nx calibration. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxResetCalibration.nxc](#).

### 6.22.2.8 `byte ACCLNxSensitivity (const byte port, const byte i2caddr)` `[inline]`

Read ACCL-Nx sensitivity value. Read the mindsensors ACCL-Nx sensitivity value. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The sensitivity value.

#### Examples:

[ex\\_ACCLNxSensitivity.nxc](#).

### 6.22.2.9 `int ACCLNxXOffset (const byte port, const byte i2caddr)` `[inline]`

Read ACCL-Nx X offset value. Read the mindsensors ACCL-Nx sensor's X offset value. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The X offset value.

#### Examples:

[ex\\_ACCLNxXOffset.nxc](#).

### 6.22.2.10 `int ACCLNxXRange (const byte port, const byte i2caddr)` `[inline]`

Read ACCL-Nx X range value. Read the mindsensors ACCL-Nx sensor's X range value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The X range value.

**Examples:**

[ex\\_ACCLNxXRange.nxc](#).

**6.22.2.11 int ACCLNxYOffset (const byte port, const byte i2caddr) [inline]**

Read ACCL-Nx Y offset value. Read the mindsensors ACCL-Nx sensor's Y offset value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Y offset value.

**Examples:**

[ex\\_ACCLNxYOffset.nxc](#).

**6.22.2.12 int ACCLNxYRange (const byte port, const byte i2caddr) [inline]**

Read ACCL-Nx Y range value. Read the mindsensors ACCL-Nx sensor's Y range value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Y range value.

**Examples:**

[ex\\_ACCLNxYRange.nxc](#).

**6.22.2.13 int ACCLNxZOffset (const byte *port*, const byte *i2caddr*) [inline]**

Read ACCL-Nx Z offset value. Read the mindsensors ACCL-Nx sensor's Z offset value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Z offset value.

**Examples:**

[ex\\_ACCLNxZOffset.nxc](#).

**6.22.2.14 int ACCLNxZRange (const byte *port*, const byte *i2caddr*) [inline]**

Read ACCL-Nx Z range value. Read the mindsensors ACCL-Nx sensor's Z range value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Z range value.

**Examples:**

[ex\\_ACCLNxZRange.nxc](#).

**6.22.2.15 int DISTNxDistance (const byte *port*, const byte *i2caddr*) [inline]**

Read DISTNx distance value. Read the mindsensors DISTNx sensor's distance value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

[ex\\_DISTNxDistance.nxc](#).

**6.22.2.16 char DISTNxGP2D12 (const byte *port*, const byte *i2caddr*) [inline]**

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2D12.nxc](#).

**6.22.2.17 char DISTNxGP2D120 (const byte *port*, const byte *i2caddr*) [inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2D120.nxc](#).

**6.22.2.18 char DISTNxGP2YA02 (const byte port, const byte i2caddr)  
[inline]**

Configure DISTNx as GP2YA02. Configure the mindsensors DISTNx sensor as GP2YA02. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA02.nxc](#).

**6.22.2.19 char DISTNxGP2YA21 (const byte port, const byte i2caddr)  
[inline]**

Configure DISTNx as GP2YA21. Configure the mindsensors DISTNx sensor as GP2YA21. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA21.nxc](#).

**6.22.2.20 int DISTNxMaxDistance (const byte port, const byte i2caddr)  
[inline]**

Read DISTNx maximum distance value. Read the mindsensors DISTNx sensor's maximum distance value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The maximum distance value.

**Examples:**

[ex\\_DISTNxMaxDistance.nxc](#).

**6.22.2.21 int DISTNxMinDistance (const byte port, const byte i2caddr)  
[inline]**

Read DISTNx minimum distance value. Read the mindsensors DISTNx sensor's minimum distance value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

[ex\\_DISTNxMinDistance.nxc](#).

**6.22.2.22 byte DISTNxModuleType (const byte *port*, const byte *i2caddr*)  
[inline]**

Read DISTNx module type value. Read the mindsensors DISTNx sensor's module type value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The module type value.

**Examples:**

[ex\\_DISTNxModuleType.nxc](#).

**6.22.2.23 byte DISTNxNumPoints (const byte *port*, const byte *i2caddr*)  
[inline]**

Read DISTNx num points value. Read the mindsensors DISTNx sensor's num points value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The num points value.

**Examples:**

[ex\\_DISTNxNumPoints.nxc](#).

**6.22.2.24 int DISTNxVoltage (const byte *port*, const byte *i2caddr*) [inline]**

Read DISTNx voltage value. Read the mindsensors DISTNx sensor's voltage value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The voltage value.

**Examples:**

[ex\\_DISTNxVoltage.nxc](#).

**6.22.2.25 char MSADPAOff (const byte *port*, const byte *i2caddr*) [inline]**

Turn off mindsensors ADPA mode. Turn ADPA mode off for the mindsensors device on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSADPAOff.nxc](#).

**6.22.2.26 char MSADPAOn (const byte *port*, const byte *i2caddr*) [inline]**

Turn on mindsensors ADPA mode. Turn ADPA mode on for the mindsensors device on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSADPAOn.nxc](#).

**6.22.2.27 char MSDeenergize (const byte port, const byte i2caddr) [inline]**

Turn off power to device. Turn power off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSDeenergize.nxc](#).

**6.22.2.28 char MSEnergize (const byte port, const byte i2caddr) [inline]**

Turn on power to device. Turn the power on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSEnergize.nxc](#).

**6.22.2.29 char MSIRTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [inline]**

MSIRTrain function. Control an IR Train receiver set to the specified channel using the mindsensors NRLink device. Valid function values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channels are [TRAIN\\_CHANNEL\\_1](#) through [TRAIN\\_CHANNEL\\_3](#) and [TRAIN\\_CHANNEL\\_ALL](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The IR Train channel. See [IR Train channel constants](#).

*func* The IR Train function. See [PF/IR Train function constants](#)

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSIRTrain.nxc](#).

**6.22.2.30 char MSPFComboDirect (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

MSPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the mindsensors NRLink device. Commands for outa and outb are [PF\\_CMD\\_STOP](#), [PF\\_CMD\\_REV](#), [PF\\_CMD\\_FWD](#), and [PF\\_CMD\\_BRAKE](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- outa* The Power Function command for output A. See [Power Function command constants](#).
- outb* The Power Function command for output B. See [Power Function command constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFComboDirect.nxc](#).

**6.22.2.31 char MSPFComboPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

MSPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Valid output values are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#), [PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- outa* The Power Function PWM command for output A. See [Power Function PWM option constants](#).
- outb* The Power Function PWM command for output B. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFComboPWM.nxc](#).

**6.22.2.32** `char MSPFRawOutput (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2) [inline]`

MSPFRawOutput function. Control a Power Function receiver set to the specified channel using the mindsensors NRLink device. Build the raw data stream using the 3 nibbles (4 bit values). The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*nibble0* The first raw data nibble.

*nibble1* The second raw data nibble.

*nibble2* The third raw data nibble.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFRawOutput.nxc](#).

**6.22.2.33** `char MSPFRepeat (const byte port, const byte i2caddr, const byte count, const unsigned int delay) [inline]`

MSPFRepeat function. Repeat sending the last Power Function command using the mindsensors NRLink device. Specify the number of times to repeat the command and the number of milliseconds of delay between each repetition. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*count* The number of times to repeat the command.

*delay* The number of milliseconds to delay between each repetition.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFRepeat.nxc](#).

**6.22.2.34 char MSPFSingleOutputCST (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

MSPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_CST\\_CLEAR1\\_CLEAR2](#), [PF\\_CST\\_SET1\\_CLEAR2](#), [PF\\_CST\\_CLEAR1\\_SET2](#), [PF\\_CST\\_SET1\\_SET2](#), [PF\\_CST\\_INCREMENT\\_PWM](#), [PF\\_CST\\_DECREMENT\\_PWM](#), [PF\\_CST\\_FULL\\_FWD](#), [PF\\_CST\\_FULL\\_REV](#), and [PF\\_CST\\_TOGGLE\\_DIR](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The Power Function channel. See [Power Function channel constants](#).

*out* The Power Function output. See [Power Function output constants](#).

*func* The Power Function CST function. See [Power Function CST options constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFSingleOutputCST.nxc](#).

**6.22.2.35 char MSPFSingleOutputPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

MSPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions

are `PF_PWM_FLOAT`, `PF_PWM_FWD1`, `PF_PWM_FWD2`, `PF_PWM_FWD3`, `PF_PWM_FWD4`, `PF_PWM_FWD5`, `PF_PWM_FWD6`, `PF_PWM_FWD7`, `PF_PWM_BRAKE`, `PF_PWM_REV7`, `PF_PWM_REV6`, `PF_PWM_REV5`, `PF_PWM_REV4`, `PF_PWM_REV3`, `PF_PWM_REV2`, and `PF_PWM_REV1`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- func* The Power Function PWM function. See [Power Function PWM option constants](#).

**Returns:**

The function call result. `NO_ERR` or [Communications specific errors](#).

**Examples:**

`ex_MSPFSingleOutputPWM.nxc`.

**6.22.2.36 char MSPFSinglePin (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *pin*, const byte *func*, bool *cont*) [inline]**

MSPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the mindsensors NLink device. Select the desired output using `PF_OUT_A` or `PF_OUT_B`. Select the desired pin using `PF_PIN_C1` or `PF_PIN_C2`. Valid functions are `PF_FUNC_NOCHANGE`, `PF_FUNC_CLEAR`, `PF_FUNC_SET`, and `PF_FUNC_TOGGLE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. Specify whether the mode by passing true (continuous) or false (timeout) as the final parameter. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).

- out* The Power Function output. See [Power Function output constants](#).
- pin* The Power Function pin. See [Power Function pin constants](#).
- func* The Power Function single pin function. See [Power Function single pin function constants](#).
- cont* Control whether the mode is continuous or timeout.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFSinglePin.nxc](#).

**6.22.2.37 char MSPFTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [[inline](#)]**

MSPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device as if it were an IR Train receiver. Valid function values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channels are PF\_CHANNEL\_1 through PF\_CHANNEL\_4. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- func* The Power Function train function. See [PF/IR Train function constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFTrain.nxc](#).

**6.22.2.38** void MSRCXAbsVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [**inline**]

MSRCXAbsVar function. Send the AbsVar command to an RCX.

**Parameters:**

- varnum* The variable number to change.
- src* The RCX source. See [RCX and Scout source constants](#).
- value* The RCX value.

**Examples:**

[ex\\_MSRCXAbsVar.nxc](#).

**6.22.2.39** void MSRCXAddToDatalog (const byte *src*, const unsigned int *value*) [**inline**]

MSRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

**Parameters:**

- src* The RCX source. See [RCX and Scout source constants](#).
- value* The RCX value.

**Examples:**

[ex\\_MSRCXAddToDatalog.nxc](#).

**6.22.2.40** void MSRCXAndVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [**inline**]

MSRCXAndVar function. Send the AndVar command to an RCX.

**Parameters:**

- varnum* The variable number to change.
- src* The RCX source. See [RCX and Scout source constants](#).
- value* The RCX value.

**Examples:**

[ex\\_MSRCXAndVar.nxc](#).

**6.22.2.41 int MSRCXBatteryLevel (void) [inline]**

MSRCXBatteryLevel function. Send the BatteryLevel command to an RCX to read the current battery level.

**Returns:**

The RCX battery level.

**Examples:**

[ex\\_MSRCXBatteryLevel.nxc](#).

**6.22.2.42 void MSRCXBoot (void) [inline]**

MSRCXBoot function. Send the Boot command to an RCX.

**Examples:**

[ex\\_MSRCXBoot.nxc](#).

**6.22.2.43 void MSRCXCalibrateEvent (const byte *evt*, const byte *low*, const byte *hi*, const byte *hyst*) [inline]**

MSRCXCalibrateEvent function. Send the CalibrateEvent command to an RCX.

**Parameters:**

- evt* The event number.
- low* The low threshold.
- hi* The high threshold.
- hyst* The hysteresis value.

**Examples:**

[ex\\_MSRCXCalibrateEvent.nxc](#).

**6.22.2.44 void MSRCXClearAllEvents (void) [inline]**

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

**Examples:**

[ex\\_MSRCXClearAllEvents.nxc](#).

**6.22.2.45 void MSRCXClearCounter (const byte counter) [inline]**

MSRCXClearCounter function. Send the ClearCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

[ex\\_MSRCXClearCounter.nxc](#).

**6.22.2.46 void MSRCXClearMsg (void) [inline]**

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

[ex\\_MSRCXClearMsg.nxc](#).

**6.22.2.47 void MSRCXClearSensor (const byte port) [inline]**

MSRCXClearSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

[ex\\_MSRCXClearSensor.nxc](#).

**6.22.2.48 void MSRCXClearSound (void) [inline]**

MSRCXClearSound function. Send the ClearSound command to an RCX.

**Examples:**

[ex\\_MSRCXClearSound.nxc](#).

**6.22.2.49 void MSRCXClearTimer (const byte *timer*) [inline]**

MSRCXClearTimer function. Send the ClearTimer command to an RCX.

**Parameters:**

*timer* The timer to clear.

**Examples:**

[ex\\_MSRCXClearTimer.nxc](#).

**6.22.2.50 void MSRCXCreateDatalog (const unsigned int *size*) [inline]**

MSRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

**Parameters:**

*size* The new datalog size.

**Examples:**

[ex\\_MSRCXCreateDatalog.nxc](#).

**6.22.2.51 void MSRCXDecCounter (const byte *counter*) [inline]**

MSRCXDecCounter function. Send the DecCounter command to an RCX.

**Parameters:**

*counter* The counter to decrement.

**Examples:**

[ex\\_MSRCXDecCounter.nxc](#).

**6.22.2.52 void MSRCXDeleteSub (const byte *s*) [inline]**

MSRCXDeleteSub function. Send the DeleteSub command to an RCX.

**Parameters:**

*s* The subroutine number to delete.

**Examples:**

[ex\\_MSRCXDeleteSub.nxc](#).

**6.22.2.53 void MSRCXDeleteSubs (void) [inline]**

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

[ex\\_MSRCXDeleteSubs.nxc](#).

**6.22.2.54 void MSRCXDeleteTask (const byte *t*) [inline]**

MSRCXDeleteTask function. Send the DeleteTask command to an RCX.

**Parameters:**

*t* The task number to delete.

**Examples:**

[ex\\_MSRCXDeleteTask.nxc](#).

**6.22.2.55 void MSRCXDeleteTasks (void) [inline]**

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

[ex\\_MSRCXDeleteTasks.nxc](#).

**6.22.2.56 void MSRCXDisableOutput (const byte *outputs*) [inline]**

MSRCXDisableOutput function. Send the DisableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to disable. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXDisableOutput.nxc](#).

**6.22.2.57 void MSRCXDivVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXDivVar function. Send the DivVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXDivVar.nxc](#).

**6.22.2.58 void MSRCXEnableOutput (const byte *outputs*) [inline]**

MSRCXEnableOutput function. Send the EnableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to enable. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXEnableOutput.nxc](#).

**6.22.2.59 void MSRCXEvent (const byte *src*, const unsigned int *value*)  
[inline]**

MSRCXEvent function. Send the Event command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXEvent.nxc](#).

**6.22.2.60 void MSRCXFloat (const byte *outputs*) [inline]**

MSRCXFloat function. Send commands to an RCX to float the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to float. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXFloat.nxc](#).

**6.22.2.61 void MSRCXFwd (const byte *outputs*) [inline]**

MSRCXFwd function. Send commands to an RCX to set the specified outputs to the forward direction.

**Parameters:**

*outputs* The RCX output(s) to set forward. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXFwd.nxc](#).

**6.22.2.62 void MSRCXIncCounter (const byte counter) [inline]**

MSRCXIncCounter function. Send the IncCounter command to an RCX.

**Parameters:**

*counter* The counter to increment.

**Examples:**

[ex\\_MSRCXIncCounter.nxc](#).

**6.22.2.63 void MSRCXInvertOutput (const byte outputs) [inline]**

MSRCXInvertOutput function. Send the InvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to invert. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXInvertOutput.nxc](#).

**6.22.2.64 void MSRCXMulVar (const byte varnum, const byte src, unsigned int value) [inline]**

MSRCXMulVar function. Send the MulVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXMulVar.nxc](#).

**6.22.2.65 void MSRCXMuteSound (void) [inline]**

MSRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

[ex\\_MSRCXMuteSound.nxc](#).

**6.22.2.66 void MSRCXObvertOutput (const byte outputs) [inline]**

MSRCXObvertOutput function. Send the ObvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to obvert. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXObvertOutput.nxc](#).

**6.22.2.67 void MSRCXOff (const byte outputs) [inline]**

MSRCXOff function. Send commands to an RCX to turn off the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn off. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOff.nxc](#).

**6.22.2.68 void MSRCXOn (const byte *outputs*) [inline]**

MSRCXOn function. Send commands to an RCX to turn on the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOn.nxc](#).

**6.22.2.69 void MSRCXOnFor (const byte *outputs*, const unsigned int *ms*) [inline]**

MSRCXOnFor function. Send commands to an RCX to turn on the specified outputs in the forward direction for the specified duration.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

*ms* The number of milliseconds to leave the outputs on

**Examples:**

[ex\\_MSRCXOnFor.nxc](#).

**6.22.2.70 void MSRCXOnFwd (const byte *outputs*) [inline]**

MSRCXOnFwd function. Send commands to an RCX to turn on the specified outputs in the forward direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the forward direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOnFwd.nxc](#).

**6.22.2.71 void MSRCXOnRev (const byte *outputs*) [inline]**

MSRCXOnRev function. Send commands to an RCX to turn on the specified outputs in the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOnRev.nxc](#).

**6.22.2.72 void MSRCXOrVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXOrVar function. Send the OrVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXOrVar.nxc](#).

**6.22.2.73 void MSRCXPBTurnOff (void) [inline]**

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

[ex\\_MSRCXPBTurnOff.nxc](#).

**6.22.2.74 void MSRCXPing (void) [inline]**

MSRCXPing function. Send the Ping command to an RCX.

**Examples:**

[ex\\_MSRCXPing.nxc](#).

**6.22.2.75 void MSRCXPlaySound (const byte *snd*) [inline]**

MSRCXPlaySound function. Send the PlaySound command to an RCX.

**Parameters:**

*snd* The sound number to play.

**Examples:**

[ex\\_MSRCXPlaySound.nxc](#).

**6.22.2.76 void MSRCXPlayTone (const unsigned int *freq*, const byte *duration*) [inline]**

MSRCXPlayTone function. Send the PlayTone command to an RCX.

**Parameters:**

*freq* The frequency of the tone to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_MSRCXPlayTone.nxc](#).

**6.22.2.77 void MSRCXPlayToneVar (const byte *varnum*, const byte *duration*) [inline]**

MSRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

**Parameters:**

*varnum* The variable containing the tone frequency to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_MSRCXPlayToneVar.nxc](#).

**6.22.2.78 int MSRCXPoll (const byte *src*, const byte *value*) [inline]**

MSRCXPoll function. Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Returns:**

The value read from the specified port and value.

**Examples:**

[ex\\_MSRCXPoll.nxc](#).

**6.22.2.79 int MSRCXPollMemory (const unsigned int *address*) [inline]**

MSRCXPollMemory function. Send the PollMemory command to an RCX.

**Parameters:**

*address* The RCX memory address.

**Returns:**

The value read from the specified address.

**Examples:**

[ex\\_MSRCXPollMemory.nxc](#).

**6.22.2.80 void MSRCXRemote (unsigned int *cmd*) [inline]**

MSRCXRemote function. Send the Remote command to an RCX.

**Parameters:**

*cmd* The RCX IR remote command to send. See [RCX IR remote constants](#).

**Examples:**

[ex\\_MSRCXRemote.nxc](#).

**6.22.2.81 void MSRCXReset (void) [inline]**

MSRCXReset function. Send the Reset command to an RCX.

**Examples:**

[ex\\_MSRCXReset.nxc](#).

**6.22.2.82 void MSRCXRev (const byte *outputs*) [inline]**

MSRCXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXRev.nxc](#).

**6.22.2.83 void MSRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]**

MSRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSelectDisplay.nxc](#).

**6.22.2.84 void MSRCXSelectProgram (const byte *prog*) [inline]**

MSRCXSelectProgram function. Send the SelectProgram command to an RCX.

**Parameters:**

*prog* The program number to select.

**Examples:**

[ex\\_MSRCXSelectProgram.nxc](#).

**6.22.2.85 void MSRCXSendSerial (const byte *first*, const byte *count*) [inline]**

MSRCXSendSerial function. Send the SendSerial command to an RCX.

**Parameters:**

*first* The first byte address.

*count* The number of bytes to send.

**Examples:**

[ex\\_MSRCXSendSerial.nxc](#).

**6.22.2.86 void MSRCXSet (const byte *dstsrc*, const byte *dstval*, const byte *src*, unsigned int *value*) [inline]**

MSRCXSet function. Send the Set command to an RCX.

**Parameters:**

*dstsrc* The RCX destination source. See [RCX and Scout source constants](#).

*dstval* The RCX destination value.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSet.nxc](#).

**6.22.2.87 void MSRCXSetDirection (const byte *outputs*, const byte *dir*)  
[inline]**

MSRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_MSRCXSetDirection.nxc](#).

**6.22.2.88 void MSRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)  
[inline]**

MSRCXSetEvent function. Send the SetEvent command to an RCX.

**Parameters:**

*evt* The event number to set.

*src* The RCX source. See [RCX and Scout source constants](#).

*type* The event type.

**Examples:**

[ex\\_MSRCXSetEvent.nxc](#).

**6.22.2.89** void MSRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)  
[inline]

MSRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_MSRCXSetGlobalDirection.nxc](#).

**6.22.2.90** void MSRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)  
[inline]

MSRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_MSRCXSetGlobalOutput.nxc](#).

**6.22.2.91** void MSRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*,  
const byte *pwrval*) [inline]

MSRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set max power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_MSRCXSetMaxPower.nxc](#).

**6.22.2.92 void MSRCXSetMessage (const byte *msg*) [inline]**

MSRCXSetMessage function. Send the SetMessage command to an RCX.

**Parameters:**

*msg* The numeric message to send.

**Examples:**

[ex\\_MSRCXSetMessage.nxc](#).

**6.22.2.93 void MSRCXSetNRLinkPort (const byte *port*, const byte *i2caddr*) [inline]**

MSRCXSetNRLinkPort function. Set the global port in advance of using the MSRCX\* and MSScout\* API functions for sending RCX and Scout messages over the mindsensors NRLink device. The port must be configured as a Low-speed port before using any of the mindsensors RCX and Scout NRLink functions.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Examples:**

[ex\\_MSRCXSetNRLinkPort.nxc](#).

**6.22.2.94 void MSRCXSetOutput (const byte *outputs*, const byte *mode*) [inline]**

MSRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

**Parameters:**

*outputs* The RCX output(s) to set mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_MSRCXSetOutput.nxc](#).

**6.22.2.95 void MSRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]**

MSRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_MSRCXSetPower.nxc](#).

**6.22.2.96 void MSRCXSetPriority (const byte *p*) [inline]**

MSRCXSetPriority function. Send the SetPriority command to an RCX.

**Parameters:**

*p* The new task priority.

**Examples:**

[ex\\_MSRCXSetPriority.nxc](#).

**6.22.2.97** `void MSRCXSetSensorMode (const byte port, const byte mode)`  
`[inline]`

MSRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

**Parameters:**

*port* The RCX sensor port.  
*mode* The RCX sensor mode.

**Examples:**

[ex\\_MSRCXSetSensorMode.nxc](#).

**6.22.2.98** `void MSRCXSetSensorType (const byte port, const byte type)`  
`[inline]`

MSRCXSetSensorType function. Send the SetSensorType command to an RCX.

**Parameters:**

*port* The RCX sensor port.  
*type* The RCX sensor type.

**Examples:**

[ex\\_MSRCXSetSensorType.nxc](#).

**6.22.2.99** `void MSRCXSetSleepTime (const byte t)` `[inline]`

MSRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

**Parameters:**

*t* The new sleep time value.

**Examples:**

[ex\\_MSRCXSetSleepTime.nxc](#).

**6.22.2.100 void MSRCXSetTxPower (const byte *pwr*) [inline]**

MSRCXSetTxPower function. Send the SetTxPower command to an RCX.

**Parameters:**

*pwr* The IR transmit power level.

**Examples:**

[ex\\_MSRCXSetTxPower.nxc](#).

**6.22.2.101 void MSRCXSetUserDisplay (const byte *src*, const unsigned int *value*, const byte *precision*) [inline]**

MSRCXSetUserDisplay function. Send the SetUserDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

*precision* The number of digits of precision.

**Examples:**

[ex\\_MSRCXSetUserDisplay.nxc](#).

**6.22.2.102 void MSRCXSetVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXSetVar function. Send the SetVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSetVar.nxc](#).

**6.22.2.103** void MSRCXSetWatch (const byte *hours*, const byte *minutes*)  
[inline]

MSRCXSetWatch function. Send the SetWatch command to an RCX.

**Parameters:**

*hours* The new watch time hours value.

*minutes* The new watch time minutes value.

**Examples:**

[ex\\_MSRCXSetWatch.nxc](#).

**6.22.2.104** void MSRCXSgnVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSgnVar function. Send the SgnVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSgnVar.nxc](#).

**6.22.2.105** void MSRCXStartTask (const byte *t*) [inline]

MSRCXStartTask function. Send the StartTask command to an RCX.

**Parameters:**

*t* The task number to start.

**Examples:**

[ex\\_MSRCXStartTask.nxc](#).

**6.22.2.106 void MSRCXStopAllTasks (void) [inline]**

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

[ex\\_MSRCXStopAllTasks.nxc](#).

**6.22.2.107 void MSRCXStopTask (const byte *t*) [inline]**

MSRCXStopTask function. Send the StopTask command to an RCX.

**Parameters:**

*t* The task number to stop.

**Examples:**

[ex\\_MSRCXStopTask.nxc](#).

**6.22.2.108 void MSRCXSubVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXSubVar function. Send the SubVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSubVar.nxc](#).

**6.22.2.109 void MSRCXSumVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXSumVar function. Send the SumVar command to an RCX.

**Parameters:**

- varnum* The variable number to change.
- src* The RCX source. See [RCX and Scout source constants](#).
- value* The RCX value.

**Examples:**

[ex\\_MSRCXSumVar.nxc](#).

**6.22.2.110 void MSRCXToggle (const byte *outputs*) [inline]**

MSRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

**Parameters:**

- outputs* The RCX output(s) to toggle. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXToggle.nxc](#).

**6.22.2.111 void MSRCXUnlock (void) [inline]**

MSRCXUnlock function. Send the Unlock command to an RCX.

**Examples:**

[ex\\_MSRCXUnlock.nxc](#).

**6.22.2.112 void MSRCXUnmuteSound (void) [inline]**

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

[ex\\_MSRCXUnmuteSound.nxc](#).

**6.22.2.113 int MSReadValue (const byte *port*, const byte *i2caddr*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one, two, or four byte value from a mind-sensors sensor. The value must be stored with the least significant byte (LSB) first (i.e., little endian). Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*reg* The device register to read.

*numbytes* The number of bytes to read. Only 1, 2 or 4 byte values are supported.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSReadValue.nxc](#).

**6.22.2.114 void MSScoutCalibrateSensor (void) [inline]**

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

[ex\\_MSScoutCalibrateSensor.nxc](#).

**6.22.2.115 void MSScoutMuteSound (void) [inline]**

MSScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

[ex\\_MSScoutMuteSound.nxc](#).

**6.22.2.116 void MSScoutSelectSounds (const byte *grp*) [inline]**

MSScoutSelectSounds function. Send the SelectSounds command to a Scout.

**Parameters:**

*grp* The Scout sound group to select.

**Examples:**

[ex\\_MSScoutSelectSounds.nxc](#).

**6.22.2.117 void MSScoutSendVLL (const byte *src*, const unsigned int *value*) [inline]**

MSScoutSendVLL function. Send the SendVLL command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSendVLL.nxc](#).

**6.22.2.118 void MSScoutSetCounterLimit (const byte *ctr*, const byte *src*, const unsigned int *value*) [inline]**

MSScoutSetCounterLimit function. Send the SetCounterLimit command to a Scout.

**Parameters:**

*ctr* The counter for which to set the limit.

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetCounterLimit.nxc](#).

**6.22.2.119 void MSScoutSetEventFeedback (const byte *src*, const unsigned int *value*) [inline]**

MSScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetEventFeedback.nxc](#).

**6.22.2.120 void MSScoutSetLight (const byte *x*) [inline]**

MSScoutSetLight function. Send the SetLight command to a Scout.

**Parameters:**

*x* Set the light on or off using this value. See [Scout light constants](#).

**Examples:**

[ex\\_MSScoutSetLight.nxc](#).

**6.22.2.121 void MSScoutSetScoutMode (const byte *mode*) [inline]**

MSScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

**Parameters:**

*mode* Set the scout mode. See [Scout mode constants](#).

**Examples:**

[ex\\_MSScoutSetScoutMode.nxc](#).

**6.22.2.122** void MSScoutSetScoutRules (const byte *m*, const byte *t*, const byte *l*, const byte *tm*, const byte *fx*) [**inline**]

MSScoutSetScoutRules function. Send the SetScoutRules command to a Scout.

**Parameters:**

- m* Scout motion rule. See [Scout motion rule constants](#).
- t* Scout touch rule. See [Scout touch rule constants](#).
- l* Scout light rule. See [Scout light rule constants](#).
- tm* Scout transmit rule. See [Scout transmit rule constants](#).
- fx* Scout special effects rule. See [Scout special effect constants](#).

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**6.22.2.123** void MSScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

**Parameters:**

- src* The Scout source. See [RCX and Scout source constants](#).
- value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorClickTime.nxc](#).

**6.22.2.124** void MSScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorHysteresis.nxc](#).

**6.22.2.125 void MSScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]**

MSScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorLowerLimit.nxc](#).

**6.22.2.126 void MSScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]**

MSScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorUpperLimit.nxc](#).

**6.22.2.127** void MSScoutSetTimerLimit (const byte *tmr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetTimerLimit function. Send the SetTimerLimit command to a Scout.

**Parameters:**

- tmr* The timer for which to set a limit.
- src* The Scout source. See [RCX and Scout source constants](#).
- value* The Scout value.

**Examples:**

[ex\\_MSScoutSetTimerLimit.nxc](#).

**6.22.2.128** void MSScoutUnmuteSound (void) [inline]

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

[ex\\_MSScoutUnmuteSound.nxc](#).

**6.22.2.129** char NRLink2400 (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in 2400 baud mode. Configure the mindsensors NRLink device in 2400 baud mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400.nxc](#).

**6.22.2.130 char NRLink4800 (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in 4800 baud mode. Configure the mindsensors NRLink device in 4800 baud mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800.nxc](#).

**6.22.2.131 char NRLinkFlush (const byte *port*, const byte *i2caddr*) [inline]**

Flush NRLink buffers. Flush the mindsensors NRLink device buffers. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkFlush.nxc](#).

**6.22.2.132 char NRLinkIRLong (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in IR long mode. Configure the mindsensors NRLink device in IR long mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkIRLong.nxc](#).

**6.22.2.133 char NRLinkIRShort (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in IR short mode. Configure the mindsensors NRLink device in IR short mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkIRShort.nxc](#).

**6.22.2.134 char NRLinkSetPF (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in power function mode. Configure the mindsensors NRLink device in power function mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetPF.nxc](#).

**6.22.2.135 char NRLinkSetRCX (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in RCX mode. Configure the mindsensors NRLink device in RCX mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetRCX.nxc](#).

**6.22.2.136 char NRLinkSetTrain (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in IR train mode. Configure the mindsensors NRLink device in IR train mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetTrain.nxc](#).

**6.22.2.137** `byte NRLinkStatus (const byte port, const byte i2caddr)`  
`[inline]`

Read NRLink status. Read the status of the mindsensors NRLink device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors NRLink status.

**Examples:**

[ex\\_NRLinkStatus.nxc](#).

**6.22.2.138** `char NRLinkTxRaw (const byte port, const byte i2caddr)`  
`[inline]`

Configure NRLink in raw IR transmit mode. Configure the mindsensors NRLink device in raw IR transmit mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkTxRaw.nxc](#).

**6.22.2.139** `char NXTHIDAsciiMode (const byte &port, const byte &i2caddr)`  
`[inline]`

Set NXTHID into ASCII data mode. Set the NXTHID device into ASCII data mode. Only printable characters can be transmitted in this mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**6.22.2.140 char NXTHIDDirectMode (const byte & port, const byte & i2caddr) [inline]**

Set NXTHID into direct data mode. Set the NXTHID device into direct data mode. Any character can be transmitted while in this mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**6.22.2.141 char NXTHIDLoadCharacter (const byte & port, const byte & i2caddr, const byte & modifier, const byte & character) [inline]**

Load NXTHID character. Load a character into the NXTHID device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- modifier* The key modifier. See the [MindSensors NXTTHID modifier keys](#) group.
- character* The character.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTTHID.nxc](#).

**6.22.2.142 char NXTTHIDTransmit (const byte & port, const byte & i2caddr) [inline]**

Transmit NXTTHID character. Transmit a single character to a computer using the NXTTHID device. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTTHID.nxc](#).

**6.22.2.143 char NXTLineLeaderAverage (const byte & port, const byte & i2caddr) [inline]**

Read NXTLineLeader average. Read the mindsensors NXTLineLeader device's average value. The average is a weighted average of the bits set to 1 based on the position.

The left most bit has a weight of 10, second bit has a weight of 20, and so forth. When all 8 sensors are over a black surface the average will be 45. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader average value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.144 char NXTLineLeaderCalibrateBlack (const byte & port, const byte & i2caddr) [inline]**

Calibrate NXTLineLeader black color. Store calibration data for the black color. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.145 char NXTLineLeaderCalibrateWhite (const byte & port, const byte & i2caddr) [inline]**

Calibrate NXTLineLeader white color. Store calibration data for the white color. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.146 char NXTLineLeaderInvert (const byte & port, const byte & i2caddr) [inline]**

Invert NXTLineLeader colors. Invert color sensing so that the device can detect a white line on a black background. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.147 char NXTLineLeaderPowerDown (const byte & port, const byte & i2caddr) [inline]**

Powerdown NXTLineLeader device. Put the NXTLineLeader to sleep so that it does not consume power when it is not required. The device wakes up on its own when any I2C communication happens or you can specifically wake it up by using the [NXTLineLeaderPowerUp](#) command. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.148 char NXTLineLeaderPowerUp (const byte & port, const byte & i2caddr) [inline]**

Powerup NXTLineLeader device. Wake up the NXTLineLeader device so that it can be used. The device can be put to sleep using the [NXTLineLeaderPowerDown](#) command. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.149 char NXTLineLeaderReset (const byte & port, const byte & i2caddr) [inline]**

Reset NXTLineLeader color inversion. Reset the NXTLineLeader color detection back to its default state (black line on a white background). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.150 byte NXTLineLeaderResult (const byte & port, const byte & i2caddr) [inline]**

Read NXTLineLeader result. Read the mindsensors NXTLineLeader device's result value. This is a single byte showing the 8 sensor's readings. Each bit corresponding to the sensor where the line is seen is set to 1, otherwise it is set to 0. When all 8 sensors are over a black surface the result will be 255 (b11111111). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader result value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.151 char NXTLineLeaderSnapshot (const byte & port, const byte & i2caddr) [inline]**

Take NXTLineLeader line snapshot. Takes a snapshot of the line under the sensor and tracks that position in subsequent tracking operations. This function also will set color inversion if it sees a white line on a black background. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.152 char NXTLineLeaderSteering (const byte & port, const byte & i2caddr) [inline]**

Read NXTLineLeader steering. Read the mindsensors NXTLineLeader device's steering value. This is the power returned by the sensor to correct your course. Add this value to your left motor and subtract it from your right motor. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader steering value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.153 int NXTPowerMeterCapacityUsed (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter capacity used. Read the mindsensors NXTPowerMeter device's capacity used since the last reset command. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter capacity used value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.154 long NXTPowerMeterElapsedTime (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter elapsed time. Read the mindsensors NXTPowerMeter device's elapsed time since the last reset command. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter elapsed time value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.155 int NXTPowerMeterErrorCount (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter error count. Read the mindsensors NXTPowerMeter device's error count value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter error count value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.156 int NXTPowerMeterMaxCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter maximum current. Read the mindsensors NXTPowerMeter device's maximum current value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter maximum current value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.157 int NXTPowerMeterMaxVoltage (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter maximum voltage. Read the mindsensors NXTPowerMeter device's maximum voltage value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter maximum voltage value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.158 int NXTPowerMeterMinCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter minimum current. Read the mindsensors NXTPowerMeter device's minimum current value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter minimum current value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.159 int NXTPowerMeterMinVoltage (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter minimum voltage. Read the mindsensors NXTPowerMeter device's minimum voltage value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter minimum voltage value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.160 int NXTPowerMeterPresentCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter present current. Read the mindsensors NXTPowerMeter device's present current value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present current.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.161 int NXTPowerMeterPresentPower (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter present power. Read the mindsensors NXTPowerMeter device's present power value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present power value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.162** `int NXTPowerMeterPresentVoltage (const byte & port, const byte & i2caddr) [inline]`

Read NXTPowerMeter present voltage. Read the mindsensors NXTPowerMeter device's present voltage value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present voltage.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.163** `char NXTPowerMeterResetCounters (const byte & port, const byte & i2caddr) [inline]`

Reset NXTPowerMeter counters. Reset the NXTPowerMeter counters back to zero. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.164 long NXTPowerMeterTotalPowerConsumed (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter total power consumed. Read the mindsensors NXTPowerMeter device's total power consumed since the last reset command. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter total power consumed value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.22.2.165 byte NXTServoBatteryVoltage (const byte & port, const byte & i2caddr) [inline]**

Read NXTServo battery voltage value. Read the mindsensors NXTServo device's battery voltage value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The battery level.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.166** `char NXTServoEditMacro (const byte & port, const byte & i2caddr) [inline]`

Edit NXTServo macro. Put the NXTServo device into macro edit mode. This operation changes the I2C address of the device to 0x40. Macros are written to EEPROM addresses between 0x21 and 0xFF. Use [NXTServoQuitEdit](#) to return the device to its normal operation mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.167** `char NXTServoGotoMacroAddress (const byte & port, const byte & i2caddr, const byte & macro) [inline]`

Goto NXTServo macro address. Run the macro found at the specified EEPROM macro address. This command re-initializes the macro environment. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*macro* The EEPROM macro address.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.168 char NXTServoHaltMacro (const byte & port, const byte & i2caddr) [inline]**

Halt NXTServo macro. Halt a macro executing on the NXTServo device. This command re-initializes the macro environment. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.169 char NXTServoInit (const byte & port, const byte & i2caddr, const byte servo) [inline]**

Initialize NXTServo servo properties. Store the initial speed and position properties of the servo motor 'n'. Current speed and position values of the nth servo is read from the servo speed register and servo position register and written to permanent memory. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.170** `char NXTServoPauseMacro (const byte & port, const byte & i2caddr) [inline]`

Pause NXTServo macro. Pause a macro executing on the NXTServo device. This command will pause the currently executing macro, and save the environment for subsequent resumption. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.171** `unsigned int NXTServoPosition (const byte & port, const byte & i2caddr, const byte servo) [inline]`

Read NXTServo servo position value. Read the mindsensors NXTServo device's servo position value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

The specified servo's position value.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.172 char NXTServoQuitEdit (const byte & *port*) [inline]**

Quit NXTServo macro edit mode. Stop editing NXTServo device macro EEPROM memory. Use [NXTServoEditMacro](#) to start editing a macro. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.173 char NXTServoReset (const byte & *port*, const byte & *i2caddr*) [inline]**

Reset NXTServo properties. Reset NXTServo device properties to factory defaults. Initial position = 1500. Initial speed = 0. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.174** `char NXTServoResumeMacro (const byte & port, const byte & i2caddr) [inline]`

Resume NXTServo macro. Resume a macro executing on the NXTServo device. This command resumes executing a macro where it was paused last, using the same environment. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.175** `byte NXTServoSpeed (const byte & port, const byte & i2caddr, const byte servo) [inline]`

Read NXTServo servo speed value. Read the mindsensors NXTServo device's servo speed value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

The specified servo's speed value.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.176** `bool PFMateSend (const byte & port, const byte & i2caddr, const byte & channel, const byte & motors, const byte & cmdA, const byte & spdA, const byte & cmdB, const byte & spdB) [inline]`

Send PFMate command. Send a PFMate command to the power function IR receiver. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The power function IR receiver channel. See the [PFMate channel constants](#) group.
- motors* The motor(s) to control. See the [PFMate motor constants](#) group.
- cmdA* The power function command for motor A.
- spdA* The power function speed for motor A.
- cmdB* The power function command for motor B.
- spdB* The power function speed for motor B.

**Returns:**

The function call result.

**Examples:**

[ex\\_PFMate.nxc](#).

**6.22.2.177** `bool PFMateSendRaw (const byte & port, const byte & i2caddr, const byte & channel, const byte & b1, const byte & b2) [inline]`

Send raw PFMate command. Send a raw PFMate command to the power function IR receiver. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The power function IR receiver channel. See the [PFMate channel constants](#) group.

*b1* Raw byte 1.

*b2* Raw byte 2.

**Returns:**

The function call result.

**Examples:**

[ex\\_PFMate.nxc](#).

**6.22.2.178 char PSPNxAnalog (const byte & port, const byte & i2caddr)  
[inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_PSPNxAnalog.nxc](#), and [ex\\_ReadSensorMSPlayStation.nxc](#).

**6.22.2.179 char PSPNxDigital (const byte & port, const byte & i2caddr)  
[inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_PSPNxDigital.nxc](#).

**6.22.2.180 bool ReadNRLinkBytes (const byte *port*, const byte *i2caddr*, byte & *data*[ ]) [inline]**

Read data from NRLink. Read data from the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*data* A byte array that will contain the data read from the device on output.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadNRLinkBytes.nxc](#).

**6.22.2.181 bool ReadSensorMSAccel (const byte *port*, const byte *i2caddr*, int & *x*, int & *y*, int & *z*) [inline]**

Read mindsensors acceleration values. Read X, Y, and Z axis acceleration values from the mindsensors Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*x* The output x-axis acceleration.

*y* The output y-axis acceleration.

*z* The output z-axis acceleration.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSAccel.nxc](#).

**6.22.2.182** `bool ReadSensorMSPlayStation (const byte port, const byte i2caddr, byte & btnset1, byte & btnset2, byte & xleft, byte & yleft, byte & xright, byte & yright) [inline]`

Read mindsensors playstation controller values. Read playstation controller values from the mindsensors playstation sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*btnset1* The button set 1 values. See [MindSensors PSP-Nx button set 1 constants](#).

*btnset2* The button set 2 values. See [MindSensors PSP-Nx button set 2 constants](#).

*xleft* The left joystick x value.

*yleft* The left joystick y value.

*xright* The right joystick x value.

*yright* The right joystick y value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.22.2.183** `bool ReadSensorMSRTClock (const byte port, byte & sec, byte & min, byte & hrs, byte & dow, byte & date, byte & month, byte & year) [inline]`

Read mindsensors RTClock values. Read real-time clock values from the Mindsensors RTClock sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*sec* The seconds.

*min* The minutes.

*hrs* The hours.

*dow* The day of week number.

*date* The day.

*month* The month.

*year* The year.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSRTClock.nxc](#).

**6.22.2.184** `bool ReadSensorMSTilt (const byte & port, const byte & i2caddr, byte & x, byte & y, byte & z) [inline]`

Read mindsensors tilt values. Read X, Y, and Z axis tilt values from the mindsensors tilt sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

- x* The output x-axis tilt.
- y* The output y-axis tilt.
- z* The output z-axis tilt.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSTilt.nxc](#).

**6.22.2.185 char RunNRLinkMacro (const byte *port*, const byte *i2caddr*, const byte *macro*) [inline]**

Run NRLink macro. Run the specified mindsensors NRLink device macro. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- macro* The address of the macro to execute.

**Returns:**

The function call result.

**Examples:**

[ex\\_RunNRLinkMacro.nxc](#).

**6.22.2.186 int SensorMSCompass (const byte & *port*, const byte *i2caddr*) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors compass value

**Examples:**

[ex\\_SensorMSCompass.nxc](#).

**6.22.2.187 int SensorMSDROD (const byte & port) [inline]**

Read mindsensors DROD value. Return the Mindsensors DROD sensor value.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors DROD value

**Examples:**

[ex\\_SensorMSDROD.nxc](#).

**6.22.2.188 int SensorMSPressure (const byte & port) [inline]**

Read mindsensors pressure sensor. Read the pressure sensor value of the mindsensors pressure sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The pressure reading.

**Examples:**

[ex\\_SensorMSPressure.nxc](#).

**6.22.2.189 int SensorMSPressureRaw (const byte & port) [inline]**

Read mindsensors raw pressure value. Return the Mindsensors pressure sensor raw value.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors raw pressure value

**Examples:**

[ex\\_SensorMSPressureRaw.nxc](#).

**6.22.2.190 char SensorNXTSumoEyes (const byte & port)**

Read mindsensors NXTSumoEyes obstacle zone. Return the Mindsensors NXTSumoEyes sensor obstacle zone value. The port should be configured for the NXTSumoEyes device using [SetSensorNXTSumoEyes](#) before calling this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors NXTSumoEyes obstacle zone value. See [MindSensors NXTSumoEyes constants](#).

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.22.2.191 int SensorNXTSumoEyesRaw (const byte & port) [inline]**

Read mindsensors NXTSumoEyes raw value. Return the Mindsensors NXTSumoEyes raw sensor value. The port should be configured for the NXTSumoEyes device using [SetSensorNXTSumoEyes](#) before calling this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors NXTSumoEyes raw value

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.22.2.192 char SetACCLNxSensitivity (const byte *port*, const byte *i2caddr*, byte *slevel*) [inline]**

Set ACCL-Nx sensitivity. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*slevel* The sensitivity level. See [MindSensors ACCL-Nx sensitivity level constants](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_SetACCLNxSensitivity.nxc](#).

**6.22.2.193 char SetNXTLineLeaderKdFactor (const byte & *port*, const byte & *i2caddr*, const byte & *value*) [inline]**

Write NXTLineLeader Kd factor. Write a Kd divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Kd value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Kd factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.194 char SetNXTLineLeaderKdValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kd value. Write a Kd value to the NXTLineLeader device. This value divided by PID Factor for Kd is the Derivative value for the PID control. Suggested value is 8 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Kd value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.195 char SetNXTLineLeaderKiFactor (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Ki factor. Write a Ki divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Ki value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Ki factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.196 char SetNXTLineLeaderKiValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Ki value. Write a Ki value to the NXTLineLeader device. This value divided by PID Factor for Ki is the Integral value for the PID control. Suggested value is 0 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Ki value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.197 char SetNXTLineLeaderKpFactor (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kp factor. Write a Kp divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Kp value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Kp factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.198 char SetNXTLineLeaderKpValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kp value. Write a Kp value to the NXTLineLeader device. This value divided by PID Factor for Kp is the Proportional value for the PID control. Suggested value is 25 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Kp value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.199 char SetNXTLineLeaderSetpoint (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader setpoint. Write a new setpoint value to the NXTLineLeader device. The Set Point is a value you can ask sensor to maintain the average to. The default value is 45, whereby the line is maintained in center of the sensor. If you need to maintain line towards left of the sensor, set the Set Point to a lower value (minimum: 10). If you need it to be towards on the right of the sensor, set it to higher value (maximum: 80). Set point is also useful while tracking an edge of dark and light areas. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new setpoint value (10..80).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.22.2.200 char SetNXTServoPosition (const byte & port, const byte & i2caddr, const byte servo, const byte & pos) [inline]**

Set NXTServo servo motor position. Set the position of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

*pos* The servo position. See [MindSensors NXTServo position constants](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.201** `char SetNXTServoQuickPosition (const byte & port, const byte & i2caddr, const byte servo, const byte & qpos) [inline]`

Set NXTServo servo motor quick position. Set the quick position of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

*qpos* The servo quick position. See [MindSensors NXTServo quick position constants](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.202** `char SetNXTServoSpeed (const byte & port, const byte & i2caddr, const byte servo, const byte & speed) [inline]`

Set NXTServo servo motor speed. Set the speed of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.
- speed* The servo speed. (0..255)

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.22.2.203 void SetSensorMSDROD (const byte & port, bool bActive) [inline]**

Configure a mindsensors DROD sensor. Configure the specified port for a mindsensors DROD sensor.

**Parameters:**

- port* The port to configure. See [Input port constants](#).
- bActive* A flag indicating whether to configure the sensor in active or inactive mode.

**Examples:**

[ex\\_setsensormsdrod.nxc](#).

**6.22.2.204 void SetSensorMSPressure (const byte & port) [inline]**

Configure a mindsensors pressure sensor. Configure the specified port for a mindsensors pressure sensor.

**Parameters:**

- port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_setsensormspressure.nxc](#).

**6.22.2.205** `void SetSensorNXTSumoEyes (const byte & port, bool bLong)`  
[inline]

Configure a mindsensors SumoEyes sensor. Configure the specified port for a mind-sensors SumoEyes sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bLong* A flag indicating whether to configure the sensor in long range or short range mode.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.22.2.206** `char WriteNRLinkBytes (const byte port, const byte i2caddr, const byte data[])` [inline]

Write data to NRLink. Write data to the mindsensors NRLink device on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*data* A byte array containing the data to write.

**Returns:**

The function call result.

**Examples:**

[ex\\_writenrlinkbytes.nxc](#).

## 6.23 Codatex API Functions

Functions for accessing and modifying Codatex devices.

## Modules

- [Codatex device constants](#)

*Constants that are for use with Codatex devices.*

## Functions

- bool [RFIDInit](#) (const byte &port)  
*RFIDInit function.*
- bool [RFIDMode](#) (const byte &port, const byte &mode)  
*RFIDMode function.*
- byte [RFIDStatus](#) (const byte &port)  
*RFIDStatus function.*
- bool [RFIDRead](#) (const byte &port, byte &output[ ])  
*RFIDRead function.*
- bool [RFIDStop](#) (const byte &port)  
*RFIDStop function.*
- bool [RFIDReadSingle](#) (const byte &port, byte &output[ ])  
*RFIDReadSingle function.*
- bool [RFIDReadContinuous](#) (const byte &port, byte &output[ ])  
*RFIDReadContinuous function.*

### 6.23.1 Detailed Description

Functions for accessing and modifying Codatex devices.

### 6.23.2 Function Documentation

#### 6.23.2.1 bool RFIDInit (const byte &port) [inline]

RFIDInit function. Initialize the Codatex RFID sensor.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDInit.nxc](#).

**6.23.2.2 bool RFIDMode (const byte & port, const byte & mode) [inline]**

RFIDMode function. Configure the Codatex RFID sensor mode.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*mode* The RFID sensor mode. See the [Codatex RFID sensor modes](#) group.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDMode.nxc](#).

**6.23.2.3 bool RFIDRead (const byte & port, byte & output[]) [inline]**

RFIDRead function. Read the Codatex RFID sensor value.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDRead.nxc](#).

**6.23.2.4 bool RFIDReadContinuous (const byte & port, byte & output[ ])  
[inline]**

RFIDReadContinuous function. Set the Codatex RFID sensor into continuous mode, if necessary, and read the RFID data.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDReadContinuous.nxc](#).

**6.23.2.5 bool RFIDReadSingle (const byte & port, byte & output[ ])  
[inline]**

RFIDReadSingle function. Set the Codatex RFID sensor into single mode and read the RFID data.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDReadSingle.nxc](#).

### 6.23.2.6 byte RFIDStatus (const byte & port) [inline]

RFIDStatus function. Read the Codatex RFID sensor status.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The RFID sensor status.

**Examples:**

[ex\\_RFIDStatus.nxc](#).

### 6.23.2.7 bool RFIDStop (const byte & port) [inline]

RFIDStop function. Stop the Codatex RFID sensor.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDStop.nxc](#).

## 6.24 Dexter Industries API Functions

Functions for accessing and modifying Dexter Industries devices.

**Modules**

- [Dexter Industries device constants](#)

*Constants that are for use with Dexter Industries devices.*

### Functions

- bool [SensorDIGPSStatus](#) (byte port)  
*SensorDIGPSStatus function.*
- long [SensorDIGPSTime](#) (byte port)  
*SensorDIGPSTime function.*
- long [SensorDIGPSLatitude](#) (byte port)  
*SensorDIGPSLatitude function.*
- long [SensorDIGPSLongitude](#) (byte port)  
*SensorDIGPSLongitude function.*
- long [SensorDIGPSVelocity](#) (byte port)  
*SensorDIGPSVelocity function.*
- int [SensorDIGPSHeading](#) (byte port)  
*SensorDIGPSHeading function.*
- long [SensorDIGPSDistanceToWaypoint](#) (byte port)  
*SensorDIGPSDistanceToWaypoint function.*
- int [SensorDIGPSHeadingToWaypoint](#) (byte port)  
*SensorDIGPSHeadingToWaypoint function.*
- int [SensorDIGPSRelativeHeading](#) (byte port)  
*SensorDIGPSRelativeHeading function.*
- bool [SetSensorDIGPSWaypoint](#) (byte port, long latitude, long longitude)  
*SetSensorDIGPSWaypoint function.*
- bool [SetSensorDIGyroEx](#) (const byte port, byte scale, byte odr, byte bw)  
*SetSensorDIGyroEx function.*
- bool [SetSensorDIGyro](#) (const byte port)  
*SetSensorDIGyro function.*
- bool [ReadSensorDIGyroRaw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIGyroRaw function.*
- bool [ReadSensorDIGyro](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIGyro function.*

- int [SensorDIGyroTemperature](#) (const byte port)  
*SensorDIGyroTemperature function.*
- byte [SensorDIGyroStatus](#) (const byte port)  
*SensorDIGyroStatus function.*
- bool [SetSensorDIAcclEx](#) (const byte port, byte mode)  
*SetSensorDIAcclEx function.*
- bool [SetSensorDIAccl](#) (const byte port)  
*SetSensorDIAccl function.*
- bool [ReadSensorDIAcclRaw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAcclRaw function.*
- bool [ReadSensorDIAccl](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAccl function.*
- bool [ReadSensorDIAccl8Raw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAccl8Raw function.*
- bool [ReadSensorDIAccl8](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAccl8 function.*
- byte [SensorDIAcclStatus](#) (const byte port)  
*SensorDIAcclStatus function.*
- bool [ReadSensorDIAcclDrift](#) (const byte port, int &x, int &y, int &z)  
*ReadSensorDIAcclDrift function.*
- bool [SetSensorDIAcclDrift](#) (const byte port, int x, int y, int z)  
*SetSensorDIAcclDrift function.*

### 6.24.1 Detailed Description

Functions for accessing and modifying Dexter Industries devices.

### 6.24.2 Function Documentation

#### 6.24.2.1 `bool ReadSensorDIAccl (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIAccl function. Read the scaled Dexter Industries IMU Accl X, Y, and Z axis 10-bit values.

##### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the scaled X, Y, and Z 10-bit values.

##### Returns:

The boolean function call result.

##### Examples:

[ex\\_diaccl.nxc](#).

#### 6.24.2.2 `bool ReadSensorDIAccl8 (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIAccl8 function. Read the scaled Dexter Industries IMU Accl X, Y, and Z axis 8-bit values.

##### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the scaled X, Y, and Z 8-bit values.

##### Returns:

The boolean function call result.

##### Examples:

[ex\\_diaccl.nxc](#).

### 6.24.2.3 `bool ReadSensorDIAccl8Raw (const byte port, VectorType & vector) [inline]`

ReadSensorDIAccl8Raw function. Read the raw Dexter Industries IMU Accl X, Y, and Z axis 8-bit values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the raw X, Y, and Z 8-bit values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_diaccl.nxc](#).

### 6.24.2.4 `bool ReadSensorDIAcclDrift (const byte port, int & x, int & y, int & z) [inline]`

ReadSensorDIAcclDrift function. Read the Dexter Industries IMU Accl X, Y, and Z axis 10-bit drift values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*x* The X axis 10-bit drift value.

*y* The Y axis 10-bit drift value.

*z* The Z axis 10-bit drift value.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_diaccl.nxc](#).

#### 6.24.2.5 `bool ReadSensorDIAcclRaw (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIAcclRaw function. Read the raw Dexter Industries IMU Accl X, Y, and Z axis 10-bit values.

##### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type `VectorType` which will contain the raw X, Y, and Z 10-bit values.

##### Returns:

The boolean function call result.

##### Examples:

[ex\\_diaccl.nxc](#).

#### 6.24.2.6 `bool ReadSensorDIGyro (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIGyro function. Read the scaled Dexter Industries IMU Gyro X, Y, and Z axis values.

##### Parameters:

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type `VectorType` which will contain the scaled X, Y, and Z values.

##### Returns:

The boolean function call result.

##### Examples:

[ex\\_digyro.nxc](#).

### 6.24.2.7 `bool ReadSensorDIGyroRaw (const byte port, VectorType & vector)` `[inline]`

`ReadSensorDIGyroRaw` function. Read the raw Dexter Industries IMU Gyro X, Y, and Z axis values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the raw X, Y, and Z values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_digyro.nxc](#).

### 6.24.2.8 `byte SensorDIAcclStatus (const byte port)` `[inline]`

`SensorDIAcclStatus` function. Read the Dexter Industries IMU Accl status value.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

#### Returns:

The status value.

#### Examples:

[ex\\_diaccl.nxc](#).

### 6.24.2.9 `long SensorDIGPSDistanceToWaypoint (byte port)` `[inline]`

`SensorDIGPSDistanceToWaypoint` function. Read the distance remaining to reach the current waypoint in meters.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The distance to the waypoint in meters

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.10 int SensorDIGPSHeading (byte *port*) [inline]**

SensorDIGPSHeading function. Read the current heading in degrees.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current heading in degrees

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.11 int SensorDIGPSHeadingToWaypoint (byte *port*) [inline]**

SensorDIGPSHeadingToWaypoint function. Read the heading required to reach the current waypoint.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The heading to the waypoint in degrees

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.12 long SensorDIGPSLatitude (byte *port*) [inline]**

SensorDIGPSLatitude function. Read the integer latitude reported by the GPS (ddddddd; Positive = North; Negative = South).

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The integer latitude

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.13 long SensorDIGPSLongitude (byte *port*) [inline]**

SensorDIGPSLongitude function. Read the integer longitude reported by the GPS (ddddddd; Positive = East; Negative = West).

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The integer longitude

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.14 int SensorDIGPSRelativeHeading (byte *port*) [inline]**

SensorDIGPSRelativeHeading function. Read the angle travelled since last request. Resets the request coordinates on the GPS sensor. Sends the angle of travel since the last call.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The relative heading in degrees

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.15 bool SensorDIGPSStatus (byte *port*) [inline]**

SensorDIGPSStatus function. Read the status of the GPS satellite link.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean GPS status

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.16 long SensorDIGPSTime (byte *port*) [inline]**

SensorDIGPSTime function. Read the current time reported by the GPS in UTC.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current time in UTC

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.17 long SensorDIGPSVelocity (byte *port*) [inline]**

SensorDIGPSVelocity function. Read the current velocity in cm/s.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current velocity in cm/s

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.18 byte SensorDIGyroStatus (const byte *port*) [inline]**

SensorDIGyroStatus function. Read the Dexter Industries IMU Gyro status value.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The status value.

**Examples:**

[ex\\_digyro.nxc](#).

**6.24.2.19 int SensorDIGyroTemperature (const byte *port*) [inline]**

SensorDIGyroTemperature function. Read the Dexter Industries IMU Gyro temperature value.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The temperature value.

**Examples:**

[ex\\_digyro.nxc](#).

**6.24.2.20 bool SetSensorDIAccl (const byte *port*) [inline]**

SetSensorDIAccl function. Configure DIAccl device on the specified port with default mode of 2G.

**Parameters:**

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**6.24.2.21 bool SetSensorDIAcclDrift (const byte *port*, int *x*, int *y*, int *z*) [inline]**

SetSensorDIAcclDrift function. Set the Dexter Industries IMU Accl X, Y, and Z axis 10-bit drift values.

**Parameters:**

- port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- x* The X axis 10-bit drift value.
- y* The Y axis 10-bit drift value.
- z* The Z axis 10-bit drift value.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**6.24.2.22 bool SetSensorDIAcclEx (const byte port, byte mode) [inline]**

SetSensorDIAcclEx function. Configure DIAccl device on the specified port with the specified mode.

**Parameters:**

- port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- mode* The mode of the device (2G, 4G, or 8G). See the [Dexter Industries IMU Accelerometer mode control register constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**6.24.2.23 bool SetSensorDIGPSWaypoint (byte port, long latitude, long longitude) [inline]**

SetSensorDIGPSWaypoint function. Set the coordinates of the waypoint destination. The GPS sensor uses this to calculate the heading and distance required to reach the waypoint.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*latitude* The latitude of the waypoint.

*longitude* The longitude of the waypoint.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digps.nxc](#).

**6.24.2.24 bool SetSensorDIGyro (const byte port) [inline]**

SetSensorDIGyro function. Configure DIGyro device on the specified port with default scale of 500dps, output data rate of 100hz, and bandwidth level 1.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digyro.nxc](#).

**6.24.2.25 bool SetSensorDIGyroEx (const byte port, byte scale, byte odr, byte bw) [inline]**

SetSensorDIGyroEx function. Configure DIGyro device on the specified port with the specified scale, output data rate, and bandwidth.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

- scale* The full scale of the device (250dps, 500dps, or 2000dps). See the [Dexter Industries IMU Gyro control register 4 constants](#) group. You may use a constant or a variable.
- odr* The output data rate of the device (100hz, 200hz, 400hz, or 800hz). See the [Dexter Industries IMU Gyro control register 1 constants](#) group. You may use a constant or a variable.
- bw* The bandwidth of the device. See the [Dexter Industries IMU Gyro control register 1 constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digyro.nxc](#).

## 6.25 Microinfinity API Functions

Functions for accessing and modifying Microinfinity devices.

**Modules**

- [Microinfinity types](#)  
*Types used by various Microinfinity device functions.*
- [Microinfinity functions](#)  
*Functions for interfacing with Microinfinity devices.*
- [Microinfinity device constants](#)  
*Constants that are for use with Microinfinity devices.*

### 6.25.1 Detailed Description

Functions for accessing and modifying Microinfinity devices.

## 6.26 RIC Macro Wrappers

Macro wrappers for use in defining RIC byte arrays.

**Defines**

- #define **RICSetValue**(\_data, \_idx, \_newval) \_data[(\_idx)] = (\_newval)&0xFF; \_data[(\_idx)+1] = (\_newval)>>8  
*Set the value of an element in an RIC data array.*
- #define **RICImgPoint**(\_X, \_Y) (\_X)&0xFF, (\_X)>>8, (\_Y)&0xFF, (\_Y)>>8  
*Output an RIC ImgPoint structure.*
- #define **RICImgRect**(\_Pt, \_W, \_H) \_Pt, (\_W)&0xFF, (\_W)>>8, (\_H)&0xFF, (\_H)>>8  
*Output an RIC ImgRect structure.*
- #define **RICOpDescription**(\_Options, \_Width, \_Height) 8, 0, 0, 0, (\_Options)&0xFF, (\_Options)>>8, (\_Width)&0xFF, (\_Width)>>8, (\_Height)&0xFF, (\_Height)>>8  
*Output an RIC Description opcode.*
- #define **RICOpCopyBits**(\_CopyOptions, \_DataAddr, \_SrcRect, \_DstPoint) 18, 0, 3, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, (\_DataAddr)&0xFF, (\_DataAddr)>>8, \_SrcRect, \_DstPoint  
*Output an RIC CopyBits opcode.*
- #define **RICOpPixel**(\_CopyOptions, \_Point, \_Value) 10, 0, 4, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Value)&0xFF, (\_Value)>>8  
*Output an RIC Pixel opcode.*
- #define **RICOpLine**(\_CopyOptions, \_Point1, \_Point2) 12, 0, 5, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point1, \_Point2  
*Output an RIC Line opcode.*
- #define **RICOpRect**(\_CopyOptions, \_Point, \_Width, \_Height) 12, 0, 6, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Width)&0xFF, (\_Width)>>8, (\_Height)&0xFF, (\_Height)>>8  
*Output an RIC Rect opcode.*
- #define **RICOpCircle**(\_CopyOptions, \_Point, \_Radius) 10, 0, 7, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Radius)&0xFF, (\_Radius)>>8  
*Output an RIC Circle opcode.*

- #define `RICOpNumBox`(\_CopyOptions, \_Point, \_Value) 10, 0, 8, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Value)&0xFF, (\_Value)>>8

*Output an RIC NumBox opcode.*
- #define `RICOpSprite`(\_DataAddr, \_Rows, \_BytesPerRow, \_SpriteData) ((\_Rows\*\_BytesPerRow)+((\_Rows\*\_BytesPerRow)%2)+8)&0xFF, ((\_Rows\*\_BytesPerRow)+((\_Rows\*\_BytesPerRow)%2)+8)>>8, 1, 0, (\_DataAddr)&0xFF, (\_DataAddr)>>8, (\_Rows)&0xFF, (\_Rows)>>8, (\_BytesPerRow)&0xFF, (\_BytesPerRow)>>8, \_SpriteData

*Output an RIC Sprite opcode.*
- #define `RICSpriteData`(...) \_\_VA\_ARGS\_\_

*Output RIC sprite data.*
- #define `RICOpVarMap`(\_DataAddr, \_MapCount, \_MapFunction) ((\_MapCount\*4)+6)&0xFF, ((\_MapCount\*4)+6)>>8, 2, 0, (\_DataAddr)&0xFF, (\_DataAddr)>>8, (\_MapCount)&0xFF, (\_MapCount)>>8, \_MapFunction

*Output an RIC VarMap opcode.*
- #define `RICMapElement`(\_Domain, \_Range) (\_Domain)&0xFF, (\_Domain)>>8, (\_Range)&0xFF, (\_Range)>>8

*Output an RIC map element.*
- #define `RICMapFunction`(\_MapElement,...) \_MapElement, \_\_VA\_ARGS\_\_

*Output an RIC VarMap function.*
- #define `RICArg`(\_arg) ((\_arg)|0x1000)

*Output an RIC parameterized argument.*
- #define `RICMapArg`(\_mapidx, \_arg) ((\_arg)|0x1000|(((mapidx)&0xF)<<8))

*Output an RIC parameterized and mapped argument.*
- #define `RICOpPolygon`(\_CopyOptions, \_Count, \_ThePoints) ((\_Count\*4)+6)&0xFF, ((\_Count\*4)+6)>>8, 10, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, (\_Count)&0xFF, (\_Count)>>8, \_ThePoints

*Output an RIC Polygon opcode.*
- #define `RICPolygonPoints`(\_pPoint1, \_pPoint2,...) \_pPoint1, \_pPoint2, \_\_VA\_ARGS\_\_

*Output RIC polygon points.*

- #define `RICOpEllipse(_CopyOptions, _Point, _RadiusX, _RadiusY)` 12, 0, 9, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_RadiusX)&0xFF, (\_RadiusX)>>8, (\_RadiusY)&0xFF, (\_RadiusY)>>8

*Output an RIC Ellipse opcode.*

### 6.26.1 Detailed Description

Macro wrappers for use in defining RIC byte arrays.

### 6.26.2 Define Documentation

#### 6.26.2.1 #define `RICArg(_arg)` ((\_arg)|0x1000)

Output an RIC parameterized argument.

#### Parameters:

`_arg` The argument that you want to parameterize.

#### Examples:

[ex\\_dispgaoutex.nxc](#).

#### 6.26.2.2 #define `RICImgPoint(_X, _Y)` (\_X)&0xFF, (\_X)>>8, (\_Y)&0xFF, (\_Y)>>8

Output an RIC `ImgPoint` structure.

#### Parameters:

`_X` The X coordinate.

`_Y` The Y coordinate.

#### Examples:

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**6.26.2.3** `#define RICImgRect(_Pt, _W, _H) _Pt, (_W)&0xFF, (_W)>>8,  
(_H)&0xFF, (_H)>>8`

Output an RIC `ImgRect` structure.

**Parameters:**

*\_Pt* An `ImgPoint`. See [RICImgPoint](#).

*\_W* The rectangle width.

*\_H* The rectangle height.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**6.26.2.4** `#define RICMapArg(_mapidx, _arg) ((_arg)|0x1000|(((_-  
mapidx)&0xF)<<8))`

Output an RIC parameterized and mapped argument.

**Parameters:**

*\_mapidx* The varmap data address.

*\_arg* The parameterized argument you want to pass through a varmap.

**6.26.2.5** `#define RICMapElement(_Domain, _Range) (_Domain)&0xFF,  
(_Domain)>>8, (_Range)&0xFF, (_Range)>>8`

Output an RIC map element.

**Parameters:**

*\_Domain* The map element domain.

*\_Range* The map element range.

**6.26.2.6 #define RICMapFunction(\_MapElement, ...) \_MapElement,  
\_\_VA\_ARGS\_\_**

Output an RIC VarMap function.

**Parameters:**

*\_MapElement* An entry in the varmap function. At least 2 elements are required.  
See [RICMapElement](#).

**6.26.2.7 #define RICOpCircle(\_CopyOptions, \_Point, \_Radius) 10,  
0, 7, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point,  
(\_Radius)&0xFF, (\_Radius)>>8**

Output an RIC Circle opcode.

**Parameters:**

*\_CopyOptions* Circle copy options. See [Drawing option constants](#).  
*\_Point* The circle's center point. See [RICImgPoint](#).  
*\_Radius* The circle's radius.

**6.26.2.8 #define RICOpCopyBits(\_CopyOptions, \_DataAddr, \_SrcRect,  
\_DstPoint) 18, 0, 3, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8,  
(\_DataAddr)&0xFF, (\_DataAddr)>>8, \_SrcRect, \_DstPoint**

Output an RIC CopyBits opcode.

**Parameters:**

*\_CopyOptions* CopyBits copy options. See [Drawing option constants](#).  
*\_DataAddr* The address of the sprite from which to copy data.  
*\_SrcRect* The rectangular portion of the sprite to copy. See [RICImgRect](#).  
*\_DstPoint* The LCD coordinate to which to copy the data. See [RICImgPoint](#).

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```
6.26.2.9 #define RICOpDescription(_Options, _Width, _Height) 8, 0, 0, 0,
          (_Options)&0xFF, (_Options)>>8, (_Width)&0xFF, (_Width)>>8,
          (_Height)&0xFF, (_Height)>>8
```

Output an RIC Description opcode.

**Parameters:**

*\_Options* RIC options.  
*\_Width* The total RIC width.  
*\_Height* The total RIC height.

**Examples:**

[ex\\_dispgaoutex.nxc](#).

```
6.26.2.10 #define RICOpEllipse(_CopyOptions, _Point, _RadiusX,
          _RadiusY) 12, 0, 9, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8,
          _Point, (_RadiusX)&0xFF, (_RadiusX)>>8, (_RadiusY)&0xFF,
          (_RadiusY)>>8
```

Output an RIC Ellipse opcode.

**Parameters:**

*\_CopyOptions* Ellipse copy options. See [Drawing option constants](#).  
*\_Point* The center of the ellipse. See [RICImgPoint](#).  
*\_RadiusX* The x-axis radius of the ellipse.  
*\_RadiusY* The y-axis radius of the ellipse.

```
6.26.2.11 #define RICOpLine(_CopyOptions, _Point1, _Point2) 12, 0, 5, 0,
          (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point1, _Point2
```

Output an RIC Line opcode.

**Parameters:**

*\_CopyOptions* Line copy options. See [Drawing option constants](#).  
*\_Point1* The starting point of the line. See [RICImgPoint](#).  
*\_Point2* The ending point of the line. See [RICImgPoint](#).

```
6.26.2.12 #define RICOpNumBox(_CopyOptions, _Point, _Value) 10, 0, 8, 0,  
          (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_Value)&0xFF,  
          (_Value)>>8
```

Output an RIC NumBox opcode.

**Parameters:**

- \_CopyOptions* NumBox copy options. See [Drawing option constants](#).
- \_Point* The numbox bottom left corner. See [RICImgPoint](#).
- \_Value* The number to draw.

```
6.26.2.13 #define RICOpPixel(_CopyOptions, _Point, _Value) 10, 0, 4, 0,  
          (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_Value)&0xFF,  
          (_Value)>>8
```

Output an RIC Pixel opcode.

**Parameters:**

- \_CopyOptions* Pixel copy options. See [Drawing option constants](#).
- \_Point* The pixel coordinate. See [RICImgPoint](#).
- \_Value* The pixel value (unused).

```
6.26.2.14 #define RICOpPolygon(_CopyOptions, _Count,  
          _ThePoints) ((_Count*4)+6)&0xFF, ((_Count*4)+6)>>8, 10, 0,  
          (_CopyOptions)&0xFF, (_CopyOptions)>>8, (_Count)&0xFF,  
          (_Count)>>8, _ThePoints
```

Output an RIC Polygon opcode.

**Parameters:**

- \_CopyOptions* Polygon copy options. See [Drawing option constants](#).
- \_Count* The number of points in the polygon.
- \_ThePoints* The list of polygon points. See [RICPolygonPoints](#).

```
6.26.2.15 #define RICOpRect(_CopyOptions, _Point, _Width, _Height) 12,
           0, 6, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,
           (_Width)&0xFF, (_Width)>>8, (_Height)&0xFF, (_Height)>>8
```

Output an RIC Rect opcode.

**Parameters:**

*\_CopyOptions* Rect copy options. See [Drawing option constants](#).

*\_Point* The rectangle's top left corner. See [RICImgPoint](#).

*\_Width* The rectangle's width.

*\_Height* The rectangle's height.

```
6.26.2.16 #define RICOpSprite(_DataAddr, _Rows,
           _BytesPerRow, _SpriteData) ((_Rows*_ -
           BytesPerRow)+((_Rows*_BytesPerRow)%2)+8)&0xFF,
           (( _Rows*_BytesPerRow)+((_Rows*_BytesPerRow)%2)+8)>>8, 1, 0,
           (_DataAddr)&0xFF, (_DataAddr)>>8, (_Rows)&0xFF, (_Rows)>>8,
           (_BytesPerRow)&0xFF, (_BytesPerRow)>>8, _SpriteData
```

Output an RIC Sprite opcode.

**Parameters:**

*\_DataAddr* The address of the sprite.

*\_Rows* The number of rows of data.

*\_BytesPerRow* The number of bytes per row.

*\_SpriteData* The actual sprite data. See [RICSpriteData](#).

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```
6.26.2.17 #define RICOpVarMap(_DataAddr, _MapCount, _ -
           MapFunction) ((_MapCount*4)+6)&0xFF, ((_MapCount*4)+6)>>8,
           2, 0, (_DataAddr)&0xFF, (_DataAddr)>>8, (_MapCount)&0xFF,
           (_MapCount)>>8, _MapFunction
```

Output an RIC VarMap opcode.

**Parameters:**

- \_DataAddr* The address of the varmap.
- \_MapCount* The number of points in the function.
- \_MapFunction* The definition of the varmap function. See [RICMapFunction](#).

**6.26.2.18** `#define RICPolygonPoints(_pPoint1, _pPoint2, ...) _pPoint1, _pPoint2, __VA_ARGS__`

Output RIC polygon points.

**Parameters:**

- \_pPoint1* The first polygon point. See [RICImgPoint](#).
- \_pPoint2* The second polygon point (at least 3 points are required). See [RICImgPoint](#).

**6.26.2.19** `#define RICSetValue(_data, _idx, _newval) _data[_idx] = (_newval)&0xFF; _data[_idx+1] = (_newval)>>8`

Set the value of an element in an RIC data array.

**Parameters:**

- \_data* The RIC data array
- \_idx* The array index to update
- \_newval* The new value to write into the RIC data array

**6.26.2.20** `#define RICSpriteData(...) __VA_ARGS__`

Output RIC sprite data.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**6.27 NXT firmware module names**

Constant string names for all the NXT firmware modules.

## Defines

- #define `CommandModuleName` "Command.mod"
- #define `IOCtrlModuleName` "IOCtrl.mod"
- #define `LoaderModuleName` "Loader.mod"
- #define `SoundModuleName` "Sound.mod"
- #define `ButtonModuleName` "Button.mod"
- #define `UIModuleName` "Ui.mod"
- #define `InputModuleName` "Input.mod"
- #define `OutputModuleName` "Output.mod"
- #define `LowSpeedModuleName` "Low Speed.mod"
- #define `DisplayModuleName` "Display.mod"
- #define `CommModuleName` "Comm.mod"

### 6.27.1 Detailed Description

Constant string names for all the NXT firmware modules.

### 6.27.2 Define Documentation

#### 6.27.2.1 #define `ButtonModuleName` "Button.mod"

The button module name

#### 6.27.2.2 #define `CommandModuleName` "Command.mod"

The command module name

### Examples:

`ex_sysiomapread.nxc.`

#### 6.27.2.3 #define `CommModuleName` "Comm.mod"

The Comm module name

#### 6.27.2.4 #define `DisplayModuleName` "Display.mod"

The display module name

**6.27.2.5 #define InputModuleName "Input.mod"**

The input module name.

**6.27.2.6 #define IOCtrlModuleName "IOCtrl.mod"**

The IOCtrl module name

**6.27.2.7 #define LoaderModuleName "Loader.mod"**

The Loader module name

**6.27.2.8 #define LowSpeedModuleName "Low Speed.mod"**

The low speed module name

**6.27.2.9 #define OutputModuleName "Output.mod"**

The output module name

**6.27.2.10 #define SoundModuleName "Sound.mod"**

The sound module name

**Examples:**

[ex\\_sysiomapwrite.nxc](#).

**6.27.2.11 #define UIModuleName "Ui.mod"**

The Ui module name

## 6.28 NXT firmware module IDs

Constant numeric IDs for all the NXT firmware modules.

**Defines**

- #define [CommandModuleID](#) 0x00010001
- #define [IOCtrlModuleID](#) 0x00060001

- #define `LoaderModuleID` 0x00090001
- #define `SoundModuleID` 0x00080001
- #define `ButtonModuleID` 0x00040001
- #define `UIModuleID` 0x000C0001
- #define `InputModuleID` 0x00030001
- #define `OutputModuleID` 0x00020001
- #define `LowSpeedModuleID` 0x000B0001
- #define `DisplayModuleID` 0x000A0001
- #define `CommModuleID` 0x00050001

### 6.28.1 Detailed Description

Constant numeric IDs for all the NXT firmware modules.

### 6.28.2 Define Documentation

#### 6.28.2.1 #define `ButtonModuleID` 0x00040001

The button module ID

#### 6.28.2.2 #define `CommandModuleID` 0x00010001

The command module ID

#### Examples:

`ex_reladdressof.nxc`, `ex_RemoteIOMapRead.nxc`, `ex_RemoteIOMapWriteBytes.nxc`, `ex_RemoteIOMapWriteValue.nxc`, and `ex_sysiomapreadbyid.nxc`.

#### 6.28.2.3 #define `CommModuleID` 0x00050001

The Comm module ID

#### 6.28.2.4 #define `DisplayModuleID` 0x000A0001

The display module ID

#### 6.28.2.5 #define `InputModuleID` 0x00030001

The input module ID

**6.28.2.6 #define IOCtrlModuleID 0x00060001**

The IOCtrl module ID

**6.28.2.7 #define LoaderModuleID 0x00090001**

The Loader module ID

**6.28.2.8 #define LowSpeedModuleID 0x000B0001**

The low speed module ID

**6.28.2.9 #define OutputModuleID 0x00020001**

The output module ID

**6.28.2.10 #define SoundModuleID 0x00080001**

The sound module ID

**Examples:**[ex\\_sysiomapwritebyid.nxc.](#)**6.28.2.11 #define UIModuleID 0x000C0001**

The Ui module ID

**6.29 Miscellaneous NBC/NXC constants**

Miscellaneous constants for use in NBC and NXC.

**Modules**

- [Type aliases](#)

*Short type aliases indicating signed/unsigned and bit count for each type.*

- [Property constants](#)

*Use these constants for specifying the property for the `GetProperty` and `SetProperty` direct commands.*

- [Data type limits](#)

*Constants that define various data type limits.*

### Defines

- `#define TRUE 1`
- `#define FALSE 0`
- `#define NA 0xFFFF`
- `#define PI 3.141593`
- `#define RADIANS_PER_DEGREE PI/180`
- `#define DEGREES_PER_RADIAN 180/PI`

#### 6.29.1 Detailed Description

Miscellaneous constants for use in NBC and NXC.

#### 6.29.2 Define Documentation

##### 6.29.2.1 `#define DEGREES_PER_RADIAN 180/PI`

Used for converting from radians to degrees

##### 6.29.2.2 `#define FALSE 0`

A false value

##### 6.29.2.3 `#define NA 0xFFFF`

The specified argument does not apply (aka unwired)

#### Examples:

[ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), and [ex\\_ArraySumSqr.nxc](#).

#### 6.29.2.4 #define PI 3.141593

A constant for PI

##### Examples:

[ex\\_dispfnout.nxc](#), and [ex\\_string.nxc](#).

#### 6.29.2.5 #define RADIANS\_PER\_DEGREE PI/180

Used for converting from degrees to radians

##### Examples:

[ex\\_sin\\_cos.nxc](#).

#### 6.29.2.6 #define TRUE 1

A true value

##### Examples:

[ex\\_syscommbtconnection.nxc](#).

## 6.30 Third-party NXT devices

Documentation for NXT devices made by companies other than LEGO such as HiTechnic, mindsensors.com, and CodaTex.

### Modules

- [RCX constants](#)

*Constants that are for use with devices that communicate with the RCX or Scout programmable bricks via IR such as the HiTechnic IRLink or the MindSensors nRLink.*

- [HiTechnic/mindsensors Power Function/IR Train constants](#)

*Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.*

- [HiTechnic API Functions](#)

*Functions for accessing and modifying HiTechnic devices.*

- [MindSensors API Functions](#)  
*Functions for accessing and modifying MindSensors devices.*
- [Codatex API Functions](#)  
*Functions for accessing and modifying Codatex devices.*
- [Dexter Industries API Functions](#)  
*Functions for accessing and modifying Dexter Industries devices.*
- [Microinfinity API Functions](#)  
*Functions for accessing and modifying Microinfinity devices.*

### 6.30.1 Detailed Description

Documentation for NXT devices made by companies other than LEGO such as HiTechnic, mindsensors.com, and CodaTex.

## 6.31 Standard-C API functions

Documentation for various Standard-C library routines.

### Modules

- [cmath API](#)  
*Standard C cmath API functions.*
- [cstdio API](#)  
*Standard C cstdio API functions.*
- [cstdlib API](#)  
*Standard C cstdlib API functions and types.*
- [cstring API](#)  
*Standard C cstring API functions.*
- [ctype API](#)  
*Standard C ctype API functions.*

### 6.31.1 Detailed Description

Documentation for various Standard-C library routines.

## 6.32 A simple 3D graphics library

Documentation for a simple 3D graphics library.

### Modules

- [Graphics library begin modes](#)  
*Constants that are used to specify the polygon surface begin mode.*
- [Graphics library actions](#)  
*Constants that are used to specify a graphics library action.*
- [Graphics library settings](#)  
*Constants that are used to configure the graphics library settings.*
- [Graphics library cull mode](#)  
*Constants to use when setting the graphics library cull mode.*

### Functions

- void [glInit](#) ()  
*Initialize graphics library.*
- void [glSet](#) (int glType, int glValue)  
*Set graphics library options.*
- int [glBeginObject](#) ()  
*Begin defining an object.*
- void [glEndObject](#) ()  
*Stop defining an object.*
- void [glObjectAction](#) (int glObjectId, int glAction, int glValue)  
*Perform an object action.*
- void [glAddVertex](#) (int glX, int glY, int glZ)

*Add a vertex to an object.*

- void `glBegin` (int glBeginMode)  
*Begin a new polygon for the current object.*
- void `glEnd` ()  
*Finish a polygon for the current object.*
- void `glBeginRender` ()  
*Begin a new render.*
- void `glCallObject` (int glObjectId)  
*Call a graphic object.*
- void `glFinishRender` ()  
*Finish the current render.*
- void `glSetAngleX` (int glValue)  
*Set the X axis angle.*
- void `glAddToAngleX` (int glValue)  
*Add to the X axis angle.*
- void `glSetAngleY` (int glValue)  
*Set the Y axis angle.*
- void `glAddToAngleY` (int glValue)  
*Add to the Y axis angle.*
- void `glSetAngleZ` (int glValue)  
*Set the Z axis angle.*
- void `glAddToAngleZ` (int glValue)  
*Add to the Z axis angle.*
- int `glSin32768` (int glAngle)  
*Table-based sine scaled by 32768.*
- int `glCos32768` (int glAngle)  
*Table-based cosine scaled by 32768.*
- int `glBox` (int glMode, int glSizeX, int glSizeY, int glSizeZ)

*Create a 3D box.*

- int [glCube](#) (int glMode, int glSize)  
*Create a 3D cube.*
- int [glPyramid](#) (int glMode, int glSizeX, int glSizeY, int glSizeZ)  
*Create a 3D pyramid.*

### 6.32.1 Detailed Description

Documentation for a simple 3D graphics library. The library code was written by Arno van der Vegt.

### 6.32.2 Function Documentation

#### 6.32.2.1 void [glAddToAngleX](#) (int *glValue*) [**inline**]

Add to the X axis angle. Add the specified value to the existing X axis angle.

##### Parameters:

*glValue* The value to add to the X axis angle.

##### Examples:

[glBoxDemo.nxc](#), and [glCircleDemo.nxc](#).

#### 6.32.2.2 void [glAddToAngleY](#) (int *glValue*) [**inline**]

Add to the Y axis angle. Add the specified value to the existing Y axis angle.

##### Parameters:

*glValue* The value to add to the Y axis angle.

##### Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.3 void glAddToAngleZ (int *glValue*) [inline]**

Add to the Z axis angle. Add the specified value to the existing Z axis angle.

**Parameters:**

*glValue* The value to add to the Z axis angle.

**6.32.2.4 void glAddVertex (int *glX*, int *glY*, int *glZ*) [inline]**

Add a vertex to an object. Add a vertex to an object currently being defined. This function should only be used between [glBegin](#) and [glEnd](#) which are themselves nested within a [glBeginObject](#) and [glEndObject](#) pair.

**Parameters:**

*glX* The X axis coordinate.

*glY* The Y axis coordinate.

*glZ* The Z axis coordinate.

**6.32.2.5 void glBegin (int *glBeginMode*) [inline]**

Begin a new polygon for the current object. Start defining a polygon surface for the current graphics object using the specified begin mode.

**Parameters:**

*glBeginMode* The desired mode. See [Graphics library begin modes](#).

**6.32.2.6 int glBeginObject () [inline]**

Begin defining an object. Start the process of defining a graphics library object using low level functions such as [glBegin](#), [glAddVertex](#), and [glEnd](#).

**Returns:**

The object index of the new object being created.

**6.32.2.7 void glBeginRender () [inline]**

Begin a new render. Start the process of rendering the existing graphic objects.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.8 int glBox (int glMode, int glSizeX, int glSizeY, int glSizeZ) [inline]**

Create a 3D box. Define a 3D box using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with width, height, and depth specified via the *glSizeX*, *glSizeY*, and *glSizeZ* parameters.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSizeX* The X axis size (width).

*glSizeY* The Y axis size (height).

*glSizeZ* The Z axis size (depth).

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.9 void glCallObject (int glObjectId) [inline]**

Call a graphic object. Tell the graphics library that you want it to include the specified object in the render.

**Parameters:**

*glObjectId* The desired object id.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.10 int glCos32768 (int *glAngle*) [inline]**

Table-based cosine scaled by 32768. Return the cosine of the specified angle in degrees. The result is scaled by 32768.

**Parameters:**

*glAngle* The angle in degrees.

**Returns:**

The cosine value scaled by 32768.

**6.32.2.11 int glCube (int *glMode*, int *glSize*) [inline]**

Create a 3D cube. Define a 3D cube using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with equal width, height, and depth specified via the *glSize* parameter.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSize* The cube's width, height, and depth.

**Examples:**

[glBoxDemo.nxc](#).

**6.32.2.12 void glEnd () [inline]**

Finish a polygon for the current object. Stop defining a polygon surface for the current graphics object.

**6.32.2.13 void glEndObject () [inline]**

Stop defining an object. Finish the process of defining a graphics library object. Call this function after you have completed the object definition.

**6.32.2.14 void glFinishRender () [inline]**

Finish the current render. Rotate the vertex list, clear the screen, and draw the rendered objects to the LCD.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.15 void glInit () [inline]**

Initialize graphics library. Setup all the necessary data for the graphics library to function. Call this function before any other graphics library routine.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.16 void glObjectAction (int glObjectId, int glAction, int glValue) [inline]**

Perform an object action. Execute the specified action on the specified object.

**Parameters:**

*glObjectId* The object id.

*glAction* The action to perform on the object. See [Graphics library actions](#).

*glValue* The setting value.

**Examples:**

[glBoxDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.17** `int glPyramid (int glMode, int glSizeX, int glSizeY, int glSizeZ)`  
`[inline]`

Create a 3D pyramid. Define a 3D pyramid using the specified begin mode for all faces. The center of the pyramid is at the origin of the XYZ axis with width, height, and depth specified via the *glSizeX*, *glSizeY*, and *glSizeZ* parameters.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSizeX* The X axis size (width).

*glSizeY* The Y axis size (height).

*glSizeZ* The Z axis size (depth).

**6.32.2.18** `void glSet (int glType, int glValue)` `[inline]`

Set graphics library options. Adjust graphic library settings for circle size and cull mode.

**Parameters:**

*glType* The setting type. See [Graphics library settings](#).

*glValue* The setting value. For culling modes see [Graphics library cull mode](#).

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.19** `void glSetAngleX (int glValue)` `[inline]`

Set the X axis angle. Set the X axis angle to the specified value.

**Parameters:**

*glValue* The new X axis angle.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.32.2.20 void glSetAngleY (int glValue) [inline]**

Set the Y axis angle. Set the Y axis angle to the specified value.

**Parameters:**

*glValue* The new Y axis angle.

**6.32.2.21 void glSetAngleZ (int glValue) [inline]**

Set the Z axis angle. Set the Z axis angle to the specified value.

**Parameters:**

*glValue* The new Z axis angle.

**6.32.2.22 int glSin32768 (int glAngle) [inline]**

Table-based sine scaled by 32768. Return the sine of the specified angle in degrees. The result is scaled by 32768.

**Parameters:**

*glAngle* The angle in degrees.

**Returns:**

The sine value scaled by 32768.

**6.33 Type aliases**

Short type aliases indicating signed/unsigned and bit count for each type.

**Defines**

- #define **u8** unsigned char
- #define **s8** char
- #define **u16** unsigned int
- #define **s16** int
- #define **u32** unsigned long
- #define **s32** long

### 6.33.1 Detailed Description

Short type aliases indicating signed/unsigned and bit count for each type.

### 6.33.2 Define Documentation

#### 6.33.2.1 #define s16 int

Signed 16 bit type

#### 6.33.2.2 #define s32 long

Signed 32 bit type

#### 6.33.2.3 #define s8 char

Signed 8 bit type

#### 6.33.2.4 #define u16 unsigned int

Unsigned 16 bit type

#### 6.33.2.5 #define u32 unsigned long

Unsigned 32 bit type

#### 6.33.2.6 #define u8 unsigned char

Unsigned 8 bit type

## 6.34 Input port constants

Input port constants are used when calling NXC sensor control API functions.

### Defines

- #define [S1](#) 0
- #define [S2](#) 1
- #define [S3](#) 2
- #define [S4](#) 3

### 6.34.1 Detailed Description

Input port constants are used when calling NXC sensor control API functions.

### 6.34.2 Define Documentation

#### 6.34.2.1 #define S1 0

Input port 1

#### Examples:

```

ex_ACCLNxCalibrateX.nxc,      ex_ACCLNxCalibrateXEnd.nxc,      ex_-
ACCLNxCalibrateY.nxc,      ex_ACCLNxCalibrateYEnd.nxc,      ex_-
ACCLNxCalibrateZ.nxc,      ex_ACCLNxCalibrateZEnd.nxc,      ex_-
ACCLNxResetCalibration.nxc,  ex_ACCLNxSensitivity.nxc,      ex_-
ACCLNxXOffset.nxc,      ex_ACCLNxXRange.nxc,      ex_ACCLNxYOffset.nxc,
ex_ACCLNxYRange.nxc,      ex_ACCLNxZOffset.nxc,      ex_ACCLNxZRange.nxc,
ex_ClearSensor.nxc,      ex_ColorADRaw.nxc,      ex_ColorBoolean.nxc,
ex_ColorCalibration.nxc,      ex_ColorCalibrationState.nxc,      ex_-
ColorCalLimits.nxc,      ex_ColorSensorRaw.nxc,      ex_ColorSensorValue.nxc,
ex_ConfigureTemperatureSensor.nxc,      ex_CustomSensorActiveStatus.nxc,
ex_CustomSensorPercentFullScale.nxc,      ex_CustomSensorZeroOffset.nxc,
ex_diaacl.nxc,      ex_digps.nxc,      ex_digyro.nxc,      ex_DISTNxDistance.nxc,      ex_-
DISTNxGP2D12.nxc,      ex_DISTNxGP2D120.nxc,      ex_DISTNxGP2YA02.nxc,
ex_DISTNxGP2YA21.nxc,      ex_DISTNxMaxDistance.nxc,      ex_-
DISTNxMinDistance.nxc,      ex_DISTNxModuleType.nxc,      ex_-
DISTNxNumPoints.nxc,      ex_DISTNxVoltage.nxc,      ex_GetInput.nxc,      ex_-
GetLSInputBuffer.nxc,      ex_GetLSOutputBuffer.nxc,      ex_HTIRTrain.nxc,
ex_HTPFComboDirect.nxc,      ex_HTPFComboPWM.nxc,      ex_-
HTPFRawOutput.nxc,      ex_HTPFRepeat.nxc,      ex_HTPFSingleOutputCST.nxc,
ex_HTPFSingleOutputPWM.nxc,      ex_HTPFSinglePin.nxc,      ex_HTPFTrain.nxc,
ex_HTRCXAddToDatalog.nxc,      ex_HTRCXClearSensor.nxc,      ex_-
HTRCXSetIRLinkPort.nxc,      ex_HTRCXSetSensorMode.nxc,      ex_-
HTRCXSetSensorType.nxc,      ex_I2CBytesReady.nxc,      ex_I2CCheckStatus.nxc,
ex_i2cdeviceid.nxc,      ex_i2cdeviceinfo.nxc,      ex_I2CRead.nxc,      ex_-
I2CSendCommand.nxc,      ex_I2CStatus.nxc,      ex_i2cvendorid.nxc,      ex_-
i2cversion.nxc,      ex_I2CWrite.nxc,      ex_LowspeedBytesReady.nxc,      ex_-
LowspeedCheckStatus.nxc,      ex_LowspeedRead.nxc,      ex_LowspeedStatus.nxc,
ex_LowspeedWrite.nxc,      ex_LSChannelState.nxc,      ex_LSErrorType.nxc,
ex_LSInputBufferBytesToRx.nxc,      ex_LSInputBufferInPtr.nxc,      ex_-
LSInputBufferOutPtr.nxc,      ex_LSMODE.nxc,      ex_LSOutputBufferBytesToRx.nxc,
ex_LSOutputBufferInPtr.nxc,      ex_LSOutputBufferOutPtr.nxc,      ex_-
MSADPAOff.nxc,      ex_MSADPAOn.nxc,      ex_MSDeenergize.nxc,      ex_-
MSEnergize.nxc,      ex_MSIRTrain.nxc,      ex_MSPFComboDirect.nxc,      ex_-

```

MSPFComboPWM.nxc, ex\_MSPFRawOutput.nxc, ex\_MSPFRepeat.nxc,  
ex\_MSPFSingleOutputCST.nxc, ex\_MSPFSingleOutputPWM.nxc, ex\_  
MSPFSinglePin.nxc, ex\_MSPFTrain.nxc, ex\_MSRCXAddToDatalog.nxc,  
ex\_MSRCXClearSensor.nxc, ex\_MSRCXSetNRLinkPort.nxc, ex\_  
MSRCXSetSensorMode.nxc, ex\_MSRCXSetSensorType.nxc, ex\_  
MSRCXSumVar.nxc, ex\_MSReadValue.nxc, ex\_NRLink2400.nxc, ex\_  
NRLink4800.nxc, ex\_NRLinkFlush.nxc, ex\_NRLinkIRLong.nxc, ex\_  
NRLinkIRShort.nxc, ex\_NRLinkSetPF.nxc, ex\_NRLinkSetRCX.nxc,  
ex\_NRLinkSetTrain.nxc, ex\_NRLinkStatus.nxc, ex\_NRLinkTxRaw.nxc,  
ex\_NXTHID.nxc, ex\_NXTLineLeader.nxc, ex\_NXTPowerMeter.nxc,  
ex\_NXTServo.nxc, ex\_NXTSumoEyes.nxc, ex\_PFMate.nxc, ex\_  
proto.nxc, ex\_PSPNxAnalog.nxc, ex\_PSPNxDigital.nxc, ex\_  
readi2cregister.nxc, ex\_ReadNRLinkBytes.nxc, ex\_ReadSensorColorEx.nxc,  
ex\_ReadSensorColorRaw.nxc, ex\_ReadSensorEMeter.nxc, ex\_  
ReadSensorHTAccel.nxc, ex\_ReadSensorHTColor.nxc, ex\_  
ReadSensorHTColor2Active.nxc, ex\_ReadSensorHTIRReceiver.nxc, ex\_  
ReadSensorHTIRReceiverEx.nxc, ex\_ReadSensorHTIRSeeker2AC.nxc, ex\_  
ReadSensorHTIRSeeker2DC.nxc, ex\_ReadSensorHTNormalizedColor.nxc, ex\_  
ReadSensorHTNormalizedColor2Active.nxc, ex\_ReadSensorHTRawColor.nxc,  
ex\_ReadSensorHTRawColor2.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc,  
ex\_ReadSensorMSAccel.nxc, ex\_ReadSensorMSPlayStation.nxc,  
ex\_ReadSensorMSRTClock.nxc, ex\_ReadSensorMSTilt.nxc, ex\_  
ReadSensorUSEx.nxc, ex\_RemoteLowSpeedRead.nxc, ex\_  
RemoteLowSpeedWrite.nxc, ex\_RemoteResetScaledValue.nxc, ex\_  
RemoteSetInputMode.nxc, ex\_RFIDInit.nxc, ex\_RFIDMode.nxc, ex\_  
RFIDRead.nxc, ex\_RFIDReadContinuous.nxc, ex\_RFIDReadSingle.nxc,  
ex\_RFIDStatus.nxc, ex\_RFIDStop.nxc, ex\_RunNRLinkMacro.nxc, ex\_  
Sensor.nxc, ex\_SensorBoolean.nxc, ex\_SensorDigiPinsDirection.nxc,  
ex\_SensorDigiPinsOutputLevel.nxc, ex\_SensorDigiPinsStatus.nxc, ex\_  
SensorHTColorNum.nxc, ex\_SensorHTCompass.nxc, ex\_SensorHTEOPD.nxc,  
ex\_SensorHTGyro.nxc, ex\_SensorHTIRSeeker2ACDir.nxc, ex\_  
SensorHTIRSeeker2Addr.nxc, ex\_SensorHTIRSeeker2DCDir.nxc,  
ex\_SensorHTIRSeekerDir.nxc, ex\_SensorHTMagnet.nxc, ex\_  
SensorInvalid.nxc, ex\_SensorMode.nxc, ex\_SensorMSCompass.nxc,  
ex\_SensorMSDROD.nxc, ex\_SensorMSPressure.nxc, ex\_  
SensorMSPressureRaw.nxc, ex\_SensorNormalized.nxc, ex\_SensorRaw.nxc,  
ex\_SensorScaled.nxc, ex\_SensorTemperature.nxc, ex\_SensorType.nxc, ex\_  
SensorValue.nxc, ex\_SensorValueBool.nxc, ex\_SensorValueRaw.nxc, ex\_  
SetACCLNxSensitivity.nxc, ex\_SetCustomSensorActiveStatus.nxc, ex\_  
SetCustomSensorPercentFullScale.nxc, ex\_SetCustomSensorZeroOffset.nxc,  
ex\_sethtcolor2mode.nxc, ex\_sethtirseeker2mode.nxc, ex\_SetInput.nxc, ex\_  
SetSensor.nxc, ex\_setsensorboolean.nxc, ex\_setsensorcolorblue.nxc, ex\_  
setsensorcolorfull.nxc, ex\_setsensorcolorgreen.nxc, ex\_setsensorcolornone.nxc,  
ex\_setsensorcolorred.nxc, ex\_SetSensorDigiPinsDirection.nxc, ex\_  
SetSensorDigiPinsOutputLevel.nxc, ex\_SetSensorDigiPinsStatus.nxc, ex\_  
SetSensorEMeter.nxc, ex\_setsensorhteopd.nxc, ex\_SetSensorHTGyro.nxc, ex\_

SetSensorHTMagnet.nxc, ex\_SetSensorLight.nxc, ex\_SetSensorLowspeed.nxc, ex\_SetSensorMode.nxc, ex\_setsensormsdrod.nxc, ex\_setsensormspressure.nxc, ex\_SetSensorSound.nxc, ex\_SetSensorTemperature.nxc, ex\_SetSensorTouch.nxc, ex\_SetSensorType.nxc, ex\_SetSensorUltrasonic.nxc, ex\_superpro.nxc, ex\_SysColorSensorRead.nxc, ex\_syscommmlscheckstatus.nxc, ex\_syscommmlsread.nxc, ex\_syscommmlswrite.nxc, ex\_syscommmlswriteex.nxc, ex\_SysComputeCalibValue.nxc, ex\_sysinputpinfunction.nxc, ex\_writei2cregister.nxc, ex\_writenrlinkbytes.nxc, and ex\_xg1300.nxc.

#### 6.34.2.2 #define S2 1

Input port 2

#### 6.34.2.3 #define S3 2

Input port 3

#### Examples:

ex\_ReadSensorHTBarometric.nxc.

#### 6.34.2.4 #define S4 3

Input port 4

#### Examples:

ex\_I2CBytes.nxc, ex\_ReadSensorHTAngle.nxc, ex\_ResetSensorHTAngle.nxc, and ex\_SensorUS.nxc.

## 6.35 Sensor type constants

Use sensor type constants to configure an input port for a specific type of sensor.

#### Defines

- #define `SENSOR_TYPE_NONE` `IN_TYPE_NO_SENSOR`
- #define `SENSOR_TYPE_TOUCH` `IN_TYPE_SWITCH`
- #define `SENSOR_TYPE_TEMPERATURE` `IN_TYPE_TEMPERATURE`
- #define `SENSOR_TYPE_LIGHT` `IN_TYPE_REFLECTION`
- #define `SENSOR_TYPE_ROTATION` `IN_TYPE_ANGLE`

- #define [SENSOR\\_TYPE\\_LIGHT\\_ACTIVE](#) IN\_TYPE\_LIGHT\_ACTIVE
- #define [SENSOR\\_TYPE\\_LIGHT\\_INACTIVE](#) IN\_TYPE\_LIGHT\_INACTIVE
- #define [SENSOR\\_TYPE\\_SOUND\\_DB](#) IN\_TYPE\_SOUND\_DB
- #define [SENSOR\\_TYPE\\_SOUND\\_DBA](#) IN\_TYPE\_SOUND\_DBA
- #define [SENSOR\\_TYPE\\_CUSTOM](#) IN\_TYPE\_CUSTOM
- #define [SENSOR\\_TYPE\\_LOWSPEED](#) IN\_TYPE\_LOWSPEED
- #define [SENSOR\\_TYPE\\_LOWSPEED\\_9V](#) IN\_TYPE\_LOWSPEED\_9V
- #define [SENSOR\\_TYPE\\_HIGHSPEED](#) IN\_TYPE\_HISPEED
- #define [SENSOR\\_TYPE\\_COLORFULL](#) IN\_TYPE\_COLORFULL
- #define [SENSOR\\_TYPE\\_COLORRED](#) IN\_TYPE\_COLORRED
- #define [SENSOR\\_TYPE\\_COLORGREEN](#) IN\_TYPE\_COLORGREEN
- #define [SENSOR\\_TYPE\\_COLORBLUE](#) IN\_TYPE\_COLORBLUE
- #define [SENSOR\\_TYPE\\_COLORNONE](#) IN\_TYPE\_COLORNONE

### 6.35.1 Detailed Description

Use sensor type constants to configure an input port for a specific type of sensor.

See also:

[SetSensorType\(\)](#)

### 6.35.2 Define Documentation

#### 6.35.2.1 #define [SENSOR\\_TYPE\\_COLORBLUE](#) IN\_TYPE\_COLORBLUE

NXT 2.0 color sensor with blue light

#### 6.35.2.2 #define [SENSOR\\_TYPE\\_COLORFULL](#) IN\_TYPE\_COLORFULL

NXT 2.0 color sensor in full color mode

#### 6.35.2.3 #define [SENSOR\\_TYPE\\_COLORGREEN](#) IN\_TYPE\_COLORGREEN

NXT 2.0 color sensor with green light

#### 6.35.2.4 #define [SENSOR\\_TYPE\\_COLORNONE](#) IN\_TYPE\_COLORNONE

NXT 2.0 color sensor with no light

**6.35.2.5 #define SENSOR\_TYPE\_COLORRED IN\_TYPE\_COLORRED**

NXT 2.0 color sensor with red light

**6.35.2.6 #define SENSOR\_TYPE\_CUSTOM IN\_TYPE\_CUSTOM**

NXT custom sensor

**6.35.2.7 #define SENSOR\_TYPE\_HIGHSPEED IN\_TYPE\_HISPEED**

NXT Hi-speed port (only S4)

**6.35.2.8 #define SENSOR\_TYPE\_LIGHT IN\_TYPE\_REFLECTION**

RCX light sensor

**6.35.2.9 #define SENSOR\_TYPE\_LIGHT\_ACTIVE IN\_TYPE\_LIGHT\_-  
ACTIVE**

NXT light sensor with light

**6.35.2.10 #define SENSOR\_TYPE\_LIGHT\_INACTIVE IN\_TYPE\_LIGHT\_-  
INACTIVE**

NXT light sensor without light

**6.35.2.11 #define SENSOR\_TYPE\_LOWSPEED IN\_TYPE\_LOWSPEED**

NXT I2C digital sensor

**Examples:**[ex\\_RemoteSetInputMode.nxc.](#)**6.35.2.12 #define SENSOR\_TYPE\_LOWSPEED\_9V IN\_TYPE\_LOWSPEED\_-  
9V**

NXT I2C digital sensor with 9V power

**6.35.2.13 #define SENSOR\_TYPE\_NONE IN\_TYPE\_NO\_SENSOR**

No sensor configured

**6.35.2.14 #define SENSOR\_TYPE\_ROTATION IN\_TYPE\_ANGLE**

RCX rotation sensor

**6.35.2.15 #define SENSOR\_TYPE\_SOUND\_DB IN\_TYPE\_SOUND\_DB**

NXT sound sensor with dB scaling

**Examples:**[ex\\_SetInput.nxc](#).**6.35.2.16 #define SENSOR\_TYPE\_SOUND\_DBA IN\_TYPE\_SOUND\_DBA**

NXT sound sensor with dBA scaling

**6.35.2.17 #define SENSOR\_TYPE\_TEMPERATURE IN\_TYPE\_-  
TEMPERATURE**

RCX temperature sensor

**6.35.2.18 #define SENSOR\_TYPE\_TOUCH IN\_TYPE\_SWITCH**

NXT or RCX touch sensor

**Examples:**[ex\\_HTRCXSetSensorType.nxc](#), [ex\\_MSRCXSetSensorType.nxc](#), and [ex\\_-SetSensorType.nxc](#).**6.36 Sensor mode constants**

Use sensor mode constants to configure an input port for the desired sensor mode.

**Defines**

- #define [SENSOR\\_MODE\\_RAW](#) IN\_MODE\_RAW

- #define [SENSOR\\_MODE\\_BOOL](#) IN\_MODE\_BOOLEAN
- #define [SENSOR\\_MODE\\_EDGE](#) IN\_MODE\_TRANSITIONCNT
- #define [SENSOR\\_MODE\\_PULSE](#) IN\_MODE\_PERIODCOUNTER
- #define [SENSOR\\_MODE\\_PERCENT](#) IN\_MODE\_PCTFULLSCALE
- #define [SENSOR\\_MODE\\_CELSIUS](#) IN\_MODE\_CELSIUS
- #define [SENSOR\\_MODE\\_FAHRENHEIT](#) IN\_MODE\_FAHRENHEIT
- #define [SENSOR\\_MODE\\_ROTATION](#) IN\_MODE\_ANGLESTEP

### 6.36.1 Detailed Description

Use sensor mode constants to configure an input port for the desired sensor mode.

See also:

[SetSensorMode\(\)](#)

### 6.36.2 Define Documentation

#### 6.36.2.1 #define SENSOR\_MODE\_BOOL IN\_MODE\_BOOLEAN

Boolean value (0 or 1)

Examples:

[ex\\_HTRCXSetSensorMode.nxc](#), and [ex\\_MSRCXSetSensorMode.nxc](#).

#### 6.36.2.2 #define SENSOR\_MODE\_CELSIUS IN\_MODE\_CELSIUS

RCX temperature sensor value in degrees celcius

#### 6.36.2.3 #define SENSOR\_MODE\_EDGE IN\_MODE\_TRANSITIONCNT

Counts the number of boolean transitions

#### 6.36.2.4 #define SENSOR\_MODE\_FAHRENHEIT IN\_MODE\_FAHRENHEIT

RCX temperature sensor value in degrees fahrenheit

#### 6.36.2.5 #define SENSOR\_MODE\_PERCENT IN\_MODE\_PCTFULLSCALE

Scaled value from 0 to 100

**6.36.2.6 #define SENSOR\_MODE\_PULSE IN\_MODE\_PERIODCOUNTER**

Counts the number of boolean periods

**6.36.2.7 #define SENSOR\_MODE\_RAW IN\_MODE\_RAW**

Raw value from 0 to 1023

**Examples:**

[ex\\_RemoteSetInputMode.nxc](#), and [ex\\_SetSensorMode.nxc](#).

**6.36.2.8 #define SENSOR\_MODE\_ROTATION IN\_MODE\_ANGLESTEP**

RCX rotation sensor (16 ticks per revolution)

**6.37 Combined sensor type and mode constants**

Use the combined sensor type and mode constants to configure both the sensor mode and type in a single function call.

**Defines**

- #define `_SENSOR_CFG`(\_type, \_mode) (((\_type)<<8)+(\_mode))
- #define `SENSOR_TOUCH` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_BOOL)`
- #define `SENSOR_LIGHT` `_SENSOR_CFG(SENSOR_TYPE_LIGHT, SENSOR_MODE_PERCENT)`
- #define `SENSOR_ROTATION` `_SENSOR_CFG(SENSOR_TYPE_ROTATION, SENSOR_MODE_ROTATION)`
- #define `SENSOR_CELSIUS` `_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_CELSIUS)`
- #define `SENSOR_FAHRENHEIT` `_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_FAHRENHEIT)`
- #define `SENSOR_PULSE` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_PULSE)`
- #define `SENSOR_EDGE` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_EDGE)`
- #define `SENSOR_NXTLIGHT` `_SENSOR_CFG(SENSOR_TYPE_LIGHT_ACTIVE, SENSOR_MODE_PERCENT)`
- #define `SENSOR_SOUND` `_SENSOR_CFG(SENSOR_TYPE_SOUND_DB, SENSOR_MODE_PERCENT)`

- #define [SENSOR\\_LOWSPEED\\_9V](#) `_SENSOR_CFG(SENSOR_TYPE_-  
LOWSPEED_9V, SENSOR_MODE_RAW)`
- #define [SENSOR\\_LOWSPEED](#) `_SENSOR_CFG(SENSOR_TYPE_-  
LOWSPEED, SENSOR_MODE_RAW)`
- #define [SENSOR\\_COLORFULL](#) `_SENSOR_CFG(SENSOR_TYPE_-  
COLORFULL, SENSOR_MODE_RAW)`
- #define [SENSOR\\_COLORRED](#) `_SENSOR_CFG(SENSOR_TYPE_-  
COLORRED, SENSOR_MODE_PERCENT)`
- #define [SENSOR\\_COLORGREEN](#) `_SENSOR_CFG(SENSOR_TYPE_-  
COLORGREEN, SENSOR_MODE_PERCENT)`
- #define [SENSOR\\_COLORBLUE](#) `_SENSOR_CFG(SENSOR_TYPE_-  
COLORBLUE, SENSOR_MODE_PERCENT)`
- #define [SENSOR\\_COLORNONE](#) `_SENSOR_CFG(SENSOR_TYPE_-  
COLORNONE, SENSOR_MODE_PERCENT)`

### 6.37.1 Detailed Description

Use the combined sensor type and mode constants to configure both the sensor mode and type in a single function call.

See also:

[SetSensor\(\)](#)

### 6.37.2 Define Documentation

#### 6.37.2.1 #define `_SENSOR_CFG(_type, _mode) (((_type)<<8)+(_mode))`

Macro for defining [SetSensor](#) combined type and mode constants

#### 6.37.2.2 #define `SENSOR_CELSIUS _SENSOR_CFG(SENSOR_TYPE_- TEMPERATURE, SENSOR_MODE_CELSIUS)`

RCX temperature sensor in celcius mode

#### 6.37.2.3 #define `SENSOR_COLORBLUE _SENSOR_CFG(SENSOR_TYPE_- COLORBLUE, SENSOR_MODE_PERCENT)`

NXT 2.0 color sensor (blue) in percent mode

**6.37.2.4 #define SENSOR\_COLORFULL \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORFULL, SENSOR\_MODE\_RAW)**

NXT 2.0 color sensor (full) in raw mode

**6.37.2.5 #define SENSOR\_COLORGREEN \_SENSOR\_CFG(SENSOR\_-  
TYPE\_COLORGREEN, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

**6.37.2.6 #define SENSOR\_COLORNONE \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORNONE, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

**6.37.2.7 #define SENSOR\_COLORRED \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORRED, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (red) in percent mode

**6.37.2.8 #define SENSOR\_EDGE \_SENSOR\_CFG(SENSOR\_TYPE\_TOUCH,  
SENSOR\_MODE\_EDGE)**

Touch sensor in edge mode

**6.37.2.9 #define SENSOR\_FAHRENHEIT \_SENSOR\_CFG(SENSOR\_TYPE\_-  
TEMPERATURE, SENSOR\_MODE\_FAHRENHEIT)**

RCX temperature sensor in fahrenheit mode

**6.37.2.10 #define SENSOR\_LIGHT \_SENSOR\_CFG(SENSOR\_TYPE\_-  
LIGHT, SENSOR\_MODE\_PERCENT)**

RCX Light sensor in percent mode

**6.37.2.11 #define SENSOR\_LOWSPEED \_SENSOR\_CFG(SENSOR\_TYPE\_-  
LOWSPEED, SENSOR\_MODE\_RAW)**

NXT I2C sensor without 9V power in raw mode

**6.37.2.12** `#define SENSOR_LOWSPEED_9V _SENSOR_CFG(SENSOR_TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)`

NXT I2C sensor with 9V power in raw mode

**6.37.2.13** `#define SENSOR_NXTLIGHT _SENSOR_CFG(SENSOR_TYPE_LIGHT_ACTIVE, SENSOR_MODE_PERCENT)`

NXT light sensor in active mode

**6.37.2.14** `#define SENSOR_PULSE _SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_PULSE)`

Touch sensor in pulse mode

**6.37.2.15** `#define SENSOR_ROTATION _SENSOR_CFG(SENSOR_TYPE_ROTATION, SENSOR_MODE_ROTATION)`

RCX rotation sensor in rotation mode

**6.37.2.16** `#define SENSOR_SOUND _SENSOR_CFG(SENSOR_TYPE_SOUND_DB, SENSOR_MODE_PERCENT)`

NXT sound sensor (dB) in percent mode

**6.37.2.17** `#define SENSOR_TOUCH _SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_BOOL)`

Touch sensor in boolean mode

#### Examples:

[ex\\_SetSensor.nxc](#).

## 6.38 Input module types

Types used by various input module functions.

### Data Structures

- struct [ColorSensorReadType](#)

*Parameters for the ColorSensorRead system call.*

- struct [InputValueType](#)  
*Parameters for the RemoteGetInputValues function.*
- struct [InputPinFunctionType](#)  
*Parameters for the InputPinFunction system call.*

### 6.38.1 Detailed Description

Types used by various input module functions.

## 6.39 Input module functions

Functions for accessing and modifying input module features.

### Modules

- [Basic analog sensor value names](#)  
*Read analog sensor values using these names.*

### Functions

- void [SetSensorType](#) (const byte &port, byte type)  
*Set sensor type.*
- void [SetSensorMode](#) (const byte &port, byte mode)  
*Set sensor mode.*
- void [ClearSensor](#) (const byte &port)  
*Clear a sensor value.*
- void [ResetSensor](#) (const byte &port)  
*Reset the sensor port.*
- void [SetSensor](#) (const byte &port, const unsigned int config)  
*Set sensor configuration.*
- void [SetSensorTouch](#) (const byte &port)

*Configure a touch sensor.*

- void [SetSensorLight](#) (const byte &port, bool bActive=true)  
*Configure a light sensor.*
- void [SetSensorSound](#) (const byte &port, bool bdBScaling=true)  
*Configure a sound sensor.*
- void [SetSensorLowspeed](#) (const byte &port, bool bIsPowered=true)  
*Configure an I2C sensor.*
- void [SetSensorUltrasonic](#) (const byte &port)  
*Configure an ultrasonic sensor.*
- void [SetSensorEMeter](#) (const byte &port)  
*Configure an EMeter sensor.*
- void [SetSensorTemperature](#) (const byte &port)  
*Configure a temperature sensor.*
- void [SetSensorColorFull](#) (const byte &port)  
*Configure an NXT 2.0 full color sensor.*
- void [SetSensorColorRed](#) (const byte &port)  
*Configure an NXT 2.0 red light sensor.*
- void [SetSensorColorGreen](#) (const byte &port)  
*Configure an NXT 2.0 green light sensor.*
- void [SetSensorColorBlue](#) (const byte &port)  
*Configure an NXT 2.0 blue light sensor.*
- void [SetSensorColorNone](#) (const byte &port)  
*Configure an NXT 2.0 no light sensor.*
- variant [GetInput](#) (const byte &port, const byte field)  
*Get an input field value.*
- void [SetInput](#) (const byte &port, const int field, variant value)  
*Set an input field value.*
- unsigned int [Sensor](#) (const byte &port)

*Read sensor scaled value.*

- bool [SensorBoolean](#) (const byte port)  
*Read sensor boolean value.*
- byte [SensorDigiPinsDirection](#) (const byte port)  
*Read sensor digital pins direction.*
- byte [SensorDigiPinsOutputLevel](#) (const byte port)  
*Read sensor digital pins output level.*
- byte [SensorDigiPinsStatus](#) (const byte port)  
*Read sensor digital pins status.*
- bool [SensorInvalid](#) (const byte &port)  
*Read sensor invalid data flag.*
- byte [SensorMode](#) (const byte &port)  
*Read sensor mode.*
- unsigned int [SensorNormalized](#) (const byte &port)  
*Read sensor normalized value.*
- unsigned int [SensorRaw](#) (const byte &port)  
*Read sensor raw value.*
- unsigned int [SensorScaled](#) (const byte &port)  
*Read sensor scaled value.*
- byte [SensorType](#) (const byte &port)  
*Read sensor type.*
- unsigned int [SensorValue](#) (const byte &port)  
*Read sensor scaled value.*
- bool [SensorValueBool](#) (const byte port)  
*Read sensor boolean value.*
- unsigned int [SensorValueRaw](#) (const byte &port)  
*Read sensor raw value.*
- byte [CustomSensorActiveStatus](#) (byte port)

*Get the custom sensor active status.*

- byte [CustomSensorPercentFullScale](#) (byte port)  
*Get the custom sensor percent full scale.*
- unsigned int [CustomSensorZeroOffset](#) (byte port)  
*Get the custom sensor zero offset.*
- void [SetCustomSensorActiveStatus](#) (byte port, byte activeStatus)  
*Set active status.*
- void [SetCustomSensorPercentFullScale](#) (byte port, byte pctFullScale)  
*Set percent full scale.*
- void [SetCustomSensorZeroOffset](#) (byte port, int zeroOffset)  
*Set custom zero offset.*
- void [SetSensorBoolean](#) (byte port, bool value)  
*Set sensor boolean value.*
- void [SetSensorDigiPinsDirection](#) (byte port, byte direction)  
*Set digital pins direction.*
- void [SetSensorDigiPinsOutputLevel](#) (byte port, byte outputLevel)  
*Set digital pins output level.*
- void [SetSensorDigiPinsStatus](#) (byte port, byte status)  
*Set digital pins status.*
- void [SysColorSensorRead](#) ([ColorSensorReadType](#) &args)  
*Read LEGO color sensor.*
- int [ReadSensorColorEx](#) (const byte &port, int &colorval, unsigned int &raw[ ], unsigned int &norm[ ], int &scaled[ ])  
*Read LEGO color sensor extra.*
- int [ReadSensorColorRaw](#) (const byte &port, unsigned int &rawVals[ ])  
*Read LEGO color sensor raw values.*
- unsigned int [ColorADRaw](#) (byte port, byte color)  
*Read a LEGO color sensor AD raw value.*
- bool [ColorBoolean](#) (byte port, byte color)

*Read a LEGO color sensor boolean value.*

- long [ColorCalibration](#) (byte port, byte point, byte color)  
*Read a LEGO color sensor calibration point value.*
- byte [ColorCalibrationState](#) (byte port)  
*Read LEGO color sensor calibration state.*
- unsigned int [ColorCalLimits](#) (byte port, byte point)  
*Read a LEGO color sensor calibration limit value.*
- unsigned int [ColorSensorRaw](#) (byte port, byte color)  
*Read a LEGO color sensor raw value.*
- unsigned int [ColorSensorValue](#) (byte port, byte color)  
*Read a LEGO color sensor scaled value.*
- void [SysInputPinFunction](#) ([InputPinFunctionType](#) &args)  
*Execute the Input module pin function.*

### 6.39.1 Detailed Description

Functions for accessing and modifying input module features.

### 6.39.2 Function Documentation

#### 6.39.2.1 void ClearSensor (const byte & port) [**inline**]

Clear a sensor value. Clear the value of a sensor - only affects sensors that are configured to measure a cumulative quantity such as rotation or a pulse count.

#### Parameters:

*port* The port to clear. See [Input port constants](#).

#### Examples:

[ex\\_ClearSensor.nxc](#).

**6.39.2.2 unsigned int ColorADRaw (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor AD raw value. This function lets you directly access a specific LEGO color sensor AD raw value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The AD raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorADRaw.nxc](#).

**6.39.2.3 bool ColorBoolean (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor boolean value. This function lets you directly access a specific LEGO color sensor boolean value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The boolean value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorBoolean.nxc](#).

#### 6.39.2.4 long ColorCalibration (byte *port*, byte *point*, byte *color*) [inline]

Read a LEGO color sensor calibration point value. This function lets you directly access a specific LEGO color calibration point value. The port, point, and color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*point* The calibration point. See [Color calibration constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The calibration point value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalibration.nxc](#).

#### 6.39.2.5 byte ColorCalibrationState (byte *port*) [inline]

Read LEGO color sensor calibration state. This function lets you directly access the LEGO color calibration state. The port must be a constant.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The calibration state.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalibrationState.nxc](#).

**6.39.2.6 unsigned int ColorCalLimits (byte *port*, byte *point*) [inline]**

Read a LEGO color sensor calibration limit value. This function lets you directly access a specific LEGO color calibration limit value. The port and the point must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*point* The calibration point. See [Color calibration constants](#).

**Returns:**

The calibration limit value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalLimits.nxc](#).

**6.39.2.7 unsigned int ColorSensorRaw (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor raw value. This function lets you directly access a specific LEGO color sensor raw value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorSensorRaw.nxc](#).

**6.39.2.8 unsigned int ColorSensorValue (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor scaled value. This function lets you directly access a specific LEGO color sensor scaled value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The scaled value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorSensorValue.nxc](#).

**6.39.2.9 byte CustomSensorActiveStatus (byte *port*) [inline]**

Get the custom sensor active status. Return the custom sensor active status value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor active status.

**Examples:**

[ex\\_CustomSensorActiveStatus.nxc](#).

**6.39.2.10** byte CustomSensorPercentFullScale (byte *port*) [**inline**]

Get the custom sensor percent full scale. Return the custom sensor percent full scale value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor percent full scale.

**Examples:**

[ex\\_CustomSensorPercentFullScale.nxc](#).

**6.39.2.11** unsigned int CustomSensorZeroOffset (byte *port*) [**inline**]

Get the custom sensor zero offset. Return the custom sensor zero offset value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor zero offset.

**Examples:**

[ex\\_CustomSensorZeroOffset.nxc](#).

**6.39.2.12** variant GetInput (const byte & *port*, const byte *field*) [**inline**]

Get an input field value. Return the value of the specified field of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

*field* An input field constant. See [Input field constants](#).

**Returns:**

The input field value.

**Examples:**

[ex\\_GetInput.nxc](#).

**6.39.2.13 int ReadSensorColorEx (const byte & port, int & colorval, unsigned int & raw[], unsigned int & norm[], int & scaled[]) [inline]**

Read LEGO color sensor extra. This function lets you read the LEGO color sensor. It returns the color value, and three arrays containing raw, normalized, and scaled color values for red, green, blue, and none indices.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*colorval* The color value. See [Color values](#).

*raw* An array containing four raw color values. See [Color sensor array indices](#).

*norm* An array containing four normalized color values. See [Color sensor array indices](#).

*scaled* An array containing four scaled color values. See [Color sensor array indices](#).

**Returns:**

The function call result.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ReadSensorColorEx.nxc](#).

**6.39.2.14 int ReadSensorColorRaw (const byte & port, unsigned int & rawVals[]) [inline]**

Read LEGO color sensor raw values. This function lets you read the LEGO color sensor. It returns an array containing raw color values for red, green, blue, and none indices.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*rawVals* An array containing four raw color values. See [Color sensor array indices](#).

**Returns:**

The function call result.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ReadSensorColorRaw.nxc](#).

**6.39.2.15 void ResetSensor (const byte & port) [inline]**

Reset the sensor port. Sets the invalid data flag on the specified port and waits for it to become valid again. After changing the type or the mode of a sensor port you must call this function to give the firmware time to reconfigure the sensor port.

**Parameters:**

*port* The port to reset. See [Input port constants](#).

**Examples:**

[ex\\_ResetSensor.nxc](#).

**6.39.2.16 unsigned int Sensor (const byte & port) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)).

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_Sensor.nxc](#), and [ex\\_SysComputeCalibValue.nxc](#).

**6.39.2.17 bool SensorBoolean (const byte port) [inline]**

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling SetSensorMode.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's boolean value.

**Examples:**

[ex\\_SensorBoolean.nxc](#).

**6.39.2.18 byte SensorDigiPinsDirection (const byte port) [inline]**

Read sensor digital pins direction. Return the digital pins direction value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins direction.

**Examples:**

[ex\\_SensorDigiPinsDirection.nxc](#).

**6.39.2.19** byte SensorDigiPinsOutputLevel (const byte *port*) [**inline**]

Read sensor digital pins output level. Return the digital pins output level value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins output level.

**Examples:**

[ex\\_SensorDigiPinsOutputLevel.nxc](#).

**6.39.2.20** byte SensorDigiPinsStatus (const byte *port*) [**inline**]

Read sensor digital pins status. Return the digital pins status value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins status.

**Examples:**

[ex\\_SensorDigiPinsStatus.nxc](#).

**6.39.2.21** bool SensorInvalid (const byte & *port*) [**inline**]

Read sensor invalid data flag. Return the value of the InvalidData flag of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's invalid data flag.

**Examples:**

[ex\\_SensorInvalid.nxc](#).

**6.39.2.22 byte SensorMode (const byte & port) [inline]**

Read sensor mode. Return the mode of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's mode. See [Sensor mode constants](#).

**Examples:**

[ex\\_SensorMode.nxc](#).

**6.39.2.23 unsigned int SensorNormalized (const byte & port) [inline]**

Read sensor normalized value. Return the normalized value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's normalized value.

**Examples:**

[ex\\_SensorNormalized.nxc](#).

**6.39.2.24 unsigned int SensorRaw (const byte & port) [inline]**

Read sensor raw value. Return the raw value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's raw value.

**Examples:**

[ex\\_SensorRaw.nxc](#).

**6.39.2.25 unsigned int SensorScaled (const byte & port) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)) or the [Sensor](#) function.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_SensorScaled.nxc](#).

**6.39.2.26 byte SensorType (const byte & port) [inline]**

Read sensor type. Return the type of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's type. See [Sensor type constants](#).

**Examples:**

[ex\\_SensorType.nxc](#).

**6.39.2.27 unsigned int SensorValue (const byte & port) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)) or the [Sensor](#) function.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_SensorValue.nxc](#).

**6.39.2.28 bool SensorValueBool (const byte port) [inline]**

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling [SetSensorMode](#).

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's boolean value.

**Examples:**

[ex\\_SensorValueBool.nxc](#).

**6.39.2.29 unsigned int SensorValueRaw (const byte & port) [inline]**

Read sensor raw value. Return the raw value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's raw value.

**Examples:**

[ex\\_SensorValueRaw.nxc](#).

**6.39.2.30 void SetCustomSensorActiveStatus (byte port, byte activeStatus) [inline]**

Set active status. Sets the active status value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*activeStatus* The new active status value.

**Examples:**

[ex\\_SetCustomSensorActiveStatus.nxc](#).

**6.39.2.31** void SetCustomSensorPercentFullScale (byte *port*, byte *pctFullScale*)  
[inline]

Set percent full scale. Sets the percent full scale value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*pctFullScale* The new percent full scale value.

**Examples:**

[ex\\_SetCustomSensorPercentFullScale.nxc](#).

**6.39.2.32** void SetCustomSensorZeroOffset (byte *port*, int *zeroOffset*)  
[inline]

Set custom zero offset. Sets the zero offset value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*zeroOffset* The new zero offset value.

**Examples:**

[ex\\_SetCustomSensorZeroOffset.nxc](#).

**6.39.2.33** void SetInput (const byte & *port*, const int *field*, variant *value*)  
[inline]

Set an input field value. Set the specified field of the sensor on the specified port to the value provided.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

*field* An input field constant. See [Input field constants](#).

*value* The new value, which may be any valid expression.

**Examples:**

[ex\\_SetInput.nxc](#).

**6.39.2.34 void SetSensor (const byte & *port*, const unsigned int *config*) [inline]**

Set sensor configuration. Set the type and mode of the given sensor to the specified configuration, which must be a special constant containing both type and mode information.

**See also:**

[SetSensorType\(\)](#), [SetSensorMode\(\)](#), and [ResetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*config* The configuration constant containing both the type and mode. See [Combined sensor type and mode constants](#).

**Examples:**

[ex\\_SetSensor.nxc](#).

**6.39.2.35 void SetSensorBoolean (byte *port*, bool *value*) [inline]**

Set sensor boolean value. Sets the boolean value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The new boolean value.

**6.39.2.36 void SetSensorColorBlue (const byte & *port*) [inline]**

Configure an NXT 2.0 blue light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in blue light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorblue.nxc](#).

**6.39.2.37 void SetSensorColorFull (const byte & port) [inline]**

Configure an NXT 2.0 full color sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in full color mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorfull.nxc](#), and [ex\\_SysColorSensorRead.nxc](#).

**6.39.2.38 void SetSensorColorGreen (const byte & port) [inline]**

Configure an NXT 2.0 green light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in green light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorgreen.nxc](#).

**6.39.2.39 void SetSensorColorNone (const byte & *port*) [inline]**

Configure an NXT 2.0 no light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in no light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolornone.nxc](#).

**6.39.2.40 void SetSensorColorRed (const byte & *port*) [inline]**

Configure an NXT 2.0 red light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in red light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorred.nxc](#).

**6.39.2.41 void SetSensorDigiPinsDirection (byte *port*, byte *direction*) [inline]**

Set digital pins direction. Sets the digital pins direction value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*direction* The new digital pins direction value.

**Examples:**

[ex\\_SetSensorDigiPinsDirection.nxc](#).

**6.39.2.42 void SetSensorDigiPinsOutputLevel (byte *port*, byte *outputLevel*)  
[inline]**

Set digital pins output level. Sets the digital pins output level value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*outputLevel* The new digital pins output level value.

**Examples:**

[ex\\_SetSensorDigiPinsOutputLevel.nxc](#).

**6.39.2.43 void SetSensorDigiPinsStatus (byte *port*, byte *status*) [inline]**

Set digital pins status. Sets the digital pins status value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*status* The new digital pins status value.

**Examples:**

[ex\\_SetSensorDigiPinsStatus.nxc](#).

**6.39.2.44 void SetSensorEMeter (const byte & *port*) [inline]**

Configure an EMeter sensor. Configure the sensor on the specified port as an EMeter sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorEMeter.nxc](#).

**6.39.2.45 void SetSensorLight (const byte & port, bool bActive = true)  
[inline]**

Configure a light sensor. Configure the sensor on the specified port as an NXT light sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bActive* A boolean flag indicating whether to configure the port as an active or inactive light sensor. The default value for this optional parameter is true.

**Examples:**

[ex\\_SetSensorLight.nxc](#).

**6.39.2.46 void SetSensorLowspeed (const byte & port, bool bIsPowered = true)  
[inline]**

Configure an I2C sensor. Configure the sensor on the specified port as an I2C digital sensor for either powered (9 volt) or unpowered devices.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bIsPowered* A boolean flag indicating whether to configure the port for powered or unpowered I2C devices. The default value for this optional parameter is true.

**Examples:**

[ex\\_digps.nxc](#), [ex\\_HTRCXSetIRLinkPort.nxc](#), [ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#),

[ex\\_PFMate.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_ResetSensorHTAngle.nxc](#), [ex\\_SetSensorLowspeed.nxc](#), [ex\\_superpro.nxc](#), and [ex\\_xg1300.nxc](#).

#### 6.39.2.47 void SetSensorMode (const byte & *port*, byte *mode*) [**inline**]

Set sensor mode. Set a sensor's mode, which should be one of the predefined sensor mode constants. A slope parameter for boolean conversion, if desired, may be added to the mode. After changing the type or the mode of a sensor port you must call [ResetSensor](#) to give the firmware time to reconfigure the sensor port.

See also:

[SetSensorType\(\)](#), [SetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*mode* The desired sensor mode. See [Sensor mode constants](#).

**Examples:**

[ex\\_SetSensorMode.nxc](#).

#### 6.39.2.48 void SetSensorSound (const byte & *port*, bool *bdbScaling* = true) [**inline**]

Configure a sound sensor. Configure the sensor on the specified port as a sound sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bdbScaling* A boolean flag indicating whether to configure the port as a sound sensor with dB or dBA scaling. The default value for this optional parameter is true, meaning dB scaling.

**Examples:**

[ex\\_SetSensorSound.nxc](#).

**6.39.2.49 void SetSensorTemperature (const byte & port) [inline]**

Configure a temperature sensor. Configure the sensor on the specified port as a temperature sensor. Use this to setup the temperature sensor rather than [SetSensorLowSpeed](#) so that the sensor is properly configured in 12-bit conversion mode.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorTemperature.nxc](#).

**6.39.2.50 void SetSensorTouch (const byte & port) [inline]**

Configure a touch sensor. Configure the sensor on the specified port as a touch sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_ReadSensorHTTouchMultiplexer.nxc](#), and [ex\\_SetSensorTouch.nxc](#).

**6.39.2.51 void SetSensorType (const byte & port, byte type) [inline]**

Set sensor type. Set a sensor's type, which must be one of the predefined sensor type constants. After changing the type or the mode of a sensor port you must call [ResetSensor](#) to give the firmware time to reconfigure the sensor port.

**See also:**

[SetSensorMode\(\)](#), [SetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*type* The desired sensor type. See [Sensor type constants](#).

**Examples:**

[ex\\_SetSensorType.nxc](#).

**6.39.2.52 void SetSensorUltrasonic (const byte & port) [inline]**

Configure an ultrasonic sensor. Configure the sensor on the specified port as an ultrasonic sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorUltrasonic.nxc](#).

**6.39.2.53 void SysColorSensorRead (ColorSensorReadType & args) [inline]**

Read LEGO color sensor. This function lets you read the LEGO color sensor given the parameters you pass in via the [ColorSensorReadType](#) structure.

**Parameters:**

*args* The [ColorSensorReadType](#) structure containing the required parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysColorSensorRead.nxc](#).

**6.39.2.54 void SysInputPinFunction (InputPinFunctionType & args) [inline]**

Execute the Input module pin function. This function lets you execute the Input module's pin function using the values specified via the [InputPinFunctionType](#) structure.

**Parameters:**

*args* The [InputPinFunctionType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysinputpinfunction.nxc](#).

## 6.40 Basic analog sensor value names

Read analog sensor values using these names.

**Defines**

- #define [SENSOR\\_1](#) Sensor(S1)
- #define [SENSOR\\_2](#) Sensor(S2)
- #define [SENSOR\\_3](#) Sensor(S3)
- #define [SENSOR\\_4](#) Sensor(S4)

### 6.40.1 Detailed Description

Read analog sensor values using these names. Returns the current scaled value of the sensor on the specified port.

### 6.40.2 Define Documentation

#### 6.40.2.1 #define [SENSOR\\_1](#) Sensor(S1)

Read the value of the analog sensor on port S1

#### 6.40.2.2 #define [SENSOR\\_2](#) Sensor(S2)

Read the value of the analog sensor on port S2

#### 6.40.2.3 #define [SENSOR\\_3](#) Sensor(S3)

Read the value of the analog sensor on port S3

#### 6.40.2.4 #define SENSOR\_4 Sensor(S4)

Read the value of the analog sensor on port S4

## 6.41 Output module types

Types used by various output module functions.

### Data Structures

- struct [OutputStateType](#)  
*Parameters for the [RemoteGetOutputState](#) function.*

#### 6.41.1 Detailed Description

Types used by various output module functions.

## 6.42 Output module functions

Functions for accessing and modifying output module features.

### Functions

- void [SetMotorPwnFreq](#) (byte n)  
*Set motor regulation frequency.*
- void [SetMotorRegulationTime](#) (byte n)  
*Set regulation time.*
- void [SetMotorRegulationOptions](#) (byte n)  
*Set regulation options.*
- void [OnFwdSyncPID](#) (byte outputs, char pwr, char turnpct, byte p, byte i, byte d)  
*Run motors forward synchronised with PID factors.*
- void [OnFwdSyncExPID](#) (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d)  
*Run motors forward synchronised and reset counters with PID factors.*

- void **OnRevSyncPID** (byte outputs, char pwr, char turnpct, byte p, byte i, byte d)  
*Run motors backward synchronised with PID factors.*
- void **OnRevSyncExPID** (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d)  
*Run motors backward synchronised and reset counters with PID factors.*
- void **OnFwdRegPID** (byte outputs, char pwr, byte regmode, byte p, byte i, byte d)  
*Run motors forward regulated with PID factors.*
- void **OnFwdRegExPID** (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d)  
*Run motors forward regulated and reset counters with PID factors.*
- void **OnRevRegPID** (byte outputs, char pwr, byte regmode, byte p, byte i, byte d)  
*Run motors reverse regulated with PID factors.*
- void **OnRevRegExPID** (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d)  
*Run motors backward regulated and reset counters with PID factors.*
- void **Off** (byte outputs)  
*Turn motors off.*
- void **OffEx** (byte outputs, const byte reset)  
*Turn motors off and reset counters.*
- void **Coast** (byte outputs)  
*Coast motors.*
- void **CoastEx** (byte outputs, const byte reset)  
*Coast motors and reset counters.*
- void **Float** (byte outputs)  
*Float motors.*
- void **OnFwd** (byte outputs, char pwr)  
*Run motors forward.*
- void **OnFwdEx** (byte outputs, char pwr, const byte reset)

*Run motors forward and reset counters.*

- void **OnRev** (byte outputs, char pwr)  
*Run motors backward.*
- void **OnRevEx** (byte outputs, char pwr, const byte reset)  
*Run motors backward and reset counters.*
- void **OnFwdReg** (byte outputs, char pwr, byte regmode)  
*Run motors forward regulated.*
- void **OnFwdRegEx** (byte outputs, char pwr, byte regmode, const byte reset)  
*Run motors forward regulated and reset counters.*
- void **OnRevReg** (byte outputs, char pwr, byte regmode)  
*Run motors forward regulated.*
- void **OnRevRegEx** (byte outputs, char pwr, byte regmode, const byte reset)  
*Run motors backward regulated and reset counters.*
- void **OnFwdSync** (byte outputs, char pwr, char turnpct)  
*Run motors forward synchronised.*
- void **OnFwdSyncEx** (byte outputs, char pwr, char turnpct, const byte reset)  
*Run motors forward synchronised and reset counters.*
- void **OnRevSync** (byte outputs, char pwr, char turnpct)  
*Run motors backward synchronised.*
- void **OnRevSyncEx** (byte outputs, char pwr, char turnpct, const byte reset)  
*Run motors backward synchronised and reset counters.*
- void **RotateMotor** (byte outputs, char pwr, long angle)  
*Rotate motor.*
- void **RotateMotorPID** (byte outputs, char pwr, long angle, byte p, byte i, byte d)  
*Rotate motor with PID factors.*
- void **RotateMotorEx** (byte outputs, char pwr, long angle, char turnpct, bool sync, bool stop)  
*Rotate motor.*

- void [RotateMotorExpID](#) (byte outputs, char pwr, long angle, char turnpct, bool sync, bool stop, byte p, byte i, byte d)  
*Rotate motor.*
- void [ResetTachoCount](#) (byte outputs)  
*Reset tachometer counter.*
- void [ResetBlockTachoCount](#) (byte outputs)  
*Reset block-relative counter.*
- void [ResetRotationCount](#) (byte outputs)  
*Reset program-relative counter.*
- void [ResetAllTachoCounts](#) (byte outputs)  
*Reset all tachometer counters.*
- void [SetOutput](#) (byte outputs, byte field1, variant val1,..., byte fieldN, variant valN)  
*Set output fields.*
- variant [GetOutput](#) (byte output, const byte field)  
*Get output field value.*
- byte [MotorMode](#) (byte output)  
*Get motor mode.*
- char [MotorPower](#) (byte output)  
*Get motor power level.*
- char [MotorActualSpeed](#) (byte output)  
*Get motor actual speed.*
- long [MotorTachoCount](#) (byte output)  
*Get motor tachometer counter.*
- long [MotorTachoLimit](#) (byte output)  
*Get motor tachometer limit.*
- byte [MotorRunState](#) (byte output)  
*Get motor run state.*
- char [MotorTurnRatio](#) (byte output)  
*Get motor turn ratio.*

- byte [MotorRegulation](#) (byte output)  
*Get motor regulation mode.*
- bool [MotorOverload](#) (byte output)  
*Get motor overload status.*
- byte [MotorRegPValue](#) (byte output)  
*Get motor P value.*
- byte [MotorRegIValue](#) (byte output)  
*Get motor I value.*
- byte [MotorRegDValue](#) (byte output)  
*Get motor D value.*
- long [MotorBlockTachoCount](#) (byte output)  
*Get motor block-relative counter.*
- long [MotorRotationCount](#) (byte output)  
*Get motor program-relative counter.*
- byte [MotorOutputOptions](#) (byte output)  
*Get motor options.*
- byte [MotorMaxSpeed](#) (byte output)  
*Get motor max speed.*
- byte [MotorMaxAcceleration](#) (byte output)  
*Get motor max acceleration.*
- byte [MotorPwnFreq](#) ()  
*Get motor regulation frequency.*
- byte [MotorRegulationTime](#) ()  
*Get motor regulation time.*
- byte [MotorRegulationOptions](#) ()  
*Get motor regulation options.*
- void [PosRegEnable](#) (byte output, byte p=PID\_3, byte i=PID\_1, byte d=PID\_1)  
*Enable absolute position regulation with PID factors.*

- void [PosRegSetAngle](#) (byte output, long angle)  
*Change the current value for set angle.*
- void [PosRegAddAngle](#) (byte output, long angle\_add)  
*Add to the current value for set angle.*
- void [PosRegSetMax](#) (byte output, byte max\_speed, byte max\_acceleration)  
*Set maximum limits.*

### 6.42.1 Detailed Description

Functions for accessing and modifying output module features.

### 6.42.2 Function Documentation

#### 6.42.2.1 void [Coast](#) (byte *outputs*) [**inline**]

Coast motors. Turn off the specified outputs, making them coast to a stop.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

#### Examples:

[ex\\_coast.nxc](#).

#### 6.42.2.2 void [CoastEx](#) (byte *outputs*, const byte *reset*) [**inline**]

Coast motors and reset counters. Turn off the specified outputs, making them coast to a stop.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a

single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_coastex.nxc](#).

**6.42.2.3 void Float (byte *outputs*) [inline]**

Float motors. Make outputs float. Float is an alias for Coast.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

**Examples:**

[ex\\_float.nxc](#).

**6.42.2.4 variant GetOutput (byte *output*, const byte *field*) [inline]**

Get output field value. Get the value of the specified field for the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

*field* Output port field to access, this should be a constant, see [Output field constants](#).

**Returns:**

The requested output field value.

**Examples:**

[ex\\_getoutput.nxc](#).

#### 6.42.2.5 char MotorActualSpeed (byte *output*) [inline]

Get motor actual speed. Get the actual speed value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The actual speed value of the specified output.

**Examples:**

[ex\\_motoractualspeed.nxc](#).

#### 6.42.2.6 long MotorBlockTachoCount (byte *output*) [inline]

Get motor block-relative counter. Get the block-relative position counter value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The block-relative position counter value of the specified output.

**Examples:**

[ex\\_motorblocktachocount.nxc](#).

#### 6.42.2.7 byte MotorMaxAcceleration (byte *output*) [inline]

Get motor max acceleration. Get the max acceleration value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The max acceleration value of the specified output.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.8 byte MotorMaxSpeed (byte *output*) [inline]**

Get motor max speed. Get the max speed value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The max speed value of the specified output.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.9 byte MotorMode (byte *output*) [inline]**

Get motor mode. Get the mode of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The mode of the specified output.

**Examples:**

[ex\\_motormode.nxc](#).

**6.42.2.10 byte MotorOutputOptions (byte *output*) [inline]**

Get motor options. Get the options value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The options value of the specified output.

**Examples:**

[ex\\_motoroutputoptions.nxc](#).

**6.42.2.11 bool MotorOverload (byte *output*) [inline]**

Get motor overload status. Get the overload value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The overload value of the specified output.

**Examples:**

[ex\\_motoroverload.nxc](#).

**6.42.2.12 char MotorPower (byte *output*) [inline]**

Get motor power level. Get the power level of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The power level of the specified output.

**Examples:**

[ex\\_motorpower.nxc](#).

**6.42.2.13 byte MotorPwnFreq () [inline]**

Get motor regulation frequency. Get the current motor regulation frequency in milliseconds.

**Returns:**

The motor regulation frequency.

**Examples:**

[ex\\_motorpwnfreq.nxc](#).

**6.42.2.14 byte MotorRegDValue (byte *output*) [inline]**

Get motor D value. Get the derivative PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The derivative PID value of the specified output.

**Examples:**

[ex\\_motorregdvalue.nxc](#).

**6.42.2.15 byte MotorRegIValue (byte *output*) [inline]**

Get motor I value. Get the integral PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The integral PID value of the specified output.

**Examples:**

[ex\\_motorregivalue.nxc](#).

**6.42.2.16 byte MotorRegPValue (byte *output*) [inline]**

Get motor P value. Get the proportional PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The proportional PID value of the specified output.

**Examples:**

[ex\\_motorregpvalue.nxc](#).

**6.42.2.17 byte MotorRegulation (byte *output*) [inline]**

Get motor regulation mode. Get the regulation value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The regulation value of the specified output.

**Examples:**

[ex\\_motorregulation.nxc](#).

**6.42.2.18 byte MotorRegulationOptions () [inline]**

Get motor regulation options. Get the current motor regulation options.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Returns:**

The motor regulation options.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.19 byte MotorRegulationTime () [inline]**

Get motor regulation time. Get the current motor regulation time in milliseconds.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Returns:**

The motor regulation time.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.20 long MotorRotationCount (byte output) [inline]**

Get motor program-relative counter. Get the program-relative position counter value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The program-relative position counter value of the specified output.

**Examples:**

[ex\\_motorrotationcount.nxc](#), and [util\\_rpm.nxc](#).

**6.42.2.21 byte MotorRunState (byte output) [inline]**

Get motor run state. Get the RunState value of the specified output, see [Output port run state constants](#).

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The RunState value of the specified output.

**Examples:**

[ex\\_motorrynstate.nxc](#).

**6.42.2.22 long MotorTachoCount (byte output) [inline]**

Get motor tachometer counter. Get the tachometer count value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The tachometer count value of the specified output.

**Examples:**

[ex\\_motortachocount.nxc](#).

**6.42.2.23 long MotorTachoLimit (byte output) [inline]**

Get motor tachometer limit. Get the tachometer limit value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The tachometer limit value of the specified output.

**Examples:**

[ex\\_motortacholimit.nxc](#).

**6.42.2.24 char MotorTurnRatio (byte output) [inline]**

Get motor turn ratio. Get the turn ratio value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The turn ratio value of the specified output.

**Examples:**

[ex\\_motorturnratio.nxc](#).

**6.42.2.25 void Off (byte *outputs*) [inline]**

Turn motors off. Turn the specified outputs off (with braking).

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

**Examples:**

[ex\\_off.nxc](#).

**6.42.2.26 void OffEx (byte *outputs*, const byte *reset*) [inline]**

Turn motors off and reset counters. Turn the specified outputs off (with braking).

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_offex.nxc](#).

**6.42.2.27 void OnFwd (byte *outputs*, char *pwr*) [inline]**

Run motors forward. Set outputs to forward direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

**Examples:**

[ex\\_onfwd.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

**6.42.2.28 void OnFwdEx (byte *outputs*, char *pwr*, const byte *reset*) [inline]**

Run motors forward and reset counters. Set outputs to forward direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdex.nxc](#).

**6.42.2.29 void OnFwdReg (byte *outputs*, char *pwr*, byte *regmode*) [inline]**

Run motors forward regulated. Run the specified outputs forward using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

**Examples:**

[ex\\_onfwdreg.nxc](#).

**6.42.2.30** `void OnFwdRegEx (byte outputs, char pwr, byte regmode, const byte reset) [inline]`

Run motors forward regulated and reset counters. Run the specified outputs forward using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdregex.nxc](#).

**6.42.2.31** `void OnFwdRegExPID (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d) [inline]`

Run motors forward regulated and reset counters with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- regmode* Regulation mode, see [Output port regulation mode constants](#).
- reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).
- p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdregexpid.nxc](#).

**6.42.2.32** `void OnFwdRegPID (byte outputs, char pwr, byte regmode, byte p, byte i, byte d) [inline]`

Run motors forward regulated with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- regmode* Regulation mode, see [Output port regulation mode constants](#).
- p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdregpid.nxc](#).

**6.42.2.33 void OnFwdSync (byte *outputs*, char *pwr*, char *turnpct*) [inline]**

Run motors forward synchronised. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

**Examples:**

[ex\\_onfwdsync.nxc](#).

**6.42.2.34 void OnFwdSyncEx (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*) [inline]**

Run motors forward synchronised and reset counters. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdsyncex.nxc](#).

**6.42.2.35 void OnFwdSyncExPID (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*, byte *p*, byte *i*, byte *d*) [inline]**

Run motors forward synchronised and reset counters with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdsyncexpid.nxc](#).

**6.42.2.36 void OnFwdSyncPID (byte *outputs*, char *pwr*, char *turnpct*, byte *p*, byte *i*, byte *d*) [inline]**

Run motors forward synchronised with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdsyncpid.nxc](#).

**6.42.2.37 void OnRev (byte outputs, char pwr) [inline]**

Run motors backward. Set outputs to reverse direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

**Examples:**

[ex\\_onrev.nxc](#).

**6.42.2.38 void OnRevEx (byte outputs, char pwr, const byte reset) [inline]**

Run motors backward and reset counters. Set outputs to reverse direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onrevex.nxc](#).

**6.42.2.39 void OnRevReg (byte outputs, char pwr, byte regmode) [inline]**

Run motors forward regulated. Run the specified outputs in reverse using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

**Examples:**

[ex\\_onrevreg.nxc](#).

**6.42.2.40 void OnRevRegEx (byte outputs, char pwr, byte regmode, const byte reset) [inline]**

Run motors backward regulated and reset counters. Run the specified outputs in reverse using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a

single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

#### Examples:

[ex\\_onrevregex.nxc](#).

#### 6.42.2.41 void OnRevRegExPID (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*, byte *p*, byte *i*, byte *d*) [inline]

Run motors backward regulated and reset counters with PID factors. Run the specified outputs in reverse using the specified regulation mode. Specify proportional, integral, and derivative factors.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

#### Examples:

[ex\\_onrevregexpid.nxc](#).

**6.42.2.42 void OnRevRegPID (byte *outputs*, char *pwr*, byte *regmode*, byte *p*, byte *i*, byte *d*) [inline]**

Run motors reverse regulated with PID factors. Run the specified outputs in reverse using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevregpid.nxc](#).

**6.42.2.43 void OnRevSync (byte *outputs*, char *pwr*, char *turnpct*) [inline]**

Run motors backward synchronised. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

**Examples:**

[ex\\_onrevsync.nxc](#).

**6.42.2.44 void OnRevSyncEx (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*) [inline]**

Run motors backward synchronised and reset counters. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onrevsyncex.nxc](#).

**6.42.2.45 void OnRevSyncExPID (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*, byte *p*, byte *i*, byte *d*) [inline]**

Run motors backward synchronised and reset counters with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevsyncexpid.nxc](#).

**6.42.2.46** `void OnRevSyncPID (byte outputs, char pwr, char turnpct, byte p, byte i, byte d) [inline]`

Run motors backward synchronised with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevsyncpid.nxc](#).

**6.42.2.47 void PosRegAddAngle (byte *output*, long *angle\_add*) [inline]**

Add to the current value for set angle. Add an offset to the current set position. Returns immediately, but keep regulating.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*angle\_add* Value to add to the current set position, in degree. Can be negative. Can be greater than 360 degree to make several turns.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.48 void PosRegEnable (byte *output*, byte *p* = PID\_3, byte *i* = PID\_1, byte *d* = PID\_1) [inline]**

Enable absolute position regulation with PID factors. Enable absolute position regulation on the specified output. Motor is kept regulated as long as this is enabled. Optionally specify proportional, integral, and derivative factors.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_3](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_1](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_1](#).

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.49 void PosRegSetAngle (byte *output*, long *angle*) [inline]**

Change the current value for set angle. Make the absolute position regulation going toward the new provided angle. Returns immediately, but keep regulating.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*angle* New set position, in degree. The 0 angle corresponds to the position of the motor when absolute position regulation was first enabled. Can be negative. Can be greater than 360 degree to make several turns.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.50 void PosRegSetMax (byte *output*, byte *max\_speed*, byte *max\_acceleration*) [inline]**

Set maximum limits. Set maximum speed and acceleration.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*max\_speed* Maximum speed, or 0 to disable speed limiting.

*max\_acceleration* Maximum acceleration, or 0 to disable acceleration limiting. The *max\_speed* parameter should not be 0 if this is not 0.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.51 void ResetAllTachoCounts (byte *outputs*) [inline]**

Reset all tachometer counters. Reset all three position counters and reset the current tachometer limit goal for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resetalltachocounts.nxc](#).

**6.42.2.52 void ResetBlockTachoCount (byte *outputs*) [inline]**

Reset block-relative counter. Reset the block-relative position counter for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resetblocktachocount.nxc](#).

**6.42.2.53 void ResetRotationCount (byte *outputs*) [inline]**

Reset program-relative counter. Reset the program-relative position counter for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resetrotationcount.nxc](#).

**6.42.2.54 void ResetTachoCount (byte *outputs*) [inline]**

Reset tachometer counter. Reset the tachometer count and tachometer limit goal for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resettachocount.nxc](#).

**6.42.2.55 void RotateMotor (byte *outputs*, char *pwr*, long *angle*) [inline]**

Rotate motor. Run the specified outputs forward for the specified number of degrees.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*angle* Angle limit, in degree. Can be negative to reverse direction.

**Examples:**

[ex\\_rotatemotor.nxc](#).

**6.42.2.56 void RotateMotorEx (byte *outputs*, char *pwr*, long *angle*, char *turnpct*, bool *sync*, bool *stop*) [inline]**

Rotate motor. Run the specified outputs forward for the specified number of degrees.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*angle* Angle limit, in degree. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*sync* Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.

*stop* Specify whether the motor(s) should brake at the end of the rotation.

**Examples:**

[ex\\_rotatemotorex.nxc](#).

**6.42.2.57 void RotateMotorExpID (byte *outputs*, char *pwr*, long *angle*, char *turnpct*, bool *sync*, bool *stop*, byte *p*, byte *i*, byte *d*) [inline]**

Rotate motor. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*angle* Angle limit, in degree. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*sync* Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.

*stop* Specify whether the motor(s) should brake at the end of the rotation.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_rotatemotorexpid.nxc](#).

**6.42.2.58 void RotateMotorPID (byte *outputs*, char *pwr*, long *angle*, byte *p*, byte *i*, byte *d*) [inline]**

Rotate motor with PID factors. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*angle* Angle limit, in degree. Can be negative to reverse direction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_rotatemotorpid.nxc](#).

**6.42.2.59 void SetMotorPwnFreq (byte *n*) [inline]**

Set motor regulation frequency. Set the motor regulation frequency in milliseconds. By default this is set to 100ms.

**Parameters:**

*n* The motor regulation frequency.

**Examples:**

[ex\\_SetMotorPwnFreq.nxc](#).

**6.42.2.60 void SetMotorRegulationOptions (byte *n*) [inline]**

Set regulation options. Set the motor regulation options.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*n* The motor regulation options.

**Examples:**

[ex\\_PosReg.nxc](#).

**6.42.2.61 void SetMotorRegulationTime (byte *n*) [inline]**

Set regulation time. Set the motor regulation time in milliseconds. By default this is set to 100ms.

**Parameters:**

*n* The motor regulation time.

**Examples:**

[ex\\_PosReg.nxc](#).

### 6.42.2.62 void SetOutput (byte *outputs*, byte *field1*, variant *val1*, ..., byte *fieldN*, variant *valN*) [**inline**]

Set output fields. Set the specified field of the outputs to the value provided. The field must be a valid output field constant. This function takes a variable number of field/value pairs.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

*field1* The 1st output port field to access, this should be a constant, see [Output field constants](#).

*val1* Value to set for the 1st field.

*fieldN* The Nth output port field to access, this should be a constant, see [Output field constants](#).

*valN* The value to set for the Nth field.

#### Examples:

[ex\\_setoutput.nxc](#).

## 6.43 Display module types

Types used by various display module functions.

#### Data Structures

- struct [LocationType](#)  
*A point on the NXT LCD screen.*
- struct [SizeType](#)  
*Width and height dimensions for the DrawRect system call.*
- struct [DrawTextType](#)  
*Parameters for the DrawText system call.*
- struct [DrawPointType](#)  
*Parameters for the DrawPoint system call.*

- struct [DrawLineType](#)  
*Parameters for the DrawLine system call.*
- struct [DrawCircleType](#)  
*Parameters for the DrawCircle system call.*
- struct [DrawRectType](#)  
*Parameters for the DrawRect system call.*
- struct [DrawGraphicType](#)  
*Parameters for the DrawGraphic system call.*
- struct [SetScreenModeType](#)  
*Parameters for the SetScreenMode system call.*
- struct [DisplayExecuteFunctionType](#)  
*Parameters for the DisplayExecuteFunction system call.*
- struct [DrawGraphicArrayType](#)  
*Parameters for the DrawGraphicArray system call.*
- struct [DrawPolygonType](#)  
*Parameters for the DrawPolygon system call.*
- struct [DrawEllipseType](#)  
*Parameters for the DrawEllipse system call.*
- struct [DrawFontType](#)  
*Parameters for the DrawFont system call.*

#### 6.43.1 Detailed Description

Types used by various display module functions.

## 6.44 Display module functions

Functions for accessing and modifying display module features.

### Functions

- void [ResetScreen](#) ()  
*Reset LCD screen.*
- char [CircleOut](#) (int x, int y, byte radius, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a circle.*
- char [LineOut](#) (int x1, int y1, int x2, int y2, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a line.*
- char [PointOut](#) (int x, int y, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a point.*
- char [RectOut](#) (int x, int y, int width, int height, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a rectangle.*
- char [TextOut](#) (int x, int y, string str, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw text.*
- char [NumOut](#) (int x, int y, variant value, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a number.*
- char [EllipseOut](#) (int x, int y, byte radiusX, byte radiusY, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw an ellipse.*
- char [PolyOut](#) ([LocationType](#) points[], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a polygon.*
- char [FontTextOut](#) (int x, int y, string filename, string str, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw text with font.*
- char [FontNumOut](#) (int x, int y, string filename, variant value, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a number with font.*

- char [GraphicOut](#) (int x, int y, string filename, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image.*
- char [GraphicArrayOut](#) (int x, int y, byte data[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image from byte array.*
- char [GraphicOutEx](#) (int x, int y, string filename, byte vars[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image with parameters.*
- char [GraphicArrayOutEx](#) (int x, int y, byte data[ ], byte vars[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image from byte array with parameters.*
- void [GetDisplayNormal](#) (const byte x, const byte line, unsigned int cnt, byte &data[ ])  
*Read pixel data from the normal display buffer.*
- void [SetDisplayNormal](#) (const byte x, const byte line, unsigned int cnt, byte data[ ])  
*Write pixel data to the normal display buffer.*
- void [GetDisplayPopup](#) (const byte x, const byte line, unsigned int cnt, byte &data[ ])  
*Read pixel data from the popup display buffer.*
- void [SetDisplayPopup](#) (const byte x, const byte line, unsigned int cnt, byte data[ ])  
*Write pixel data to the popup display buffer.*
- unsigned long [DisplayEraseMask](#) ()  
*Read the display erase mask value.*
- unsigned long [DisplayUpdateMask](#) ()  
*Read the display update mask value.*
- unsigned long [DisplayFont](#) ()  
*Read the display font memory address.*
- unsigned long [DisplayDisplay](#) ()

*Read the display memory address.*

- byte `DisplayFlags ()`  
*Read the display flags.*
- byte `DisplayTextLinesCenterFlags ()`  
*Read the display text lines center flags.*
- void `SysDrawText (DrawTextType &args)`  
*Draw text.*
- void `SysDrawPoint (DrawPointType &args)`  
*Draw a point.*
- void `SysDrawLine (DrawLineType &args)`  
*Draw a line.*
- void `SysDrawCircle (DrawCircleType &args)`  
*Draw a circle.*
- void `SysDrawRect (DrawRectType &args)`  
*Draw a rectangle.*
- void `SysDrawGraphic (DrawGraphicType &args)`  
*Draw a graphic (RIC file).*
- void `SysSetScreenMode (SetScreenModeType &args)`  
*Set the screen mode.*
- void `SysDisplayExecuteFunction (DisplayExecuteFunctionType &args)`  
*Execute any Display module command.*
- byte `DisplayContrast ()`  
*Read the display contrast setting.*
- void `SysDrawGraphicArray (DrawGraphicArrayType &args)`  
*Draw a graphic image from a byte array.*
- void `SysDrawPolygon (DrawPolygonType &args)`  
*Draw a polygon.*
- void `SysDrawEllipse (DrawEllipseType &args)`

*Draw an ellipse.*

- void **SysDrawFont** (**DrawFontType** &args)  
*Draw text using a custom font.*
- void **ClearScreen** ()  
*Clear LCD screen.*
- void **ClearLine** (byte line)  
*Clear a line on the LCD screen.*
- void **SetDisplayFont** (unsigned long fontaddr)  
*Set the display font memory address.*
- void **SetDisplayDisplay** (unsigned long dispaddr)  
*Set the display memory address.*
- void **SetDisplayEraseMask** (unsigned long eraseMask)  
*Set the display erase mask.*
- void **SetDisplayFlags** (byte flags)  
*Set the display flags.*
- void **SetDisplayTextLinesCenterFlags** (byte ctrFlags)  
*Set the display text lines center flags.*
- void **SetDisplayUpdateMask** (unsigned long updateMask)  
*Set the display update mask.*
- void **SetDisplayContrast** (byte contrast)  
*Set the display contrast.*

#### 6.44.1 Detailed Description

Functions for accessing and modifying display module features.

#### 6.44.2 Function Documentation

- 6.44.2.1** **char CircleOut** (int *x*, int *y*, byte *radius*, unsigned long *options* = **DRAW\_OPT\_NORMAL**) [**inline**]

Draw a circle. This function lets you draw a circle on the screen with its center at the specified *x* and *y* location, using the specified radius. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawCircle](#), [DrawCircleType](#)

**Parameters:**

*x* The *x* value for the center of the circle.

*y* The *y* value for the center of the circle.

*radius* The radius of the circle.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_CircleOut.nxc](#), and [ex\\_file\\_system.nxc](#).

#### 6.44.2.2 void ClearLine (byte *line*) [**inline**]

Clear a line on the LCD screen. This function lets you clear a single line on the NXT LCD.

**Parameters:**

*line* The line you want to clear. See [Line number constants](#).

**Examples:**

[ex\\_clearline.nxc](#), and [ex\\_joystickmsg.nxc](#).

#### 6.44.2.3 void ClearScreen () [**inline**]

Clear LCD screen. This function lets you clear the NXT LCD to a blank screen.

**Examples:**

[ex\\_ClearScreen.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispftout.nxc](#),  
[ex\\_dispgout.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_-  
ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_-  
SetAbortFlag.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_string.nxc](#), [ex\\_sysdrawpolygon.nxc](#),  
[ex\\_sysmemorymanager.nxc](#), and [ex\\_xg1300.nxc](#).

**6.44.2.4 byte DisplayContrast () [inline]**

Read the display contrast setting. This function lets you read the current display contrast setting.

**Returns:**

The current display contrast (byte).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_contrast.nxc](#).

**6.44.2.5 unsigned long DisplayDisplay () [inline]**

Read the display memory address. This function lets you read the current display memory address.

**Returns:**

The current display memory address.

**Examples:**

[ex\\_DisplayDisplay.nxc](#), and [ex\\_dispmisc.nxc](#).

**6.44.2.6 unsigned long DisplayEraseMask () [inline]**

Read the display erase mask value. This function lets you read the current display erase mask value.

**Returns:**

The current display erase mask value.

**Examples:**

[ex\\_DisplayEraseMask.nxc](#), and [ex\\_dispmisc.nxc](#).

**6.44.2.7 byte DisplayFlags () [inline]**

Read the display flags. This function lets you read the current display flags. Valid flag values are listed in the [Display flags](#) group.

**Returns:**

The current display flags.

**Examples:**

[ex\\_DisplayFlags.nxc](#), and [ex\\_dispmisc.nxc](#).

**6.44.2.8 unsigned long DisplayFont () [inline]**

Read the display font memory address. This function lets you read the current display font memory address.

**Returns:**

The current display font memory address.

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_displayfont.nxc](#), and [ex\\_setdisplayfont.nxc](#).

**6.44.2.9 byte DisplayTextLinesCenterFlags () [inline]**

Read the display text lines center flags. This function lets you read the current display text lines center flags.

**Returns:**

The current display text lines center flags.

**Examples:**

[ex\\_DisplayTextLinesCenterFlags.nxc](#), and [ex\\_dispmisc.nxc](#).

**6.44.2.10 unsigned long DisplayUpdateMask () [inline]**

Read the display update mask value. This function lets you read the current display update mask value.

**Returns:**

The current display update mask.

**Examples:**

[ex\\_DisplayUpdateMask.nxc](#), and [ex\\_dispmisc.nxc](#).

**6.44.2.11 char EllipseOut (int *x*, int *y*, byte *radiusX*, byte *radiusY*, unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw an ellipse. This function lets you draw an ellipse on the screen with its center at the specified *x* and *y* location, using the specified radii. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawEllipse](#), [DrawEllipseType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*x* The *x* value for the center of the ellipse.

*y* The *y* value for the center of the ellipse.

*radiusX* The *x* axis radius.

*radiusY* The y axis radius.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_EllipseOut.nxc](#).

**6.44.2.12 char FontNumOut (int x, int y, string filename, variant value, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a number with font. Draw a numeric value on the screen at the specified x and y location using a custom RIC font. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. See the [Font drawing option constants](#) for options specific to the font drawing functions.

**See also:**

[FontTextOut](#), [SysDrawFont](#), [DrawFontType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*x* The x value for the start of the number output.

*y* The y value for the start of the number output.

*filename* The filename of the RIC font.

*value* The value to output to the LCD screen. Any numeric type is supported.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispfnout.nxc](#).

**6.44.2.13 char FontTextOut (int x, int y, string filename, string str, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw text with font. Draw a text value on the screen at the specified x and y location using a custom RIC font. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. See the [Font drawing option constants](#) for options specific to the font drawing functions.

**See also:**

[FontNumOut](#), [SysDrawFont](#), [DrawFontType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

- x* The x value for the start of the text output.
- y* The y value for the start of the text output.
- filename* The filename of the RIC font.
- str* The text to output to the LCD screen.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispftout.nxc](#).

**6.44.2.14 void GetDisplayNormal (const byte x, const byte line, unsigned int cnt, byte & data[]) [inline]**

Read pixel data from the normal display buffer. Read "cnt" bytes from the normal display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use [TEXTLINE\\_1](#) through [TEXTLINE\\_8](#) for the "line" parameter.

**Parameters:**

- x* The desired x position from which to read pixel data.
- line* The desired line from which to read pixel data.
- cnt* The number of bytes of pixel data to read.
- data* The array of bytes into which pixel data is read.

**Examples:**

[ex\\_GetDisplayNormal.nxc](#).

**6.44.2.15 void GetDisplayPopup (const byte x, const byte line, unsigned int cnt, byte & data[ ]) [inline]**

Read pixel data from the popup display buffer. Read "cnt" bytes from the popup display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

- x* The desired x position from which to read pixel data.
- line* The desired line from which to read pixel data.
- cnt* The number of bytes of pixel data to read.
- data* The array of bytes into which pixel data is read.

**Examples:**

[ex\\_GetDisplayPopup.nxc](#).

**6.44.2.16 char GraphicArrayOut (int x, int y, byte data[], unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image from byte array. Draw a graphic image byte array on the screen at the specified x and y location. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

**Parameters:**

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgaout.nxc](#).

**6.44.2.17 char GraphicArrayOutEx (int x, int y, byte data[], byte vars[], unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image from byte array with parameters. Draw a graphic image byte array on the screen at the specified x and y location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

**Parameters:**

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- vars* The byte array of parameters.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgaoutex.nxc](#).

**6.44.2.18 char GraphicOut (int x, int y, string filename, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image. Draw a graphic image file on the screen at the specified x and y location. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphic](#), [DrawGraphicType](#)

**Parameters:**

*x* The x value for the position of the graphic image.

*y* The y value for the position of the graphic image.

*filename* The filename of the RIC graphic image.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_GraphicOut.nxc](#).

**6.44.2.19 char GraphicOutEx (int x, int y, string filename, byte vars[], unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image with parameters. Draw a graphic image file on the screen at the specified x and y location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphic](#), [DrawGraphicType](#)

**Parameters:**

*x* The x value for the position of the graphic image.  
*y* The y value for the position of the graphic image.  
*filename* The filename of the RIC graphic image.  
*vars* The byte array of parameters.  
*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgoutex.nxc](#), and [ex\\_GraphicOutEx.nxc](#).

**6.44.2.20** `char LineOut (int x1, int y1, int x2, int y2, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a line. This function lets you draw a line on the screen from x1, y1 to x2, y2. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawLine](#), [DrawLineType](#)

**Parameters:**

*x1* The x value for the start of the line.  
*y1* The y value for the start of the line.  
*x2* The x value for the end of the line.  
*y2* The y value for the end of the line.  
*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_LineOut.nxc](#).

#### 6.44.2.21 `char NumOut (int x, int y, variant value, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a number. Draw a numeric value on the screen at the specified x and y location. The y value must be a multiple of 8. Valid line number constants are listed in the [Line number constants](#) group. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawText](#), [DrawTextType](#)

**Parameters:**

- x* The x value for the start of the number output.
- y* The text line number for the number output.
- value* The value to output to the LCD screen. Any numeric type is supported.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_ArrayBuild.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_buttonpressed.nxc](#), [ex\\_contrast.nxc](#), [ex\\_ctype.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getchar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_motoroutputoptions.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_Pos.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_SizeOf.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_superpro.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_SysReadLastResponse.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), [ex\\_SysWriteSemData.nxc](#), [ex\\_UnflattenVar.nxc](#), and [ex\\_xg1300.nxc](#).

**6.44.2.22** `char PointOut (int x, int y, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a point. This function lets you draw a point on the screen at x, y. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawPoint](#), [DrawPointType](#)

**Parameters:**

- x* The x value for the point.
- y* The y value for the point.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_PointOut.nxc](#), [ex\\_sin\\_cos.nxc](#), and [ex\\_sind\\_cosd.nxc](#).

**6.44.2.23** `char PolyOut (LocationType points[ ], unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a polygon. This function lets you draw a polygon on the screen using an array of points. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawPolygon](#), [DrawPolygonType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

- points* An array of [LocationType](#) points that define the polygon.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_PolyOut.nxc](#).

**6.44.2.24 char RectOut (int x, int y, int width, int height, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a rectangle. This function lets you draw a rectangle on the screen at x, y with the specified width and height. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawRect](#), [DrawRectType](#)

**Parameters:**

*x* The x value for the top left corner of the rectangle.

*y* The y value for the top left corner of the rectangle.

*width* The width of the rectangle.

*height* The height of the rectangle.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_RectOut.nxc](#).

**6.44.2.25 void ResetScreen () [inline]**

Reset LCD screen. This function lets you restore the standard NXT running program screen.

**Examples:**

[ex\\_ResetScreen.nxc](#).

**6.44.2.26 void SetDisplayContrast (byte *contrast*) [inline]**

Set the display contrast. This function lets you set the display contrast setting.

**Parameters:**

*contrast* The desired display contrast.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_contrast.nxc](#), and [ex\\_setdisplaycontrast.nxc](#).

**6.44.2.27 void SetDisplayDisplay (unsigned long *dispaddr*) [inline]**

Set the display memory address. This function lets you set the current display memory address.

**Parameters:**

*dispaddr* The new display memory address.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayDisplay.nxc](#).

**6.44.2.28 void SetDisplayEraseMask (unsigned long *eraseMask*) [inline]**

Set the display erase mask. This function lets you set the current display erase mask.

**Parameters:**

*eraseMask* The new display erase mask.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayEraseMask.nxc](#).

**6.44.2.29 void SetDisplayFlags (byte *flags*) [inline]**

Set the display flags. This function lets you set the current display flags.

**Parameters:**

*flags* The new display flags. See [Display flags](#).

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayFlags.nxc](#).

**6.44.2.30 void SetDisplayFont (unsigned long *fontaddr*) [inline]**

Set the display font memory address. This function lets you set the current display font memory address.

**Parameters:**

*fontaddr* The new display font memory address.

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_displayfont.nxc](#), and [ex\\_setdisplayfont.nxc](#).

**6.44.2.31 void SetDisplayNormal (const byte *x*, const byte *line*, unsigned int *cnt*, byte *data*[]) [inline]**

Write pixel data to the normal display buffer. Write "cnt" bytes to the normal display memory from the data array. Start writing at the specified x, line coordinate. Each byte of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

*x* The desired x position where you wish to write pixel data.

*line* The desired line where you wish to write pixel data.

*cnt* The number of bytes of pixel data to write.

*data* The array of bytes from which pixel data is read.

**Examples:**

[ex\\_SetDisplayNormal.nxc](#).

**6.44.2.32 void SetDisplayPopup (const byte x, const byte line, unsigned int cnt, byte data[]) [inline]**

Write pixel data to the popup display buffer. Write "cnt" bytes to the popup display memory from the data array. Start writing at the specified x, line coordinate. Each byte of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

*x* The desired x position where you wish to write pixel data.

*line* The desired line where you wish to write pixel data.

*cnt* The number of bytes of pixel data to write.

*data* The array of bytes from which pixel data is read.

**Examples:**

[ex\\_SetDisplayPopup.nxc](#).

**6.44.2.33 void SetDisplayTextLinesCenterFlags (byte ctrFlags) [inline]**

Set the display text lines center flags. This function lets you set the current display text lines center flags.

**Parameters:**

*ctrFlags* The new display text lines center flags.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayTextLinesCenterFlags.nxc](#).

**6.44.2.34 void SetDisplayUpdateMask (unsigned long *updateMask*)  
[inline]**

Set the display update mask. This function lets you set the current display update mask.

**Parameters:**

*updateMask* The new display update mask.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayUpdateMask.nxc](#).

**6.44.2.35 void SysDisplayExecuteFunction (DisplayExecuteFunctionType &  
*args*) [inline]**

Execute any Display module command. This function lets you directly execute the Display module's primary drawing function using the values specified via the [DisplayExecuteFunctionType](#) structure.

**Parameters:**

*args* The [DisplayExecuteFunctionType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

**6.44.2.36 void SysDrawCircle (DrawCircleType & *args*) [inline]**

Draw a circle. This function lets you draw a circle on the NXT LCD given the parameters you pass in via the [DrawCircleType](#) structure.

**Parameters:**

*args* The [DrawCircleType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawcircle.nxc](#).

**6.44.2.37 void SysDrawEllipse (DrawEllipseType & args) [inline]**

Draw an ellipse. This function lets you draw an ellipse on the NXT LCD given the parameters you pass in via the [DrawEllipseType](#) structure.

**Parameters:**

*args* The [DrawEllipseType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_SysDrawEllipse.nxc](#).

**6.44.2.38 void SysDrawFont (DrawFontType & args) [inline]**

Draw text using a custom font. This function lets you draw text on the NXT LCD using a custom font with parameters you pass in via the [DrawFontType](#) structure.

**Parameters:**

*args* The [DrawFontType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**6.44.2.39 void SysDrawGraphic (DrawGraphicType & args) [inline]**

Draw a graphic (RIC file). This function lets you draw a graphic image (RIC file) on the NXT LCD given the parameters you pass in via the [DrawGraphicType](#) structure.

**Parameters:**

*args* The [DrawGraphicType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

**6.44.2.40 void SysDrawGraphicArray (DrawGraphicArrayType & args) [inline]**

Draw a graphic image from a byte array. This function lets you draw a graphic image on the NXT LCD given the parameters you pass in via the [DrawGraphicArrayType](#) structure.

**Parameters:**

*args* The [DrawGraphicArrayType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysdrawgraphicarray.nxc](#).

**6.44.2.41 void SysDrawLine (DrawLineType & args) [inline]**

Draw a line. This function lets you draw a line on the NXT LCD given the parameters you pass in via the [DrawLineType](#) structure.

**Parameters:**

*args* The [DrawLineType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawline.nxc](#).

**6.44.2.42 void SysDrawPoint (DrawPointType & args) [inline]**

Draw a point. This function lets you draw a pixel on the NXT LCD given the parameters you pass in via the [DrawPointType](#) structure.

**Parameters:**

*args* The [DrawPointType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawpoint.nxc](#).

**6.44.2.43 void SysDrawPolygon (DrawPolygonType & args) [inline]**

Draw a polygon. This function lets you draw a polygon on the NXT LCD given the parameters you pass in via the [DrawPolygonType](#) structure.

**Parameters:**

*args* The [DrawPolygonType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysdrawpolygon.nxc](#).

**6.44.2.44 void SysDrawRect (DrawRectType & args) [inline]**

Draw a rectangle. This function lets you draw a rectangle on the NXT LCD given the parameters you pass in via the [DrawRectType](#) structure.

**Parameters:**

*args* The [DrawRectType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawrect.nxc](#).

**6.44.2.45 void SysDrawText (DrawTextType & args) [inline]**

Draw text. This function lets you draw text on the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

**Parameters:**

*args* The [DrawTextType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawtext.nxc](#).

**6.44.2.46 void SysSetScreenMode (SetScreenModeType & args) [inline]**

Set the screen mode. This function lets you set the screen mode of the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

**Parameters:**

*args* The [SetScreenModeType](#) structure containing the screen mode parameters.

**Examples:**

[ex\\_syssetscreenmode.nxc](#).

**6.44.2.47 char TextOut (int x, int y, string str, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw text. Draw a text value on the screen at the specified x and y location. The y value must be a multiple of 8. Valid line number constants are listed in the [Line number constants](#) group. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawText](#), [DrawTextType](#)

**Parameters:**

*x* The x value for the start of the text output.

**y** The text line number for the text output.

**str** The text to output to the LCD screen.

**options** The optional drawing options.

#### Returns:

The result of the drawing operation.

#### Examples:

`ex_acos.nxc`, `ex_acosd.nxc`, `ex_addressof.nxc`, `ex_addressofex.nxc`, `ex_ArrayMax.nxc`, `ex_ArrayMean.nxc`, `ex_ArrayMin.nxc`, `ex_ArrayOp.nxc`, `ex_ArraySort.nxc`, `ex_ArrayStd.nxc`, `ex_ArraySum.nxc`, `ex_ArraySumSqr.nxc`, `ex_asin.nxc`, `ex_asind.nxc`, `ex_atan.nxc`, `ex_atan2.nxc`, `ex_atan2d.nxc`, `ex_atand.nxc`, `ex_clearline.nxc`, `ex_copy.nxc`, `ex_ctype.nxc`, `ex_DataMode.nxc`, `ex_delete_data_file.nxc`, `ex_diaccl.nxc`, `ex_digyro.nxc`, `ex_disp gout.nxc`, `ex_displayfont.nxc`, `ex_file_system.nxc`, `ex_findfirstfile.nxc`, `ex_findnextfile.nxc`, `ex_GetBrickDataAddress.nxc`, `ex_HTGyroTest.nxc`, `ex_i2cdeviceid.nxc`, `ex_i2cdeviceinfo.nxc`, `ex_i2cvendorid.nxc`, `ex_i2cversion.nxc`, `ex_isnan.nxc`, `ex_labs.nxc`, `ex_leftstr.nxc`, `ex_midstr.nxc`, `ex_ReadSensorHTBarometric.nxc`, `ex_ReadSensorHTTouchMultiplexer.nxc`, `ex_ReadSensorMSPlayStation.nxc`, `ex_reladdressof.nxc`, `ex_rightstr.nxc`, `ex_RS485Receive.nxc`, `ex_RS485Send.nxc`, `ex_SetAbortFlag.nxc`, `ex_setdisplayfont.nxc`, `ex_SetLongAbort.nxc`, `ex_StrCatOld.nxc`, `ex_string.nxc`, `ex_StrReplace.nxc`, `ex_strtod.nxc`, `ex_strtol.nxc`, `ex_strtoul.nxc`, `ex_SubStr.nxc`, `ex_syscommbtconnection.nxc`, `ex_SysCommBTOff.nxc`, `ex_SysCommHSControl.nxc`, `ex_SysCommHSRead.nxc`, `ex_SysCommHSControl.nxc`, `ex_SysComputeCalibValue.nxc`, `ex_SysDataLogWrite.nxc`, `ex_sysfilefindfirst.nxc`, `ex_sysfilefindnext.nxc`, `ex_sysfileread.nxc`, `ex_syslistfiles.nxc`, `ex_sysmessageread.nxc`, `ex_tan.nxc`, `ex_tand.nxc`, `ex_TextOut.nxc`, `ex_xg1300.nxc`, `util_battery_1.nxc`, `util_battery_2.nxc`, and `util_rpm.nxc`.

## 6.45 Sound module types

Types used by various sound module functions.

#### Data Structures

- struct [Tone](#)  
*Type used with the `PlayTones` API function.*
- struct [SoundPlayFileType](#)  
*Parameters for the `SoundPlayFile` system call.*

- struct [SoundPlayToneType](#)  
*Parameters for the SoundPlayTone system call.*
- struct [SoundGetStateType](#)  
*Parameters for the SoundGetState system call.*
- struct [SoundSetStateType](#)  
*Parameters for the SoundSetState system call.*

### 6.45.1 Detailed Description

Types used by various sound module functions.

## 6.46 Sound module functions

Functions for accessing and modifying sound module features.

### Functions

- char [PlayFile](#) (string filename)  
*Play a file.*
- char [PlayFileEx](#) (string filename, byte volume, bool loop)  
*Play a file with extra options.*
- char [PlayTone](#) (unsigned int frequency, unsigned int duration)  
*Play a tone.*
- char [PlayToneEx](#) (unsigned int frequency, unsigned int duration, byte volume, bool loop)  
*Play a tone with extra options.*
- byte [SoundState](#) ()  
*Get sound module state.*
- byte [SoundFlags](#) ()  
*Get sound module flags.*
- byte [StopSound](#) ()

*Stop sound.*

- unsigned int **SoundFrequency** ()  
*Get sound frequency.*
- unsigned int **SoundDuration** ()  
*Get sound duration.*
- unsigned int **SoundSampleRate** ()  
*Get sample rate.*
- byte **SoundMode** ()  
*Get sound mode.*
- byte **SoundVolume** ()  
*Get volume.*
- void **SetSoundDuration** (unsigned int duration)  
*Set sound duration.*
- void **SetSoundFlags** (byte flags)  
*Set sound module flags.*
- void **SetSoundFrequency** (unsigned int frequency)  
*Set sound frequency.*
- void **SetSoundMode** (byte mode)  
*Set sound mode.*
- void **SetSoundModuleState** (byte state)  
*Set sound module state.*
- void **SetSoundSampleRate** (unsigned int sampleRate)  
*Set sample rate.*
- void **SetSoundVolume** (byte volume)  
*Set sound volume.*
- void **SysSoundPlayFile** (**SoundPlayFileType** &args)  
*Play sound file.*
- void **SysSoundPlayTone** (**SoundPlayToneType** &args)

*Play tone.*

- void `SysSoundGetState` (`SoundGetStateType` &args)  
*Get sound state.*
- void `SysSoundSetState` (`SoundSetStateType` &args)  
*Set sound state.*
- void `PlaySound` (const int &aCode)  
*Play a system sound.*
- void `PlayTones` (`Tone` tones[ ])  
*Play multiple tones.*

### 6.46.1 Detailed Description

Functions for accessing and modifying sound module features.

### 6.46.2 Function Documentation

#### 6.46.2.1 `char PlayFile (string filename) [inline]`

Play a file. Play the specified file. The filename may be any valid string expression. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

#### Parameters:

*filename* The name of the sound or melody file to play.

#### Examples:

`ex_PlayFile.nxc.`

#### 6.46.2.2 `char PlayFileEx (string filename, byte volume, bool loop) [inline]`

Play a file with extra options. Play the specified file. The filename may be any valid string expression. Volume should be a number from 0 (silent) to 4 (loudest). Play the file repeatedly if loop is true. The sound file can either be an RSO file containing PCM

or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

**Parameters:**

*filename* The name of the sound or melody file to play.

*volume* The desired tone volume.

*loop* A boolean flag indicating whether to play the file repeatedly.

**Examples:**

[ex\\_PlayFileEx.nxc](#).

**6.46.2.3 void PlaySound (const int & aCode)**

Play a system sound. Play a sound that mimics the RCX system sounds using one of the [RCX and Scout sound constants](#).

| aCode                             | Resulting Sound |
|-----------------------------------|-----------------|
| <a href="#">SOUND_CLICK</a>       | key click sound |
| <a href="#">SOUND_DOUBLE_BEEP</a> | double beep     |
| <a href="#">SOUND_DOWN</a>        | sweep down      |
| <a href="#">SOUND_UP</a>          | sweep up        |
| <a href="#">SOUND_LOW_BEEP</a>    | error sound     |
| <a href="#">SOUND_FAST_UP</a>     | fast sweep up   |

**Parameters:**

*aCode* The system sound to play. See [RCX and Scout sound constants](#).

**Examples:**

[ex\\_playsound.nxc](#).

**6.46.2.4 char PlayTone (unsigned int *frequency*, unsigned int *duration*)  
[inline]**

Play a tone. Play a single tone of the specified frequency and duration. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). The tone is played at the loudest sound level supported by the firmware and it is not looped.

**Parameters:**

*frequency* The desired tone frequency, in Hz.

*duration* The desired tone duration, in ms.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_PlayTone.nxc](#), and [ex\\_yield.nxc](#).

**6.46.2.5 char PlayToneEx (unsigned int *frequency*, unsigned int *duration*, byte *volume*, bool *loop*) [inline]**

Play a tone with extra options. Play a single tone of the specified frequency, duration, and volume. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). Volume should be a number from 0 (silent) to 4 (loudest). Play the tone repeatedly if loop is true.

**Parameters:**

*frequency* The desired tone frequency, in Hz.

*duration* The desired tone duration, in ms.

*volume* The desired tone volume.

*loop* A boolean flag indicating whether to play the tone repeatedly.

**Examples:**

[ex\\_PlayToneEx.nxc](#).

**6.46.2.6 void PlayTones (Tone *tones*[])**

Play multiple tones. Play a series of tones contained in the tones array. Each element in the array is an instance of the [Tone](#) structure, containing a frequency and a duration.

**Parameters:**

*tones* The array of tones to play.

**Examples:**

[ex\\_playtones.nxc](#).

**6.46.2.7 void SetSoundDuration (unsigned int *duration*) [inline]**

Set sound duration. Set the sound duration.

**See also:**

[SoundDuration\(\)](#)

**Parameters:**

*duration* The new sound duration

**Examples:**

[ex\\_SetSoundDuration.nxc](#).

**6.46.2.8 void SetSoundFlags (byte *flags*) [inline]**

Set sound module flags. Set the sound module flags. See the [SoundFlags constants](#) group.

**See also:**

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Parameters:**

*flags* The new sound module flags

**Examples:**

[ex\\_SetSoundFlags.nxc](#).

**6.46.2.9 void SetSoundFrequency (unsigned int *frequency*) [inline]**

Set sound frequency. Set the sound frequency.

**See also:**

[SoundFrequency\(\)](#)

**Parameters:**

*frequency* The new sound frequency

**Examples:**

[ex\\_SetSoundFrequency.nxc](#).

**6.46.2.10 void SetSoundMode (byte *mode*) [inline]**

Set sound mode. Set the sound mode. See the [SoundMode constants](#) group.

**See also:**

[SoundMode\(\)](#)

**Parameters:**

*mode* The new sound mode

**Examples:**

[ex\\_SetSoundMode.nxc](#).

**6.46.2.11 void SetSoundModuleState (byte *state*) [inline]**

Set sound module state. Set the sound module state. See the [SoundState constants](#) group.

**See also:**

[SoundState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Parameters:**

*state* The new sound state

**Examples:**

[ex\\_SetSoundModuleState.nxc](#).

**6.46.2.12 void SetSoundSampleRate (unsigned int *sampleRate*) [inline]**

Set sample rate. Set the sound sample rate.

**See also:**

[SoundSampleRate\(\)](#)

**Parameters:**

*sampleRate* The new sample rate

**Examples:**

[ex\\_SetSoundSampleRate.nxc](#).

**6.46.2.13 void SetSoundVolume (byte *volume*) [inline]**

Set sound volume. Set the sound volume.

**See also:**

[SoundVolume\(\)](#)

**Parameters:**

*volume* The new volume

**Examples:**

[ex\\_SetSoundVolume.nxc](#).

**6.46.2.14 unsigned int SoundDuration () [inline]**

Get sound duration. Return the current sound duration.

**See also:**

[SetSoundDuration\(\)](#)

**Returns:**

The current sound duration.

**Examples:**

[ex\\_SoundDuration.nxc](#).

**6.46.2.15 byte SoundFlags () [inline]**

Get sound module flags. Return the current sound module flags. See the [SoundFlags constants](#) group.

**See also:**

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Returns:**

The current sound module flags.

**Examples:**

[ex\\_SoundFlags.nxc](#).

**6.46.2.16 unsigned int SoundFrequency () [inline]**

Get sound frequency. Return the current sound frequency.

**See also:**

[SetSoundFrequency\(\)](#)

**Returns:**

The current sound frequency.

**Examples:**

[ex\\_SoundFrequency.nxc](#).

**6.46.2.17 byte SoundMode () [inline]**

Get sound mode. Return the current sound mode. See the [SoundMode constants](#) group.

**See also:**[SetSoundMode\(\)](#)**Returns:**

The current sound mode.

**Examples:**[ex\\_SoundMode.nxc](#).**6.46.2.18 unsigned int SoundSampleRate () [inline]**

Get sample rate. Return the current sound sample rate.

**See also:**[SetSoundSampleRate\(\)](#)**Returns:**

The current sound sample rate.

**Examples:**[ex\\_SoundSampleRate.nxc](#).**6.46.2.19 byte SoundState () [inline]**

Get sound module state. Return the current sound module state. See the [SoundState constants](#) group.

**See also:**[SetSoundModuleState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)**Returns:**

The current sound module state.

**Examples:**[ex\\_SoundState.nxc](#).

**6.46.2.20** `byte SoundVolume () [inline]`

Get volume. Return the current sound volume.

**See also:**

[SetSoundVolume\(\)](#)

**Returns:**

The current sound volume.

**Examples:**

[ex\\_SoundVolume.nxc](#).

**6.46.2.21** `byte StopSound () [inline]`

Stop sound. Stop playing of the current tone or file.

**Returns:**

The result

**Todo**

?

**Examples:**

[ex\\_StopSound.nxc](#).

**6.46.2.22** `void SysSoundGetState (SoundGetStateType & args) [inline]`

Get sound state. This function lets you retrieve information about the sound module state via the [SoundGetStateType](#) structure.

**Parameters:**

*args* The [SoundGetStateType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundgetstate.nxc](#).

**6.46.2.23 void SysSoundPlayFile (SoundPlayFileType & args) [inline]**

Play sound file. This function lets you play a sound file given the parameters you pass in via the [SoundPlayFileType](#) structure. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

**Parameters:**

*args* The [SoundPlayFileType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundplayfile.nxc](#).

**6.46.2.24 void SysSoundPlayTone (SoundPlayToneType & args) [inline]**

Play tone. This function lets you play a tone given the parameters you pass in via the [SoundPlayToneType](#) structure.

**Parameters:**

*args* The [SoundPlayToneType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundplaytone.nxc](#).

**6.46.2.25 void SysSoundSetState (SoundSetStateType & args) [inline]**

Set sound state. This function lets you set sound module state settings via the [SoundSetStateType](#) structure.

**Parameters:**

*args* The [SoundSetStateType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundsetstate.nxc](#).

## 6.47 LowSpeed module types

Types used by various low speed module functions.

### Data Structures

- struct [CommLSWriteType](#)  
*Parameters for the CommLSWrite system call.*
- struct [CommLSReadType](#)  
*Parameters for the CommLSRead system call.*
- struct [CommLSCheckStatusType](#)  
*Parameters for the CommLSCheckStatus system call.*
- struct [CommLSWriteExType](#)  
*Parameters for the CommLSWriteEx system call.*

### 6.47.1 Detailed Description

Types used by various low speed module functions.

## 6.48 LowSpeed module functions

Functions for accessing and modifying low speed module features.

### Modules

- [Low level LowSpeed module functions](#)  
*Low level functions for accessing low speed module features.*
- [LowSpeed module system call functions](#)  
*System call functions for accessing low speed module features.*

### Functions

- byte [SensorUS](#) (const byte port)  
*Read ultrasonic sensor value.*

- char [ReadSensorUSEx](#) (const byte port, byte &values[ ])   
*Read multiple ultrasonic sensor values.*
- char [ReadSensorEMeter](#) (const byte &port, float &vIn, float &aIn, float &vOut, float &aOut, int &joules, float &wIn, float &wOut)   
*Read the LEGO EMeter values.*
- char [ConfigureTemperatureSensor](#) (const byte &port, const byte &config)   
*Configure LEGO Temperature sensor options.*
- float [SensorTemperature](#) (const byte &port)   
*Read the LEGO Temperature sensor value.*
- long [LowspeedStatus](#) (const byte port, byte &bytesready)   
*Get lowspeed status.*
- long [LowspeedCheckStatus](#) (const byte port)   
*Check lowspeed status.*
- byte [LowspeedBytesReady](#) (const byte port)   
*Get lowspeed bytes ready.*
- long [LowspeedWrite](#) (const byte port, byte retlen, byte buffer[ ])   
*Write lowspeed data.*
- long [LowspeedRead](#) (const byte port, byte buflen, byte &buffer[ ])   
*Read lowspeed data.*
- long [I2CStatus](#) (const byte port, byte &bytesready)   
*Get I2C status.*
- long [I2CCheckStatus](#) (const byte port)   
*Check I2C status.*
- byte [I2CBytesReady](#) (const byte port)   
*Get I2C bytes ready.*
- long [I2CWrite](#) (const byte port, byte retlen, byte buffer[ ])   
*Write I2C data.*
- long [I2CRead](#) (const byte port, byte buflen, byte &buffer[ ])   
*Read I2C data.*

- long [I2CBytes](#) (const byte port, byte inbuf[ ], byte &count, byte &outbuf[ ])   
*Perform an I2C write/read transaction.*
- char [ReadI2CRegister](#) (byte port, byte i2caddr, byte reg, byte &out)   
*Read I2C register.*
- char [WriteI2CRegister](#) (byte port, byte i2caddr, byte reg, byte val)   
*Write I2C register.*
- string [I2CDeviceInfo](#) (byte port, byte i2caddr, byte info)   
*Read I2C device information.*
- string [I2CVersion](#) (byte port, byte i2caddr)   
*Read I2C device version.*
- string [I2CVendorId](#) (byte port, byte i2caddr)   
*Read I2C device vendor.*
- string [I2CDeviceId](#) (byte port, byte i2caddr)   
*Read I2C device identifier.*
- long [I2CSendCommand](#) (byte port, byte i2caddr, byte cmd)   
*Send an I2C command.*

### 6.48.1 Detailed Description

Functions for accessing and modifying low speed module features.

### 6.48.2 Function Documentation

#### 6.48.2.1 char [ConfigureTemperatureSensor](#) (const byte & *port*, const byte & *config*) [`inline`]

Configure LEGO Temperature sensor options. Set various LEGO Temperature sensor options.

#### Parameters:

*port* The port to which the temperature sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*config* The temperature sensor configuration settings. See [LEGO temperature sensor constants](#) for configuration constants that can be ORed or added together.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible Result values.

**Examples:**

[ex\\_ConfigureTemperatureSensor.nxc](#).

**6.48.2.2 long I2CBytes (const byte port, byte inbuf[], byte & count, byte & outbuf[]) [inline]**

Perform an I2C write/read transaction. This method writes the bytes contained in the input buffer (inbuf) to the I2C device on the specified port, checks for the specified number of bytes to be ready for reading, and then tries to read the specified number (count) of bytes from the I2C device into the output buffer (outbuf).

This is a higher-level wrapper around the three main I2C functions. It also maintains a "last good read" buffer and returns values from that buffer if the I2C communication transaction fails.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*inbuf* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

*count* The number of bytes that should be returned by the I2C device. On output count is set to the number of bytes in outbuf.

*outbuf* A byte array that contains the data read from the internal I2C buffer.

**Returns:**

Returns true or false indicating whether the I2C transaction succeeded or failed.

**See also:**

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [I2CRead](#), [LowSpeedRead](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CBytes.nxc](#).

**6.48.2.3 byte I2CBytesReady (const byte *port*) [inline]**

Get I2C bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [LowSpeedBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CBytesReady.nxc](#).

**6.48.2.4 long I2CCheckStatus (const byte *port*) [inline]**

Check I2C status. This method checks the status of the I2C communication on the specified port.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while this function returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [LowspeedStatus](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

**Examples:**

[ex\\_I2CCheckStatus.nxc](#).

**6.48.2.5 string I2CDeviceId (byte port, byte i2caddr) [inline]**

Read I2C device identifier. Read standard I2C device identifier. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device identifier.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

**6.48.2.6 string I2CDeviceInfo (byte port, byte i2caddr, byte info) [inline]**

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

*info* A value indicating the type of device information you are requesting. See [Standard I2C constants](#).

**Returns:**

A string containing the requested device information.

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

**6.48.2.7 long I2CRead (const byte port, byte buflen, byte & buffer[ ])  
[inline]**

Read I2C data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*buflen* The initial size of the output buffer.

*buffer* A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is `NO_ERR` then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CRead.nxc](#).

**6.48.2.8 long I2CSendCommand (byte *port*, byte *i2caddr*, byte *cmd*)  
[inline]**

Send an I2C command. Send a command to an I2C device at the standard command register: [I2C\\_REG\\_CMD](#). The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

*cmd* The command to send to the I2C device.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_I2CSendCommand.nxc](#).

**6.48.2.9 long I2CStatus (const byte *port*, byte & *bytesready*) [inline]**

Get I2C status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*bytesready* The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible return values. If the return value is `NO_ERR` then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while [I2CStatus](#) returns `STAT_COMM_PENDING`.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [LowSpeedStatus](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedCheckStatus](#)

**Examples:**

[ex\\_I2CStatus.nxc](#).

#### 6.48.2.10 `string I2CVendorId (byte port, byte i2caddr) [inline]`

Read I2C device vendor. Read standard I2C device vendor. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device vendor.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

#### 6.48.2.11 `string I2CVersion (byte port, byte i2caddr) [inline]`

Read I2C device version. Read standard I2C device version. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device version.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

**6.48.2.12 long I2CWrite (const byte port, byte retlen, byte buffer[ ]  
[inline])**

Write I2C data. This method starts a transaction to write the bytes contained in the array buffer to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*retlen* The number of bytes that should be returned by the I2C device.

*buffer* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is `NO_ERR` then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CWrite.nxc](#).

**6.48.2.13 byte LowspeedBytesReady (const byte *port*) [inline]**

Get lowspeed bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful `LowspeedWrite` call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

**Examples:**

[ex\\_LowspeedBytesReady.nxc](#).

**6.48.2.14 long LowspeedCheckStatus (const byte *port*) [inline]**

Check lowspeed status. This method checks the status of the I2C communication on the specified port.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while [LowspeedCheckStatus](#) returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

**Examples:**

[ex\\_LowspeedCheckStatus.nxc](#).

**6.48.2.15 long LowspeedRead (const byte port, byte buflen, byte & buffer[])  
[inline]**

Read lowspeed data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

**Parameters:**

**port** The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**buflen** The initial size of the output buffer.

**buffer** A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

**Examples:**

[ex\\_LowspeedRead.nxc](#).

**6.48.2.16 long LowspeedStatus (const byte port, byte & bytesready)  
[inline]**

Get lowspeed status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*bytesready* The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while LowspeedStatus returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [I2CCheckStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

**Examples:**

[ex\\_LowspeedStatus.nxc](#).

**6.48.2.17 long LowspeedWrite (const byte port, byte retlen, byte buffer[])  
[inline]**

Write lowspeed data. This method starts a transaction to write the bytes contained in the array buffer to the I2C device on the specified port. It also tells the I2C device the

number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*retlen* The number of bytes that should be returned by the I2C device.

*buffer* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_LowSpeedWrite.nxc](#).

**6.48.2.18 char ReadI2CRegister (byte port, byte i2caddr, byte reg, byte & out) [inline]**

Read I2C register. Read a single byte from an I2C device register.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*i2caddr* The I2C device address.

*reg* The I2C device register from which to read a single byte.

*out* The single byte read from the I2C device.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_readi2cregister.nxc](#).

**6.48.2.19** `char ReadSensorEMeter (const byte &port, float &vIn, float &aIn, float &vOut, float &aOut, int &joules, float &wIn, float &wOut) [inline]`

Read the LEGO EMeter values. Read all the LEGO EMeter register values. They must all be read at once to ensure data coherency.

**Parameters:**

*port* The port to which the LEGO EMeter sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vIn* Input voltage

*aIn* Input current

*vOut* Output voltage

*aOut* Output current

*joules* The number of joules stored in the EMeter

*wIn* The number of watts generated

*wOut* The number of watts consumed

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_ReadSensorEMeter.nxc](#).

**6.48.2.20** `char ReadSensorUSEx (const byte port, byte &values[]) [inline]`

Read multiple ultrasonic sensor values. Return eight ultrasonic sensor distance values.

**Parameters:**

*port* The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*values* An array of bytes that will contain the 8 distance values read from the ultrasonic sensor.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_ReadSensorUSEx.nxc](#).

**6.48.2.21 float SensorTemperature (const byte & port) [inline]**

Read the LEGO Temperature sensor value. Return the temperature sensor value in degrees celcius. Since a temperature sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. The port must be configured as a temperature sensor port before using this function. Use [SetSensorTemperature](#) to configure the port.

**Parameters:**

*port* The port to which the temperature sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The temperature sensor value in degrees celcius.

**Examples:**

[ex\\_SensorTemperature.nxc](#).

**6.48.2.22 byte SensorUS (const byte port) [inline]**

Read ultrasonic sensor value. Return the ultrasonic sensor distance value. Since an ultrasonic sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The ultrasonic sensor distance value (0..255)

**Examples:**

[ex\\_SensorUS.nxc](#).

**6.48.2.23 char WriteI2CRegister (byte port, byte i2caddr, byte reg, byte val) [inline]**

Write I2C register. Write a single byte to an I2C device register.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*i2caddr* The I2C device address.

*reg* The I2C device register to which to write a single byte.

*val* The byte to write to the I2C device.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_writei2cregister.nxc](#).

**6.49 Low level LowSpeed module functions**

Low level functions for accessing low speed module features.

**Functions**

- void [GetLSInputBuffer](#) (const byte port, const byte offset, byte cnt, byte &data[])

*Get I2C input buffer data.*

- void [GetLSOutputBuffer](#) (const byte port, const byte offset, byte cnt, byte &data[ ])

*Get I2C output buffer data.*

- byte [LSInputBufferInPtr](#) (const byte port)

*Get I2C input buffer in-pointer.*

- byte [LSInputBufferOutPtr](#) (const byte port)

*Get I2C input buffer out-pointer.*

- byte [LSInputBufferBytesToRx](#) (const byte port)

*Get I2C input buffer bytes to rx.*

- byte [LSOutputBufferInPtr](#) (const byte port)

*Get I2C output buffer in-pointer.*

- byte [LSOutputBufferOutPtr](#) (const byte port)

*Get I2C output buffer out-pointer.*

- byte [LSOutputBufferBytesToRx](#) (const byte port)

*Get I2C output buffer bytes to rx.*

- byte [LSMode](#) (const byte port)

*Get I2C mode.*

- byte [LSChannelState](#) (const byte port)

*Get I2C channel state.*

- byte [LSErrorType](#) (const byte port)

*Get I2C error type.*

- byte [LSSpeed](#) ()

*Get I2C state.*

- byte [LSSpeed](#) ()

*Get I2C speed.*

- byte [LSNoRestartOnRead](#) ()

*Get I2C no restart on read setting.*

- void [SetI2COptions](#) (byte port, byte options)

*Set I2C options.*

### 6.49.1 Detailed Description

Low level functions for accessing low speed module features.

### 6.49.2 Function Documentation

#### 6.49.2.1 void GetLSInputBuffer (const byte *port*, const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get I2C input buffer data. This method reads count bytes of data from the I2C input buffer for the specified port and writes it to the buffer provided.

#### Parameters:

*port* A constant port number (S1..S4). See [Input port constants](#).

*offset* A constant offset into the I2C input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the I2C input buffer.

#### Examples:

[ex\\_GetLSInputBuffer.nxc](#).

#### 6.49.2.2 void GetLSOutputBuffer (const byte *port*, const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get I2C output buffer data. This method reads cnt bytes of data from the I2C output buffer for the specified port and writes it to the buffer provided.

#### Parameters:

*port* A constant port number (S1..S4). See [Input port constants](#).

*offset* A constant offset into the I2C output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the I2C output buffer.

**Examples:**

[ex\\_GetLSOutputBuffer.nxc](#).

**6.49.2.3 byte LSChannelState (const byte port) [inline]**

Get I2C channel state. This method returns the value of the I2C channel state for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port channel state. See [LSChannelState constants](#).

**Examples:**

[ex\\_LSChannelState.nxc](#).

**6.49.2.4 byte LSErrorType (const byte port) [inline]**

Get I2C error type. This method returns the value of the I2C error type for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port error type. See [LSErrorType constants](#).

**Examples:**

[ex\\_LSErrorType.nxc](#).

**6.49.2.5 byte LSInputBufferBytesToRx (const byte port) [inline]**

Get I2C input buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's bytes to rx value.

**Examples:**

[ex\\_LSInputBufferBytesToRx.nxc](#).

**6.49.2.6 byte LSInputBufferIntPtr (const byte port) [inline]**

Get I2C input buffer in-pointer. This method returns the value of the input pointer of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's in-pointer value.

**Examples:**

[ex\\_LSInputBufferIntPtr.nxc](#).

**6.49.2.7 byte LSInputBufferOutPtr (const byte port) [inline]**

Get I2C input buffer out-pointer. This method returns the value of the output pointer of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's out-pointer value.

**Examples:**

[ex\\_LSInputBufferOutPtr.nxc](#).

**6.49.2.8** byte `LSMode (const byte port) [inline]`

Get I2C mode. This method returns the value of the I2C mode for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port mode. See [LSMode constants](#).

**Examples:**

[ex\\_LSMODE.nxc](#).

**6.49.2.9** byte `LSNoRestartOnRead () [inline]`

Get I2C no restart on read setting. This method returns the value of the I2C no restart on read field.

**Returns:**

The I2C no restart on read field. See [LSNoRestartOnRead constants](#).

**Examples:**

[ex\\_LSNORestartOnRead.nxc](#).

**6.49.2.10** byte `LSOutputBufferBytesToRx (const byte port) [inline]`

Get I2C output buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's bytes to rx value.

**Examples:**

[ex\\_LSOutputBufferBytesToRx.nxc](#).

**6.49.2.11** byte `LSOutputBufferInPtr (const byte port) [inline]`

Get I2C output buffer in-pointer. This method returns the value of the input pointer of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's in-pointer value.

**Examples:**

[ex\\_LSOutputBufferInPtr.nxc](#).

**6.49.2.12** byte `LSOutputBufferOutPtr (const byte port) [inline]`

Get I2C output buffer out-pointer. This method returns the value of the output pointer of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's out-pointer value.

**Examples:**

[ex\\_LSOutputBufferOutPtr.nxc](#).

**6.49.2.13** byte `LSSpeed () [inline]`

Get I2C speed. This method returns the value of the I2C speed.

**Returns:**

The I2C speed.

**Warning:**

This function is unimplemented within the firmware.

**Examples:**

[ex\\_LSSpeed.nxc](#).

**6.49.2.14 byte LSSState () [inline]**

Get I2C state. This method returns the value of the I2C state.

**Returns:**

The I2C state. See [LSSState constants](#).

**Examples:**

[ex\\_LSSState.nxc](#).

**6.49.2.15 void SetI2COptions (byte port, byte options) [inline]**

Set I2C options. This method lets you modify I2C options. Use this function to turn on or off the fast I2C mode and also control whether the standard I2C mode performs a restart prior to the read operation.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*port* The port whose I2C options you wish to change. See the [Input port constants](#) group. You may use a constant or a variable.

*options* The new option value. See [I2C option constants](#).

**6.50 LowSpeed module system call functions**

System call functions for accessing low speed module features.

## Functions

- void [SysCommLSWrite](#) ([CommLSWriteType](#) &args)  
*Write to a Lowspeed sensor.*
- void [SysCommLSRead](#) ([CommLSReadType](#) &args)  
*Read from a Lowspeed sensor.*
- void [SysCommLSCheckStatus](#) ([CommLSCheckStatusType](#) &args)  
*Check Lowspeed sensor status.*
- void [SysCommLSWriteEx](#) ([CommLSWriteExType](#) &args)  
*Write to a Lowspeed sensor (extra).*

### 6.50.1 Detailed Description

System call functions for accessing low speed module features.

### 6.50.2 Function Documentation

#### 6.50.2.1 void [SysCommLSCheckStatus](#) ([CommLSCheckStatusType](#) & *args*) [[inline](#)]

Check Lowspeed sensor status. This function lets you check the status of an I2C (Lowspeed) sensor transaction using the values specified via the [CommLSCheckStatusType](#) structure.

#### Parameters:

*args* The [CommLSCheckStatusType](#) structure containing the needed parameters.

#### Examples:

[ex\\_syscommlscheckstatus.nxc](#).

#### 6.50.2.2 void [SysCommLSRead](#) ([CommLSReadType](#) & *args*) [[inline](#)]

Read from a Lowspeed sensor. This function lets you read from an I2C (Lowspeed) sensor using the values specified via the [CommLSReadType](#) structure.

**Parameters:**

*args* The [CommLSReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommlsread.nxc](#).

**6.50.2.3 void SysCommLSWrite (CommLSWriteType & args) [inline]**

Write to a Lowspeed sensor. This function lets you write to an I2C (Lowspeed) sensor using the values specified via the [CommLSWriteType](#) structure.

**Parameters:**

*args* The [CommLSWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommlswrite.nxc](#).

**6.50.2.4 void SysCommLSWriteEx (CommLSWriteExType & args) [inline]**

Write to a Lowspeed sensor (extra). This function lets you write to an I2C (Lowspeed) sensor using the values specified via the [CommLSWriteExType](#) structure. This is the same as the SysCommLSWrite function except that you also can specify whether or not the Lowspeed module should issue a restart command to the I2C device before beginning to read data from the device.

**Parameters:**

*args* The [CommLSWriteExType](#) structure containing the desired parameters.

**Examples:**

[ex\\_syscommlswriteex.nxc](#).

**6.51 Command module types**

Types used by various Command module functions.

**Data Structures**

- struct [GetStartTickType](#)  
*Parameters for the GetStartTick system call.*
- struct [KeepAliveType](#)  
*Parameters for the KeepAlive system call.*
- struct [IOMapReadType](#)  
*Parameters for the IOMapRead system call.*
- struct [IOMapWriteType](#)  
*Parameters for the IOMapWrite system call.*
- struct [IOMapReadByIDType](#)  
*Parameters for the IOMapReadByID system call.*
- struct [IOMapWriteByIDType](#)  
*Parameters for the IOMapWriteByID system call.*
- struct [DatalogWriteType](#)  
*Parameters for the DatalogWrite system call.*
- struct [DatalogGetTimesType](#)  
*Parameters for the DatalogGetTimes system call.*
- struct [ReadSemDataType](#)  
*Parameters for the ReadSemData system call.*
- struct [WriteSemDataType](#)  
*Parameters for the WriteSemData system call.*
- struct [UpdateCalibCacheInfoType](#)  
*Parameters for the UpdateCalibCacheInfo system call.*
- struct [ComputeCalibValueType](#)  
*Parameters for the ComputeCalibValue system call.*
- struct [MemoryManagerType](#)  
*Parameters for the MemoryManager system call.*
- struct [ReadLastResponseType](#)  
*Parameters for the ReadLastResponse system call.*

### 6.51.1 Detailed Description

Types used by various Command module functions.

## 6.52 Command module functions

Functions for accessing and modifying Command module features.

### Modules

- [Comparison Constants](#)  
*Logical comparison operators for use in BranchTest and BranchComp.*
- [Array API functions](#)  
*Functions for use with NXC array types.*

### Functions

- unsigned long [CurrentTick](#) ()  
*Read the current system tick.*
- unsigned long [FirstTick](#) ()  
*Get the first tick.*
- long [ResetSleepTimer](#) ()  
*Reset the sleep timer.*
- void [SysCall](#) (byte funcID, variant &args)  
*Call any system function.*
- void [SysGetStartTick](#) ([GetStartTickType](#) &args)  
*Get start tick.*
- void [SysKeepAlive](#) ([KeepAliveType](#) &args)  
*Keep alive.*
- void [SysIOMapRead](#) ([IOMapReadType](#) &args)  
*Read from IOMap by name.*
- void [SysIOMapWrite](#) ([IOMapWriteType](#) &args)

*Write to IOMap by name.*

- void [SysIOMapReadByID](#) ([IOMapReadByIDType](#) &args)  
*Read from IOMap by identifier.*
- void [SysIOMapWriteByID](#) ([IOMapWriteByIDType](#) &args)  
*Write to IOMap by identifier.*
- void [SysDatalogWrite](#) ([DatalogWriteType](#) &args)  
*Write to the datalog.*
- void [SysDatalogGetTimes](#) ([DatalogGetTimesType](#) &args)  
*Get datalog times.*
- void [SysReadSemData](#) ([ReadSemDataType](#) &args)  
*Read semaphore data.*
- void [SysWriteSemData](#) ([WriteSemDataType](#) &args)  
*Write semaphore data.*
- void [SysUpdateCalibCacheInfo](#) ([UpdateCalibCacheInfoType](#) &args)  
*Update calibration cache information.*
- void [SysComputeCalibValue](#) ([ComputeCalibValueType](#) &args)  
*Compute calibration values.*
- char [GetMemoryInfo](#) (bool Compact, unsigned int &PoolSize, unsigned int &DataspaceSize)  
*Read memory information.*
- void [SysMemoryManager](#) ([MemoryManagerType](#) &args)  
*Read memory information.*
- char [GetLastResponseInfo](#) (bool Clear, byte &Length, byte &Command, byte &Buffer[ ])  
*Read last response information.*
- void [SysReadLastResponse](#) ([ReadLastResponseType](#) &args)  
*Read last response information.*
- void [Wait](#) (unsigned long ms)  
*Wait some milliseconds.*

- void [Yield](#) ()  
*Yield to another task.*
- void [StopAllTasks](#) ()  
*Stop all tasks.*
- void [Stop](#) (bool bvalue)  
*Stop the running program.*
- void [ExitTo](#) (task newTask)  
*Exit to another task.*
- void [Precedes](#) (task task1, task task2,..., task taskN)  
*Declare tasks that this task precedes.*
- void [Follows](#) (task task1, task task2,..., task taskN)  
*Declare tasks that this task follows.*
- void [Acquire](#) (mutex m)  
*Acquire a mutex.*
- void [Release](#) (mutex m)  
*Acquire a mutex.*
- void [StartTask](#) (task t)  
*Start a task.*
- void [StopTask](#) (task t)  
*Stop a task.*
- void [BranchTest](#) (const byte cmp, constant void lbl, variant value)  
*Branch if test is true.*
- void [BranchComp](#) (const byte cmp, constant void lbl, variant v1, variant v2)  
*Branch if compare is true.*
- void [SetIOMapBytes](#) (string moduleName, unsigned int offset, unsigned int count, byte data[ ])  
*Set IOMap bytes by name.*
- void [SetIOMapValue](#) (string moduleName, unsigned int offset, variant value)  
*Set IOMap value by name.*

- void [GetIOMapBytes](#) (string moduleName, unsigned int offset, unsigned int count, byte &data[ ])   
*Get IOMap bytes by name.*
- void [GetIOMapValue](#) (string moduleName, unsigned int offset, variant &value)   
*Get IOMap value by name.*
- void [GetLowSpeedModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])   
*Get Lowspeed module IOMap bytes.*
- void [GetDisplayModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])   
*Get Display module IOMap bytes.*
- void [GetCommModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])   
*Get Comm module IOMap bytes.*
- void [GetCommandModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])   
*Get Command module IOMap bytes.*
- void [SetCommandModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])   
*Set Command module IOMap bytes.*
- void [SetLowSpeedModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])   
*Set Lowspeed module IOMap bytes.*
- void [SetDisplayModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])   
*Set Display module IOMap bytes.*
- void [SetCommModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])   
*Set Comm module IOMap bytes.*
- void [SetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte data[ ])

*Set IOMap bytes by ID.*

- void [SetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant value)

*Set IOMap value by ID.*

- void [GetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte &data[ ])

*Get IOMap bytes by ID.*

- void [GetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant &value)

*Get IOMap value by ID.*

- void [SetCommandModuleValue](#) (unsigned int offset, variant value)

*Set Command module IOMap value.*

- void [SetIOCtrlModuleValue](#) (unsigned int offset, variant value)

*Set IOCtrl module IOMap value.*

- void [SetLoaderModuleValue](#) (unsigned int offset, variant value)

*Set Loader module IOMap value.*

- void [SetUIModuleValue](#) (unsigned int offset, variant value)

*Set Ui module IOMap value.*

- void [SetSoundModuleValue](#) (unsigned int offset, variant value)

*Set Sound module IOMap value.*

- void [SetButtonModuleValue](#) (unsigned int offset, variant value)

*Set Button module IOMap value.*

- void [SetInputModuleValue](#) (unsigned int offset, variant value)

*Set Input module IOMap value.*

- void [SetOutputModuleValue](#) (unsigned int offset, variant value)

*Set Output module IOMap value.*

- void [SetLowSpeedModuleValue](#) (unsigned int offset, variant value)

*Set Lowspeed module IOMap value.*

- void [SetDisplayModuleValue](#) (unsigned int offset, variant value)

*Set Display module IOMap value.*

- void [SetCommModuleValue](#) (unsigned int offset, variant value)  
*Set Comm module IOMap value.*
- void [GetCommandModuleValue](#) (unsigned int offset, variant &value)  
*Get Command module IOMap value.*
- void [GetLoaderModuleValue](#) (unsigned int offset, variant &value)  
*Get Loader module IOMap value.*
- void [GetSoundModuleValue](#) (unsigned int offset, variant &value)  
*Get Sound module IOMap value.*
- void [GetButtonModuleValue](#) (unsigned int offset, variant &value)  
*Get Button module IOMap value.*
- void [GetUIModuleValue](#) (unsigned int offset, variant &value)  
*Get Ui module IOMap value.*
- void [GetInputModuleValue](#) (unsigned int offset, variant &value)  
*Get Input module IOMap value.*
- void [GetOutputModuleValue](#) (unsigned int offset, variant &value)  
*Get Output module IOMap value.*
- void [GetLowSpeedModuleValue](#) (unsigned int offset, variant &value)  
*Get LowSpeed module IOMap value.*
- void [GetDisplayModuleValue](#) (unsigned int offset, variant &value)  
*Get Display module IOMap value.*
- void [GetCommModuleValue](#) (unsigned int offset, variant &value)  
*Get Comm module IOMap value.*

### 6.52.1 Detailed Description

Functions for accessing and modifying Command module features.

### 6.52.2 Function Documentation

#### 6.52.2.1 void Acquire (mutex *m*) [`inline`]

Acquire a mutex. Acquire the specified mutex variable. If another task already has acquired the mutex then the current task will be suspended until the mutex is released by the other task. This function is used to ensure that the current task has exclusive access to a shared resource, such as the display or a motor. After the current task has finished using the shared resource the program should call Release to allow other tasks to acquire the mutex.

##### Parameters:

*m* The mutex to acquire.

##### Examples:

[ex\\_Acquire.nxc](#), and [ex\\_Release.nxc](#).

#### 6.52.2.2 void BranchComp (const byte *cmp*, constant void *lbl*, variant *v1*, variant *v2*) [`inline`]

Branch if compare is true. Branch to the specified label if the two values compare with a true result.

##### Parameters:

*cmp* The constant comparison code. See the [Comparison Constants](#) for valid values.

*lbl* The name of the label where code should continue executing if the comparison is true.

*v1* The first value that you want to compare.

*v2* The second value that you want to compare.

##### Warning:

You cannot use NXC expressions with this function

##### Examples:

[ex\\_nbcopt.nxc](#).

### 6.52.2.3 void BranchTest (const byte *cmp*, constant void *lbl*, variant *value*) [inline]

Branch if test is true. Branch to the specified label if the variable compares to zero with a true result.

#### Parameters:

*cmp* The constant comparison code. See the [Comparison Constants](#) for valid values.

*lbl* The name of the label where code should continue executing if the test is true.

*value* The value that you want to compare against zero.

#### Warning:

You cannot use NXC expressions with this function

#### Examples:

[ex\\_nbcopt.nxc](#).

### 6.52.2.4 unsigned long CurrentTick () [inline]

Read the current system tick. This function lets you current system tick count.

#### Returns:

The current system tick count.

#### Examples:

[ex\\_CurrentTick.nxc](#), [ex\\_dispgout.nxc](#), and [util\\_rpm.nxc](#).

### 6.52.2.5 void ExitTo (task *newTask*) [inline]

Exit to another task. Immediately exit the current task and start executing the specified task.

#### Parameters:

*newTask* The task to start executing after exiting the current task.

**Examples:**

[alternating\\_tasks.nxc](#).

**6.52.2.6 unsigned long FirstTick () [inline]**

Get the first tick. Return an unsigned 32-bit value, which is the system timing value (called a "tick") in milliseconds at the time that the program began running.

**Returns:**

The tick count at the start of program execution.

**Examples:**

[ex\\_FirstTick.nxc](#).

**6.52.2.7 void Follows (task *task1*, task *task2*, ..., task *taskN*) [inline]**

Declare tasks that this task follows. Schedule this task to follow the specified tasks so that it will execute once any of the specified tasks has completed executing. This statement should occur once within a task - preferably at the start of the task definition. If multiple tasks declare that they follow the same task then they will all execute simultaneously unless other dependencies prevent them from doing so. Any number of tasks may be listed in the Follows statement.

**Parameters:**

*task1* The first task that this task follows.

*task2* The second task that this task follows.

*taskN* The last task that this task follows.

**Examples:**

[ex\\_Follows.nxc](#).

**6.52.2.8 void GetButtonModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Button module IOMap value. Read a value from the Button module IOMap structure. You provide the offset into the Button module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Button module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.9 void GetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[ ]) [inline]**

Get Command module IOMap bytes. Read one or more bytes of data from Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the data should be read. See [Command module IOMAP offsets](#).

*count* The number of bytes to read from the specified Command module IOMap offset.

*data* A byte array that will contain the data read from the Command module IOMap.

**6.52.2.10 void GetCommandModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Command module IOMap value. Read a value from the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Command module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.11 void GetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Comm module IOMap bytes. Read one or more bytes of data from Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the data should be read. See [Comm module IOMAP offsets](#).

*count* The number of bytes to read from the specified Comm module IOMap offset.

*data* A byte array that will contain the data read from the Comm module IOMap.

**6.52.2.12 void GetCommModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Comm module IOMap value. Read a value from the Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Comm module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.13 void GetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Display module IOMap bytes. Read one or more bytes of data from Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the data should be read. See [Display module IOMAP offsets](#).

*count* The number of bytes to read from the specified Display module IOMap offset.

*data* A byte array that will contain the data read from the Display module IOMap.

**6.52.2.14 void GetDisplayModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Display module IOMap value. Read a value from the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Display module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.15 void GetInputModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Input module IOMap value. Read a value from the Input module IOMap structure. You provide the offset into the Input module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Input module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.16 void GetIOMapBytes (string *moduleName*, unsigned int *offset*,  
unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by name. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*moduleName* The module name of the IOMap. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be read

*count* The number of bytes to read from the specified IOMap offset.

*data* A byte array that will contain the data read from the IOMap

**6.52.2.17 void GetIOMapBytesByID (unsigned long *moduleId*, unsigned int *offset*, unsigned int *count*, byte & *data*) [inline]**

Get IOMap bytes by ID. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*moduleId* The module ID of the IOMap. See [NXT firmware module IDs](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be read.

*count* The number of bytes to read from the specified IOMap offset.

*data* A byte array that will contain the data read from the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.52.2.18 void GetIOMapValue (string *moduleName*, unsigned int *offset*, variant & *value*) [inline]**

Get IOMap value by name. Read a value from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

**Parameters:**

*moduleName* The module name of the IOMap. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read

*value* A variable that will contain the value read from the IOMap

**6.52.2.19 void GetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant & *value*) [inline]**

Get IOMap value by ID. Read a value from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

**Parameters:**

*moduleId* The module ID of the IOMap. See [NXT firmware module IDs](#).

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read.

*value* A variable that will contain the value read from the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.52.2.20 char GetLastResponseInfo (bool *Clear*, byte & *Length*, byte & *Command*, byte & *Buffer*[]) [inline]**

Read last response information. Read the last direct or system command response packet received by the NXT. Optionally clear the response after retrieving the information.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Parameters:**

*Clear* A boolean value indicating whether to clear the response or not.

**Length** The response packet length.

**Command** The original command byte.

**Buffer** The response packet buffer.

**Returns:**

The response status code.

**Examples:**

[ex\\_GetLastResponseInfo.nxc](#).

**6.52.2.21 void GetLoaderModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Loader module IOMap value. Read a value from the Loader module IOMap structure. You provide the offset into the Loader module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

***offset*** The number of bytes offset from the start of the IOMap structure where the value should be read. See [Loader module IOMAP offsets](#).

***value*** A variable that will contain the value read from the IOMap.

**6.52.2.22 void GetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Lowspeed module IOMap bytes. Read one or more bytes of data from Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

***offset*** The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be read. See [Low speed module IOMAP offsets](#).

***count*** The number of bytes to read from the specified Lowspeed module IOMap offset.

*data* A byte array that will contain the data read from the LowSpeed module IOMap.

**6.52.2.23 void GetLowSpeedModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get LowSpeed module IOMap value. Read a value from the LowSpeed module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Low speed module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.24 char GetMemoryInfo (bool *Compact*, unsigned int & *PoolSize*, unsigned int & *DataspaceSize*) [inline]**

Read memory information. Read the current pool size and dataspace size. Optionally compact the dataspace before returning the information. Running programs have a maximum of 32k bytes of memory available. The amount of free RAM can be calculated by subtracting the value returned by this function from [POOL\\_MAX\\_SIZE](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*Compact* A boolean value indicating whether to compact the dataspace or not.

*PoolSize* The current pool size.

*DataspaceSize* The current dataspace size.

**Returns:**

The function call result. It will be [NO\\_ERR](#) if the compact operation is not performed. Otherwise it will be the result of the compact operation.

**Examples:**

[ex\\_getmemoryinfo.nxc](#).

**6.52.2.25 void GetOutputModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Output module IOMap value. Read a value from the Output module IOMap structure. You provide the offset into the Output module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Output module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.26 void GetSoundModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Sound module IOMap value. Read a value from the Sound module IOMap structure. You provide the offset into the Sound module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Sound module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.27 void GetUIModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Ui module IOMap value. Read a value from the Ui module IOMap structure. You provide the offset into the Ui module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Ui module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**6.52.2.28 void Precedes (task *task1*, task *task2*, ..., task *taskN*) [inline]**

Declare tasks that this task precedes. Schedule the listed tasks for execution once the current task has completed executing. The tasks will all execute simultaneously unless other dependencies prevent them from doing so. This statement should be used once within a task - preferably at the start of the task definition. Any number of tasks may be listed in the Precedes statement.

**Parameters:**

*task1* The first task to start executing after the current task ends.

*task2* The second task to start executing after the current task ends.

*taskN* The last task to start executing after the current task ends.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_Precedes.nxc](#), and [ex\\_yield.nxc](#).

**6.52.2.29 void Release (mutex *m*) [inline]**

Acquire a mutex. Release the specified mutex variable. Use this to relinquish a mutex so that it can be acquired by another task. Release should always be called after a matching call to Acquire and as soon as possible after a shared resource is no longer needed.

**Parameters:**

*m* The mutex to release.

**Examples:**

[ex\\_Acquire.nxc](#), and [ex\\_Release.nxc](#).

**6.52.2.30 long ResetSleepTimer () [inline]**

Reset the sleep timer. This function lets you reset the sleep timer.

**Returns:**

The result of resetting the sleep timer.

**Examples:**

[ex\\_ResetSleepTimer.nxc](#).

**6.52.2.31 void SetButtonModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Button module IOMap value. Set one of the fields of the Button module IOMap structure to a new value. You provide the offset into the Button module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Button module IOMap structure where the new value should be written. See [Button module IOMAP offsets](#).

*value* A variable containing the new value to write to the Button module IOMap.

**6.52.2.32 void SetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]**

Set Command module IOMap bytes. Modify one or more bytes of data in the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the data should be written. See [Command module IOMAP offsets](#).

*count* The number of bytes to write at the specified Command module IOMap offset.

*data* The byte array containing the data to write to the Command module IOMap.

**6.52.2.33 void SetCommandModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Command module IOMap value. Set one of the fields of the Command module IOMap structure to a new value. You provide the offset into the Command module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the new value should be written. See [Command module IOMAP offsets](#).

*value* A variable containing the new value to write to the Command module IOMap.

**6.52.2.34 void SetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[ ]) [inline]**

Set Comm module IOMap bytes. Modify one or more bytes of data in an IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the data should be written. See [Comm module IOMAP offsets](#).

*count* The number of bytes to write at the specified Comm module IOMap offset.

*data* The byte array containing the data to write to the Comm module IOMap.

**6.52.2.35 void SetCommModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Comm module IOMap value. Set one of the fields of the Comm module IOMap structure to a new value. You provide the offset into the Comm module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the new value should be written. See [Comm module IOMAP offsets](#).

*value* A variable containing the new value to write to the Comm module IOMap.

**6.52.2.36 void SetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]**

Set Display module IOMap bytes. Modify one or more bytes of data in the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the data should be written. See [Display module IOMAP offsets](#).

*count* The number of bytes to write at the specified Display module IOMap offset.

*data* The byte array containing the data to write to the Display module IOMap.

**6.52.2.37 void SetDisplayModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Display module IOMap value. Set one of the fields of the Display module IOMap structure to a new value. You provide the offset into the Display module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the new value should be written. See [Display module IOMAP offsets](#).

*value* A variable containing the new value to write to the Display module IOMap.

**6.52.2.38 void SetInputModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Input module IOMap value. Set one of the fields of the Input module IOMap structure to a new value. You provide the offset into the Input module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Input module IOMap structure where the new value should be written. See [Input module IOMAP offsets](#).

*value* A variable containing the new value to write to the Input module IOMap.

**6.52.2.39 void SetIOCtrlModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set IOCtrl module IOMap value. Set one of the fields of the IOCtrl module IOMap structure to a new value. You provide the offset into the IOCtrl module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the IOCtrl module IOMap structure where the new value should be written. See [IOCtrl module IOMAP offsets](#).

*value* A variable containing the new value to write to the IOCtrl module IOMap.

**6.52.2.40 void SetIOMapBytes (string *moduleName*, unsigned int *offset*, unsigned int *count*, byte *data*[ ]) [inline]**

Set IOMap bytes by name. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*moduleName* The module name of the IOMap to modify. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be written

*count* The number of bytes to write at the specified IOMap offset.

*data* The byte array containing the data to write to the IOMap

**6.52.2.41 void SetIOMapBytesByID (unsigned long *moduleId*, unsigned int *offset*, unsigned int *count*, byte *data*[ ]) [inline]**

Set IOMap bytes by ID. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*moduleId* The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be written.

*count* The number of bytes to write at the specified IOMap offset.

*data* The byte array containing the data to write to the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.52.2.42 void SetIOMapValue (string *moduleName*, unsigned int *offset*, variant *value*) [inline]**

Set IOMap value by name. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*moduleName* The module name of the IOMap to modify. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the new value should be written

*value* A variable containing the new value to write to the IOMap

**6.52.2.43 void SetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant *value*) [inline]**

Set IOMap value by ID. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*moduleId* The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

*offset* The number of bytes offset from the start of the IOMap structure where the new value should be written.

*value* A variable containing the new value to write to the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.52.2.44 void SetLoaderModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Loader module IOMap value. Set one of the fields of the Loader module IOMap structure to a new value. You provide the offset into the Loader module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Loader module IOMap structure where the new value should be written. See [Loader module IOMAP offsets](#).

*value* A variable containing the new value to write to the Loader module IOMap.

**6.52.2.45 void SetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]**

Set Lowspeed module IOMap bytes. Modify one or more bytes of data in the Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be written. See [Low speed module IOMAP offsets](#).

*count* The number of bytes to write at the specified Lowspeed module IOMap offset.

*data* The byte array containing the data to write to the Lowspeed module IOMap.

**6.52.2.46 void SetLowSpeedModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Lowspeed module IOMap value. Set one of the fields of the Lowspeed module IOMap structure to a new value. You provide the offset into the Lowspeed module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Lowspeed module IOMap structure where the new value should be written. See [Low speed module IOMAP offsets](#).

*value* A variable containing the new value to write to the Lowspeed module IOMap.

**6.52.2.47 void SetOutputModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Output module IOMap value. Set one of the fields of the Output module IOMap structure to a new value. You provide the offset into the Output module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Output module IOMap structure where the new value should be written. See [Output module IOMAP offsets](#).

*value* A variable containing the new value to write to the Output module IOMap.

**6.52.2.48 void SetSoundModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Sound module IOMap value. Set one of the fields of the Sound module IOMap structure to a new value. You provide the offset into the Sound module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Sound module IOMap structure where the new value should be written. See [Sound module IOMAP offsets](#).

*value* A variable containing the new value to write to the Sound module IOMap.

**6.52.2.49 void SetUIModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Ui module IOMap value. Set one of the fields of the Ui module IOMap structure to a new value. You provide the offset into the Ui module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Ui module IOMap structure where the new value should be written. See [Ui module IOMAP offsets](#).

*value* A variable containing the new value to write to the Ui module IOMap.

**6.52.2.50 void StartTask (task *t*) [inline]**

Start a task. Start the specified task.

**Parameters:**

*t* The task to start.

**Examples:**

[ex\\_StartTask.nxc](#).

**6.52.2.51 void Stop (bool *bvalue*) [inline]**

Stop the running program. Stop the running program if *bvalue* is true. This will halt the program completely, so any code following this command will be ignored.

**Parameters:**

*bvalue* If this value is true the program will stop executing.

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_Stop.nxc](#).

**6.52.2.52 void StopAllTasks () [inline]**

Stop all tasks. Stop all currently running tasks. This will halt the program completely, so any code following this command will be ignored.

**Examples:**

[ex\\_StopAllTasks.nxc](#).

**6.52.2.53 void StopTask (task *t*) [inline]**

Stop a task. Stop the specified task.

**Parameters:**

*t* The task to stop.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_StopTask.nxc](#).

**6.52.2.54 void SysCall (byte *funcID*, variant & *args*) [inline]**

Call any system function. This generic macro can be used to call any system function. No type checking is performed so you need to make sure you use the correct structure type given the selected system function ID. This is, however, the fastest possible way to call a system function in NXC.

Valid function ID constants are defined in the [System Call function constants](#) group.

**Parameters:**

*funcID* The function ID constant corresponding to the function to be called.

*args* The structure containing the needed parameters.

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_syscall.nxc](#).

**6.52.2.55 void SysComputeCalibValue (ComputeCalibValueType & *args*) [inline]**

Compute calibration values. This function lets you compute calibration values using the values specified via the [ComputeCalibValueType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [ComputeCalibValueType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysComputeCalibValue.nxc](#).

**6.52.2.56 void SysDatalogGetTimes (DatalogGetTimesType & args) [inline]**

Get datalog times. This function lets you get datalog times using the values specified via the [DatalogGetTimesType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [DatalogGetTimesType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_sysdataloggettimes.nxc](#).

**6.52.2.57 void SysDatalogWrite (DatalogWriteType & args) [inline]**

Write to the datalog. This function lets you write to the datalog using the values specified via the [DatalogWriteType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [DatalogWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysDatalogWrite.nxc](#).

**6.52.2.58 void SysGetStartTick (GetStartTickType & args) [inline]**

Get start tick. This function lets you obtain the tick value at the time your program began executing via the [GetStartTickType](#) structure.

**Parameters:**

*args* The [GetStartTickType](#) structure receiving results.

**Examples:**

[ex\\_sysgetstarttick.nxc](#).

**6.52.2.59 void SysIOMapRead (IOMapReadType & args) [inline]**

Read from IOMap by name. This function lets you read data from a firmware module's IOMap using the values specified via the [IOMapReadType](#) structure.

**Parameters:**

*args* The [IOMapReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysiomapread.nxc](#).

**6.52.2.60 void SysIOMapReadByID (IOMapReadByIDType & args) [inline]**

Read from IOMap by identifier. This function lets you read data from a firmware module's IOMap using the values specified via the [IOMapReadByIDType](#) structure. This function can be as much as three times faster than using SysIOMapRead since it does not have to do a string lookup using the ModuleName.

**Parameters:**

*args* The [IOMapReadByIDType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**6.52.2.61 void SysIOMapWrite (IOMapWriteType & args) [inline]**

Write to IOMap by name. This function lets you write data to a firmware module's IOMap using the values specified via the [IOMapWriteType](#) structure.

**Parameters:**

*args* The [IOMapWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysiomapwrite.nxc](#).

**6.52.2.62 void SysIOMapWriteByID (IOMapWriteByIDType & args) [inline]**

Write to IOMap by identifier. This function lets you write data to a firmware module's IOMap using the values specified via the [IOMapWriteByIDType](#) structure. This function can be as much as three times faster than using SysIOMapWrite since it does not have to do a string lookup using the ModuleName.

**Parameters:**

*args* The [IOMapWriteByIDType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**6.52.2.63 void SysKeepAlive (KeepAliveType & args) [inline]**

Keep alive. This function lets you reset the sleep timer via the [KeepAliveType](#) structure.

**Parameters:**

*args* The [KeepAliveType](#) structure receiving results.

**Examples:**

[ex\\_syskeepalive.nxc](#).

**6.52.2.64 void SysMemoryManager (MemoryManagerType & args)  
[inline]**

Read memory information. This function lets you read memory information using the values specified via the [MemoryManagerType](#) structure.

**Parameters:**

*args* The [MemoryManagerType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysmemorymanager.nxc](#).

**6.52.2.65 void SysReadLastResponse (ReadLastResponseType & args)  
[inline]**

Read last response information. This function lets you read the last system or direct command response received by the NXT using the values specified via the [ReadLastResponseType](#) structure.

**Parameters:**

*args* The [ReadLastResponseType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_SysReadLastResponse.nxc](#).

**6.52.2.66 void SysReadSemData (ReadSemDataType & args) [inline]**

Read semaphore data. This function lets you read global motor semaphore data using the values specified via the [ReadSemDataType](#) structure.

**Parameters:**

*args* The [ReadSemDataType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysReadSemData.nxc](#).

**6.52.2.67 void SysUpdateCalibCacheInfo (UpdateCalibCacheInfoType & args) [inline]**

Update calibration cache information. This function lets you update calibration cache information using the values specified via the [UpdateCalibCacheInfoType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [UpdateCalibCacheInfoType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

**6.52.2.68 void SysWriteSemData (WriteSemDataType & args) [inline]**

Write semaphore data. This function lets you write global motor semaphore data using the values specified via the [WriteSemDataType](#) structure.

**Parameters:**

*args* The [WriteSemDataType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysWriteSemData.nxc](#).

**6.52.2.69 void Wait (unsigned long *ms*) [inline]**

Wait some milliseconds. Make a task sleep for specified amount of time (in 1000ths of a second).

**Parameters:**

*ms* The number of milliseconds to sleep.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_CircleOut.nxc](#), [ex\\_clearline.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_contrast.nxc](#), [ex\\_copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispfout.nxc](#), [ex\\_dispfunc.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getchar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_onfwdsyncpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_PFMate.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_PosReg.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_ResetSensorHTAngle.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommHSControl.nxc](#),

[ex\\_SysCommHSRead.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysdrawpolygon.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_UnflattenVar.nxc](#), [ex\\_wait.nxc](#), [ex\\_xg1300.nxc](#), [ex\\_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

#### 6.52.2.70 void Yield () [inline]

Yield to another task. Make a task yield to another concurrently running task.

#### Examples:

[ex\\_yield.nxc](#).

## 6.53 Comparison Constants

Logical comparison operators for use in BranchTest and BranchComp.

### Defines

- #define [LT](#) 0x00
- #define [GT](#) 0x01
- #define [LTEQ](#) 0x02
- #define [GTEQ](#) 0x03
- #define [EQ](#) 0x04
- #define [NEQ](#) 0x05

#### 6.53.1 Detailed Description

Logical comparison operators for use in BranchTest and BranchComp.

#### 6.53.2 Define Documentation

##### 6.53.2.1 #define EQ 0x04

The first value is equal to the second.

### 6.53.2.2 #define GT 0x01

The first value is greater than the second.

#### Examples:

[ex\\_nbcopt.nxc](#).

### 6.53.2.3 #define GTEQ 0x03

The first value is greater than or equal to the second.

### 6.53.2.4 #define LT 0x00

The first value is less than the second.

### 6.53.2.5 #define LTEQ 0x02

The first value is less than or equal to the second.

### 6.53.2.6 #define NEQ 0x05

The first value is not equal to the second.

## 6.54 Array API functions

Functions for use with NXC array types.

### Functions

- void [ArrayBuild](#) (variant &aout[ ], variant src1, variant src2,..., variant srcN)  
*Build an array.*
- unsigned int [ArrayLen](#) (variant data[ ])  
*Get array length.*
- void [ArrayInit](#) (variant &aout[ ], variant value, unsigned int count)  
*Initialize an array.*
- void [ArraySubset](#) (variant &aout[ ], variant asrc[ ], unsigned int idx, unsigned int len)

*Copy an array subset.*

- void [ArrayIndex](#) (variant &out, variant asrc[ ], unsigned int idx)  
*Extract item from an array.*
- void [ArrayReplace](#) (variant &asrc[ ], unsigned int idx, variant value)  
*Replace items in an array.*
- variant [ArraySum](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the sum of the elements in a numeric array.*
- variant [ArrayMean](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the mean of the elements in a numeric array.*
- variant [ArraySumSqr](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the sum of the squares of the elements in a numeric array.*
- variant [ArrayStd](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the standard deviation of the elements in a numeric array.*
- variant [ArrayMin](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the minimum of the elements in a numeric array.*
- variant [ArrayMax](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the maximum of the elements in a numeric array.*
- void [ArraySort](#) (variant &dest[ ], const variant &src[ ], unsigned int idx, unsigned int len)  
*Sort the elements in a numeric array.*
- void [ArrayOp](#) (const byte op, variant &dest, const variant &src[ ], unsigned int idx, unsigned int len)  
*Operate on numeric arrays.*

### 6.54.1 Detailed Description

Functions for use with NXC array types.

### 6.54.2 Function Documentation

#### 6.54.2.1 void ArrayBuild (variant & *aout*[], variant *src1*, variant *src2*, ..., variant *srcN*) [inline]

Build an array. Build a new array from the specified source(s). The sources can be of any type so long as the number of dimensions is equal to or one less than the number of dimensions in the output array and the type is compatible with the type of the output array. If a source is an array with the same number of dimensions as the output array then all of its elements are added to the output array.

##### Parameters:

*aout* The output array to build.

*src1* The first source to build into the output array.

*src2* The second source to build into the output array.

*srcN* The first source to build into the output array.

##### Warning:

You cannot use NXC expressions with this function

##### Examples:

[ex\\_ArrayBuild.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_SysCommHSWrite.nxc](#), [ex\\_SysDatalogWrite.nxc](#), and [ex\\_sysmemorymanager.nxc](#).

#### 6.54.2.2 void ArrayIndex (variant & *out*, variant *asrc*[], unsigned int *idx*) [inline]

Extract item from an array. Extract one element from an array. The output type depends on the type of the source array.

##### Parameters:

*out* The output value.

*asrc* The input array from which to extract an item.

*idx* The index of the item to extract.

##### Warning:

You cannot use NXC expressions with this function

**Examples:**

[ex\\_nbcopt.nxc](#).

**6.54.2.3 void ArrayInit (variant & aout[ ], variant value, unsigned int count) [inline]**

Initialize an array. Initialize the array to contain count elements with each element equal to the value provided. To initialize a multi-dimensional array, the value should be an array of N-1 dimensions, where N is the number of dimensions in the array being initialized.

**Parameters:**

*aout* The output array to initialize.

*value* The value to initialize each element to.

*count* The number of elements to create in the output array.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayInit.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_nbcopt.nxc](#), [ex\\_sysdrawgraphic.nxc](#), and [ex\\_sysmemorymanager.nxc](#).

**6.54.2.4 unsigned int ArrayLen (variant data[ ]) [inline]**

Get array length. Return the length of the specified array. Any type of array of up to four dimensions can be passed into this function.

**Parameters:**

*data* The array whose length you need to read.

**Returns:**

The length of the specified array.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayLen.nxc](#), [ex\\_atan2.nxc](#), [ex\\_atan2d.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_tan.nxc](#), and [ex\\_tand.nxc](#).

**6.54.2.5 variant ArrayMax (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the maximum of the elements in a numeric array. This function calculates the maximum of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from idx to the end of the array).

**Returns:**

The maximum of len elements from the src numeric array (starting from idx).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMax.nxc](#), and [ex\\_ArraySort.nxc](#).

**6.54.2.6 variant ArrayMean (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the mean of the elements in a numeric array. This function calculates the mean of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The mean value of *len* elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMean.nxc](#).

**6.54.2.7 variant ArrayMin (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the minimum of the elements in a numeric array. This function calculates the minimum of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The minimum of *len* elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMin.nxc](#), and [ex\\_ArraySort.nxc](#).

**6.54.2.8 void ArrayOp (const byte *op*, variant & *dest*, const variant & *src*[], unsigned int *idx*, unsigned int *len*) [inline]**

Operate on numeric arrays. This function lets you perform various operations on numeric arrays.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*op* The array operation. See [Array operation constants](#).

*dest* The destination variant type (scalar or array, depending on the operation).

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the specified process. Pass [NA](#) to include the rest of the elements in the *src* array (from *idx* to the end of the array).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayOp.nxc](#).

**6.54.2.9 void ArrayReplace (variant & *asrc*[], unsigned int *idx*, variant *value*) [inline]**

Replace items in an array. Replace one or more items in the specified source array. The items are replaced starting at the specified index. If the value provided has the same number of dimensions as the source array then multiple items in the source are replaced. If the value provided has one less dimension than the source array then one item will be replaced. Other differences between the source array and the new value dimensionality are not supported.

**Parameters:**

*asrc* The input array to be modified

*idx* The index of the item to replace.

*value* The new value or values to put into the source array.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_nbcopt.nxc](#).

**6.54.2.10 void ArraySort (variant & *dest*[], const variant & *src*[], unsigned int *idx*, unsigned int *len*) [inline]**

Sort the elements in a numeric array. This function sorts all or a subset of the elements in the numeric *src* array in ascending order and saves the results in the numeric *dest* array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*dest* The destination numeric array.

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the sorting process. Pass [NA](#) to include the rest of the elements in the *src* array (from *idx* to the end of the array).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySort.nxc](#).

**6.54.2.11 variant ArrayStd (const variant & src[ ], unsigned int idx, unsigned int len) [inline]**

Calculate the standard deviation of the elements in a numeric array. This function calculates the standard deviation of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from idx to the end of the array).

**Returns:**

The standard deviation of len elements from the src numeric array (starting from idx).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayStd.nxc](#).

**6.54.2.12 void ArraySubset (variant & aout[ ], variant asrc[ ], unsigned int idx, unsigned int len) [inline]**

Copy an array subset. Copy a subset of the source array starting at the specified index and containing the specified number of elements into the destination array.

**Parameters:**

- argout* The output array containing the subset.
- asrc* The input array from which to copy a subset.
- idx* The start index of the array subset.
- len* The length of the array subset.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySubset.nxc](#).

**6.54.2.13 variant ArraySum (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the sum of the elements in a numeric array. This function calculates the sum of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

- src* The source numeric array.
- idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.
- len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from idx to the end of the array).

**Returns:**

The sum of len elements from the src numeric array (starting from idx).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySum.nxc](#).

#### 6.54.2.14 variant ArraySumSqr (const variant & src[], unsigned int idx, unsigned int len) [inline]

Calculate the sum of the squares of the elements in a numeric array. This function calculates the sum of the squares of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from idx to the end of the array).

**Returns:**

The sum of the squares of len elements from the src numeric array (starting from idx).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySumSqr.nxc](#).

## 6.55 IOCtrl module types

Types used by various IOCtrl module functions. Types used by various IOCtrl module functions.

## 6.56 IOCtrl module functions

Functions for accessing and modifying IOCtrl module features.

## Functions

- void [PowerDown \(\)](#)  
*Power down the NXT.*
- void [SleepNow \(\)](#)  
*Put the brick to sleep immediately.*
- void [RebootInFirmwareMode \(\)](#)  
*Reboot the NXT in firmware download mode.*

### 6.56.1 Detailed Description

Functions for accessing and modifying IOCtrl module features.

### 6.56.2 Function Documentation

#### 6.56.2.1 void PowerDown () [inline]

Power down the NXT. This function powers down the NXT. The running program will terminate as a result of this action.

#### Examples:

[ex\\_PowerDown.nxc](#).

#### 6.56.2.2 void RebootInFirmwareMode () [inline]

Reboot the NXT in firmware download mode. This function lets you reboot the NXT into SAMBA or firmware download mode. The running program will terminate as a result of this action.

#### Examples:

[ex\\_RebootInFirmwareMode.nxc](#).

### 6.56.2.3 void SleepNow () [inline]

Put the brick to sleep immediately. This function lets you immediately put the NXT to sleep. The running program will terminate as a result of this action.

#### Examples:

[ex\\_SleepNow.nxc](#).

## 6.57 Comm module types

Types used by various Comm module functions.

### Data Structures

- struct [MessageWriteType](#)  
*Parameters for the MessageWrite system call.*
- struct [MessageReadType](#)  
*Parameters for the MessageRead system call.*
- struct [CommBTCheckStatusType](#)  
*Parameters for the CommBTCheckStatus system call.*
- struct [CommBTWriteType](#)  
*Parameters for the CommBTWrite system call.*
- struct [JoystickMessageType](#)  
*The JoystickMessageType structure.*
- struct [CommExecuteFunctionType](#)  
*Parameters for the CommExecuteFunction system call.*
- struct [CommHSControlType](#)  
*Parameters for the CommHSControl system call.*
- struct [CommHSCheckStatusType](#)  
*Parameters for the CommHSCheckStatus system call.*
- struct [CommHSReadWriteType](#)  
*Parameters for the CommHSReadWrite system call.*

- struct [CommBTOnOffType](#)  
*Parameters for the CommBTOnOff system call.*
- struct [CommBTConnectionType](#)  
*Parameters for the CommBTConnection system call.*

### 6.57.1 Detailed Description

Types used by various Comm module functions.

## 6.58 Comm module functions

Functions for accessing and modifying Comm module features.

### Modules

- [Direct Command functions](#)  
*Functions for sending direct commands to another NXT.*
- [System Command functions](#)  
*Functions for sending system commands to another NXT.*

### Functions

- char [JoystickMessageRead](#) (byte queue, [JoystickMessageType](#) &msg)  
*Read a joystick message from a queue/mailbox.*
- char [SendMessage](#) (byte queue, string msg)  
*Send a message to a queue/mailbox.*
- char [ReceiveMessage](#) (byte queue, bool clear, string &msg)  
*Read a message from a queue/mailbox.*
- char [BluetoothStatus](#) (byte conn)  
*Check bluetooth status.*
- char [BluetoothWrite](#) (byte conn, byte buffer[ ])  
*Write to a bluetooth connection.*

- char [RemoteConnectionWrite](#) (byte conn, byte buffer[ ])   
*Write to a remote connection.*
- bool [RemoteConnectionIdle](#) (byte conn)   
*Check if remote connection is idle.*
- char [SendRemoteBool](#) (byte conn, byte queue, bool bval)   
*Send a boolean value to a remote mailbox.*
- char [SendRemoteNumber](#) (byte conn, byte queue, long val)   
*Send a numeric value to a remote mailbox.*
- char [SendRemoteString](#) (byte conn, byte queue, string str)   
*Send a string value to a remote mailbox.*
- char [SendResponseBool](#) (byte queue, bool bval)   
*Write a boolean value to a local response mailbox.*
- char [SendResponseNumber](#) (byte queue, long val)   
*Write a numeric value to a local response mailbox.*
- char [SendResponseString](#) (byte queue, string str)   
*Write a string value to a local response mailbox.*
- char [ReceiveRemoteBool](#) (byte queue, bool clear, bool &bval)   
*Read a boolean value from a queue/mailbox.*
- char [ReceiveRemoteMessageEx](#) (byte queue, bool clear, string &str, long &val, bool &bval)   
*Read a value from a queue/mailbox.*
- char [ReceiveRemoteNumber](#) (byte queue, bool clear, long &val)   
*Read a numeric value from a queue/mailbox.*
- char [ReceiveRemoteString](#) (byte queue, bool clear, string &str)   
*Read a string value from a queue/mailbox.*
- void [UseRS485](#) (void)   
*Use the RS485 port.*
- char [RS485Control](#) (byte cmd, byte baud, unsigned int mode)

*Control the RS485 port.*

- byte [RS485DataAvailable](#) (void)  
*Check for RS485 available data.*
- char [RS485Initialize](#) (void)  
*Initialize RS485 port.*
- char [RS485Disable](#) (void)  
*Disable RS485.*
- char [RS485Enable](#) (void)  
*Enable RS485.*
- char [RS485Read](#) (byte &buffer[ ])  
*Read RS485 data.*
- char [RS485ReadEx](#) (byte &buffer[ ], byte buflen)  
*Read limited RS485 data.*
- byte [RS485SendingData](#) (void)  
*Is RS485 sending data.*
- void [RS485Status](#) (byte &sendingData, byte &dataAvail)  
*Check RS485 status.*
- char [RS485Uart](#) (byte baud, unsigned int mode)  
*Configure RS485 UART.*
- char [RS485Write](#) (byte buffer[ ])  
*Write RS485 data.*
- char [SendRS485Bool](#) (bool bval)  
*Write RS485 boolean.*
- char [SendRS485Number](#) (long val)  
*Write RS485 numeric.*
- char [SendRS485String](#) (string str)  
*Write RS485 string.*
- void [GetBTInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])

*Get bluetooth input buffer data.*

- void [GetBTOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get bluetooth output buffer data.*
- void [GetHSInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get hi-speed port input buffer data.*
- void [GetHSOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get hi-speed port output buffer data.*
- void [GetUSBInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb input buffer data.*
- void [GetUSBOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb output buffer data.*
- void [GetUSBPollBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb poll buffer data.*
- string [BTDeviceName](#) (const byte devidx)  
*Get bluetooth device name.*
- string [BTConnectionName](#) (const byte conn)  
*Get bluetooth device name.*
- string [BTConnectionPinCode](#) (const byte conn)  
*Get bluetooth device pin code.*
- string [BrickDataName](#) (void)  
*Get NXT name.*
- void [GetBTDeviceAddress](#) (const byte devidx, byte &data[ ])  
*Get bluetooth device address.*
- void [GetBTConnectionAddress](#) (const byte conn, byte &data[ ])  
*Get bluetooth device address.*
- void [GetBrickDataAddress](#) (byte &data[ ])  
*Get NXT address.*
- long [BTDeviceClass](#) (const byte devidx)

*Get bluetooth device class.*

- byte **BTDeviceStatus** (const byte devidx)  
*Get bluetooth device status.*
- long **BTConnectionClass** (const byte conn)  
*Get bluetooth device class.*
- byte **BTConnectionHandleNum** (const byte conn)  
*Get bluetooth device handle number.*
- byte **BTConnectionStreamStatus** (const byte conn)  
*Get bluetooth device stream status.*
- byte **BTConnectionLinkQuality** (const byte conn)  
*Get bluetooth device link quality.*
- int **BrickDataBluecoreVersion** (void)  
*Get NXT bluecore version.*
- byte **BrickDataBtStateStatus** (void)  
*Get NXT bluetooth state status.*
- byte **BrickDataBtHardwareStatus** (void)  
*Get NXT bluetooth hardware status.*
- byte **BrickDataTimeoutValue** (void)  
*Get NXT bluetooth timeout value.*
- byte **BTInputBufferInPtr** (void)  
*Get bluetooth input buffer in-pointer.*
- byte **BTInputBufferOutPtr** (void)  
*Get bluetooth input buffer out-pointer.*
- byte **BTOutputBufferInPtr** (void)  
*Get bluetooth output buffer in-pointer.*
- byte **BTOutputBufferOutPtr** (void)  
*Get bluetooth output buffer out-pointer.*
- byte **HSInputBufferInPtr** (void)

*Get hi-speed port input buffer in-pointer.*

- byte [HSInputBufferOutPtr](#) (void)  
*Get hi-speed port input buffer out-pointer.*
- byte [HSOutputBufferInPtr](#) (void)  
*Get hi-speed port output buffer in-pointer.*
- byte [HSOutputBufferOutPtr](#) (void)  
*Get hi-speed port output buffer out-pointer.*
- byte [USBInputBufferInPtr](#) (void)  
*Get usb port input buffer in-pointer.*
- byte [USBInputBufferOutPtr](#) (void)  
*Get usb port input buffer out-pointer.*
- byte [USBOutputBufferInPtr](#) (void)  
*Get usb port output buffer in-pointer.*
- byte [USBOutputBufferOutPtr](#) (void)  
*Get usb port output buffer out-pointer.*
- byte [USBPollBufferInPtr](#) (void)  
*Get usb port poll buffer in-pointer.*
- byte [USBPollBufferOutPtr](#) (void)  
*Get usb port poll buffer out-pointer.*
- byte [BTDeviceCount](#) (void)  
*Get bluetooth device count.*
- byte [BTDeviceNameCount](#) (void)  
*Get bluetooth device name count.*
- byte [HSFlags](#) (void)  
*Get hi-speed port flags.*
- byte [HSSpeed](#) (void)  
*Get hi-speed port speed.*
- byte [HSState](#) (void)

*Get hi-speed port state.*

- byte **HSAddress** (void)  
*Get hi-speed port address.*
- int **HSMode** (void)  
*Get hi-speed port mode.*
- int **BTDataMode** (void)  
*Get Bluetooth data mode.*
- int **HSDDataMode** (void)  
*Get hi-speed port datamode.*
- byte **USBState** (void)  
*Get USB state.*
- void **SetBTInputBuffer** (const byte offset, byte cnt, byte data[ ])  
*Set bluetooth input buffer data.*
- void **SetBTInputBufferInPtr** (byte n)  
*Set bluetooth input buffer in-pointer.*
- void **SetBTInputBufferOutPtr** (byte n)  
*Set bluetooth input buffer out-pointer.*
- void **SetBTOutputBuffer** (const byte offset, byte cnt, byte data[ ])  
*Set bluetooth output buffer data.*
- void **SetBTOutputBufferInPtr** (byte n)  
*Set bluetooth output buffer in-pointer.*
- void **SetBTOutputBufferOutPtr** (byte n)  
*Set bluetooth output buffer out-pointer.*
- void **SetHSInputBuffer** (const byte offset, byte cnt, byte data[ ])  
*Set hi-speed port input buffer data.*
- void **SetHSInputBufferInPtr** (byte n)  
*Set hi-speed port input buffer in-pointer.*
- void **SetHSInputBufferOutPtr** (byte n)

*Set hi-speed port input buffer out-pointer.*

- void [SetHSOutputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set hi-speed port output buffer data.*
- void [SetHSOutputBufferInPtr](#) (byte n)  
*Set hi-speed port output buffer in-pointer.*
- void [SetHSOutputBufferOutPtr](#) (byte n)  
*Set hi-speed port output buffer out-pointer.*
- void [SetUSBInputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB input buffer data.*
- void [SetUSBInputBufferInPtr](#) (byte n)  
*Set USB input buffer in-pointer.*
- void [SetUSBInputBufferOutPtr](#) (byte n)  
*Set USB input buffer out-pointer.*
- void [SetUSBOutputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB output buffer data.*
- void [SetUSBOutputBufferInPtr](#) (byte n)  
*Set USB output buffer in-pointer.*
- void [SetUSBOutputBufferOutPtr](#) (byte n)  
*Set USB output buffer out-pointer.*
- void [SetUSBPollBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB poll buffer data.*
- void [SetUSBPollBufferInPtr](#) (byte n)  
*Set USB poll buffer in-pointer.*
- void [SetUSBPollBufferOutPtr](#) (byte n)  
*Set USB poll buffer out-pointer.*
- void [SetHSFlags](#) (byte hsFlags)  
*Set hi-speed port flags.*
- void [SetHSSpeed](#) (byte hsSpeed)

*Set hi-speed port speed.*

- void [SetHSState](#) (byte hsState)  
*Set hi-speed port state.*
- void [SetHSAddress](#) (byte hsAddress)  
*Set hi-speed port address.*
- void [SetHSMode](#) (unsigned int hsMode)  
*Set hi-speed port mode.*
- void [SetBTDataMode](#) (const byte dataMode)  
*Set Bluetooth data mode.*
- void [SetHSDataMode](#) (const byte dataMode)  
*Set hi-speed port data mode.*
- void [SetUSBState](#) (byte usbState)  
*Set USB state.*
- void [SysMessageWrite](#) ([MessageWriteType](#) &args)  
*Write message.*
- void [SysMessageRead](#) ([MessageReadType](#) &args)  
*Read message.*
- void [SysCommBTWrite](#) ([CommBTWriteType](#) &args)  
*Write data to a Bluetooth connection.*
- void [SysCommBTCheckStatus](#) ([CommBTCheckStatusType](#) &args)  
*Check Bluetooth connection status.*
- void [SysCommExecuteFunction](#) ([CommExecuteFunctionType](#) &args)  
*Execute any Comm module command.*
- void [SysCommHSControl](#) ([CommHSControlType](#) &args)  
*Control the hi-speed port.*
- void [SysCommHSCheckStatus](#) ([CommHSCheckStatusType](#) &args)  
*Check the hi-speed port status.*
- void [SysCommHSRead](#) ([CommHSReadWriteType](#) &args)

*Read from the hi-speed port.*

- void [SysCommHSWrite](#) ([CommHSReadWriteType](#) &args)  
*Write to the hi-speed port.*
- void [SysCommBTONOff](#) ([CommBTONOffType](#) &args)  
*Turn on or off the bluetooth subsystem.*
- void [SysCommBTConnection](#) ([CommBTConnectionType](#) &args)  
*Connect or disconnect a bluetooth device.*

### 6.58.1 Detailed Description

Functions for accessing and modifying Comm module features.

### 6.58.2 Function Documentation

#### 6.58.2.1 char [BluetoothStatus](#) (byte *conn*) [[inline](#)]

Check bluetooth status. Check the status of the bluetooth subsystem for the specified connection slot.

#### Parameters:

*conn* The connection slot (0..3). Connections 0 through 3 are for bluetooth connections. See [Remote connection constants](#).

#### Returns:

The bluetooth status for the specified connection.

#### Examples:

[ex\\_BluetoothStatus.nxc](#), and [ex\\_syscommbtconnection.nxc](#).

#### 6.58.2.2 char [BluetoothWrite](#) (byte *conn*, byte *buffer*[]) [[inline](#)]

Write to a bluetooth connection. This method tells the NXT firmware to write the data in the buffer to the device on the specified Bluetooth connection. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3). Connections 0 through 3 are for bluetooth connections. See [Remote connection constants](#).

*buffer* The data to be written (up to 128 bytes)

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_BluetoothWrite.nxc](#).

**6.58.2.3 int BrickDataBluecoreVersion (void) [inline]**

Get NXT bluecore version. This method returns the bluecore version of the NXT.

**Returns:**

The NXT's bluecore version number.

**Examples:**

[ex\\_BrickDataBluecoreVersion.nxc](#).

**6.58.2.4 byte BrickDataBtHardwareStatus (void) [inline]**

Get NXT bluetooth hardware status. This method returns the Bluetooth hardware status of the NXT.

**Returns:**

The NXT's bluetooth hardware status.

**Examples:**

[ex\\_BrickDataBtHardwareStatus.nxc](#).

**6.58.2.5 byte BrickDataBtStateStatus (void) [inline]**

Get NXT bluetooth state status. This method returns the Bluetooth state status of the NXT.

**Returns:**

The NXT's bluetooth state status.

**Examples:**

[ex\\_BrickDataBtStateStatus.nxc](#).

**6.58.2.6 string BrickDataName (void) [inline]**

Get NXT name. This method returns the name of the NXT.

**Returns:**

The NXT's bluetooth name.

**Examples:**

[ex\\_BrickDataName.nxc](#).

**6.58.2.7 byte BrickDataTimeoutValue (void) [inline]**

Get NXT bluetooth timeout value. This method returns the Bluetooth timeout value of the NXT.

**Returns:**

The NXT's bluetooth timeout value.

**Examples:**

[ex\\_BrickDataTimeoutValue.nxc](#).

**6.58.2.8 long BTConnectionClass (const byte *conn*) [inline]**

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The class of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionClass.nxc](#).

**6.58.2.9 byte BTConnectionHandleNum (const byte *conn*) [inline]**

Get bluetooth device handle number. This method returns the handle number of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The handle number of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionHandleNum.nxc](#).

**6.58.2.10 byte BTConnectionLinkQuality (const byte *conn*) [inline]**

Get bluetooth device link quality. This method returns the link quality of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The link quality of the specified connection slot (unimplemented).

**Warning:**

This function is not implemented at the firmware level.

**Examples:**

[ex\\_BTConnectionLinkQuality.nxc](#).

**6.58.2.11 string BTConnectionName (const byte *conn*) [inline]**

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The name of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionName.nxc](#).

**6.58.2.12 string BTConnectionPinCode (const byte *conn*) [inline]**

Get bluetooth device pin code. This method returns the pin code of the device at the specified index in the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The pin code for the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionPinCode.nxc](#).

**6.58.2.13** `byte BTConnectionStreamStatus (const byte conn) [inline]`

Get bluetooth device stream status. This method returns the stream status of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The stream status of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionStreamStatus.nxc](#).

**6.58.2.14** `int BTDataMode (void) [inline]`

Get Bluetooth data mode. This method returns the value of the Bluetooth data mode.

**Returns:**

The Bluetooth data mode. See [Data mode constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**6.58.2.15** `long BTDeviceClass (const byte devidx) [inline]`

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The device class of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceClass.nxc](#).

**6.58.2.16 byte BTDeviceCount (void) [inline]**

Get bluetooth device count. This method returns the number of devices defined within the Bluetooth device table.

**Returns:**

The count of known bluetooth devices.

**Examples:**

[ex\\_BTDeviceCount.nxc](#).

**6.58.2.17 string BTDeviceName (const byte *devidx*) [inline]**

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The device name of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceName.nxc](#).

**6.58.2.18 byte BTDeviceNameCount (void) [inline]**

Get bluetooth device name count. This method returns the number of device names defined within the Bluetooth device table. This usually has the same value as BTDeviceCount but it can differ in some instances.

**Returns:**

The count of known bluetooth device names.

**Examples:**

[ex\\_BTDeviceNameCount.nxc](#).

**6.58.2.19 byte BTDeviceStatus (const byte *devidx*) [inline]**

Get bluetooth device status. This method returns the status of the device at the specified index within the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The status of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceStatus.nxc](#).

**6.58.2.20 byte BTInputBufferInPtr (void) [inline]**

Get bluetooth input buffer in-pointer. This method returns the value of the input pointer of the Bluetooth input buffer.

**Returns:**

The bluetooth input buffer's in-pointer value.

**Examples:**

[ex\\_BTInputBufferInPtr.nxc](#).

**6.58.2.21 byte BTInputBufferOutPtr (void) [inline]**

Get bluetooth input buffer out-pointer. This method returns the value of the output pointer of the Bluetooth input buffer.

**Returns:**

The bluetooth input buffer's out-pointer value.

**Examples:**

[ex\\_BTInputBufferOutPtr.nxc](#).

**6.58.2.22 byte BTOutputBufferInPtr (void) [inline]**

Get bluetooth output buffer in-pointer. This method returns the value of the input pointer of the Bluetooth output buffer.

**Returns:**

The bluetooth output buffer's in-pointer value.

**Examples:**

[ex\\_BTOutputBufferInPtr.nxc](#).

**6.58.2.23 byte BTOutputBufferOutPtr (void) [inline]**

Get bluetooth output buffer out-pointer. This method returns the value of the output pointer of the Bluetooth output buffer.

**Returns:**

The bluetooth output buffer's out-pointer value.

**Examples:**

[ex\\_BTOutputBufferOutPtr.nxc](#).

**6.58.2.24 void GetBrickDataAddress (byte & data[]) [inline]**

Get NXT address. This method reads the address of the NXT and stores it in the data buffer provided.

**Parameters:**

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBrickDataAddress.nxc](#).

**6.58.2.25** `void GetBTConnectionAddress (const byte conn, byte & data[ ])`  
`[inline]`

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth connection table and stores it in the data buffer provided.

**Parameters:**

*conn* The connection slot (0..3).

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBTConnectionAddress.nxc](#).

**6.58.2.26** `void GetBTDeviceAddress (const byte devidx, byte & data[ ])`  
`[inline]`

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth device table and stores it in the data buffer provided.

**Parameters:**

*devidx* The device table index.

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBTDeviceAddress.nxc](#).

**6.58.2.27** `void GetBTInputBuffer (const byte offset, byte cnt, byte & data[ ])`  
`[inline]`

Get bluetooth input buffer data. This method reads count bytes of data from the Bluetooth input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the bluetooth input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the bluetooth input buffer.

**Examples:**

[ex\\_GetBTInputBuffer.nxc](#).

**6.58.2.28** `void GetBTOutputBuffer (const byte offset, byte cnt, byte & data[ ])`  
`[inline]`

Get bluetooth output buffer data. This method reads count bytes of data from the Bluetooth output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the bluetooth output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the bluetooth output buffer.

**Examples:**

[ex\\_GetBTOutputBuffer.nxc](#).

**6.58.2.29** `void GetHSInputBuffer (const byte offset, byte cnt, byte & data[ ])`  
`[inline]`

Get hi-speed port input buffer data. This method reads count bytes of data from the hi-speed port input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the hi-speed port input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the hi-speed port input buffer.

**Examples:**

[ex\\_GetHSInputBuffer.nxc](#).

**6.58.2.30 void GetHSOutputBuffer (const byte *offset*, byte *cnt*, byte & *data* [ ]  
[inline])**

Get hi-speed port output buffer data. This method reads count bytes of data from the hi-speed port output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the hi-speed port output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the hi-speed port output buffer.

**Examples:**

[ex\\_GetHSOutputBuffer.nxc](#).

**6.58.2.31 void GetUSBInputBuffer (const byte *offset*, byte *cnt*, byte & *data* [ ]  
[inline])**

Get usb input buffer data. This method reads count bytes of data from the usb input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb input buffer.

**Examples:**

[ex\\_GetUSBInputBuffer.nxc](#).

**6.58.2.32 void GetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[])  
[inline]**

Get usb output buffer data. This method reads count bytes of data from the usb output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb output buffer.

**Examples:**

[ex\\_GetUSBOutputBuffer.nxc](#).

**6.58.2.33 void GetUSBPollBuffer (const byte *offset*, byte *cnt*, byte & *data*[])  
[inline]**

Get usb poll buffer data. This method reads count bytes of data from the usb poll buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb poll buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb poll buffer.

**Examples:**

[ex\\_GetUSBPollBuffer.nxc](#).

**6.58.2.34** byte HSAddress (void) [`inline`]

Get hi-speed port address. This method returns the value of the hi-speed port address.

**Returns:**

The hi-speed port address. See [Hi-speed port address constants](#).

**6.58.2.35** int HSDataMode (void) [`inline`]

Get hi-speed port datamode. This method returns the value of the hi-speed port data mode.

**Returns:**

The hi-speed port data mode. See [Data mode constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**6.58.2.36** byte HSFlags (void) [`inline`]

Get hi-speed port flags. This method returns the value of the hi-speed port flags.

**Returns:**

The hi-speed port flags. See [Hi-speed port flags constants](#).

**Examples:**

[ex\\_HSFlags.nxc](#).

**6.58.2.37 byte HSInputBufferInPtr (void) [inline]**

Get hi-speed port input buffer in-pointer. This method returns the value of the input pointer of the hi-speed port input buffer.

**Returns:**

The hi-speed port input buffer's in-pointer value.

**Examples:**

[ex\\_HSInputBufferInPtr.nxc](#).

**6.58.2.38 byte HSInputBufferOutPtr (void) [inline]**

Get hi-speed port input buffer out-pointer. This method returns the value of the output pointer of the hi-speed port input buffer.

**Returns:**

The hi-speed port input buffer's out-pointer value.

**Examples:**

[ex\\_HSInputBufferOutPtr.nxc](#).

**6.58.2.39 int HSMode (void) [inline]**

Get hi-speed port mode. This method returns the value of the hi-speed port mode.

**Returns:**

The hi-speed port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_HSMode.nxc](#).

**6.58.2.40** byte HSOutputBufferInPtr (void) [`inline`]

Get hi-speed port output buffer in-pointer. This method returns the value of the input pointer of the hi-speed port output buffer.

**Returns:**

The hi-speed port output buffer's in-pointer value.

**Examples:**

[ex\\_HSOutputBufferInPtr.nxc](#).

**6.58.2.41** byte HSOutputBufferOutPtr (void) [`inline`]

Get hi-speed port output buffer out-pointer. This method returns the value of the output pointer of the hi-speed port output buffer.

**Returns:**

The hi-speed port output buffer's out-pointer value.

**Examples:**

[ex\\_HSOutputBufferOutPtr.nxc](#).

**6.58.2.42** byte HSSpeed (void) [`inline`]

Get hi-speed port speed. This method returns the value of the hi-speed port speed (baud rate).

**Returns:**

The hi-speed port speed (baud rate). See [Hi-speed port baud rate constants](#).

**Examples:**

[ex\\_HSSpeed.nxc](#).

**6.58.2.43 byte HSState (void) [inline]**

Get hi-speed port state. This method returns the value of the hi-speed port state.

**Returns:**

The hi-speed port state. See [Hi-speed port state constants](#).

**Examples:**

[ex\\_HSState.nxc](#).

**6.58.2.44 char JoystickMessageRead (byte queue, JoystickMessageType & msg) [inline]**

Read a joystick message from a queue/mailbox. Read a joystick message from a queue/mailbox.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*msg* The joystick message that is read from the mailbox. See [JoystickMessageType](#) for details.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_joystickmsg.nxc](#).

**6.58.2.45 char ReceiveMessage (byte queue, bool clear, string & msg) [inline]**

Read a message from a queue/mailbox. Read a message from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- msg* The message that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**6.58.2.46 char ReceiveRemoteBool (byte *queue*, bool *clear*, bool & *bval*) [inline]**

Read a boolean value from a queue/mailbox. Read a boolean value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- bval* The boolean value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteBool.nxc](#), and [ex\\_ReceiveRemoteNumber.nxc](#).

**6.58.2.47 char ReceiveRemoteMessageEx (byte *queue*, bool *clear*, string & *str*, long & *val*, bool & *bval*) [inline]**

Read a value from a queue/mailbox. Read a value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number. Output the value in string, number, and boolean form.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- str* The string value that is read from the mailbox.
- val* The numeric value that is read from the mailbox.
- bval* The boolean value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteMessageEx.nxc](#).

**6.58.2.48 char ReceiveRemoteNumber (byte *queue*, bool *clear*, long & *val*)  
[inline]**

Read a numeric value from a queue/mailbox. Read a numeric value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- val* The numeric value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**6.58.2.49 char ReceiveRemoteString (byte *queue*, bool *clear*, string & *str*)  
[inline]**

Read a string value from a queue/mailbox. Read a string value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- str* The string value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteString.nxc](#).

**6.58.2.50 bool RemoteConnectionIdle (byte conn) [inline]**

Check if remote connection is idle. Check whether a Bluetooth or RS485 hi-speed port connection is idle, i.e., not currently sending data.

**Parameters:**

- conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A boolean value indicating whether the connection is idle or busy.

**Warning:**

Checking the status of the RS485 hi-speed connection requires the enhanced NBC/NXC firmware

**Examples:**

[ex\\_RemoteConnectionIdle.nxc](#).

**6.58.2.51 char RemoteConnectionWrite (byte conn, byte buffer[]) [inline]**

Write to a remote connection. This method tells the NXT firmware to write the data in the buffer to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*buffer* The data to be written (up to 128 bytes)

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

Writing to the RS485 hi-speed connection requires the enhanced NBC/NXC firmware

**Examples:**

[ex\\_RemoteConnectionWrite.nxc](#).

**6.58.2.52 char RS485Control (byte *cmd*, byte *baud*, unsigned int *mode*)  
[inline]**

Control the RS485 port. Control the RS485 hi-speed port using the specified parameters.

**Parameters:**

*cmd* The control command to send to the port. See [Hi-speed port SysCommHSControl constants](#).

*baud* The baud rate for the RS485 port. See [Hi-speed port baud rate constants](#).

*mode* The RS485 port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.58.2.53 byte RS485DataAvailable (void) [inline]**

Check for RS485 available data. Check the RS485 hi-speed port for available data.

**Returns:**

The number of bytes of data available for reading.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.58.2.54 char RS485Disable (void) [inline]**

Disable RS485. Turn off the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**6.58.2.55 char RS485Enable (void) [inline]**

Enable RS485. Turn on the RS485 hi-speed port so that it can be used.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.58.2.56 char RS485Initialize (void) [inline]**

Initialize RS485 port. Initialize the RS485 UART port to its default values. The baud rate is set to 921600 and the mode is set to 8N1 (8 data bits, no parity, 1 stop bit). Data cannot be sent or received over the RS485 port until the port is configured as a hi-speed port, the port is turned on, and the UART is initialized.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.58.2.57 char RS485Read (byte & buffer[]) [inline]**

Read RS485 data. Read data from the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array that will contain the data read from the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.58.2.58 char RS485ReadEx (byte & *buffer*[], byte *buflen*) [inline]**

Read limited RS485 data. Read a limited number of bytes of data from the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array that will contain the data read from the RS485 port.

*buflen* The number of bytes you want to read.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_RS485Receive.nxc](#).

**6.58.2.59 byte RS485SendingData (void) [inline]**

Is RS485 sending data. Check whether the RS485 is actively sending data.

**Returns:**

The number of bytes of data being sent.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.58.2.60 void RS485Status (byte & *sendingData*, byte & *dataAvail*) [inline]**

Check RS485 status. Check the status of the RS485 hi-speed port.

**Parameters:**

*sendingData* The number of bytes of data being sent.

*dataAvail* The number of bytes of data available for reading.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.58.2.61 char RS485Uart (byte *baud*, unsigned int *mode*) [inline]**

Configure RS485 UART. Configure the RS485 UART parameters, including baud rate, data bits, stop bits, and parity.

**Parameters:**

*baud* The baud rate for the RS485 port. See [Hi-speed port baud rate constants](#).

*mode* The RS485 port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.58.2.62 char RS485Write (byte *buffer*[ ]) [inline]**

Write RS485 data. Write data to the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array containing the data to write to the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#).

**6.58.2.63 char SendMessage (byte *queue*, string *msg*) [inline]**

Send a message to a queue/mailbox. Write a message into a local mailbox.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*msg* The message to write to the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendMessage.nxc](#).

**6.58.2.64 char SendRemoteBool (byte *conn*, byte *queue*, bool *bval*) [inline]**

Send a boolean value to a remote mailbox. Send a boolean value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*bval* The boolean value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteBool.nxc](#).

**6.58.2.65 char SendRemoteNumber (byte *conn*, byte *queue*, long *val*)  
[inline]**

Send a numeric value to a remote mailbox. Send a numeric value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*val* The numeric value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteNumber.nxc](#).

**6.58.2.66 char SendRemoteString (byte *conn*, byte *queue*, string *str*)  
[inline]**

Send a string value to a remote mailbox. Send a string value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*str* The string value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteString.nxc](#).

**6.58.2.67 char SendResponseBool (byte *queue*, bool *bval*) [inline]**

Write a boolean value to a local response mailbox. Write a boolean value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*bval* The boolean value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseBool.nxc](#).

**6.58.2.68 char SendResponseNumber (byte *queue*, long *val*) [inline]**

Write a numeric value to a local response mailbox. Write a numeric value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*val* The numeric value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseNumber.nxc](#).

**6.58.2.69 char SendResponseString (byte *queue*, string *str*) [inline]**

Write a string value to a local response mailbox. Write a string value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*str* The string value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseString.nxc](#).

**6.58.2.70 char SendRS485Bool (bool *bval*) [inline]**

Write RS485 boolean. Write a boolean value to the RS485 hi-speed port.

**Parameters:**

*bval* A boolean value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.58.2.71 char SendRS485Number (long *val*) [inline]**

Write RS485 numeric. Write a numeric value to the RS485 hi-speed port.

**Parameters:**

*val* A numeric value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**6.58.2.72 char SendRS485String (string *str*) [inline]**

Write RS485 string. Write a string value to the RS485 hi-speed port.

**Parameters:**

*str* A string value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**6.58.2.73 void SetBTDataMode (const byte *dataMode*) [inline]**

Set Bluetooth data mode. This method sets the value of the Bluetooth data mode.

**Parameters:**

*dataMode* The Bluetooth data mode. See [Data mode constants](#). Must be a constant.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**6.58.2.74 void SetBTInputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set bluetooth input buffer data. Write cnt bytes of data to the bluetooth input buffer at offset.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetBTInputBuffer.nxc](#).

**6.58.2.75 void SetBTInputBufferIntPtr (byte *n*) [inline]**

Set bluetooth input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetBTInputBufferInPtr.nxc](#).

**6.58.2.76 void SetBTInputBufferOutPtr (byte *n*) [inline]**

Set bluetooth input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetBTInputBufferOutPtr.nxc](#).

**6.58.2.77 void SetBTOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set bluetooth output buffer data. Write *cnt* bytes of data to the bluetooth output buffer at *offset*.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetBTOutputBuffer.nxc](#).

**6.58.2.78 void SetBTOutputBufferInPtr (byte *n*) [inline]**

Set bluetooth output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetBTOutputBufferInPtr.nxc](#).

**6.58.2.79 void SetBTOutputBufferOutPtr (byte *n*) [inline]**

Set bluetooth output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetBTOutputBufferOutPtr.nxc](#).

**6.58.2.80 void SetHSAddress (byte *hsAddress*) [inline]**

Set hi-speed port address. This method sets the value of the hi-speed port address.

**Parameters:**

*hsAddress* The hi-speed port address. See [Hi-speed port address constants](#).

**6.58.2.81 void SetHSDataMode (const byte *dataMode*) [inline]**

Set hi-speed port data mode. This method sets the value of the hi-speed port data mode.

**Parameters:**

*dataMode* The hi-speed port data mode. See [Data mode constants](#). Must be a constant.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**6.58.2.82 void SetHSFlags (byte *hsFlags*) [inline]**

Set hi-speed port flags. This method sets the value of the hi-speed port flags.

**Parameters:**

*hsFlags* The hi-speed port flags. See [Hi-speed port flags constants](#).

**Examples:**

[ex\\_SetHSFlags.nxc](#).

**6.58.2.83 void SetHSInputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]**

Set hi-speed port input buffer data. Write *cnt* bytes of data to the hi-speed port input buffer at *offset*.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetHSInputBuffer.nxc](#).

**6.58.2.84 void SetHSInputBufferIntPtr (byte *n*) [inline]**

Set hi-speed port input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetHSInputBufferIntPtr.nxc](#).

**6.58.2.85 void SetHSInputBufferOutPtr (byte *n*) [inline]**

Set hi-speed port input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetHSInputBufferOutPtr.nxc](#).

**6.58.2.86 void SetHSMode (unsigned int *hsMode*) [inline]**

Set hi-speed port mode. This method sets the value of the hi-speed port mode.

**Parameters:**

*hsMode* The hi-speed port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sethsmode.nxc](#).

**6.58.2.87 void SetHSOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set hi-speed port output buffer data. Write *cnt* bytes of data to the hi-speed port output buffer at *offset*.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetHSOutputBuffer.nxc](#).

**6.58.2.88 void SetHSOutputBufferInPtr (byte *n*) [inline]**

Set hi-speed port output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetHSOutputBufferInPtr.nxc](#).

**6.58.2.89 void SetHSOutputBufferOutPtr (byte *n*) [inline]**

Set hi-speed port output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetHSOutputBufferOutPtr.nxc](#).

**6.58.2.90 void SetHSSpeed (byte *hsSpeed*) [inline]**

Set hi-speed port speed. This method sets the value of the hi-speed port speed (baud rate).

**Parameters:**

*hsSpeed* The hi-speed port speed (baud rate). See [Hi-speed port baud rate constants](#).

**Examples:**

[ex\\_SetHSSpeed.nxc](#).

**6.58.2.91 void SetHSState (byte *hsState*) [inline]**

Set hi-speed port state. This method sets the value of the hi-speed port state.

**Parameters:**

*hsState* The hi-speed port state. See [Hi-speed port state constants](#).

**Examples:**

[ex\\_SetHSState.nxc](#).

**6.58.2.92 void SetUSBInputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set USB input buffer data. Write *cnt* bytes of data to the USB input buffer at *offset*.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBInputBuffer.nxc](#).

**6.58.2.93 void SetUSBInputBufferInPtr (byte *n*) [inline]**

Set USB input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBInputBufferInPtr.nxc](#).

**6.58.2.94 void SetUSBInputBufferOutPtr (byte *n*) [inline]**

Set USB input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBInputBufferOutPtr.nxc](#).

**6.58.2.95 void SetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set USB output buffer data. Write cnt bytes of data to the USB output buffer at offset.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBOutputBuffer.nxc](#).

**6.58.2.96 void SetUSBOutputBufferInPtr (byte *n*) [inline]**

Set USB output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBOutputBufferInPtr.nxc](#).

**6.58.2.97 void SetUSBOutputBufferOutPtr (byte *n*) [inline]**

Set USB output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBOutputBufferOutPtr.nxc](#).

**6.58.2.98 void SetUSBPollBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]**

Set USB poll buffer data. Write *cnt* bytes of data to the USB poll buffer at *offset*.

**Parameters:**

*offset* A constant offset into the poll buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBPollBuffer.nxc](#).

**6.58.2.99 void SetUSBPollBufferInPtr (byte *n*) [inline]**

Set USB poll buffer in-pointer. Set the value of the poll buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBPollBufferInPtr.nxc](#).

**6.58.2.100 void SetUSBPollBufferOutPtr (byte *n*) [inline]**

Set USB poll buffer out-pointer. Set the value of the poll buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBPollBufferOutPtr.nxc](#).

**6.58.2.101 void SetUSBState (byte *usbState*) [inline]**

Set USB state. This method sets the value of the USB state.

**Parameters:**

*usbState* The USB state.

**Examples:**

[ex\\_SetUsbState.nxc](#).

**6.58.2.102 void SysCommBTCheckStatus (CommBTCheckStatusType & *args*)**

Check Bluetooth connection status. This function lets you check the status of a Bluetooth connection using the values specified via the [CommBTCheckStatusType](#) structure.

**Parameters:**

*args* The [CommBTCheckStatusType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommbtcheckstatus.nxc](#).

**6.58.2.103 void SysCommBTConnection (CommBTConnectionType & args) [inline]**

Connect or disconnect a bluetooth device. This function lets you connect or disconnect a bluetooth device using the values specified via the [CommBTConnectionType](#) structure.

**Parameters:**

*args* The [CommBTConnectionType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_syscommbtconnection.nxc](#).

**6.58.2.104 void SysCommBTOnOff (CommBTOnOffType & args) [inline]**

Turn on or off the bluetooth subsystem. This function lets you turn on or off the bluetooth subsystem using the values specified via the [CommBTOnOffType](#) structure.

**Parameters:**

*args* The [CommBTOnOffType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysCommBTOnOff.nxc](#).

**6.58.2.105 void SysCommBTWrite (CommBTWriteType & args)**

Write data to a Bluetooth connection. This function lets you write to a Bluetooth connection using the values specified via the [CommBTWriteType](#) structure.

**Parameters:**

*args* The [CommBTWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommbtwrite.nxc](#).

**6.58.2.106 void SysCommExecuteFunction (CommExecuteFunctionType & args) [inline]**

Execute any Comm module command. This function lets you directly execute the Comm module's primary function using the values specified via the [CommExecuteFunctionType](#) structure.

**Parameters:**

*args* The [CommExecuteFunctionType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_syscommexecutefunction.nxc](#).

**6.58.2.107 void SysCommHSCheckStatus (CommHSCheckStatusType & args) [inline]**

Check the hi-speed port status. This function lets you check the hi-speed port status using the values specified via the [CommHSCheckStatusType](#) structure.

**Parameters:**

*args* The [CommHSCheckStatusType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSCheckStatus.nxc](#).

**6.58.2.108 void SysCommHSControl (CommHSControlType & args)**  
[inline]

Control the hi-speed port. This function lets you control the hi-speed port using the values specified via the [CommHSControlType](#) structure.

**Parameters:**

*args* The [CommHSControlType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSControl.nxc](#).

**6.58.2.109 void SysCommHSRead (CommHSReadWriteType & args)**  
[inline]

Read from the hi-speed port. This function lets you read from the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

**Parameters:**

*args* The [CommHSReadWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSRead.nxc](#).

**6.58.2.110 void SysCommHSWrite (CommHSReadWriteType & args)**  
[inline]

Write to the hi-speed port. This function lets you write to the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

**Parameters:**

*args* The [CommHSReadWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSWrite.nxc](#).

**6.58.2.111 void SysMessageRead (MessageReadType & args)**

Read message. This function lets you read a message from a queue (aka mailbox) using the values specified via the [MessageReadType](#) structure.

**Parameters:**

*args* The [MessageReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysmessageread.nxc](#).

**6.58.2.112 void SysMessageWrite (MessageWriteType & args)**

Write message. This function lets you write a message to a queue (aka mailbox) using the values specified via the [MessageWriteType](#) structure.

**Parameters:**

*args* The [MessageWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysmessagewrite.nxc](#).

**6.58.2.113** byte `USBInputBufferInPtr` (void) [`inline`]

Get usb port input buffer in-pointer. This method returns the value of the input pointer of the usb port input buffer.

**Returns:**

The USB port input buffer's in-pointer value.

**Examples:**

[ex\\_USBInputBufferInPtr.nxc](#).

**6.58.2.114** byte `USBInputBufferOutPtr` (void) [`inline`]

Get usb port input buffer out-pointer. This method returns the value of the output pointer of the usb port input buffer.

**Returns:**

The USB port input buffer's out-pointer value.

**Examples:**

[ex\\_USBInputBufferOutPtr.nxc](#).

**6.58.2.115** byte `USBOutputBufferInPtr` (void) [`inline`]

Get usb port output buffer in-pointer. This method returns the value of the input pointer of the usb port output buffer.

**Returns:**

The USB port output buffer's in-pointer value.

**Examples:**

[ex\\_USBOutputBufferInPtr.nxc](#).

**6.58.2.116** byte `USBOutputBufferOutPtr (void) [inline]`

Get usb port output buffer out-pointer. This method returns the value of the output pointer of the usb port output buffer.

**Returns:**

The USB port output buffer's out-pointer value.

**Examples:**

[ex\\_USBOutputBufferOutPtr.nxc](#).

**6.58.2.117** byte `USBPollBufferInPtr (void) [inline]`

Get usb port poll buffer in-pointer. This method returns the value of the input pointer of the usb port poll buffer.

**Returns:**

The USB port poll buffer's in-pointer value.

**Examples:**

[ex\\_USBPollBufferInPtr.nxc](#).

**6.58.2.118** byte `USBPollBufferOutPtr (void) [inline]`

Get usb port poll buffer out-pointer. This method returns the value of the output pointer of the usb port poll buffer.

**Returns:**

The USB port poll buffer's out-pointer value.

**Examples:**

[ex\\_USBPollBufferOutPtr.nxc](#), and [ex\\_UsbState.nxc](#).

**6.58.2.119 byte USBState (void) [inline]**

Get USB state. This method returns the value of the USB state.

**Returns:**

The USB state.

**6.58.2.120 void UseRS485 (void) [inline]**

Use the RS485 port. Configure port 4 for RS485 usage.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.59 Direct Command functions**

Functions for sending direct commands to another NXT.

**Functions**

- char [RemoteKeepAlive](#) (byte conn)  
*Send a KeepAlive message.*
- char [RemoteMessageRead](#) (byte conn, byte queue)  
*Send a MessageRead message.*
- char [RemoteMessageWrite](#) (byte conn, byte queue, string msg)  
*Send a MessageWrite message.*
- char [RemotePlaySoundFile](#) (byte conn, string filename, bool bloop)  
*Send a PlaySoundFile message.*
- char [RemotePlayTone](#) (byte conn, unsigned int frequency, unsigned int duration)  
*Send a PlayTone message.*
- char [RemoteResetMotorPosition](#) (byte conn, byte port, bool brelative)  
*Send a ResetMotorPosition message.*

- char [RemoteResetScaledValue](#) (byte conn, byte port)  
*Send a ResetScaledValue message.*
- char [RemoteSetInputMode](#) (byte conn, byte port, byte type, byte mode)  
*Send a SetInputMode message.*
- char [RemoteSetOutputState](#) (byte conn, byte port, char speed, byte mode, byte regmode, char turnpct, byte runstate, unsigned long tacholimit)  
*Send a SetOutputMode message.*
- char [RemoteStartProgram](#) (byte conn, string filename)  
*Send a StartProgram message.*
- char [RemoteStopProgram](#) (byte conn)  
*Send a StopProgram message.*
- char [RemoteStopSound](#) (byte conn)  
*Send a StopSound message.*
- char [RemoteGetOutputState](#) (byte conn, [OutputStateType](#) &params)  
*Send a GetOutputState message.*
- char [RemoteGetInputValues](#) (byte conn, [InputValuesType](#) &params)  
*Send a GetInputValues message.*
- char [RemoteGetBatteryLevel](#) (byte conn, int &value)  
*Send a GetBatteryLevel message.*
- char [RemoteLowspeedGetStatus](#) (byte conn, byte &value)  
*Send a LowspeedGetStatus message.*
- char [RemoteLowspeedRead](#) (byte conn, byte port, byte &bread, byte &data[ ])  
*Send a LowspeedRead message.*
- char [RemoteGetCurrentProgramName](#) (byte conn, string &name)  
*Send a GetCurrentProgramName message.*
- char [RemoteDatalogRead](#) (byte conn, bool remove, byte &cnt, byte &log[ ])  
*Send a DatalogRead message.*
- char [RemoteGetContactCount](#) (byte conn, byte &cnt)

*Send a GetContactCount message.*

- char [RemoteGetContactName](#) (byte conn, byte idx, string &name)  
*Send a GetContactName message.*
- char [RemoteGetConnectionCount](#) (byte conn, byte &cnt)  
*Send a GetConnectionCount message.*
- char [RemoteGetConnectionName](#) (byte conn, byte idx, string &name)  
*Send a GetConnectionName message.*
- char [RemoteGetProperty](#) (byte conn, byte property, variant &value)  
*Send a GetProperty message.*
- char [RemoteResetTachoCount](#) (byte conn, byte port)  
*Send a ResetTachoCount message.*
- char [RemoteDatalogSetTimes](#) (byte conn, long synctime)  
*Send a DatalogSetTimes message.*
- char [RemoteSetProperty](#) (byte conn, byte prop, variant value)  
*Send a SetProperty message.*
- char [RemoteLowSpeedWrite](#) (byte conn, byte port, byte txlen, byte rxlen, byte data[ ])  
*Send a LowSpeedWrite message.*

### 6.59.1 Detailed Description

Functions for sending direct commands to another NXT.

### 6.59.2 Function Documentation

#### 6.59.2.1 char RemoteDatalogRead (byte conn, bool remove, byte & cnt, byte & log[ ]) `[inline]`

Send a DatalogRead message. Send the DatalogRead direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*remove* Remove the datalog message from the queue after reading it (true or false).

*cnt* The number of bytes read from the datalog.

*log* A byte array containing the datalog contents.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDatalogRead.nxc](#).

**6.59.2.2 char RemoteDatalogSetTimes (byte *conn*, long *synctime*) [inline]**

Send a DatalogSetTimes message. Send the DatalogSetTimes direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*synctime* The datalog sync time.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDatalogSetTimes.nxc](#).

### 6.59.2.3 char RemoteGetBatteryLevel (byte *conn*, int & *value*) [inline]

Send a GetBatteryLevel message. Send the GetBatteryLevel direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*value* The battery level value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetBatteryLevel.nxc](#).

### 6.59.2.4 char RemoteGetConnectionCount (byte *conn*, byte & *cnt*) [inline]

Send a GetConnectionCount message. This method sends a GetConnectionCount direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*cnt* The number of connections.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetConnectionCount.nxc](#).

**6.59.2.5 char RemoteGetConnectionName (byte *conn*, byte *idx*, string & *name*) [inline]**

Send a GetConnectionName message. Send the GetConnectionName direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*idx* The index of the connection.

*name* The name of the specified connection.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetConnectionName.nxc](#).

**6.59.2.6 char RemoteGetContactCount (byte *conn*, byte & *cnt*) [inline]**

Send a GetContactCount message. This method sends a GetContactCount direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*cnt* The number of contacts.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetContactCount.nxc](#).

**6.59.2.7 char RemoteGetContactName (byte *conn*, byte *idx*, string & *name*)  
[inline]**

Send a GetContactName message. Send the GetContactName direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*idx* The index of the contact.

*name* The name of the specified contact.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetContactName.nxc](#).

**6.59.2.8 char RemoteGetCurrentProgramName (byte *conn*, string & *name*)  
[inline]**

Send a GetCurrentProgramName message. This method sends a GetCurrentProgramName direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The current program name.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetCurrentProgramName.nxc](#).

**6.59.2.9 char RemoteGetInputValues (byte *conn*, InputValueType & *params*)  
[inline]**

Send a GetInputValues message. Send the GetInputValues direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*params* The input and output parameters for the function call. See [InputValueType](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetInputValues.nxc](#).

### 6.59.2.10 char RemoteGetOutputState (byte *conn*, OutputStateType & *params*) [inline]

Send a GetOutputState message. Send the GetOutputState direct command on the specified connection slot.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*params* The input and output parameters for the function call. See [OutputState-Type](#).

#### Returns:

A char value indicating whether the function call succeeded or not.

#### Examples:

[ex\\_RemoteGetOutputState.nxc](#).

### 6.59.2.11 char RemoteGetProperty (byte *conn*, byte *property*, variant & *value*) [inline]

Send a GetProperty message. Send the GetProperty direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*property* The property to read. See [Property constants](#).

*value* The property value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetProperty.nxc](#).

**6.59.2.12 char RemoteKeepAlive (byte *conn*) [inline]**

Send a KeepAlive message. This method sends a KeepAlive direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteKeepAlive.nxc](#).

**6.59.2.13 char RemoteLowspeedGetStatus (byte *conn*, byte & *value*) [inline]**

Send a LowspeedGetStatus message. This method sends a LowspeedGetStatus direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*value* The count of available bytes to read.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowspeedGetStatus.nxc](#).

**6.59.2.14 char RemoteLowspeedRead (byte *conn*, byte *port*, byte & *bread*, byte & *data*[]) [inline]**

Send a LowspeedRead message. Send the LowspeedRead direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port from which to read I2C data. See [Input port constants](#).

*bread* The number of bytes read.

*data* A byte array containing the data read from the I2C device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowspeedRead.nxc](#).

**6.59.2.15 char RemoteLowspeedWrite (byte *conn*, byte *port*, byte *txlen*, byte *rxlen*, byte *data*[]) [inline]**

Send a LowspeedWrite message. Send the LowspeedWrite direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The I2C port. See [Input port constants](#).

*txlen* The number of bytes you are writing to the I2C device.

*rxlen* The number of bytes want to read from the I2C device.

*data* A byte array containing the data you are writing to the device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowspeedWrite.nxc](#).

**6.59.2.16 char RemoteMessageRead (byte conn, byte queue) [inline]**

Send a MessageRead message. This method sends a MessageRead direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox to read. See [Mailbox constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteMessageRead.nxc](#).

**6.59.2.17 char RemoteMessageWrite (byte conn, byte queue, string msg) [inline]**

Send a MessageWrite message. This method sends a MessageWrite direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox to write. See [Mailbox constants](#).

*msg* The message to write to the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteMessageWrite.nxc](#).

**6.59.2.18 char RemotePlaySoundFile (byte conn, string filename, bool bloop) [inline]**

Send a PlaySoundFile message. Send the PlaySoundFile direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the sound file to play.

*bloop* A boolean value indicating whether to loop the sound file or not.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePlaySoundFile.nxc](#).

**6.59.2.19 char RemotePlayTone (byte *conn*, unsigned int *frequency*, unsigned int *duration*) [inline]**

Send a PlayTone message. Send the PlayTone direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*frequency* The frequency of the tone.

*duration* The duration of the tone.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePlayTone.nxc](#).

**6.59.2.20 char RemoteResetMotorPosition (byte *conn*, byte *port*, bool *brelative*) [inline]**

Send a ResetMotorPosition message. Send the ResetMotorPosition direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to reset.

*brelative* A flag indicating whether the counter to reset is relative.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetMotorPosition.nxc](#).

**6.59.2.21 char RemoteResetScaledValue (byte *conn*, byte *port*) [inline]**

Send a ResetScaledValue message. Send the ResetScaledValue direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port to reset.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetScaledValue.nxc](#).

**6.59.2.22 char RemoteResetTachoCount (byte *conn*, byte *port*) [inline]**

Send a ResetTachoCount message. Send the ResetTachoCount direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to reset the tachometer count on. See [Output port constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetTachoCount.nxc](#).

### 6.59.2.23 `char RemoteSetInputMode (byte conn, byte port, byte type, byte mode) [inline]`

Send a SetInputMode message. Send the SetInputMode direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port to configure. See [Input port constants](#).

*type* The sensor type. See [Sensor type constants](#).

*mode* The sensor mode. See [Sensor mode constants](#).

#### Returns:

A char value indicating whether the function call succeeded or not.

#### Examples:

[ex\\_RemoteSetInputMode.nxc](#).

### 6.59.2.24 `char RemoteSetOutputState (byte conn, byte port, char speed, byte mode, byte regmode, char turnpct, byte runstate, unsigned long tacholimit) [inline]`

Send a SetOutputMode message. Send the SetOutputMode direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to configure. See [Output port constants](#).

*speed* The motor speed. (-100..100)

*mode* The motor mode. See [Output port mode constants](#).

*regmode* The motor regulation mode. See [Output port regulation mode constants](#).

*turnpct* The motor synchronized turn percentage. (-100..100)

*runstate* The motor run state. See [Output port run state constants](#).

*tacholimit* The motor tachometer limit.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetOutputState.nxc](#).

**6.59.2.25 char RemoteSetProperty (byte *conn*, byte *prop*, variant *value*) [inline]**

Send a SetProperty message. Send the SetProperty direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*prop* The property to set. See [Property constants](#).

*value* The new property value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetProperty.nxc](#).

**6.59.2.26 char RemoteStartProgram (byte *conn*, string *filename*) [inline]**

Send a StartProgram message. Send the StartProgram direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the program to start running.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStartProgram.nxc](#).

**6.59.2.27 char RemoteStopProgram (byte conn) [inline]**

Send a StopProgram message. Send the StopProgram direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStopProgram.nxc](#).

**6.59.2.28 char RemoteStopSound (byte conn) [inline]**

Send a StopSound message. Send the StopSound direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

**conn** The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStopSound.nxc](#).

## 6.60 System Command functions

Functions for sending system commands to another NXT.

**Functions**

- char [RemoteOpenRead](#) (byte conn, string filename, byte &handle, long &size)  
*Send an OpenRead message.*
- char [RemoteOpenAppendData](#) (byte conn, string filename, byte &handle, long &size)  
*Send an OpenAppendData message.*
- char [RemoteDeleteFile](#) (byte conn, string filename)  
*Send a DeleteFile message.*
- char [RemoteFindFirstFile](#) (byte conn, string mask, byte &handle, string &name, long &size)  
*Send a FindFirstFile message.*
- char [RemoteGetFirmwareVersion](#) (byte conn, byte &pmin, byte &pmaj, byte &fmin, byte &fmaj)  
*Send a GetFirmwareVersion message.*
- char [RemoteGetBluetoothAddress](#) (byte conn, byte &btaddr[ ])  
*Send a GetBluetoothAddress message.*
- char [RemoteGetDeviceInfo](#) (byte conn, string &name, byte &btaddr[ ], byte &btsignal[ ], long &freemem)  
*Send a GetDeviceInfo message.*

- char [RemoteDeleteUserFlash](#) (byte conn)  
*Send a DeleteUserFlash message.*
- char [RemoteOpenWrite](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWrite message.*
- char [RemoteOpenWriteLinear](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWriteLinear message.*
- char [RemoteOpenWriteData](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWriteData message.*
- char [RemoteCloseFile](#) (byte conn, byte handle)  
*Send a CloseFile message.*
- char [RemoteFindNextFile](#) (byte conn, byte &handle, string &name, long &size)  
*Send a FindNextFile message.*
- char [RemotePollCommandLength](#) (byte conn, byte bufnum, byte &length)  
*Send a PollCommandLength message.*
- char [RemoteWrite](#) (byte conn, byte &handle, int &numbytes, byte data[ ])  
*Send a Write message.*
- char [RemoteRead](#) (byte conn, byte &handle, int &numbytes, byte &data[ ])  
*Send a Read message.*
- char [RemoteIOMapRead](#) (byte conn, long id, int offset, int &numbytes, byte &data[ ])  
*Send an IOMapRead message.*
- char [RemotePollCommand](#) (byte conn, byte bufnum, byte &len, byte &data[ ])  
*Send a PollCommand message.*
- char [RemoteRenameFile](#) (byte conn, string oldname, string newname)  
*Send a RenameFile message.*
- char [RemoteBluetoothFactoryReset](#) (byte conn)  
*Send a BluetoothFactoryReset message.*

- char [RemoteIOMapWriteValue](#) (byte conn, long id, int offset, variant value)  
*Send an IOMapWrite value message.*
- char [RemoteIOMapWriteBytes](#) (byte conn, long id, int offset, byte data[ ])  
*Send an IOMapWrite bytes message.*
- char [RemoteSetBrickName](#) (byte conn, string name)  
*Send a SetBrickName message.*

### 6.60.1 Detailed Description

Functions for sending system commands to another NXT.

### 6.60.2 Function Documentation

#### 6.60.2.1 char RemoteBluetoothFactoryReset (byte conn) [inline]

Send a BluetoothFactoryReset message. This method sends a BluetoothFactoryReset system command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed. This command cannot be sent over a bluetooth connection.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

#### Returns:

A char value indicating whether the function call succeeded or not.

#### Examples:

[ex\\_RemoteBluetoothFactoryReset.nxc](#).

#### 6.60.2.2 char RemoteCloseFile (byte conn, byte handle) [inline]

Send a CloseFile message. Send the CloseFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file to close.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteCloseFile.nxc](#).

**6.60.2.3 char RemoteDeleteFile (byte *conn*, string *filename*) [inline]**

Send a DeleteFile message. Send the DeleteFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to delete.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDeleteFile.nxc](#).

#### 6.60.2.4 char RemoteDeleteUserFlash (byte *conn*) [inline]

Send a DeleteUserFlash message. This method sends a DeleteUserFlash system command to the device on the specified connection.

##### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

##### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

##### Returns:

A char value indicating whether the function call succeeded or not.

##### Examples:

[ex\\_RemoteDeleteUserFlash.nxc](#).

#### 6.60.2.5 char RemoteFindFirstFile (byte *conn*, string *mask*, byte & *handle*, string & *name*, long & *size*) [inline]

Send a FindFirstFile message. Send the FindFirstFile system command on the specified connection slot.

##### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

##### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*mask* The filename mask for the files you want to find.

*handle* The handle of the found file.

*name* The name of the found file.

*size* The size of the found file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteFindFirstFile.nxc](#).

**6.60.2.6 char RemoteFindNextFile (byte *conn*, byte & *handle*, string & *name*, long & *size*) [inline]**

Send a FindNextFile message. Send the FindNextFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle returned by the last [FindFirstFile](#) or FindNextFile call.

*name* The name of the next found file.

*size* The size of the next found file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteFindNextFile.nxc](#).

**6.60.2.7 char RemoteGetBluetoothAddress (byte *conn*, byte & *btaddr*[]) [inline]**

Send a GetBluetoothAddress message. This method sends a GetBluetoothAddress system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*btaddr* The bluetooth address of the remote device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetBluetoothAddress.nxc](#).

**6.60.2.8 char RemoteGetDeviceInfo (byte conn, string & name, byte & btaddr[], byte & btsignal[], long & freemem) [inline]**

Send a GetDeviceInfo message. This method sends a GetDeviceInfo system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The name of the remote device.

*btaddr* The bluetooth address of the remote device.

*btsignal* The signal strength of each connection on the remote device.

*freemem* The number of bytes of free flash memory on the remote device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetDeviceInfo.nxc](#).

### 6.60.2.9 char RemoteGetFirmwareVersion (byte *conn*, byte & *pmin*, byte & *pmaj*, byte & *fmin*, byte & *fmaj*) [inline]

Send a GetFirmwareVersion message. This method sends a GetFirmwareVersion system command to the device on the specified connection.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*pmin* The protocol minor version byte.

*pmaj* The protocol major version byte.

*fmin* The firmware minor version byte.

*fmaj* The firmware major version byte.

#### Returns:

A char value indicating whether the function call succeeded or not.

#### Examples:

[ex\\_RemoteGetFirmwareVersion.nxc](#).

### 6.60.2.10 char RemoteIOMapRead (byte *conn*, long *id*, int *offset*, int & *numbytes*, byte & *data*[]) [inline]

Send an IOMapRead message. Send the IOMapRead system command on the specified connection slot.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module from which to read data.

*offset* The offset into the IOMap structure from which to read.

*numbytes* The number of bytes of data to read. Returns the number of bytes actually read.

*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteIOMapRead.nxc](#).

**6.60.2.11 char RemoteIOMapWriteBytes (byte conn, long id, int offset, byte data[]) [inline]**

Send an IOMapWrite bytes message. Send the IOMapWrite system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module to which to write data.

*offset* The offset into the IOMap structure to which to write.

*data* A byte array containing the data you are writing to the IOMap structure.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteIOMapWriteBytes.nxc](#).

### 6.60.2.12 char RemoteIOMapWriteValue (byte *conn*, long *id*, int *offset*, variant *value*) [inline]

Send an IOMapWrite value message. Send the IOMapWrite system command on the specified connection slot to write the value provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module to which to write data.

*offset* The offset into the IOMap structure to which to write.

*value* A scalar variable containing the value you are writing to the IOMap structure.

#### Returns:

A char value indicating whether the function call succeeded or not.

#### Examples:

[ex\\_RemoteIOMapWriteValue.nxc](#).

### 6.60.2.13 char RemoteOpenAppendData (byte *conn*, string *filename*, byte & *handle*, long & *size*) [inline]

Send an OpenAppendData message. Send the OpenAppendData system command on the specified connection slot.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for appending.

*handle* The handle of the file.

*size* The size of the file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenAppendData.nxc](#).

**6.60.2.14 char RemoteOpenRead (byte *conn*, string *filename*, byte & *handle*, long & *size*) [inline]**

Send an OpenRead message. Send the OpenRead system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for reading.

*handle* The handle of the file.

*size* The size of the file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenRead.nxc](#).

**6.60.2.15 char RemoteOpenWrite (byte *conn*, string *filename*, long *size*, byte & *handle*) [inline]**

Send an OpenWrite message. Send the OpenWrite system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWrite.nxc](#).

**6.60.2.16 char RemoteOpenWriteData (byte conn, string filename, long size, byte & handle) [inline]**

Send an OpenWriteData message. Send the OpenWriteData system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWriteData.nxc](#).

**6.60.2.17 char RemoteOpenWriteLinear (byte *conn*, string *filename*, long *size*, byte & *handle*) [inline]**

Send an OpenWriteLinear message. Send the OpenWriteLinear system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWriteLinear.nxc](#).

**6.60.2.18 char RemotePollCommand (byte *conn*, byte *bufnum*, byte & *len*, byte & *data*[]) [inline]**

Send a PollCommand message. Send the PollCommand system command on the specified connection slot to write the data provided.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*bufnum* The buffer from which to read data (0=USBPoll, 1=HiSpeed).  
*len* The number of bytes to read. Returns the number of bytes actually read.  
*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePollCommand.nxc](#).

**6.60.2.19 char RemotePollCommandLength (byte *conn*, byte *bufnum*, byte & *length*) [inline]**

Send a PollCommandLength message. Send the PollCommandLength system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*bufnum* The poll buffer you want to query (0=USBPoll, 1=HiSpeed).

*length* The number of bytes available for polling.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePollCommandLength.nxc](#).

**6.60.2.20 char RemoteRead (byte *conn*, byte & *handle*, int & *numbytes*, byte & *data*[ ]) [inline]**

Send a Read message. Send the Read system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file you are reading from.

*numbytes* The number of bytes you want to read. Returns the number of bytes actually read.

*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteRead.nxc](#).

**6.60.2.21 char RemoteRenameFile (byte *conn*, string *oldname*, string *newname*) [*inline*]**

Send a RenameFile message. Send the RenameFile system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*oldname* The old filename.

*newname* The new filename.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteRenameFile.nxc](#).

**6.60.2.22 char RemoteSetBrickName (byte *conn*, string *name*) [inline]**

Send a SetBrickName message. Send the SetBrickName system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The new brick name.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetBrickName.nxc](#).

**6.60.2.23 char RemoteWrite (byte *conn*, byte & *handle*, int & *numbytes*, byte *data*[]) [inline]**

Send a Write message. Send the Write system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file you are writing to.

*numbytes* The number of bytes actually written.

*data* A byte array containing the data you are writing.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteWrite.nxc](#).

## 6.61 Button module types

Types used by various Button module functions.

**Data Structures**

- struct [ReadButtonType](#)  
*Parameters for the ReadButton system call.*

### 6.61.1 Detailed Description

Types used by various Button module functions.

## 6.62 Button module functions

Functions for accessing and modifying Button module features.

**Functions**

- bool [ButtonPressed](#) (const byte btn, bool resetCount)  
*Check for button press.*
- byte [ButtonCount](#) (const byte btn, bool resetCount)  
*Get button press count.*
- char [ReadButtonEx](#) (const byte btn, bool reset, bool &pressed, unsigned int &count)  
*Read button information.*
- byte [ButtonPressCount](#) (const byte btn)  
*Get button press count.*

- byte [ButtonLongPressCount](#) (const byte btn)  
*Get button long press count.*
- byte [ButtonShortReleaseCount](#) (const byte btn)  
*Get button short release count.*
- byte [ButtonLongReleaseCount](#) (const byte btn)  
*Get button long release count.*
- byte [ButtonReleaseCount](#) (const byte btn)  
*Get button release count.*
- byte [ButtonState](#) (const byte btn)  
*Get button state.*
- void [SetButtonLongPressCount](#) (const byte btn, const byte n)  
*Set button long press count.*
- void [SetButtonLongReleaseCount](#) (const byte btn, const byte n)  
*Set button long release count.*
- void [SetButtonPressCount](#) (const byte btn, const byte n)  
*Set button press count.*
- void [SetButtonReleaseCount](#) (const byte btn, const byte n)  
*Set button release count.*
- void [SetButtonShortReleaseCount](#) (const byte btn, const byte n)  
*Set button short release count.*
- void [SetButtonState](#) (const byte btn, const byte state)  
*Set button state.*
- void [SysReadButton](#) ([ReadButtonType](#) &args)  
*Read button.*

### 6.62.1 Detailed Description

Functions for accessing and modifying Button module features.

### 6.62.2 Function Documentation

#### 6.62.2.1 byte ButtonCount (const byte *btn*, bool *resetCount*) [inline]

Get button press count. Return the number of times the specified button has been pressed since the last time the button press count was reset. Optionally clear the count after reading it.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*resetCount* Whether or not to reset the press counter.

**Returns:**

The button press count.

**Examples:**

[ex\\_ButtonCount.nxc](#).

#### 6.62.2.2 byte ButtonLongPressCount (const byte *btn*) [inline]

Get button long press count. Return the long press count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button long press count.

**Examples:**

[ex\\_ButtonLongPressCount.nxc](#).

#### 6.62.2.3 byte ButtonLongReleaseCount (const byte *btn*) [inline]

Get button long release count. Return the long release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button long release count.

**Examples:**

[ex\\_ButtonLongReleaseCount.nxc](#).

**6.62.2.4 byte ButtonPressCount (const byte *btn*) [inline]**

Get button press count. Return the press count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button press count.

**Examples:**

[ex\\_ButtonPressCount.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**6.62.2.5 bool ButtonPressed (const byte *btn*, bool *resetCount*) [inline]**

Check for button press. This function checks whether the specified button is pressed or not. You may optionally reset the press count.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*resetCount* Whether or not to reset the press counter.

**Returns:**

A boolean value indicating whether the button is pressed or not.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**6.62.2.6** byte ButtonReleaseCount (const byte *btn*) [**inline**]

Get button release count. Return the release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button release count.

**Examples:**

[ex\\_ButtonReleaseCount.nxc](#).

**6.62.2.7** byte ButtonShortReleaseCount (const byte *btn*) [**inline**]

Get button short release count. Return the short release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button short release count.

**Examples:**

[ex\\_ButtonShortReleaseCount.nxc](#).

**6.62.2.8** byte ButtonState (const byte *btn*) [**inline**]

Get button state. Return the state of the specified button. See [ButtonState constants](#).

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button state.

**Examples:**

[ex\\_ButtonState.nxc](#).

**6.62.2.9 char ReadButtonEx (const byte *btn*, bool *reset*, bool & *pressed*, unsigned int & *count*) [inline]**

Read button information. Read the specified button. Set the pressed and count parameters with the current state of the button. Optionally reset the press count after reading it.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*reset* Whether or not to reset the press counter.

*pressed* The button pressed state.

*count* The button press count.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadButtonEx.nxc](#).

**6.62.2.10 void SetButtonLongPressCount (const byte *btn*, const byte *n*) [inline]**

Set button long press count. Set the long press count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new long press count value.

**Examples:**

[ex\\_SetButtonLongPressCount.nxc](#).

**6.62.2.11 void SetButtonLongReleaseCount (const byte *btn*, const byte *n*)  
[inline]**

Set button long release count. Set the long release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new long release count value.

**Examples:**

[ex\\_SetButtonLongReleaseCount.nxc](#).

**6.62.2.12 void SetButtonPressCount (const byte *btn*, const byte *n*) [inline]**

Set button press count. Set the press count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new press count value.

**Examples:**

[ex\\_SetButtonPressCount.nxc](#).

**6.62.2.13 void SetButtonReleaseCount (const byte *btn*, const byte *n*)  
[inline]**

Set button release count. Set the release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new release count value.

**Examples:**

[ex\\_SetButtonReleaseCount.nxc](#).

**6.62.2.14 void SetButtonShortReleaseCount (const byte *btn*, const byte *n*) [inline]**

Set button short release count. Set the short release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new short release count value.

**Examples:**

[ex\\_SetButtonShortReleaseCount.nxc](#).

**6.62.2.15 void SetButtonState (const byte *btn*, const byte *state*) [inline]**

Set button state. Set the state of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*state* The new button state. See [ButtonState constants](#).

**Examples:**

[ex\\_SetButtonState.nxc](#).

**6.62.2.16 void SysReadButton (ReadButtonType & *args*) [inline]**

Read button. This function lets you read button state information via the [ReadButtonType](#) structure.

**Parameters:**

*args* The [ReadButtonType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

## 6.63 Ui module types

Types used by various Ui module functions.

### Data Structures

- struct [SetSleepTimeoutType](#)  
*Parameters for the SetSleepTimeout system call.*

### 6.63.1 Detailed Description

Types used by various Ui module functions.

## 6.64 Ui module functions

Functions for accessing and modifying Ui module features.

### Functions

- byte [CommandFlags](#) (void)  
*Get command flags.*
- byte [UIState](#) (void)  
*Get UI module state.*
- byte [UIButton](#) (void)  
*Read UI button.*
- byte [VMRunState](#) (void)  
*Read VM run state.*
- byte [BatteryState](#) (void)  
*Get battery state.*
- byte [BluetoothState](#) (void)  
*Get bluetooth state.*
- byte [UsbState](#) (void)  
*Get UI module USB state.*

- byte [SleepTimeout](#) (void)  
*Read sleep timeout.*
- byte [SleepTime](#) (void)  
*Read sleep time.*
- byte [SleepTimer](#) (void)  
*Read sleep timer.*
- bool [RechargeableBattery](#) (void)  
*Read battery type.*
- byte [Volume](#) (void)  
*Read volume.*
- byte [OnBrickProgramPointer](#) (void)  
*Read the on brick program pointer value.*
- byte [AbortFlag](#) (void)  
*Read abort flag.*
- byte [LongAbort](#) (void)  
*Read long abort setting.*
- unsigned int [BatteryLevel](#) (void)  
*Get battery Level.*
- void [SetCommandFlags](#) (const byte cmdFlags)  
*Set command flags.*
- void [SetUIButton](#) (byte btn)  
*Set UI button.*
- void [SetUIState](#) (byte state)  
*Set UI state.*
- void [SetVMRunState](#) (const byte vmRunState)  
*Set VM run state.*
- void [SetBatteryState](#) (byte state)  
*Set battery state.*

- void [SetBluetoothState](#) (byte state)  
*Set bluetooth state.*
- void [SetSleepTimeout](#) (const byte n)  
*Set sleep timeout.*
- void [SetSleepTime](#) (const byte n)  
*Set sleep time.*
- void [SetSleepTimer](#) (const byte n)  
*Set the sleep timer.*
- void [SetVolume](#) (byte volume)  
*Set volume.*
- void [SetOnBrickProgramPointer](#) (byte obpStep)  
*Set on-brick program pointer.*
- void [ForceOff](#) (byte num)  
*Turn off NXT.*
- void [SetAbortFlag](#) (byte abortFlag)  
*Set abort flag.*
- void [SetLongAbort](#) (bool longAbort)  
*Set long abort.*
- void [SysSetSleepTimeout](#) ([SetSleepTimeoutType](#) &args)  
*Set system sleep timeout.*

### 6.64.1 Detailed Description

Functions for accessing and modifying Ui module features.

### 6.64.2 Function Documentation

#### 6.64.2.1 byte AbortFlag (void) [`inline`]

Read abort flag. Return the enhanced NBC/NXC firmware's abort flag.

**Returns:**

The current abort flag value. See [ButtonState constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_AbortFlag.nxc](#).

**6.64.2.2 unsigned int BatteryLevel (void) [inline]**

Get battery Level. Return the battery level in millivolts.

**Returns:**

The battery level

**Examples:**

[util\\_battery\\_1.nxc](#), and [util\\_battery\\_2.nxc](#).

**6.64.2.3 byte BatteryState (void) [inline]**

Get battery state. Return battery state information (0..4).

**Returns:**

The battery state (0..4)

**Examples:**

[ex\\_BatteryState.nxc](#).

**6.64.2.4 byte BluetoothState (void) [inline]**

Get bluetooth state. Return the bluetooth state.

**Returns:**

The bluetooth state. See [BluetoothState constants](#).

**Examples:**

[ex\\_BluetoothState.nxc](#).

**6.64.2.5 byte CommandFlags (void) [inline]**

Get command flags. Return the command flags.

**Returns:**

Command flags. See [CommandFlags constants](#)

**Examples:**

[ex\\_CommandFlags.nxc](#).

**6.64.2.6 void ForceOff (byte num) [inline]**

Turn off NXT. Force the NXT to turn off if the specified value is greater than zero.

**Parameters:**

*num* If greater than zero the NXT will turn off.

**Examples:**

[ex\\_ForceOff.nxc](#).

**6.64.2.7 byte LongAbort (void) [inline]**

Read long abort setting. Return the enhanced NBC/NXC firmware's long abort setting.

**See also:**

[AbortFlag](#)

**Returns:**

The current abort flag value. See [ButtonState constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_LongAbort.nxc](#).

**6.64.2.8 byte OnBrickProgramPointer (void) [inline]**

Read the on brick program pointer value. Return the current OBP (on-brick program) step

**Returns:**

On brick program pointer (step).

**Examples:**

[ex\\_OnBrickProgramPointer.nxc](#).

**6.64.2.9 bool RechargeableBattery (void) [inline]**

Read battery type. Return whether the NXT has a rechargeable battery installed or not.

**Returns:**

Whether the battery is rechargeable or not. (false = no, true = yes)

**Examples:**

[ex\\_RechargeableBattery.nxc](#).

**6.64.2.10 void SetAbortFlag (byte abortFlag) [inline]**

Set abort flag. Set the enhanced NBC/NXC firmware's program abort flag. By default the running program can be interrupted by a short press of the escape button. You can change this to any other button state flag.

**Parameters:**

*abortFlag* The new abort flag value. See [ButtonState constants](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**6.64.2.11 void SetBatteryState (byte *state*) [inline]**

Set battery state. Set battery state information.

**Parameters:**

*state* The desired battery state (0..4).

**Examples:**

[ex\\_SetBatteryState.nxc](#).

**6.64.2.12 void SetBluetoothState (byte *state*) [inline]**

Set bluetooth state. Set the Bluetooth state.

**Parameters:**

*state* The desired bluetooth state. See [BluetoothState constants](#).

**Examples:**

[ex\\_SetBluetoothState.nxc](#).

**6.64.2.13 void SetCommandFlags (const byte *cmdFlags*) [inline]**

Set command flags. Set the command flags.

**Parameters:**

*cmdFlags* The new command flags. See [CommandFlags constants](#).

**Examples:**

[ex\\_SetCommandFlags.nxc](#).

**6.64.2.14 void SetLongAbort (bool *longAbort*) [inline]**

Set long abort. Set the enhanced NBC/NXC firmware's long abort setting (true or false). If set to true then a program has access the escape button. Aborting a program requires a long press of the escape button.

**Parameters:**

*longAbort* If true then require a long press of the escape button to abort a program, otherwise a short press will abort it.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_getchar.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**6.64.2.15 void SetOnBrickProgramPointer (byte *obpStep*) [inline]**

Set on-brick program pointer. Set the current OBP (on-brick program) step.

**Parameters:**

*obpStep* The new on-brick program step.

**Examples:**

[ex\\_SetOnBrickProgramPointer.nxc](#).

**6.64.2.16 void SetSleepTime (const byte *n*) [inline]**

Set sleep time. Set the NXT sleep timeout value to the specified number of minutes.

**Parameters:**

*n* The minutes to wait before sleeping.

**See also:**

[SetSleepTimeout](#), [SleepTimeout](#)

**Examples:**

[ex\\_setsleeptime.nxc](#).

**6.64.2.17 void SetSleepTimeout (const byte *n*) [inline]**

Set sleep timeout. Set the NXT sleep timeout value to the specified number of minutes.

**Parameters:**

*n* The minutes to wait before sleeping.

**Examples:**

[ex\\_SetSleepTimeout.nxc](#).

**6.64.2.18 void SetSleepTimer (const byte *n*) [inline]**

Set the sleep timer. Set the system sleep timer to the specified number of minutes.

**Parameters:**

*n* The minutes left on the timer.

**Examples:**

[ex\\_SetSleepTimer.nxc](#).

**6.64.2.19 void SetUIButton (byte *btn*) [inline]**

Set UI button. Set user interface button information.

**Parameters:**

*btn* A user interface button value. See [UIButton constants](#).

**Examples:**

[ex\\_SetUIButton.nxc](#).

**6.64.2.20 void SetUIState (byte *state*) [inline]**

Set UI state. Set the user interface state.

**Parameters:**

*state* A user interface state value. See [UIState constants](#).

**Examples:**

[ex\\_SetUIState.nxc](#).

**6.64.2.21 void SetVMRunState (const byte *vmRunState*) [inline]**

Set VM run state. Set VM run state information.

**Parameters:**

*vmRunState* The desired VM run state. See [VM run state constants](#).

**Warning:**

It is not a good idea to change the VM run state from within a running program unless you know what you are doing.

**Examples:**

[ex\\_SetVMRunState.nxc](#).

**6.64.2.22 void SetVolume (byte *volume*) [inline]**

Set volume. Set the user interface volume level. Valid values are from 0 to 4.

**Parameters:**

*volume* The new volume level.

**Examples:**

[ex\\_SetVolume.nxc](#).

**6.64.2.23 byte SleepTime (void) [inline]**

Read sleep time. Return the number of minutes that the NXT will remain on before it automatically shuts down.

**Returns:**

The sleep time value

**See also:**

[SleepTimeout](#)

**Examples:**

[ex\\_sleeptime.nxc](#).

**6.64.2.24 byte SleepTimeout (void) [inline]**

Read sleep timeout. Return the number of minutes that the NXT will remain on before it automatically shuts down.

**Returns:**

The sleep timeout value

**Examples:**

[ex\\_SleepTimeout.nxc](#).

**6.64.2.25 byte SleepTimer (void) [inline]**

Read sleep timer. Return the number of minutes left in the countdown to zero from the original SleepTimeout value. When the SleepTimer value reaches zero the NXT will shutdown.

**Returns:**

The sleep timer value

**Examples:**

[ex\\_SleepTimer.nxc](#).

**6.64.2.26 void SysSetSleepTimeout (SetSleepTimeoutType & args) [inline]**

Set system sleep timeout. This function lets you set the system sleep timeout value given the parameters you pass in via the [SetSleepTimeoutType](#) structure.

**Parameters:**

*args* The [SetSleepTimeoutType](#) structure containing the required parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysSetSleepTimeout.nxc](#).

**6.64.2.27 byte UIButton (void) [inline]**

Read UI button. Return user interface button information.

**Returns:**

A UI button value. See [UIButton constants](#).

**Examples:**

[ex\\_UIButton.nxc](#).

**6.64.2.28 byte UIState (void) [inline]**

Get UI module state. Return the user interface state.

**Returns:**

The UI module state. See [UIState constants](#).

**Examples:**

[ex\\_UIState.nxc](#).

**6.64.2.29 byte UsbState (void) [inline]**

Get UI module USB state. This method returns the UI module USB state.

**Returns:**

The UI module USB state. (0=disconnected, 1=connected, 2=working)

**Examples:**

[ex\\_UiUsbState.nxc](#).

**6.64.2.30 byte VMRunState (void) [inline]**

Read VM run state. Return VM run state information.

**Returns:**

VM run state. See [VM run state constants](#).

**Examples:**

[ex\\_VMRunState.nxc](#).

**6.64.2.31 byte Volume (void) [inline]**

Read volume. Return the user interface volume level. Valid values are from 0 to 4.

**Returns:**

The UI module volume. (0..4)

**Examples:**

[ex\\_Volume.nxc](#).

## 6.65 Loader module types

Types used by various Loader module functions.

**Data Structures**

- struct [FileOpenType](#)  
*Parameters for the FileOpen system call.*
- struct [FileReadWriteType](#)  
*Parameters for the FileReadWrite system call.*
- struct [FileCloseType](#)  
*Parameters for the FileClose system call.*
- struct [FileResolveHandleType](#)  
*Parameters for the FileResolveHandle system call.*
- struct [FileRenameType](#)  
*Parameters for the FileRename system call.*
- struct [FileDeleteType](#)  
*Parameters for the FileDelete system call.*
- struct [LoaderExecuteFunctionType](#)  
*Parameters for the LoaderExecuteFunction system call.*
- struct [FileFindType](#)  
*Parameters for the FileFind system call.*
- struct [FileSeekType](#)  
*Parameters for the FileSeek system call.*
- struct [FileResizeType](#)  
*Parameters for the FileResize system call.*

- struct [FileTellType](#)  
*Parameters for the FileTell system call.*
- struct [ListFilesType](#)  
*Parameters for the ListFiles system call.*

### 6.65.1 Detailed Description

Types used by various Loader module functions.

## 6.66 Loader module functions

Functions for accessing and modifying Loader module features.

### Functions

- unsigned int [FreeMemory](#) (void)  
*Get free flash memory.*
- unsigned int [CreateFile](#) (string fname, unsigned int fsize, byte &handle)  
*Create a file.*
- unsigned int [OpenFileAppend](#) (string fname, unsigned int &fsize, byte &handle)  
*Open a file for appending.*
- unsigned int [OpenFileRead](#) (string fname, unsigned int &fsize, byte &handle)  
*Open a file for reading.*
- unsigned int [CloseFile](#) (byte handle)  
*Close a file.*
- unsigned int [ResolveHandle](#) (string filename, byte &handle, bool &writeable)  
*Resolve a handle.*
- unsigned int [RenameFile](#) (string oldname, string newname)  
*Rename a file.*
- unsigned int [DeleteFile](#) (string fname)

*Delete a file.*

- unsigned int [ResizeFile](#) (string fname, const unsigned int newsize)  
*Resize a file.*
- unsigned int [CreateFileLinear](#) (string fname, unsigned int fsize, byte &handle)  
*Create a linear file.*
- unsigned int [CreateFileNonLinear](#) (string fname, unsigned int fsize, byte &handle)  
*Create a non-linear file.*
- unsigned int [OpenFileReadLinear](#) (string fname, unsigned int &fsize, byte &handle)  
*Open a linear file for reading.*
- unsigned int [FindFirstFile](#) (string &fname, byte &handle)  
*Start searching for files.*
- unsigned int [FindNextFile](#) (string &fname, byte &handle)  
*Continue searching for files.*
- unsigned int [SizeOf](#) (variant &value)  
*Calculate the size of a variable.*
- unsigned int [Read](#) (byte handle, variant &value)  
*Read a value from a file.*
- unsigned int [ReadLn](#) (byte handle, variant &value)  
*Read a value from a file plus line ending.*
- unsigned int [ReadBytes](#) (byte handle, unsigned int &length, byte &buf[ ])  
*Read bytes from a file.*
- unsigned int [ReadLnString](#) (byte handle, string &output)  
*Read a string from a file plus line ending.*
- unsigned int [Write](#) (byte handle, const variant &value)  
*Write value to file.*
- unsigned int [WriteBytes](#) (byte handle, const byte &buf[ ], unsigned int &cnt)  
*Write bytes to file.*

- unsigned int [WriteBytesEx](#) (byte handle, unsigned int &len, const byte &buf[ ])   
*Write bytes to a file with limit.*
- unsigned int [WriteLn](#) (byte handle, const variant &value)   
*Write a value and new line to a file.*
- unsigned int [WriteLnString](#) (byte handle, const string &str, unsigned int &cnt)   
*Write string and new line to a file.*
- unsigned int [WriteString](#) (byte handle, const string &str, unsigned int &cnt)   
*Write string to a file.*
- void [SysFileOpenRead](#) (FileOpenType &args)   
*Open file for reading.*
- void [SysFileOpenWrite](#) (FileOpenType &args)   
*Open and create file for writing.*
- void [SysFileOpenAppend](#) (FileOpenType &args)   
*Open file for writing at end of file.*
- void [SysFileRead](#) (FileReadWriteType &args)   
*Read from file.*
- void [SysFileWrite](#) (FileReadWriteType &args)   
*File write.*
- void [SysFileClose](#) (FileCloseType &args)   
*Close file handle.*
- void [SysFileResolveHandle](#) (FileResolveHandleType &args)   
*File resolve handle.*
- void [SysFileRename](#) (FileRenameType &args)   
*Rename file.*
- void [SysFileDelete](#) (FileDeleteType &args)   
*Delete file.*
- void [SysLoaderExecuteFunction](#) (LoaderExecuteFunctionType &args)   
*Execute any Loader module command.*

- void `SysFileFindFirst` (`FileFindType` &args)  
*Start finding files.*
- void `SysFileFindNext` (`FileFindType` &args)  
*Continue finding files.*
- void `SysFileOpenWriteLinear` (`FileOpenType` &args)  
*Open and create linear file for writing.*
- void `SysFileOpenWriteNonLinear` (`FileOpenType` &args)  
*Open and create non-linear file for writing.*
- void `SysFileOpenReadLinear` (`FileOpenType` &args)  
*Open linear file for reading.*
- void `SysFileSeek` (`FileSeekType` &args)  
*Seek to file position.*
- void `SysFileResize` (`FileResizeType` &args)  
*Resize a file.*
- void `SysFileTell` (`FileTellType` &args)  
*Return the file position.*
- void `SysListFiles` (`ListFilesType` &args)  
*List files.*

### 6.66.1 Detailed Description

Functions for accessing and modifying Loader module features.

### 6.66.2 Function Documentation

#### 6.66.2.1 `unsigned int CloseFile` (byte *handle*) [`inline`]

Close a file. Close the file associated with the specified file handle. The loader result code is returned as the value of the function call. The handle parameter must be a constant or a variable.

**Parameters:**

*handle* The file handle.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_CloseFile.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

**6.66.2.2 unsigned int CreateFile (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]**

Create a file. Create a new file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.

*fsize* The size of the file.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_CreateFile.nxc](#), and [ex\\_file\\_system.nxc](#).

**6.66.2.3 unsigned int CreateFileLinear (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]**

Create a linear file. Create a new linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be

a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.  
*fsize* The size of the file.  
*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_CreateFileLinear.nxc](#).

#### 6.66.2.4 unsigned int CreateFileNonLinear (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]

Create a non-linear file. Create a new non-linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.  
*fsize* The size of the file.  
*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_CreateFileNonLinear.nxc](#).

**6.66.2.5 unsigned int DeleteFile (string *fname*) [inline]**

Delete a file. Delete the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to delete.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_delete\\_data\\_file.nxc](#), and [ex\\_DeleteFile.nxc](#).

**6.66.2.6 unsigned int FindFirstFile (string & *fname*, byte & *handle*) [inline]**

Start searching for files. This function lets you begin iterating through files stored on the NXT.

**Parameters:**

*fname* On input this contains the filename pattern you are searching for. On output this contains the name of the first file found that matches the pattern.

*handle* The search handle input to and output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

**6.66.2.7 unsigned int FindNextFile (string & *fname*, byte & *handle*)  
[inline]**

Continue searching for files. This function lets you continue iterating through files stored on the NXT.

**Parameters:**

*fname* On output this contains the name of the next file found that matches the pattern used when the search began by calling [FindFirstFile](#).

*handle* The search handle input to and output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

**6.66.2.8 unsigned int FreeMemory (void) [inline]**

Get free flash memory. Get the number of bytes of flash memory that are available for use.

**Returns:**

The number of bytes of unused flash memory.

**Examples:**

[ex\\_FreeMemory.nxc](#).

**6.66.2.9 unsigned int OpenFileAppend (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]**

Open a file for appending. Open an existing file with the specified filename for writing. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_OpenFileAppend.nxc](#).

**6.66.2.10 unsigned int OpenFileRead (string *fname*, unsigned int & *fsize*, byte & *handle*) [`inline`]**

Open a file for reading. Open an existing file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_OpenFileRead.nxc](#).

### 6.66.2.11 unsigned int OpenFileReadLinear (string *fname*, unsigned int & *fsize*, byte & *handle*) [**inline**]

Open a linear file for reading. Open an existing linear file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

#### Parameters:

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Warning:

This function requires the enhanced NBC/NXC firmware.

#### Examples:

[ex\\_OpenFileReadLinear.nxc](#).

### 6.66.2.12 unsigned int Read (byte *handle*, variant & *value*) [**inline**]

Read a value from a file. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read.

#### Parameters:

*handle* The file handle.

*value* The variable to store the data read from the file.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Examples:

[ex\\_file\\_system.nxc](#), and [ex\\_Read.nxc](#).

**6.66.2.13 unsigned int ReadBytes (byte *handle*, unsigned int & *length*, byte & *buf*[]) [inline]**

Read bytes from a file. Read the specified number of bytes from the file associated with the specified handle. The handle parameter must be a variable. The length parameter must be a variable. The buf parameter must be an array or a string variable. The actual number of bytes read is returned in the length parameter.

**Parameters:**

*handle* The file handle.

*length* The number of bytes to read. Returns the number of bytes actually read.

*buf* The byte array where the data is stored on output.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ReadBytes.nxc](#).

**6.66.2.14 unsigned int ReadLn (byte *handle*, variant & *value*) [inline]**

Read a value from a file plus line ending. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read. The ReadLn function reads two additional bytes from the file which it assumes are a carriage return and line feed pair.

**Parameters:**

*handle* The file handle.

*value* The variable to store the data read from the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ReadLn.nxc](#).

**6.66.2.15 unsigned int ReadLnString (byte *handle*, string & *output*)**  
[inline]

Read a string from a file plus line ending. Read a string from the file associated with the specified handle. The handle parameter must be a variable. The output parameter must be a variable. Appends bytes to the output variable until a line ending (CRLF) is reached. The line ending is also read but it is not appended to the output parameter.

**Parameters:**

*handle* The file handle.

*output* The variable to store the string read from the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**6.66.2.16 unsigned int RenameFile (string *oldname*, string *newname*)**  
[inline]

Rename a file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call. The filename parameters must be constants or variables.

**Parameters:**

*oldname* The old filename.

*newname* The new filename.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_RenameFile.nxc](#).

**6.66.2.17 unsigned int ResizeFile (string *fname*, const unsigned int *newsiz*)**  
[inline]

Resize a file. Resize the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to resize.

*newsiz*e The new size for the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_resizefile.nxc](#).

**6.66.2.18 unsigned int ResolveHandle (string *filename*, byte & *handle*, bool & *writable*) [inline]**

Resolve a handle. Resolve a file handle from the specified filename. The file handle is returned in the second parameter, which must be a variable. A boolean value indicating whether the handle can be used to write to the file or not is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*filename* The name of the file for which to resolve a handle.

*handle* The file handle output from the function call.

*writable* A boolean flag indicating whether the handle is to a file open for writing (true) or reading (false).

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ResolveHandle.nxc](#).

**6.66.2.19 unsigned int SizeOf (variant & *value*) [inline]**

Calculate the size of a variable. Calculate the number of bytes required to store the contents of the variable passed into the function.

**Parameters:**

*value* The variable.

**Returns:**

The number of bytes occupied by the variable.

**Examples:**

[ex\\_SizeOf.nxc](#).

**6.66.2.20 void SysFileClose (FileCloseType & args) [inline]**

Close file handle. This function lets you close a file using the values specified via the [FileCloseType](#) structure.

**Parameters:**

*args* The [FileCloseType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileclose.nxc](#).

**6.66.2.21 void SysFileDelete (FileDeleteType & args) [inline]**

Delete file. This function lets you delete a file using the values specified via the [FileDeleteType](#) structure.

**Parameters:**

*args* The [FileDeleteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfiledelete.nxc](#).

**6.66.2.22 void SysFileFindFirst (FileFindType & args) [inline]**

Start finding files. This function lets you begin iterating through files stored on the NXT.

**Parameters:**

*args* The [FileFindType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfilefindfirst.nxc](#).

**6.66.2.23 void SysFileFindNext (FileFindType & args) [inline]**

Continue finding files. This function lets you continue iterating through files stored on the NXT.

**Parameters:**

*args* The [FileFindType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfilefindnext.nxc](#).

**6.66.2.24 void SysFileOpenAppend (FileOpenType & args) [inline]**

Open file for writing at end of file. This function lets you open an existing file that you can write to using the values specified via the [FileOpenType](#) structure.

The available length remaining in the file is returned via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenappend.nxc](#).

**6.66.2.25 void SysFileOpenRead (FileOpenType & args) [inline]**

Open file for reading. This function lets you open an existing file for reading using the values specified via the [FileOpenType](#) structure.

The number of bytes that can be read from the file is returned via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenread.nxc](#).

**6.66.2.26 void SysFileOpenReadLinear (FileOpenType & args) [inline]**

Open linear file for reading. This function lets you open an existing linear file for reading using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenreadlinear.nxc](#).

**6.66.2.27 void SysFileOpenWrite (FileOpenType & args) [inline]**

Open and create file for writing. This function lets you create a file that you can write to using the values specified via the [FileOpenType](#) structure.

The desired maximum file capacity in bytes is specified via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenwrite.nxc](#).

**6.66.2.28 void SysFileOpenWriteLinear (FileOpenType & args) [inline]**

Open and create linear file for writing. This function lets you create a linear file that you can write to using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenwritelinear.nxc](#).

**6.66.2.29 void SysFileOpenWriteNonLinear (FileOpenType & args) [inline]**

Open and create non-linear file for writing. This function lets you create a non-linear linear file that you can write to using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenwritenonlinear.nxc](#).

**6.66.2.30 void SysFileRead (FileReadWriteType & args) [inline]**

Read from file. This function lets you read from a file using the values specified via the [FileReadWriteType](#) structure.

**Parameters:**

*args* The [FileReadWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileread.nxc](#).

**6.66.2.31 void SysFileRename (FileRenameType & args) [inline]**

Rename file. This function lets you rename a file using the values specified via the [FileRenameType](#) structure.

**Parameters:**

*args* The [FileRenameType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfilerename.nxc](#).

**6.66.2.32 void SysFileResize (FileResizeType & args) [inline]**

Resize a file. This function lets you resize a file using the values specified via the [FileResizeType](#) structure.

**Parameters:**

*args* The [FileResizeType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware. It has not yet been implemented at the firmware level.

**Examples:**

[ex\\_sysfileresize.nxc](#).

**6.66.2.33 void SysFileResolveHandle (FileResolveHandleType & args) [inline]**

File resolve handle. This function lets you resolve the handle of a file using the values specified via the [FileResolveHandleType](#) structure. This will find a previously opened file handle.

**Parameters:**

*args* The [FileResolveHandleType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileresolvehandle.nxc](#).

**6.66.2.34 void SysFileSeek (FileSeekType & args) [inline]**

Seek to file position. This function lets you seek to a specific file position using the values specified via the [FileSeekType](#) structure.

**Parameters:**

*args* The [FileSeekType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileseek.nxc](#).

**6.66.2.35 void SysFileTell (FileTellType & args) [inline]**

Return the file position. This function returns the current file position in the open file specified via the [FileTellType](#) structure.

**Parameters:**

*args* The [FileTellType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**6.66.2.36 void SysFileWrite (FileReadWriteType & args) [inline]**

File write. This function lets you write to a file using the values specified via the [FileReadWriteType](#) structure.

**Parameters:**

*args* The [FileReadWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfilewrite.nxc](#).

**6.66.2.37 void SysListFiles (ListFilesType & args) [inline]**

List files. This function lets you retrieve a list of files on the NXT using the values specified via the [ListFilesType](#) structure.

**Parameters:**

*args* The [ListFilesType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syslistfiles.nxc](#).

**6.66.2.38 void SysLoaderExecuteFunction (LoaderExecuteFunctionType & args) [inline]**

Execute any Loader module command. This function lets you directly execute the Loader module's primary function using the values specified via the [LoaderExecuteFunctionType](#) structure.

**Parameters:**

*args* The [LoaderExecuteFunctionType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysloaderexecutefunction.nxc](#).

**6.66.2.39 unsigned int Write (byte handle, const variant & value) [inline]**

Write value to file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written.

**Parameters:**

*handle* The file handle.

*value* The value to write to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_Write.nxc](#).

**6.66.2.40 unsigned int WriteBytes (byte *handle*, const byte & *buf*[], unsigned int & *cnt*) [inline]**

Write bytes to file. Write the contents of the data array to the file associated with the specified handle. The handle parameter must be a variable. The cnt parameter must be a variable. The data parameter must be a byte array. The actual number of bytes written is returned in the cnt parameter.

**Parameters:**

- handle* The file handle.
- buf* The byte array or string containing the data to write.
- cnt* The number of bytes actually written to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteBytes.nxc](#).

**6.66.2.41 unsigned int WriteBytesEx (byte *handle*, unsigned int & *len*, const byte & *buf*[]) [inline]**

Write bytes to a file with limit. Write the specified number of bytes to the file associated with the specified handle. The handle parameter must be a variable. The len parameter must be a variable. The buf parameter must be a byte array or a string variable or string constant. The actual number of bytes written is returned in the len parameter.

**Parameters:**

- handle* The file handle.
- len* The maximum number of bytes to write on input. Returns the actual number of bytes written.
- buf* The byte array or string containing the data to write.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteBytesEx.nxc](#).

**6.66.2.42 unsigned int WriteLn (byte *handle*, const variant & *value*)  
[inline]**

Write a value and new line to a file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written. This function also writes a carriage return and a line feed to the file following the numeric data.

**Parameters:**

*handle* The file handle.

*value* The value to write to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteLn.nxc](#).

**6.66.2.43 unsigned int WriteLnString (byte *handle*, const string & *str*,  
unsigned int & *cnt*) [inline]**

Write string and new line to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. This function also writes a carriage return and a line feed to the file following the string data. The total number of bytes written is returned in the cnt parameter.

**Parameters:**

*handle* The file handle.

*str* The string to write to the file.

*cnt* The number of bytes actually written to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteLnString.nxc](#).

#### 6.66.2.44 unsigned int WriteString (byte *handle*, const string & *str*, unsigned int & *cnt*) [inline]

Write string to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. The actual number of bytes written is returned in the cnt parameter.

##### Parameters:

- handle* The file handle.
- str* The string to write to the file.
- cnt* The number of bytes actually written to the file.

##### Returns:

The function call result. See [Loader module error codes](#).

##### Examples:

[ex\\_WriteString.nxc](#).

## 6.67 Microinfinity types

Types used by various Microinfinity device functions.

### Data Structures

- struct [XGPacketType](#)  
*Parameters for the [ReadSensorMIXG1300L](#) function.*

#### 6.67.1 Detailed Description

Types used by various Microinfinity device functions.

## 6.68 Microinfinity functions

Functions for interfacing with Microinfinity devices.

## Functions

- bool [ResetMIXG1300L](#) (byte port)  
*ResetMIXG1300L function.*
- int [SensorMIXG1300LScale](#) (byte port)  
*SensorMIXG1300LScale function.*
- bool [SetSensorMIXG1300LScale](#) (byte port, const byte scale)  
*SetSensorMIXG1300LScale function.*
- bool [ReadSensorMIXG1300L](#) (byte port, [XGPacketType](#) &packet)  
*ReadSensorMIXG1300L function.*

### 6.68.1 Detailed Description

Functions for interfacing with Microinfinity devices.

### 6.68.2 Function Documentation

#### 6.68.2.1 bool [ReadSensorMIXG1300L](#) (byte *port*, [XGPacketType](#) & *packet*) **[inline]**

[ReadSensorMIXG1300L](#) function. Read Microinfinity CruizCore XG1300L values. Read accumulated angle, turn rate, and X, Y, and Z axis acceleration values from the Microinfinity CruizCore XG1300L sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

- port* The sensor port. See the [Input port constants](#) group.
- packet* The output XK1300L data structure. See [XGPacketType](#).

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_xg1300.nxc](#).

### 6.68.2.2 bool ResetMIXG1300L (byte *port*) [inline]

ResetMIXG1300L function. Reset the Microinfinity CruzCore XG1300L device.

During reset, the XG1300L will recomputed the bias drift value, therefore it must remain stationary. The bias drift value will change randomly over time due to temperature variations, however the internal algorithm in the XG1300L will compensate for these changes. We strongly recommend issuing a reset command to the XG1300L at the beginning of the program.

The reset function also resets the accumulate angle value to a zero. Since the accelerometers measurements are taken with respect to the sensor reference frame the reset function will have no effect in the accelerometer measurements.

Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See the [Input port constants](#) group.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_xg1300.nxc](#).

### 6.68.2.3 int SensorMIXG1300LScale (byte *port*) [inline]

SensorMIXG1300LScale function. Read the Microinfinity CruzCore XG1300L accelerometer scale. The accelerometer in the CruzCore XG1300L can be set to operate with a scale ranging from +/-2G, +/-4G, or +/-8G. Returns the scale value that the device is currently configured to use. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See the [Input port constants](#) group.

#### Returns:

The current scale value.

**Examples:**

[ex\\_xg1300.nxc](#).

**6.68.2.4 bool SetSensorMIXG1300LScale (byte port, const byte scale)  
[inline]**

SetSensorMIXG1300LScale function. Set the Microinfinity CruizCore XG1300L accelerometer scale. The accelerometer in the CruizCore XG1300L can be set to operate with a scale ranging from +/-2G, +/-4G, or +/-8G. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See the [Input port constants](#) group.

*scale* This value must be a constant. See [Microinfinity CruizCore XG1300L](#).

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_xg1300.nxc](#).

**6.69 cmath API**

Standard C cmath API functions.

**Data Structures**

- struct [VectorType](#)

*This structure is used for storing three axis values in a single object.*

**Defines**

- #define [Sqrt](#)(\_X) asm { sqrt \_\_FLTRETVAL\_\_, \_X }  
*Compute square root.*
- #define [Sin](#)(\_X) asm { sin \_\_FLTRETVAL\_\_, \_X }

*Compute sine.*

- #define **Cos**(\_X) asm { cos \_\_FLTRETVAL\_\_, \_X }  
*Compute cosine.*
- #define **Asin**(\_X) asm { asin \_\_FLTRETVAL\_\_, \_X }  
*Compute arc sine.*
- #define **Acos**(\_X) asm { acos \_\_FLTRETVAL\_\_, \_X }  
*Compute arc cosine.*
- #define **Atan**(\_X) asm { atan \_\_FLTRETVAL\_\_, \_X }  
*Compute arc tangent.*
- #define **Ceil**(\_X) asm { ceil \_\_FLTRETVAL\_\_, \_X }  
*Round up value.*
- #define **Exp**(\_X) asm { exp \_\_FLTRETVAL\_\_, \_X }  
*Compute exponential function .*
- #define **Floor**(\_X) asm { floor \_\_FLTRETVAL\_\_, \_X }  
*Round down value.*
- #define **Tan**(\_X) asm { tan \_\_FLTRETVAL\_\_, \_X }  
*Compute tangent.*
- #define **Tanh**(\_X) asm { tanh \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic tangent.*
- #define **Cosh**(\_X) asm { cosh \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic cosine.*
- #define **Sinh**(\_X) asm { sinh \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic sine.*
- #define **Log**(\_X) asm { log \_\_FLTRETVAL\_\_, \_X }  
*Compute natural logarithm.*
- #define **Log10**(\_X) asm { log10 \_\_FLTRETVAL\_\_, \_X }  
*Compute common logarithm.*
- #define **Atan2**(\_Y, \_X) asm { atan2 \_\_FLTRETVAL\_\_, \_Y, \_X }

*Compute arc tangent with 2 parameters.*

- #define **Pow**(\_Base, \_Exponent) asm { pow \_\_FLTRETVAL\_\_, \_Base, \_Exponent }
- Raise to power.*
- #define **Trunc**(\_X) asm { trunc \_\_RETVAL\_\_, \_X }
- Compute integral part.*
- #define **Frac**(\_X) asm { frac \_\_FLTRETVAL\_\_, \_X }
- Compute fractional part.*
- #define **MulDiv32**(\_A, \_B, \_C) asm { muldiv \_\_RETVAL\_\_, \_A, \_B, \_C }
- Multiply and divide.*
- #define **SinD**(\_X) asm { sind \_\_FLTRETVAL\_\_, \_X }
- Compute sine (degrees).*
- #define **CosD**(\_X) asm { cosd \_\_FLTRETVAL\_\_, \_X }
- Compute cosine (degrees).*
- #define **AsinD**(\_X) asm { asind \_\_FLTRETVAL\_\_, \_X }
- Compute arch sine (degrees).*
- #define **AcosD**(\_X) asm { acosd \_\_FLTRETVAL\_\_, \_X }
- Compute arc cosine (degrees).*
- #define **AtanD**(\_X) asm { atand \_\_FLTRETVAL\_\_, \_X }
- Compute arc tangent (degrees).*
- #define **TanD**(\_X) asm { tand \_\_FLTRETVAL\_\_, \_X }
- Compute tangent (degrees).*
- #define **TanhD**(\_X) asm { tanhd \_\_FLTRETVAL\_\_, \_X }
- Compute hyperbolic tangent (degrees).*
- #define **CoshD**(\_X) asm { coshd \_\_FLTRETVAL\_\_, \_X }
- Compute hyperbolic cosine (degrees).*
- #define **SinhD**(\_X) asm { sinhd \_\_FLTRETVAL\_\_, \_X }
- Compute hyperbolic sine (degrees).*
- #define **Atan2D**(\_Y, \_X) asm { atan2d \_\_FLTRETVAL\_\_, \_Y, \_X }

*Compute arc tangent with two parameters (degrees).*

### Functions

- float `sqrt` (float x)  
*Compute square root.*
- float `cos` (float x)  
*Compute cosine.*
- float `sin` (float x)  
*Compute sine.*
- float `tan` (float x)  
*Compute tangent.*
- float `acos` (float x)  
*Compute arc cosine.*
- float `asin` (float x)  
*Compute arc sine.*
- float `atan` (float x)  
*Compute arc tangent.*
- float `atan2` (float y, float x)  
*Compute arc tangent with 2 parameters.*
- float `cosh` (float x)  
*Compute hyperbolic cosine.*
- float `sinh` (float x)  
*Compute hyperbolic sine.*
- float `tanh` (float x)  
*Compute hyperbolic tangent.*
- float `exp` (float x)  
*Compute exponential function.*
- float `log` (float x)

*Compute natural logarithm.*

- float `log10` (float x)  
*Compute common logarithm.*
- long `trunc` (float x)  
*Compute integral part.*
- float `frac` (float x)  
*Compute fractional part.*
- float `pow` (float base, float exponent)  
*Raise to power.*
- float `ceil` (float x)  
*Round up value.*
- float `floor` (float x)  
*Round down value.*
- long `muldiv32` (long a, long b, long c)  
*Multiply and divide.*
- float `cosd` (float x)  
*Compute cosine (degrees).*
- float `sind` (float x)  
*Compute sine (degrees).*
- float `tand` (float x)  
*Compute tangent (degrees).*
- float `acosd` (float x)  
*Compute arc cosine (degrees).*
- float `asind` (float x)  
*Compute arc sine (degrees).*
- float `atand` (float x)  
*Compute arc tangent (degrees).*
- float `atan2d` (float y, float x)

*Compute arc tangent with 2 parameters (degrees).*

- float `coshd` (float x)  
*Compute hyperbolic cosine (degrees).*
- float `sinhd` (float x)  
*Compute hyperbolic sine (degrees).*
- float `tanhd` (float x)  
*Compute hyperbolic tangent (degrees).*
- byte `bcd2dec` (byte bcd)  
*Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.*
- bool `isNAN` (float value)  
*Is the value NaN.*
- char `sign` (variant num)  
*Sign value.*
- void `VectorCross` (`VectorType` a, `VectorType` b, `VectorType` &out)  
*VectorCross function.*
- float `VectorDot` (`VectorType` a, `VectorType` b)  
*VectorDot function.*
- void `VectorNormalize` (`VectorType` &a)  
*VectorNormalize function.*

### 6.69.1 Detailed Description

Standard C cmath API functions.

### 6.69.2 Define Documentation

#### 6.69.2.1 `#define Acos(_X) asm { acos __FLTRETVAL__, _X }`

Compute arc cosine. Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [acos\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc cosine of `_X`.

**6.69.2.2 #define AcosD(\_X) asm { acosd \_\_FLTRETVAL\_\_, \_X }**

Compute arc cosine (degrees). Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [acosd\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc cosine of `_X`.

**6.69.2.3 #define Asin(\_X) asm { asin \_\_FLTRETVAL\_\_, \_X }**

Compute arc sine. Computes the arc sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [asin\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc sine of `_X`.

**6.69.2.4 #define AsinD(\_X) asm { asind \_\_FLTRETVAL\_\_, \_X }**

Compute arch sine (degrees). Computes the arc sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `asind()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc sine of `_X`.

**6.69.2.5 #define Atan(\_X) asm { atan \_\_FLTRETVAL\_\_, \_X }**

Compute arc tangent. Computes the arc tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc tangent of `_X`.

**6.69.2.6 #define Atan2(\_Y, \_X) asm { atan2 \_\_FLTRETVAL\_\_, \_Y, \_X }**

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of `_Y/_X`, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan2()` instead.

**Parameters:**

`_Y` Floating point value representing a y coordinate.

`_X` Floating point value representing an x coordinate.

**Returns:**

Arc tangent of `_Y/_X`, in the interval `[-pi,+pi]` radians.

**6.69.2.7** `#define Atan2D(_Y, _X) asm { atan2d __FLTRETVAL__, _Y, _X }`

Compute arc tangent with two parameters (degrees). Computes the arc tangent of `_Y/_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan2d()` instead.

**Parameters:**

`_Y` Floating point value.

`_X` Floating point value.

**Returns:**

Arc tangent of `_Y/_X`, in the interval `[-180,+180]` degrees.

**6.69.2.8** `#define AtanD(_X) asm { atand __FLTRETVAL__, _X }`

Compute arc tangent (degrees). Computes the arc tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atand()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc tangent of `_X`.

**6.69.2.9 #define Ceil(\_X) asm { ceil \_\_FLTRETVAL\_\_, \_X }**

Round up value. Computes the smallest integral value that is not less than `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `ceil()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

The smallest integral value not less than `_X`.

**6.69.2.10 #define Cos(\_X) asm { cos \_\_FLTRETVAL\_\_, \_X }**

Compute cosine. Computes the cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cos()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Cosine of `_X`.

**6.69.2.11 #define CosD(\_X) asm { cosd \_\_FLTRETVAL\_\_, \_X }**

Compute cosine (degrees). Computes the cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cosd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Cosine of `_X`.

**6.69.2.12** `#define Cosh(_X) asm { cosh __FLTRETVAL__, _X }`

Compute hyperbolic cosine. Computes the hyperbolic cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cosh()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic cosine of `_X`.

**6.69.2.13** `#define CoshD(_X) asm { coshd __FLTRETVAL__, _X }`

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `coshd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic cosine of `_X`.

**6.69.2.14 #define Exp(\_X) asm { exp \_\_FLTRETVAL\_\_, \_X }**

Compute exponential function . Computes the base-e exponential function of `_X`, which is the e number raised to the power `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `exp()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Exponential value of `_X`.

**6.69.2.15 #define Floor(\_X) asm { floor \_\_FLTRETVAL\_\_, \_X }**

Round down value. Computes the largest integral value that is not greater than `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `floor()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

The largest integral value not greater than `_X`.

**6.69.2.16 #define Frac(\_X) asm { frac \_\_FLTRETVAL\_\_, \_X }**

Compute fractional part. Computes the fractional part of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `frac()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Fractional part of `_X`.

**6.69.2.17 #define Log(\_X) asm { log \_\_FLTRETVAL\_\_, \_X }**

Compute natural logarithm. Computes the natural logarithm of `_X`. The natural logarithm is the base-e logarithm, the inverse of the natural exponential function (`exp`). For base-10 logarithms, a specific function `Log10()` exists. Only constants or variables allowed (no expressions).

**Deprecated**

Use `log()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Natural logarithm of `_X`.

**6.69.2.18 #define Log10(\_X) asm { log10 \_\_FLTRETVAL\_\_, \_X }**

Compute common logarithm. Computes the common logarithm of `_X`. The common logarithm is the base-10 logarithm. For base-e logarithms, a specific function `Log()` exists. Only constants or variables allowed (no expressions).

**Deprecated**

Use `log10()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Common logarithm of `_X`.

```
6.69.2.19 #define MulDiv32(_A, _B, _C) asm { muldiv __RETVAL__, _A, _B,
        _C }
```

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value. Only constants or variables allowed (no expressions).

**Deprecated**

Use `muldiv32()` instead.

**Parameters:**

`_A` 32-bit long value.

`_B` 32-bit long value.

`_C` 32-bit long value.

**Returns:**

The result of multiplying `_A` times `_B` and dividing by `_C`.

```
6.69.2.20 #define Pow(_Base, _Exponent) asm { pow __FLTRETVAL__, _Base,
        _Exponent }
```

Raise to power. Computes `_Base` raised to the power `_Exponent`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `pow()` instead.

**Parameters:**

`_Base` Floating point value.

`_Exponent` Floating point value.

**Returns:**

The result of raising `_Base` to the power `_Exponent`.

**6.69.2.21 #define Sin(\_X) asm { sin \_\_FLTRETVAL\_\_, \_X }**

Compute sine. Computes the sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sin()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Sine of `_X`.

**6.69.2.22 #define SinD(\_X) asm { sind \_\_FLTRETVAL\_\_, \_X }**

Compute sine (degrees). Computes the sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sind()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Sine of `_X`.

**6.69.2.23 #define Sinh(\_X) asm { sinh \_\_FLTRETVAL\_\_, \_X }**

Compute hyperbolic sine. Computes the hyperbolic sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sinh()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic sine of `_X`.

**6.69.2.24** `#define SinhD(_X) asm { sinh __FLTRETVAL__, _X }`

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sinhd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic sine of `_X`.

**6.69.2.25** `#define Sqrt(_X) asm { sqrt __FLTRETVAL__, _X }`

Compute square root. Computes the square root of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sqrt()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Square root of `_X`.

**6.69.2.26 #define Tan(\_X) asm { tan \_\_FLTRETVAL\_\_, \_X }**

Compute tangent. Computes the tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `tan()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Tangent of `_X`.

**6.69.2.27 #define TanD(\_X) asm { tand \_\_FLTRETVAL\_\_, \_X }**

Compute tangent (degrees). Computes the sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `tand()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Tangent of `_X`.

**6.69.2.28 #define Tanh(\_X) asm { tanh \_\_FLTRETVAL\_\_, \_X }**

Compute hyperbolic tangent. Computes the hyperbolic tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `tanh()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic tangent of `_X`.

**6.69.2.29** `#define TanhD(_X) asm { tanhd __FLTRETVAL__, _X }`

Compute hyperbolic tangent (degrees). Computes the hyperbolic tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `tanhD()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic tangent of `_X`.

**6.69.2.30** `#define Trunc(_X) asm { trunc __RETVAL__, _X }`

Compute integral part. Computes the integral part of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `trunc()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Integral part of `_X`.

### 6.69.3 Function Documentation

#### 6.69.3.1 float acos (float *x*) [**inline**]

Compute arc cosine. Computes the principal value of the arc cosine of *x*, expressed in radians. In trigonometrics, arc cosine is the inverse operation of cosine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc cosine of *x*, in the interval [0,pi] radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acos.nxc](#).

#### 6.69.3.2 float acosd (float *x*) [**inline**]

Compute arc cosine (degrees). Computes the principal value of the arc cosine of *x*, expressed in degrees. In trigonometrics, arc cosine is the inverse operation of cosine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc cosine of *x*, in the interval [0,180] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acosd.nxc](#).

### 6.69.3.3 float asin (float *x*) [**inline**]

Compute arc sine. Computes the principal value of the arc sine of *x*, expressed in radians. In trigonometrics, arc sine is the inverse operation of sine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc sine of *x*, in the interval [-pi/2,+pi/2] radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_asin.nxc](#).

### 6.69.3.4 float asind (float *x*) [**inline**]

Compute arc sine (degrees). Computes the principal value of the arc sine of *x*, expressed in degrees. In trigonometrics, arc sine is the inverse operation of sine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc sine of *x*, in the interval [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_asind.nxc](#).

### 6.69.3.5 float atan(float x) [inline]

Compute arc tangent. Computes the principal value of the arc tangent of  $x$ , expressed in radians. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use [atan2\(\)](#) if you need to determine the quadrant.

**See also:**

[atan2\(\)](#)

**Parameters:**

$x$  Floating point value.

**Returns:**

Arc tangent of  $x$ , in the interval  $[-\pi/2, +\pi/2]$  radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan.nxc](#).

### 6.69.3.6 float atan2(float y, float x) [inline]

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of  $y/x$ , expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant.

**See also:**

[atan\(\)](#)

**Parameters:**

$y$  Floating point value representing a  $y$  coordinate.

$x$  Floating point value representing an  $x$  coordinate.

**Returns:**

Arc tangent of  $y/x$ , in the interval  $[-\pi, +\pi]$  radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2.nxc](#).

**6.69.3.7 float atan2d (float y, float x) [inline]**

Compute arc tangent with 2 parameters (degrees). Computes the principal value of the arc tangent of  $y/x$ , expressed in degrees. To compute the value, the function uses the sign of both arguments to determine the quadrant.

**Parameters:**

- $y$  Floating point value representing a y coordinate.
- $x$  Floating point value representing an x coordinate.

**Returns:**

Arc tangent of  $y/x$ , in the interval  $[-180, +180]$  degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2d.nxc](#).

**6.69.3.8 float atand (float x) [inline]**

Compute arc tangent (degrees). Computes the principal value of the arc tangent of  $x$ , expressed in degrees. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use `atan2d` if you need to determine the quadrant.

**Parameters:**

*x* Floating point value.

**Returns:**

Arc tangent of *x*, in the interval [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atand.nxc](#).

**6.69.3.9 byte bcd2dec (byte *bcd*) [inline]**

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

**Parameters:**

*bcd* The value you want to convert from bcd to decimal.

**Returns:**

The decimal equivalent of the binary coded decimal byte.

**Examples:**

[ex\\_bcd2dec.nxc](#).

**6.69.3.10 float ceil (float *x*) [inline]**

Round up value. Computes the smallest integral value that is not less than *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

The smallest integral value not less than *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_ceil.nxc](#).

**6.69.3.11 float cos (float *x*) [inline]**

Compute cosine. Computes the cosine of an angle of *x* radians.

**Parameters:**

*x* Floating point value representing an angle expressed in radians.

**Returns:**

Cosine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#).

**6.69.3.12 float cosd (float *x*) [inline]**

Compute cosine (degrees). Computes the cosine of an angle of *x* degrees.

**Parameters:**

*x* Floating point value representing an angle expressed in degrees.

**Returns:**

Cosine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sind\\_cosd.nxc](#).

**6.69.3.13 float cosh (float x) [inline]**

Compute hyperbolic cosine. Computes the hyperbolic cosine of  $x$ , expressed in radians.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic cosine of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_cosh.nxc](#).

**6.69.3.14 float coshd (float x) [inline]**

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of  $x$ , expressed in degrees.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic cosine of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.69.3.15 float exp (float x) [inline]**

Compute exponential function. Computes the base- $e$  exponential function of  $x$ , which is the  $e$  number raised to the power  $x$ .

**Parameters:**

$x$  Floating point value.

**Returns:**

Exponential value of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_exp.nxc](#).

**6.69.3.16 float floor (float  $x$ ) [inline]**

Round down value. Computes the largest integral value that is not greater than  $x$ .

**Parameters:**

$x$  Floating point value.

**Returns:**

The largest integral value not greater than  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_floor.nxc](#).

**6.69.3.17 float frac (float  $x$ ) [inline]**

Compute fractional part. Computes the fractional part of  $x$ .

**Parameters:**

$x$  Floating point value.

**Returns:**

Fractional part of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_frac.nxc](#).

**6.69.3.18 bool isNAN (float *value*) [`inline`]**

Is the value NaN. Returns true if the floating point value is NaN (not a number).

**Parameters:**

*value* A floating point variable.

**Returns:**

Whether the value is NaN.

**Examples:**

[ex\\_isnan.nxc](#), and [ex\\_labs.nxc](#).

**6.69.3.19 float log (float *x*) [`inline`]**

Compute natural logarithm. Computes the natural logarithm of  $x$ . The natural logarithm is the base- $e$  logarithm, the inverse of the natural exponential function (`exp`). For base-10 logarithms, a specific function `log10()` exists.

**See also:**

[log10\(\)](#), [exp\(\)](#)

**Parameters:**

$x$  Floating point value.

**Returns:**

Natural logarithm of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_log.nxc](#).

**6.69.3.20 float log10 (float *x*) [inline]**

Compute common logarithm. Computes the common logarithm of *x*. The common logarithm is the base-10 logarithm. For base-e logarithms, a specific function [log\(\)](#) exists.

**See also:**

[log\(\)](#), [exp\(\)](#)

**Parameters:**

*x* Floating point value.

**Returns:**

Common logarithm of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_log10.nxc](#).

**6.69.3.21 long multdiv32 (long *a*, long *b*, long *c*) [inline]**

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value.

**Parameters:**

*a* 32-bit long value.

*b* 32-bit long value.

*c* 32-bit long value.

**Returns:**

The result of multiplying *a* times *b* and dividing by *c*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_muldiv32.nxc](#).

**6.69.3.22 float pow (float *base*, float *exponent*) [inline]**

Raise to power. Computes base raised to the power exponent.

**Parameters:**

*base* Floating point value.

*exponent* Floating point value.

**Returns:**

The result of raising base to the power exponent.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_pow.nxc](#).

**6.69.3.23 char sign (variant *num*) [inline]**

Sign value. Return the sign of the value argument (-1, 0, or 1). Any scalar type can be passed into this function.

**Parameters:**

*num* The numeric value for which to calculate its sign value.

**Returns:**

-1 if the parameter is negative, 0 if the parameter is zero, or 1 if the parameter is positive.

**Examples:**

[ex\\_sign.nxc](#).

**6.69.3.24 float sin (float *x*) [inline]**

Compute sine. Computes the sine of an angle of *x* radians.

**Parameters:**

*x* Floating point value representing an angle expressed in radians.

**Returns:**

Sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#).

**6.69.3.25 float sind (float *x*) [inline]**

Compute sine (degrees). Computes the sine of an angle of *x* degrees.

**Parameters:**

*x* Floating point value representing an angle expressed in degrees.

**Returns:**

Sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sind\\_cosd.nxc](#).

**6.69.3.26 float sinh (float *x*) [inline]**

Compute hyperbolic sine. Computes the hyperbolic sine of *x*, expressed in radians.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sinh.nxc](#).

**6.69.3.27 float sinhd (float *x*) [inline]**

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of *x*, expressed in degrees.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.69.3.28 float sqrt (float x) [inline]**

Compute square root. Computes the square root of  $x$ .

**Parameters:**

$x$  Floating point value.

**Returns:**

Square root of  $x$ .

**Examples:**

[ex\\_isnan.nxc](#), [ex\\_labs.nxc](#), and [ex\\_sqrt.nxc](#).

**6.69.3.29 float tan (float x) [inline]**

Compute tangent. Computes the tangent of an angle of  $x$  radians.

**Parameters:**

$x$  Floating point value representing an angle expressed in radians.

**Returns:**

Tangent of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tan.nxc](#).

**6.69.3.30 float tand (float x) [inline]**

Compute tangent (degrees). Computes the tangent of an angle of  $x$  degrees.

**Parameters:**

$x$  Floating point value representing an angle expressed in degrees.

**Returns:**

Tangent of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tand.nxc](#).

**6.69.3.31 float tanh (float  $x$ ) [inline]**

Compute hyperbolic tangent. Computes the hyperbolic tangent of  $x$ , expressed in radians.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic tangent of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tanh.nxc](#).

**6.69.3.32 float tanhd (float  $x$ ) [inline]**

Compute hyperbolic tangent (degrees). Computes the hyperbolic tangent of  $x$ , expressed in degrees.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic tangent of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**6.69.3.33 long trunc (float  $x$ ) [inline]**

Compute integral part. Computes the integral part of  $x$ .

**Parameters:**

$x$  Floating point value.

**Returns:**

Integral part of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), and [ex\\_trunc.nxc](#).

**6.69.3.34 void VectorCross (VectorType  $a$ , VectorType  $b$ , VectorType &  $out$ ) [inline]**

VectorCross function. Calculate the cross-product of two vectors.

**Parameters:**

$a$  A variable of type [VectorType](#)

$b$  A variable of type [VectorType](#)

$out$  The cross-product vector.

**6.69.3.35** `float VectorDot (VectorType a, VectorType b) [inline]`

`VectorDot` function. Calculate the dot-product of two vectors.

**Parameters:**

- a* A variable of type `VectorType`
- b* A variable of type `VectorType`

**6.69.3.36** `void VectorNormalize (VectorType & a) [inline]`

`VectorNormalize` function. Normalize the vector.

**Parameters:**

- a* A variable of type `VectorType`

**6.70** `cstdio` API

Standard C `cstdio` API functions.

**Modules**

- [fseek origin constants](#)  
*Constants for use in calls to `fseek`.*

**Defines**

- `#define getc(_handle) fgetc(_handle)`  
*Get character from file.*
- `#define putc(_ch, _handle) fputc(_ch, _handle)`  
*Write character to file.*

### Functions

- int `fclose` (byte handle)  
*Close file.*
- int `remove` (string filename)  
*Remove file.*
- int `rename` (string old, string new)  
*Rename file.*
- char `fgetc` (byte handle)  
*Get character from file.*
- string `fgets` (string &str, int num, byte handle)  
*Get string from file.*
- int `feof` (byte handle)  
*Check End-of-file indicator.*
- void `set_fopen_size` (unsigned long fsize)  
*Set the default fopen file size.*
- byte `fopen` (string filename, const string mode)  
*Open file.*
- int `fflush` (byte handle)  
*Flush file.*
- unsigned long `ftell` (byte handle)  
*Get current position in file.*
- char `fputc` (char ch, byte handle)  
*Write character to file.*
- int `fputs` (string str, byte handle)  
*Write string to file.*
- void `printf` (string format, variant value)  
*Print formatted data to stdout.*
- void `fprintf` (byte handle, string format, variant value)  
*Write formatted data to file.*

- void `sprintf` (string &str, string format, variant value)  
*Write formatted data to string.*
- int `fseek` (byte handle, long offset, int origin)  
*Reposition file position indicator.*
- void `rewind` (byte handle)  
*Set position indicator to the beginning.*
- int `getchar` ()  
*Get character from stdin.*

### Variables

- unsigned long `__fopen_default_size` = 1024

### 6.70.1 Detailed Description

Standard C `stdio` API functions.

### 6.70.2 Define Documentation

#### 6.70.2.1 `#define getc(_handle) fgetc(_handle)`

Get character from file. Returns the character currently pointed to by the internal file position indicator of the file specified by the handle. The internal file position indicator is then advanced by one character to point to the next character. The functions `fgetc` and `getc` are equivalent.

#### Parameters:

*\_handle* The handle of the file from which the character is read.

#### Returns:

The character read from the file.

#### Examples:

[ex\\_getc.nxc](#).

### 6.70.2.2 `#define` `putc(_ch, _handle)` `fputc(_ch, _handle)`

Write character to file. Writes a character to the file and advances the position indicator. The character is written at the current position of the file as indicated by the internal position indicator, which is then advanced one character. If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned.

#### Parameters:

`_ch` The character to be written.

`_handle` The handle of the file where the character is to be written.

#### Returns:

The character written to the file.

#### Examples:

[ex\\_putc.nxc](#).

## 6.70.3 Function Documentation

### 6.70.3.1 `int` `fclose` (byte *handle*) [`inline`]

Close file. Close the file associated with the specified file handle. The loader result code is returned as the value of the function call.

#### Parameters:

*handle* The handle of the file to be closed.

#### Returns:

The loader result code.

#### Examples:

[ex\\_fclose.nxc](#).

### 6.70.3.2 `int` `feof` (byte *handle*) [`inline`]

Check End-of-file indicator. Checks whether the End-of-File indicator associated with the handle is set, returning a value different from zero if it is.

**Parameters:**

*handle* The handle of the file to check.

**Returns:**

Currently always returns 0.

**Examples:**

[ex\\_feof.nxc](#).

**6.70.3.3 `int fflush (byte handle) [inline]`**

Flush file. Writes any buffered data to the file. A zero value indicates success.

**Parameters:**

*handle* The handle of the file to be flushed.

**Returns:**

Currently always returns 0.

**Examples:**

[ex\\_fflush.nxc](#).

**6.70.3.4 `char fgetc (byte handle) [inline]`**

Get character from file. Returns the character currently pointed to by the internal file position indicator of the file specified by the handle. The internal file position indicator is then advanced by one character to point to the next character. The functions `fgetc` and `getc` are equivalent.

**Parameters:**

*handle* The handle of the file from which the character is read.

**Returns:**

The character read from the file.

**Examples:**

[ex\\_fgetc.nxc](#).

### 6.70.3.5 `string fgets (string & str, int num, byte handle) [inline]`

Get string from file. Reads characters from a file and stores them as a string into `str` until `(num-1)` characters have been read or either a newline or a the End-of-File is reached, whichever comes first. A newline character makes `fgets` stop reading, but it is considered a valid character and therefore it is included in the string copied to `str`. A null character is automatically appended in `str` after the characters read to signal the end of the string. Returns the string parameter.

#### Parameters:

- str* The string where the characters are stored.
- num* The maximum number of characters to be read.
- handle* The handle of the file from which the characters are read.

#### Returns:

The string read from the file.

#### Examples:

[ex\\_fgets.nxc](#).

### 6.70.3.6 `byte fopen (string filename, const string mode)`

Open file. Opens the file whose name is specified in the parameter `filename` and associates it with a file handle that can be identified in future operations by the handle that is returned. The operations that are allowed on the stream and how these are performed are defined by the mode parameter.

#### Parameters:

- filename* The name of the file to be opened.
- mode* The file access mode. Valid values are "r" - opens an existing file for reading, "w" - creates a new file and opens it for writing, and "a" - opens an existing file for appending to the end of the file.

#### Returns:

The handle to the opened file.

#### Examples:

[ex\\_fopen.nxc](#).

**6.70.3.7** `void fprintf (byte handle, string format, variant value) [inline]`

Write formatted data to file. Writes a sequence of data formatted as the *format* argument specifies to a file. After the *format* parameter, the function expects one value argument.

**Parameters:**

*handle* The handle of the file to write to.

*format* A string specifying the desired format.

*value* A value to be formatted for writing to the file.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_fprintf.nxc](#).

**6.70.3.8** `char fputc (char ch, byte handle) [inline]`

Write character to file. Writes a character to the file and advances the position indicator. The character is written at the current position of the file as indicated by the internal position indicator, which is then advanced one character. If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned.

**Parameters:**

*ch* The character to be written.

*handle* The handle of the file where the character is to be written.

**Returns:**

The character written to the file.

**Examples:**

[ex\\_fputc.nxc](#).

**6.70.3.9** `int fputs (string str, byte handle) [inline]`

Write string to file. Writes the string to the file specified by the handle. The null terminating character at the end of the string is not written to the file. If there are no errors, a non-negative value is returned. If an error occurs, EOF is returned.

**Parameters:**

*str* The string of characters to be written.

*handle* The handle of the file where the string is to be written.

**Returns:**

The number of characters written to the file.

**Examples:**

[ex\\_fputs.nxc](#).

**6.70.3.10** `int fseek (byte handle, long offset, int origin) [inline]`

Reposition file position indicator. Sets the position indicator associated with the file to a new position defined by adding offset to a reference position specified by origin.

**Parameters:**

*handle* The handle of the file.

*offset* The number of bytes to offset from origin.

*origin* Position from where offset is added. It is specified by one of the following constants: SEEK\_SET - beginning of file, SEEK\_CUR - current position of the file pointer, or SEEK\_END - end of file. [fseek origin constants](#)

**Returns:**

A value of zero if successful or non-zero otherwise. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_fseek.nxc](#).

**6.70.3.11 unsigned long ftell (byte *handle*) [inline]**

Get current position in file. Returns the current value of the file position indicator of the specified handle.

**Parameters:**

*handle* The handle of the file.

**Returns:**

The current file position in the open file.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_ftell.nxc](#).

**6.70.3.12 int getchar () [inline]**

Get character from stdin. Returns the next character from the standard input (stdin). It is equivalent to `getc` with `stdin` as its argument. On the NXT this means wait for a button press and return the value of the button pressed.

**Returns:**

The pressed button. See [Button name constants](#).

**Examples:**

[ex\\_getchar.nxc](#).

**6.70.3.13 void printf (string *format*, variant *value*) [inline]**

Print formatted data to stdout. Writes to the LCD at 0, LCD\_LINE1 a sequence of data formatted as the format argument specifies. After the format parameter, the function expects one value argument.

**Parameters:**

*format* A string specifying the desired format.

*value* A value to be formatted for writing to the LCD.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_printf.nxc](#).

**6.70.3.14 `int remove (string filename) [inline]`**

Remove file. Delete the specified file. The loader result code is returned as the value of the function call.

**Parameters:**

*filename* The name of the file to be deleted.

**Returns:**

The loader result code.

**6.70.3.15 `int rename (string old, string new) [inline]`**

Rename file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call.

**Parameters:**

*old* The name of the file to be renamed.

*new* The new name for the file.

**Returns:**

The loader result code.

**Examples:**

[ex\\_rename.nxc](#).

**6.70.3.16** `void rewind (byte handle) [inline]`

Set position indicator to the beginning. Sets the position indicator associated with stream to the beginning of the file.

**Parameters:**

*handle* The handle of the file.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_rewind.nxc](#).

**6.70.3.17** `void set_fopen_size (unsigned long fsize) [inline]`

Set the default fopen file size. Set the default size of a file created via a call to fopen.

**Parameters:**

*fsize* The default new file size for fopen.

**6.70.3.18** `void sprintf (string & str, string format, variant value) [inline]`

Write formatted data to string. Writes a sequence of data formatted as the format argument specifies to a string. After the format parameter, the function expects one value argument.

**Parameters:**

*str* The string to write to.

*format* A string specifying the desired format.

*value* A value to be formatted for writing to the string.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sprintf.nxc](#).

**6.71 fseek origin constants**

Constants for use in calls to fseek.

**Defines**

- #define [SEEK\\_SET](#) 0
- #define [SEEK\\_CUR](#) 1
- #define [SEEK\\_END](#) 2

**6.71.1 Detailed Description**

Constants for use in calls to fseek.

**6.71.2 Define Documentation****6.71.2.1 #define SEEK\_CUR 1**

Seek from the current file position

**Examples:**

[ex\\_fseek.nxc](#).

**6.71.2.2 #define SEEK\_END 2**

Seek from the end of the file

**6.71.2.3 #define SEEK\_SET 0**

Seek from the beginning of the file

**Examples:**

[ex\\_sysfileseek.nxc](#).

## 6.72 cstdlib API

Standard C cstdlib API functions and types.

### Modules

- [cstdlib API types](#)  
*Standard C cstdlib API types.*

### Functions

- void [abort](#) ()  
*Abort current process.*
- variant [abs](#) (variant num)  
*Absolute value.*
- long [srand](#) (long seed)  
*Seed the random number generator.*
- unsigned long [rand](#) ()  
*Generate random number.*
- int [Random](#) (unsigned int n=0)  
*Generate random number.*
- void [SysRandomNumber](#) ([RandomNumberType](#) &args)  
*Draw a random number.*
- void [SysRandomEx](#) ([RandomExType](#) &args)  
*Call the enhanced random number function.*
- int [atoi](#) (const string &str)  
*Convert string to integer.*
- long [atol](#) (const string &str)  
*Convert string to long integer.*
- long [labs](#) (long n)  
*Absolute value.*

- float [atof](#) (const string &str)  
*Convert string to float.*
- float [strtod](#) (const string &str, string &endptr)  
*Convert string to float.*
- long [strtol](#) (const string &str, string &endptr, int base=10)  
*Convert string to long integer.*
- long [strtoul](#) (const string &str, string &endptr, int base=10)  
*Convert string to unsigned long integer.*
- [div\\_t div](#) (int numer, int denom)  
*Integral division.*
- [ldiv\\_t ldiv](#) (long numer, long denom)  
*Integral division.*

### 6.72.1 Detailed Description

Standard C cstdlib API functions and types.

### 6.72.2 Function Documentation

#### 6.72.2.1 void abort () [**inline**]

Abort current process. Aborts the process with an abnormal program termination. The function never returns to its caller.

#### Examples:

[ex\\_abort.nxc](#).

#### 6.72.2.2 variant abs (variant *num*) [**inline**]

Absolute value. Return the absolute value of the value argument. Any scalar type can be passed into this function.

**Parameters:**

*num* The numeric value.

**Returns:**

The absolute value of *num*. The return type matches the input type.

**Examples:**

[ex\\_abs.nxc](#).

**6.72.2.3 `float atof (const string & str) [inline]`**

Convert string to float. Parses the string *str* interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. The rest of the string after the last valid character is ignored and has no effect on the behavior of this function.

A valid floating point number for `atof` is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in *str* does not form a valid floating-point number as just defined, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of a floating-point number.

**Returns:**

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

**Examples:**

[ex\\_atof.nxc](#).

#### 6.72.2.4 `int atoi (const string & str)` [`inline`]

Convert string to integer. Parses the string `str` interpreting its content as an integral number, which is returned as an `int` value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in `str` does not form a valid integral number, or if no such sequence exists because either `str` is empty or contains only whitespace characters, no conversion is performed.

##### Parameters:

*str* String beginning with the representation of an integral number.

##### Returns:

On success, the function returns the converted integral number as an `int` value. If no valid conversion could be performed a zero value is returned.

##### Examples:

[ex\\_atoi.nxc](#).

#### 6.72.2.5 `long atol (const string & str)` [`inline`]

Convert string to long integer. Parses the string `str` interpreting its content as an integral number, which is returned as a long `int` value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in `str` does not form a valid integral number, or if no such sequence exists because either `str` is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of an integral number.

**Returns:**

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

**Examples:**

[ex\\_atol.nxc](#).

**6.72.2.6 div\_t div (int numer, int denom) [inline]**

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type `div_t`, which has two members: `quot` and `rem`.

**Parameters:**

*numer* Numerator.

*denom* Denominator.

**Returns:**

The result is returned by value in a structure defined in `cstdlib`, which has two members. For `div_t`, these are, in either order: `int quot`; `int rem`.

**Examples:**

[ex\\_div.nxc](#).

**6.72.2.7 long labs (long n) [inline]**

Absolute value. Return the absolute value of parameter `n`.

**Parameters:**

*n* Integral value.

**Returns:**

The absolute value of `n`.

### 6.72.2.8 `ldiv_t ldiv (long numer, long denom) [inline]`

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type `ldiv_t`, which has two members: `quot` and `rem`.

#### Parameters:

*numer* Numerator.

*denom* Denominator.

#### Returns:

The result is returned by value in a structure defined in `cstdlib`, which has two members. For `ldiv_t`, these are, in either order: long quot; long rem.

#### Examples:

[ex\\_ldiv.nxc](#).

### 6.72.2.9 `unsigned long rand () [inline]`

Generate random number. Returns a pseudo-random integral number in the range 0 to `RAND_MAX`.

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called.

#### Returns:

An integer value between 0 and `RAND_MAX` (inclusive).

#### Examples:

[ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), and [ex\\_rand.nxc](#).

### 6.72.2.10 `int Random (unsigned int n = 0) [inline]`

Generate random number. Return a signed or unsigned 16-bit random number. If the optional argument `n` is not provided the function will return a signed value. Otherwise the returned value will range between 0 and `n` (exclusive).

**Parameters:**

*n* The maximum unsigned value desired (optional).

**Returns:**

A random number

**Examples:**

[ex\\_ArrayMax.nxc](#), [ex\\_CircleOut.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_Random.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), [ex\\_string.nxc](#), [ex\\_SysDrawEllipse.nxc](#), and [ex\\_wait.nxc](#).

**6.72.2.11 long srand (long seed) [inline]**

Seed the random number generator. Provide the random number generator with a new seed value.

**Parameters:**

*seed* The new random number generator seed. A value of zero causes the seed to be based on the current time value. A value less than zero causes the seed to be restored to the last specified seed.

**Returns:**

The new seed value (useful if you pass in 0 or -1).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Examples:**

[ex\\_srand.nxc](#).

**6.72.2.12 float strtod (const string & str, string & endptr) [inline]**

Convert string to float. Parses the string *str* interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many

characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. A string containing the rest of the string after the last valid character is stored in `endptr`.

A valid floating point number for `atof` is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in `str` does not form a valid floating-point number as just defined, or if no such sequence exists because either `str` is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of a floating-point number.

*endptr* Reference to a string, whose value is set by the function to the remaining characters in `str` after the numerical value.

**Returns:**

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

**Examples:**

[ex\\_strtod.nxc](#).

**6.72.2.13 `long strtol (const string & str, string & endptr, int base = 10) [inline]`**

Convert string to long integer. Parses the C string `str` interpreting its content as an integral number of the specified base, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in `str` is stored in `endptr`.

If the first sequence of non-whitespace characters in `str` does not form a valid integral number, or if no such sequence exists because either `str` is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

- str* String beginning with the representation of an integral number.
- endptr* Reference to a string, whose value is set by the function to the remaining characters in *str* after the numerical value.
- base* Optional and ignored if specified.

**Returns:**

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

**Warning:**

Only `base = 10` is currently supported.

**Examples:**

[ex\\_strtol.nxc](#).

**6.72.2.14 `long strtoul (const string & str, string & endptr, int base = 10)`  
[inline]**

Convert string to unsigned long integer. Parses the C string *str* interpreting its content as an unsigned integral number of the specified base, which is returned as an unsigned long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in *str* is stored in *endptr*.

If the first sequence of non-whitespace characters in *str* does not form a valid integral number, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

- str* String containing the representation of an unsigned integral number.
- endptr* Reference to a string, whose value is set by the function to the remaining characters in *str* after the numerical value.
- base* Optional and ignored if specified.

**Returns:**

On success, the function returns the converted integral number as an unsigned long int value. If no valid conversion could be performed a zero value is returned.

**Warning:**

Only base = 10 is currently supported.

**Examples:**

[ex\\_strtoul.nxc](#).

**6.72.2.15 void SysRandomEx (RandomExType & args) [inline]**

Call the enhanced random number function. This function lets you either obtain a random number or seed the random number generator via the [RandomExType](#) structure.

**Parameters:**

*args* The [RandomExType](#) structure for passing inputs and receiving output values.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Examples:**

[ex\\_sysrandomex.nxc](#).

**6.72.2.16 void SysRandomNumber (RandomNumberType & args) [inline]**

Draw a random number. This function lets you obtain a random number via the [RandomNumberType](#) structure.

**Parameters:**

*args* The [RandomNumberType](#) structure receiving results.

**Examples:**

[ex\\_sysrandomnumber.nxc](#).

## 6.73 cstdlib API types

Standard C cstdlib API types.

### Data Structures

- struct [RandomNumberType](#)  
*Parameters for the RandomNumber system call.*
- struct [RandomExType](#)  
*Parameters for the RandomEx system call.*
- struct [div\\_t](#)  
*Output type of the div function.*
- struct [ldiv\\_t](#)  
*Output type of the ldiv function.*

### 6.73.1 Detailed Description

Standard C cstdlib API types.

## 6.74 cstring API

Standard C cstring API functions.

### Functions

- variant [StrToNum](#) (string str)  
*Convert string to number.*
- unsigned int [StrLen](#) (string str)  
*Get string length.*
- byte [StrIndex](#) (string str, unsigned int idx)  
*Extract a character from a string.*
- string [NumToStr](#) (variant num)  
*Convert number to string.*

- string **StrCat** (string str1, string str2, string strN)  
*Concatenate strings.*
- string **SubStr** (string str, unsigned int idx, unsigned int len)  
*Extract a portion of a string.*
- string **Flatten** (variant num)  
*Flatten a number to a string.*
- string **StrReplace** (string str, unsigned int idx, string strnew)  
*Replace a portion of a string.*
- string **FormatNum** (string fmt, variant num)  
*Format a number.*
- string **FlattenVar** (variant x)  
*Flatten any data to a string.*
- int **UnflattenVar** (string str, variant &x)  
*Unflatten a string into a data type.*
- int **Pos** (string Substr, string S)  
*Find substring position.*
- string **ByteArrayToStr** (byte data[ ])  
*Convert a byte array to a string.*
- void **ByteArrayToStrEx** (byte data[ ], string &str)  
*Convert a byte array to a string.*
- void **StrToByteArray** (string str, byte &data[ ])  
*Convert a string to a byte array.*
- string **Copy** (string str, unsigned int idx, unsigned int len)  
*Copy a portion of a string.*
- string **MidStr** (string str, unsigned int idx, unsigned int len)  
*Copy a portion from the middle of a string.*
- string **RightStr** (string str, unsigned int size)  
*Copy a portion from the end of a string.*

- string [LeftStr](#) (string str, unsigned int size)  
*Copy a portion from the start of a string.*
- int [strlen](#) (const string &str)  
*Get string length.*
- string [strcat](#) (string &dest, const string &src)  
*Concatenate strings.*
- string [strncat](#) (string &dest, const string &src, unsigned int num)  
*Append characters from string.*
- string [strcpy](#) (string &dest, const string &src)  
*Copy string.*
- string [strncpy](#) (string &dest, const string &src, unsigned int num)  
*Copy characters from string.*
- int [strcmp](#) (const string &str1, const string &str2)  
*Compare two strings.*
- int [strncmp](#) (const string &str1, const string &str2, unsigned int num)  
*Compare characters of two strings.*
- void [memcpy](#) (variant dest, variant src, byte num)  
*Copy memory.*
- void [memmove](#) (variant dest, variant src, byte num)  
*Move memory.*
- char [memcmp](#) (variant ptr1, variant ptr2, byte num)  
*Compare two blocks of memory.*
- unsigned long [addressOf](#) (variant data)  
*Get the absolute address of a variable.*
- unsigned long [reladdressOf](#) (variant data)  
*Get the relative address of a variable.*
- unsigned long [addressOfEx](#) (variant data, bool relative)  
*Get the absolute or relative address of a variable.*

### 6.74.1 Detailed Description

Standard C cstring API functions.

### 6.74.2 Function Documentation

#### 6.74.2.1 unsigned long addressOf (variant *data*) [inline]

Get the absolute address of a variable. Get the absolute address of a variable and return it to the calling routine as an unsigned long value.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*data* A variable whose address you wish to get.

**Returns:**

The absolute address of the variable.

**Examples:**

[ex\\_addressof.nxc](#).

#### 6.74.2.2 unsigned long addressOfEx (variant *data*, bool *relative*) [inline]

Get the absolute or relative address of a variable. Get the absolute or relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*data* A variable whose address you wish to get.

*relative* A boolean flag indicating whether you want to get the relative or absolute address.

**Returns:**

The absolute or relative address of the variable.

**Examples:**

[ex\\_addressofex.nxc](#).

**6.74.2.3 string ByteArrayToStr (byte *data*[]) [inline]**

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

**See also:**

[StrToByteArray](#), [ByteArrayToStrEx](#)

**Parameters:**

*data* A byte array.

**Returns:**

A string containing data and a null terminator byte.

**Examples:**

[ex\\_ByteArrayToStr.nxc](#), and [ex\\_string.nxc](#).

**6.74.2.4 void ByteArrayToStrEx (byte *data*[], string & *str*) [inline]**

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

**See also:**

[StrToByteArray](#), [ByteArrayToStr](#)

**Parameters:**

*data* A byte array.

*str* A string variable reference which, on output, will contain data and a null terminator byte.

**Examples:**

[ex\\_ByteArrayToStrEx.nxc](#), and [ex\\_string.nxc](#).

**6.74.2.5 `string Copy (string str, unsigned int idx, unsigned int len) [inline]`**

Copy a portion of a string. Returns a substring of a string.

**Parameters:**

*str* A string  
*idx* The starting index of the substring.  
*len* The length of the substring.

**Returns:**

The specified substring.

**Examples:**

[ex\\_copy.nxc](#).

**6.74.2.6 `string Flatten (variant num) [inline]`**

Flatten a number to a string. Return a string containing the byte representation of the specified value.

**Parameters:**

*num* A number.

**Returns:**

A string containing the byte representation of the parameter *num*.

**Examples:**

[ex\\_Flatten.nxc](#), and [ex\\_string.nxc](#).

### 6.74.2.7 string FlattenVar (variant *x*) [**inline**]

Flatten any data to a string. Return a string containing the byte representation of the specified value.

**See also:**

[UnflattenVar](#)

**Parameters:**

*x* Any NXC datatype.

**Returns:**

A string containing the byte representation of the parameter *x*.

**Examples:**

[ex\\_FlattenVar.nxc](#), [ex\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

### 6.74.2.8 string FormatNum (string *fmt*, variant *num*) [**inline**]

Format a number. Return the formatted string using the format and value. Use a standard numeric sprintf format specifier within the format string. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*fmt* The string format containing a sprintf numeric format specifier.

*num* A number.

**Returns:**

A string containing the formatted numeric value.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acos.nxc](#), [ex\\_acosd.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_asin.nxc](#), [ex\\_asind.nxc](#), [ex\\_atan.nxc](#), [ex\\_atan2.nxc](#), [ex\\_atan2d.nxc](#),

[ex\\_atand.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_FormatNum.nxc](#), [ex\\_GetBrickDataAddress.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_string.nxc](#), [ex\\_tan.nxc](#), [ex\\_tand.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

#### 6.74.2.9 string LeftStr (string *str*, unsigned int *size*) [inline]

Copy a portion from the start of a string. Returns the substring of a specified length that appears at the start of a string.

##### Parameters:

*str* A string  
*size* The size or length of the substring.

##### Returns:

The substring of a specified length that appears at the start of a string.

##### Examples:

[ex\\_leftstr.nxc](#).

#### 6.74.2.10 char memcmp (variant *ptr1*, variant *ptr2*, byte *num*) [inline]

Compare two blocks of memory. Compares the variant *ptr1* to the variant *ptr2*. Returns an integral value indicating the relationship between the variables. The *num* argument is ignored.

##### Parameters:

*ptr1* A variable to be compared.  
*ptr2* A variable to be compared.  
*num* The number of bytes to compare (ignored).

##### Examples:

[ex\\_memcmp.nxc](#).

**6.74.2.11 void memcpy (variant *dest*, variant *src*, byte *num*) [inline]**

Copy memory. Copies memory contents from the source to the destination. The *num* argument is ignored.

**Parameters:**

- dest* The destination variable.
- src* The source variable.
- num* The number of bytes to copy (ignored).

**Examples:**

[ex\\_memcpy.nxc](#).

**6.74.2.12 void memmove (variant *dest*, variant *src*, byte *num*) [inline]**

Move memory. Moves memory contents from the source to the destination. The *num* argument is ignored.

**Parameters:**

- dest* The destination variable.
- src* The source variable.
- num* The number of bytes to copy (ignored).

**Examples:**

[ex\\_memmove.nxc](#).

**6.74.2.13 string MidStr (string *str*, unsigned int *idx*, unsigned int *len*) [inline]**

Copy a portion from the middle of a string. Returns the substring of a specified length that appears at a specified position in a string.

**Parameters:**

- str* A string

*idx* The starting index of the substring.

*len* The length of the substring.

**Returns:**

The substring of a specified length that appears at a specified position in a string.

**Examples:**

[ex\\_midstr.nxc](#).

**6.74.2.14 string NumToStr (variant *num*) [inline]**

Convert number to string. Return the string representation of the specified numeric value.

**Parameters:**

*num* A number.

**Returns:**

The string representation of the parameter *num*.

**Examples:**

[ex\\_NumToStr.nxc](#), [ex\\_RS485Send.nxc](#), and [ex\\_string.nxc](#).

**6.74.2.15 int Pos (string *Substr*, string *S*) [inline]**

Find substring position. Returns the index value of the first character in a specified substring that occurs in a given string. *Pos* searches for *Substr* within *S* and returns an integer value that is the index of the first character of *Substr* within *S*. *Pos* is case-sensitive. If *Substr* is not found, *Pos* returns negative one.

**Parameters:**

*Substr* A substring to search for in another string.

*S* A string that might contain the specified substring.

**Returns:**

The position of the substring in the specified string or -1 if it is not found.

**Examples:**

[ex\\_Pos.nxc](#).

**6.74.2.16 unsigned long reladdressOf (variant *data*) [inline]**

Get the relative address of a variable. Get the relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*data* A variable whose address you wish to get.

**Returns:**

The relative address of the variable.

**Examples:**

[ex\\_reladdressof.nxc](#).

**6.74.2.17 string RightStr (string *str*, unsigned int *size*) [inline]**

Copy a portion from the end of a string. Returns the substring of a specified length that appears at the end of a string.

**Parameters:**

*str* A string

*size* The size or length of the substring.

**Returns:**

The substring of a specified length that appears at the end of a string.

**Examples:**

[ex\\_rightstr.nxc](#).

**6.74.2.18 string strcat (string & dest, const string & src) [inline]**

Concatenate strings. Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_StrCat.nxc](#).

**6.74.2.19 string StrCat (string str1, string str2, string strN) [inline]**

Concatenate strings. Return a string which is the result of concatenating all of the string arguments together. This function accepts any number of parameters which may be string variables, constants, or expressions.

**Parameters:**

*str1* The first string.

*str2* The second string.

*strN* The Nth string.

**Returns:**

The concatenated string.

**Examples:**

[ex\\_GetBrickDataAddress.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_-StrReplace.nxc](#), and [util\\_battery\\_1.nxc](#).

**6.74.2.20 int strcmp (const string & *str1*, const string & *str2*) [inline]**

Compare two strings. Compares the string *str1* to the string *str2*.

**Parameters:**

*str1* A string to be compared.

*str2* A string to be compared.

**Returns:**

Returns an integral value indicating the relationship between the strings. A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in *str1* than in *str2*. A value less than zero indicates the opposite.

**Examples:**

[ex\\_strcmp.nxc](#).

**6.74.2.21 string strcpy (string & *dest*, const string & *src*) [inline]**

Copy string. Copies the string pointed by source into the array pointed by destination, including the terminating null character. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strcpy.nxc](#).

**6.74.2.22 byte StrIndex (string *str*, unsigned int *idx*) [inline]**

Extract a character from a string. Return the numeric value of the character in the specified string at the specified index. The input string parameter may be a variable, constant, or expression.

**Parameters:**

- str* A string.
- idx* The index of the character to retrieve.

**Returns:**

The numeric value of the character at the specified index.

**Examples:**

[ex\\_StrIndex.nxc](#), and [ex\\_string.nxc](#).

**6.74.2.23 `int strlen (const string & str) [inline]`**

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string.

**Parameters:**

- str* A string.

**Returns:**

The length of the string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrLen.nxc](#).

**6.74.2.24 `unsigned int StrLen (string str) [inline]`**

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string. The input string parameter may be a variable, constant, or expression.

**Parameters:**

- str* A string.

**Returns:**

The length of the string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrLenOld.nxc](#).

**6.74.2.25 string strncat (string & dest, const string & src, unsigned int num) [inline]**

Append characters from string. Appends the first num characters of source to destination, plus a terminating null-character. If the length of the string in source is less than num, only the content up to the terminating null-character is copied. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

*num* The maximum number of characters to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strncat.nxc](#).

**6.74.2.26 int strncmp (const string & str1, const string & str2, unsigned int num) [inline]**

Compare characters of two strings. Compares up to num characters of the string str1 to those of the string str2.

**Parameters:**

*str1* A string to be compared.

*str2* A string to be compared.

*num* The maximum number of characters to be compared.

**Returns:**

Returns an integral value indicating the relationship between the strings. A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in `str1` than in `str2`. A value less than zero indicates the opposite.

**Examples:**

[ex\\_strncmp.nxc](#).

**6.74.2.27 `string strncpy (string & dest, const string & src, unsigned int num)`  
[inline]**

Copy characters from `string`. Copies the first `num` characters of source to destination. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

*num* The maximum number of characters to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strncpy.nxc](#).

**6.74.2.28 `string StrReplace (string str, unsigned int idx, string strnew)`  
[inline]**

Replace a portion of a string. Return a string with the part of the string replaced (starting at the specified index) with the contents of the new string value provided in the third argument. The input string parameters may be variables, constants, or expressions.

**Parameters:**

*str* A string.

*idx* The starting point for the replace operation.

*strnew* The replacement string.

**Returns:**

The modified string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrReplace.nxc](#).

**6.74.2.29** `void StrToByteArray (string str, byte & data[ ]) [inline]`

Convert a string to a byte array. Convert the specified string to an array of byte by removing the null terminator at the end of the string. The output array variable must be a one-dimensional array of byte.

**See also:**

[ByteArrayToStr](#), [ByteArrayToStrEx](#)

**Parameters:**

*str* A string

*data* A byte array reference which, on output, will contain *str* without its null terminator.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrToByteArray.nxc](#).

**6.74.2.30** `variant StrToNum (string str) [inline]`

Convert string to number. Return the numeric value specified by the string passed to the function. If the content of the string is not a numeric value then this function returns zero. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* String beginning with the representation of a number.

*str* A string.

**Returns:**

A number.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrToNum.nxc](#).

**6.74.2.31 string SubStr (string *str*, unsigned int *idx*, unsigned int *len*) [inline]**

Extract a portion of a string. Return a sub-string from the specified input string starting at *idx* and including the specified number of characters. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* A string.

*idx* The starting point of the sub-string.

*len* The length of the sub-string.

**Returns:**

The sub-string extracted from parameter *str*.

**Examples:**

[ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), and [ex\\_SubStr.nxc](#).

**6.74.2.32 int UnflattenVar (string *str*, variant & *x*) [inline]**

Unflatten a string into a data type. Convert a string containing the byte representation of the specified variable back into the original variable type.

**See also:**

[FlattenVar](#), [Flatten](#)

**Parameters:**

*str* A string containing flattened data.

*x* A variable reference where the unflattened data is stored.

**Returns:**

A boolean value indicating whether the operation succeeded or not.

**Examples:**

[ex\\_FlattenVar.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

## 6.75 ctype API

Standard C ctype API functions.

**Functions**

- int [isupper](#) (int c)  
*Check if character is uppercase letter.*
- int [islower](#) (int c)  
*Check if character is lowercase letter.*
- int [isalpha](#) (int c)  
*Check if character is alphabetic.*
- int [isdigit](#) (int c)  
*Check if character is decimal digit.*
- int [isalnum](#) (int c)  
*Check if character is alphanumeric.*
- int [isspace](#) (int c)  
*Check if character is a white-space.*
- int [isctrl](#) (int c)  
*Check if character is a control character.*
- int [isprint](#) (int c)  
*Check if character is printable.*
- int [isgraph](#) (int c)  
*Check if character has graphical representation.*
- int [ispunct](#) (int c)

*Check if character is a punctuation.*

- `int isxdigit (int c)`  
*Check if character is hexadecimal digit.*
- `int toupper (int c)`  
*Convert lowercase letter to uppercase.*
- `int tolower (int c)`  
*Convert uppercase letter to lowercase.*

### 6.75.1 Detailed Description

Standard C ctype API functions.

### 6.75.2 Function Documentation

#### 6.75.2.1 `int isalnum (int c) [inline]`

Check if character is alphanumeric. Checks if parameter `c` is either a decimal digit or an uppercase or lowercase letter. The result is true if either `isalpha` or `isdigit` would also return true.

#### Parameters:

`c` Character to be checked.

#### Returns:

Returns a non-zero value (true) if `c` is either a digit or a letter, otherwise it returns 0 (false).

#### Examples:

[ex\\_ctype.nxc](#), and [ex\\_isalnum.nxc](#).

#### 6.75.2.2 `int isalpha (int c) [inline]`

Check if character is alphabetic. Checks if parameter `c` is either an uppercase or lowercase letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isalpha.nxc](#).

**6.75.2.3 int iscntrl (int *c*) [inline]**

Check if character is a control character. Checks if parameter *c* is a control character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a control character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_iscntrl.nxc](#).

**6.75.2.4 int isdigit (int *c*) [inline]**

Check if character is decimal digit. Checks if parameter *c* is a decimal digit character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a decimal digit, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isdigit.nxc](#).

### 6.75.2.5 int isgraph (int *c*) [**inline**]

Check if character has graphical representation. Checks if parameter *c* is a character with a graphical representation.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* has a graphical representation, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isgraph.nxc](#).

### 6.75.2.6 int islower (int *c*) [**inline**]

Check if character is lowercase letter. Checks if parameter *c* is an lowercase alphabetic letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an lowercase alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_islower.nxc](#).

### 6.75.2.7 int isprint (int *c*) [**inline**]

Check if character is printable. Checks if parameter *c* is a printable character (i.e., not a control character).

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a printable character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isprint.nxc](#).

**6.75.2.8 int ispunct (int *c*) [inline]**

Check if character is a punctuation. Checks if parameter *c* is a punctuation character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a punctuation character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_ispunct.nxc](#).

**6.75.2.9 int isspace (int *c*) [inline]**

Check if character is a white-space. Checks if parameter *c* is a white-space character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a white-space character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isspace.nxc](#).

### 6.75.2.10 int isupper (int c) [inline]

Check if character is uppercase letter. Checks if parameter *c* is an uppercase alphabetic letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an uppercase alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isupper.nxc](#).

### 6.75.2.11 int isxdigit (int c) [inline]

Check if character is hexadecimal digit. Checks if parameter *c* is a hexadecimal digit character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a hexadecimal digit character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isxdigit.nxc](#).

### 6.75.2.12 int tolower (int c) [inline]

Convert uppercase letter to lowercase. Converts parameter *c* to its lowercase equivalent if *c* is an uppercase letter and has a lowercase equivalent. If no such conversion is possible, the value returned is *c* unchanged.

**Parameters:**

*c* Uppercase letter character to be converted.

**Returns:**

The lowercase equivalent to *c*, if such value exists, or *c* (unchanged) otherwise..

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_tolower.nxc](#).

**6.75.2.13 int toupper (int *c*) [inline]**

Convert lowercase letter to uppercase. Converts parameter *c* to its uppercase equivalent if *c* is a lowercase letter and has an uppercase equivalent. If no such conversion is possible, the value returned is *c* unchanged.

**Parameters:**

*c* Lowercase letter character to be converted.

**Returns:**

The uppercase equivalent to *c*, if such value exists, or *c* (unchanged) otherwise..

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_toupper.nxc](#).

**6.76 Property constants**

Use these constants for specifying the property for the SetProperty and SetProperty direct commands.

**Defines**

- #define [RC\\_PROP\\_BTONOFF](#) 0x0
- #define [RC\\_PROP\\_SOUND\\_LEVEL](#) 0x1
- #define [RC\\_PROP\\_SLEEP\\_TIMEOUT](#) 0x2
- #define [RC\\_PROP\\_DEBUGGING](#) 0xF

### 6.76.1 Detailed Description

Use these constants for specifying the property for the GetProperty and SetProperty direct commands.

### 6.76.2 Define Documentation

#### 6.76.2.1 #define RC\_PROP\_BTONOFF 0x0

Set/get whether bluetooth is on or off

#### 6.76.2.2 #define RC\_PROP\_DEBUGGING 0xF

Set/get enhanced firmware debugging information

#### 6.76.2.3 #define RC\_PROP\_SLEEP\_TIMEOUT 0x2

Set/get the NXT sleep timeout value (times 60000)

#### 6.76.2.4 #define RC\_PROP\_SOUND\_LEVEL 0x1

Set/get the NXT sound level

### Examples:

[ex\\_RemoteGetProperty.nxc](#), and [ex\\_RemoteSetProperty.nxc](#).

## 6.77 Array operation constants

Constants for use with the NXC ArrayOp function and the NBC arrop statement.

### Defines

- #define [OPARR\\_SUM](#) 0x00
- #define [OPARR\\_MEAN](#) 0x01
- #define [OPARR\\_SUMSQR](#) 0x02
- #define [OPARR\\_STD](#) 0x03
- #define [OPARR\\_MIN](#) 0x04
- #define [OPARR\\_MAX](#) 0x05
- #define [OPARR\\_SORT](#) 0x06

### 6.77.1 Detailed Description

Constants for use with the NXC ArrayOp function and the NBC arrop statement.

### 6.77.2 Define Documentation

#### 6.77.2.1 #define OPARR\_MAX 0x05

Calculate the maximum value of the elements in the numeric input array

#### Examples:

[ex\\_ArrayOp.nxc](#).

#### 6.77.2.2 #define OPARR\_MEAN 0x01

Calculate the mean value for the elements in the numeric input array

#### 6.77.2.3 #define OPARR\_MIN 0x04

Calculate the minimum value of the elements in the numeric input array

#### 6.77.2.4 #define OPARR\_SORT 0x06

Sort the elements in the numeric input array

#### 6.77.2.5 #define OPARR\_STD 0x03

Calculate the standard deviation of the elements in the numeric input array

#### 6.77.2.6 #define OPARR\_SUM 0x00

Calculate the sum of the elements in the numeric input array

#### 6.77.2.7 #define OPARR\_SUMSQR 0x02

Calculate the sum of the squares of the elements in the numeric input array

## 6.78 System Call function constants

Constants for use in the `SysCall()` function or NBC syscall statement.

### Defines

- #define `FileOpenRead` 0
- #define `FileOpenWrite` 1
- #define `FileOpenAppend` 2
- #define `FileRead` 3
- #define `FileWrite` 4
- #define `FileClose` 5
- #define `FileResolveHandle` 6
- #define `FileRename` 7
- #define `FileDelete` 8
- #define `SoundPlayFile` 9
- #define `SoundPlayTone` 10
- #define `SoundGetState` 11
- #define `SoundSetState` 12
- #define `DrawText` 13
- #define `DrawPoint` 14
- #define `DrawLine` 15
- #define `DrawCircle` 16
- #define `DrawRect` 17
- #define `DrawGraphic` 18
- #define `SetScreenMode` 19
- #define `ReadButton` 20
- #define `CommLSWrite` 21
- #define `CommLSRead` 22
- #define `CommLSCheckStatus` 23
- #define `RandomNumber` 24
- #define `GetStartTick` 25
- #define `MessageWrite` 26
- #define `MessageRead` 27
- #define `CommBTCheckStatus` 28
- #define `CommBTWrite` 29
- #define `CommBTRead` 30
- #define `KeepAlive` 31
- #define `IOMapRead` 32
- #define `IOMapWrite` 33
- #define `ColorSensorRead` 34
- #define `CommBTONOff` 35

- #define [CommBTConnection](#) 36
- #define [CommHSWrite](#) 37
- #define [CommHSRead](#) 38
- #define [CommHSCheckStatus](#) 39
- #define [ReadSemData](#) 40
- #define [WriteSemData](#) 41
- #define [ComputeCalibValue](#) 42
- #define [UpdateCalibCacheInfo](#) 43
- #define [DatalogWrite](#) 44
- #define [DatalogGetTimes](#) 45
- #define [SetSleepTimeoutVal](#) 46
- #define [ListFiles](#) 47
- #define [InputPinFunction](#) 77
- #define [IOMapReadByID](#) 78
- #define [IOMapWriteByID](#) 79
- #define [DisplayExecuteFunction](#) 80
- #define [CommExecuteFunction](#) 81
- #define [LoaderExecuteFunction](#) 82
- #define [FileFindFirst](#) 83
- #define [FileFindNext](#) 84
- #define [FileOpenWriteLinear](#) 85
- #define [FileOpenWriteNonLinear](#) 86
- #define [FileOpenReadLinear](#) 87
- #define [CommHSControl](#) 88
- #define [CommLSWriteEx](#) 89
- #define [FileSeek](#) 90
- #define [FileResize](#) 91
- #define [DrawGraphicArray](#) 92
- #define [DrawPolygon](#) 93
- #define [DrawEllipse](#) 94
- #define [DrawFont](#) 95
- #define [MemoryManager](#) 96
- #define [ReadLastResponse](#) 97
- #define [FileTell](#) 98
- #define [RandomEx](#) 99

### 6.78.1 Detailed Description

Constants for use in the [SysCall\(\)](#) function or NBC syscall statement.

**6.78.2 Define Documentation**

**6.78.2.1 #define ColorSensorRead 34**

Read data from the NXT 2.0 color sensor

**6.78.2.2 #define CommBTCheckStatus 28**

Check the bluetooth status

**6.78.2.3 #define CommBTConnection 36**

Connect or disconnect to a known bluetooth device

**6.78.2.4 #define CommBTONOff 35**

Turn the bluetooth radio on or off

**6.78.2.5 #define CommBTRead 30**

Read from a bluetooth connection

**6.78.2.6 #define CommBTWrite 29**

Write to a bluetooth connections

**6.78.2.7 #define CommExecuteFunction 81**

Execute one of the Comm module's internal functions

**6.78.2.8 #define CommHSCheckStatus 39**

Check the status of the hi-speed port

**6.78.2.9 #define CommHSControl 88**

Control the hi-speed port

**6.78.2.10 #define CommHSRead 38**

Read data from the hi-speed port

**6.78.2.11 #define CommHSWrite 37**

Write data to the hi-speed port

**6.78.2.12 #define CommLSCheckStatus 23**

Check the status of a lowspeed (aka I2C) device

**6.78.2.13 #define CommLSRead 22**

Read from a lowspeed (aka I2C) device

**6.78.2.14 #define CommLSWrite 21**

Write to a lowspeed (aka I2C) device

**6.78.2.15 #define CommLSWriteEx 89**

Write to a lowspeed (aka I2C) device with optional restart on read

**6.78.2.16 #define ComputeCalibValue 42**

Compute a calibration value

**6.78.2.17 #define DatalogGetTimes 45**

Get datalog timing information

**6.78.2.18 #define DatalogWrite 44**

Write to the datalog

**6.78.2.19 #define DisplayExecuteFunction 80**

Execute one of the Display module's internal functions

**6.78.2.20 #define DrawCircle 16**

Draw a circle on the LCD screen

**6.78.2.21 #define DrawEllipse 94**

Draw an ellipse on the LCD screen

**6.78.2.22 #define DrawFont 95**

Draw text using a custom RIC-based font to the LCD screen

**6.78.2.23 #define DrawGraphic 18**

Draw a graphic image on the LCD screen

**6.78.2.24 #define DrawGraphicArray 92**

Draw a graphic image from a byte array to the LCD screen

**Examples:**

[ex\\_dispgout.nxc](#).

**6.78.2.25 #define DrawLine 15**

Draw a line on the LCD screen

**6.78.2.26 #define DrawPoint 14**

Draw a single pixel on the LCD screen

**6.78.2.27 #define DrawPolygon 93**

Draw a polygon on the LCD screen

**6.78.2.28 #define DrawRect 17**

Draw a rectangle on the LCD screen

**6.78.2.29 #define DrawText 13**

Draw text to one of 8 LCD lines

**Examples:**

[ex\\_syscall.nxc](#).

**6.78.2.30 #define FileClose 5**

Close the specified file

**6.78.2.31 #define FileDelete 8**

Delete a file

**6.78.2.32 #define FileFindFirst 83**

Start a search for a file using a filename pattern

**6.78.2.33 #define FileFindNext 84**

Continue searching for a file

**6.78.2.34 #define FileOpenAppend 2**

Open a file for appending to the end of the file

**6.78.2.35 #define FileOpenRead 0**

Open a file for reading

**6.78.2.36 #define FileOpenReadLinear 87**

Open a linear file for reading

**6.78.2.37 #define FileOpenWrite 1**

Open a file for writing (creates a new file)

**6.78.2.38 #define FileOpenWriteLinear 85**

Open a linear file for writing

**6.78.2.39 #define FileOpenWriteNonLinear 86**

Open a non-linear file for writing

**6.78.2.40 #define FileRead 3**

Read from the specified file

**6.78.2.41 #define FileRename 7**

Rename a file

**6.78.2.42 #define FileResize 91**

Resize a file (not yet implemented)

**6.78.2.43 #define FileResolveHandle 6**

Get a file handle for the specified filename if it is already open

**6.78.2.44 #define FileSeek 90**

Seek to a specific position in an open file

**6.78.2.45 #define FileTell 98**

Return the current file position in an open file

**6.78.2.46 #define FileWrite 4**

Write to the specified file

**6.78.2.47 #define GetStartTick 25**

Get the current system tick count

**6.78.2.48 #define InputPinFunction 77**

Execute the Input module's pin function

**6.78.2.49 #define IOMapRead 32**

Read data from one of the firmware module's IOMap structures using the module's name

**6.78.2.50 #define IOMapReadByID 78**

Read data from one of the firmware module's IOMap structures using the module's ID

**6.78.2.51 #define IOMapWrite 33**

Write data to one of the firmware module's IOMap structures using the module's name

**6.78.2.52 #define IOMapWriteByID 79**

Write data to one of the firmware module's IOMap structures using the module's ID

**6.78.2.53 #define KeepAlive 31**

Reset the NXT sleep timer

**6.78.2.54 #define ListFiles 47**

List files that match the specified filename pattern

**6.78.2.55 #define LoaderExecuteFunction 82**

Execute one of the Loader module's internal functions

**6.78.2.56 #define MemoryManager 96**

Read memory manager information, optionally compacting the dataspace first

**6.78.2.57 #define MessageRead 27**

Read a message from a mailbox

**6.78.2.58 #define MessageWrite 26**

Write a message to a mailbox

**6.78.2.59 #define RandomEx 99**

Generate a random number or seed the RNG.

**6.78.2.60 #define RandomNumber 24**

Generate a random number

**6.78.2.61 #define ReadButton 20**

Read the current button state

**6.78.2.62 #define ReadLastResponse 97**

Read the last response packet received by the NXT. Optionally clear the value after reading it.

**6.78.2.63 #define ReadSemData 40**

Read motor semaphore data

**6.78.2.64 #define SetScreenMode 19**

Set the screen mode

**6.78.2.65 #define SetSleepTimeoutVal 46**

Set the NXT sleep timeout value

**6.78.2.66 #define SoundGetState 11**

Get the current sound module state

**6.78.2.67 #define SoundPlayFile 9**

Play a sound or melody file

**6.78.2.68 #define SoundPlayTone 10**

Play a simple tone with the specified frequency and duration

**6.78.2.69 #define SoundSetState 12**

Set the sound module state

**6.78.2.70 #define UpdateCalibCacheInfo 43**

Update sensor calibration cache information

**6.78.2.71 #define WriteSemData 41**

Write motor semaphore data

**6.79 Line number constants**

Line numbers for use with DrawText system function.

**Defines**

- #define [LCD\\_LINE8](#) 0
- #define [LCD\\_LINE7](#) 8
- #define [LCD\\_LINE6](#) 16
- #define [LCD\\_LINE5](#) 24
- #define [LCD\\_LINE4](#) 32
- #define [LCD\\_LINE3](#) 40
- #define [LCD\\_LINE2](#) 48
- #define [LCD\\_LINE1](#) 56

**6.79.1 Detailed Description**

Line numbers for use with DrawText system function.

**See also:**

[SysDrawText\(\)](#), [TextOut\(\)](#), [NumOut\(\)](#)

**6.79.2 Define Documentation****6.79.2.1 #define LCD\_LINE1 56**

The 1st line of the LCD screen

**Examples:**

ex\_acos.nxc, ex\_acosd.nxc, ex\_addressof.nxc, ex\_addressofex.nxc, ex\_ArrayMax.nxc, ex\_ArrayMean.nxc, ex\_ArrayMin.nxc, ex\_ArrayOp.nxc, ex\_ArraySort.nxc, ex\_ArrayStd.nxc, ex\_ArraySum.nxc, ex\_ArraySumSqr.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc, ex\_atand.nxc, ex\_atof.nxc, ex\_atoi.nxc, ex\_atol.nxc, ex\_buttonpressed.nxc, ex\_clearline.nxc, ex\_contrast.nxc, ex\_copy.nxc, ex\_ctype.nxc, ex\_DataMode.nxc, ex\_delete\_data\_file.nxc, ex\_diaccl.nxc, ex\_digps.nxc, ex\_digyro.nxc, ex\_dispgaout.nxc, ex\_dispgout.nxc, ex\_displayfont.nxc, ex\_dispmisc.nxc, ex\_div.nxc, ex\_file\_system.nxc, ex\_findfirstfile.nxc, ex\_findnextfile.nxc, ex\_FlattenVar.nxc, ex\_GetBrickDataAddress.nxc, ex\_getchar.nxc, ex\_getmemoryinfo.nxc, ex\_HTGyroTest.nxc, ex\_i2cdeviceid.nxc, ex\_i2cdeviceinfo.nxc, ex\_i2cvendorid.nxc, ex\_i2cversion.nxc, ex\_isnan.nxc, ex\_joystickmsg.nxc, ex\_labs.nxc, ex\_ldiv.nxc, ex\_leftstr.nxc, ex\_memcmp.nxc, ex\_midstr.nxc, ex\_motoroutputoptions.nxc, ex\_NumOut.nxc, ex\_NXTLineLeader.nxc, ex\_NXTPowerMeter.nxc, ex\_NXTServo.nxc, ex\_NXTSumoEyes.nxc, ex\_Pos.nxc, ex\_proto.nxc, ex\_ReadSensorHTAngle.nxc, ex\_ReadSensorHTBarometric.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_rightstr.nxc, ex\_RS485Receive.nxc, ex\_RS485Send.nxc, ex\_SensorHTGyro.nxc, ex\_SetAbortFlag.nxc, ex\_setdisplayfont.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_string.nxc, ex\_strtod.nxc, ex\_strtol.nxc, ex\_strtoul.nxc, ex\_superpro.nxc, ex\_syscall.nxc, ex\_SysColorSensorRead.nxc, ex\_syscommbtconnection.nxc, ex\_SysCommBTONOff.nxc, ex\_SysCommHSControl.nxc, ex\_SysCommHSRead.nxc, ex\_SysCommHSCheckStatus.nxc, ex\_SysComputeCalibValue.nxc, ex\_SysDatalogWrite.nxc, ex\_sysdrawtext.nxc, ex\_sysfilefindfirst.nxc, ex\_sysfilefindnext.nxc, ex\_sysfileread.nxc, ex\_sysfilewrite.nxc, ex\_systemmemorymanager.nxc, ex\_sysmessageread.nxc, ex\_SysReadLastResponse.nxc, ex\_SysReadSemData.nxc, ex\_SysUpdateCalibCacheInfo.nxc, ex\_SysWriteSemData.nxc, ex\_UnflattenVar.nxc, and ex\_xg1300.nxc.

**6.79.2.2 #define LCD\_LINE2 48**

The 2nd line of the LCD screen

**Examples:**

ex\_acos.nxc, ex\_acosd.nxc, ex\_addressof.nxc, ex\_addressofex.nxc, ex\_ArrayMax.nxc, ex\_ArrayMean.nxc, ex\_ArrayMin.nxc, ex\_ArrayOp.nxc,

ex\_ArraySort.nxc, ex\_ArrayStd.nxc, ex\_ArraySum.nxc, ex\_ArraySumSqr.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc, ex\_atand.nxc, ex\_buttonpressed.nxc, ex\_ctype.nxc, ex\_DataMode.nxc, ex\_diaccl.nxc, ex\_digps.nxc, ex\_digyro.nxc, ex\_displayfont.nxc, ex\_dispmisc.nxc, ex\_div.nxc, ex\_file\_system.nxc, ex\_findfirstfile.nxc, ex\_findnextfile.nxc, ex\_FlattenVar.nxc, ex\_getmemoryinfo.nxc, ex\_HTGyroTest.nxc, ex\_i2cdeviceid.nxc, ex\_i2cdeviceinfo.nxc, ex\_i2cvendorid.nxc, ex\_i2cversion.nxc, ex\_isnan.nxc, ex\_joystickmsg.nxc, ex\_labs.nxc, ex\_ldiv.nxc, ex\_memcmp.nxc, ex\_NXTLineLeader.nxc, ex\_NXTPowerMeter.nxc, ex\_NXTServo.nxc, ex\_proto.nxc, ex\_ReadSensorHTAngle.nxc, ex\_ReadSensorHTBarometric.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_SetAbortFlag.nxc, ex\_setdisplayfont.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_string.nxc, ex\_strtod.nxc, ex\_strtol.nxc, ex\_strtoul.nxc, ex\_SubStr.nxc, ex\_superpro.nxc, ex\_syscommbtconnection.nxc, ex\_sysfileread.nxc, ex\_sysmemorymanager.nxc, ex\_SysReadLastResponse.nxc, ex\_UnflattenVar.nxc, ex\_xg1300.nxc, util\_battery\_1.nxc, util\_battery\_2.nxc, and util\_rpm.nxc.

### 6.79.2.3 #define LCD\_LINE3 40

The 3rd line of the LCD screen

#### Examples:

ex\_acos.nxc, ex\_acosd.nxc, ex\_ArraySort.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc, ex\_atand.nxc, ex\_buttonpressed.nxc, ex\_ctype.nxc, ex\_diaccl.nxc, ex\_digps.nxc, ex\_digyro.nxc, ex\_dispmisc.nxc, ex\_findfirstfile.nxc, ex\_findnextfile.nxc, ex\_FlattenVar.nxc, ex\_i2cdeviceid.nxc, ex\_i2cdeviceinfo.nxc, ex\_i2cvendorid.nxc, ex\_i2cversion.nxc, ex\_joystickmsg.nxc, ex\_memcmp.nxc, ex\_NXTLineLeader.nxc, ex\_NXTPowerMeter.nxc, ex\_NXTServo.nxc, ex\_proto.nxc, ex\_ReadSensorHTAngle.nxc, ex\_ReadSensorHTBarometric.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_SetAbortFlag.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_StrCatOld.nxc, ex\_string.nxc, ex\_strtod.nxc, ex\_strtol.nxc, ex\_strtoul.nxc, ex\_superpro.nxc, ex\_syscommbtconnection.nxc, ex\_TextOut.nxc, ex\_UnflattenVar.nxc, and ex\_xg1300.nxc.

### 6.79.2.4 #define LCD\_LINE4 32

The 4th line of the LCD screen

#### Examples:

ex\_acos.nxc, ex\_acosd.nxc, ex\_addressof.nxc, ex\_addressofex.nxc, ex\_ArrayBuild.nxc, ex\_ArraySort.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc,

[ex\\_atand.nxc](#), [ex\\_buttonpressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorHTTouchMultiplexer.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_SizeOf.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_superpro.nxc](#), [ex\\_sysdataloggettimes.nxc](#), and [ex\\_UnflattenVar.nxc](#).

#### 6.79.2.5 #define LCD\_LINE5 24

The 5th line of the LCD screen

##### Examples:

[ex\\_ArrayBuild.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_atan.nxc](#), [ex\\_atand.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_superpro.nxc](#), [ex\\_sysdataloggettimes.nxc](#), and [ex\\_xg1300.nxc](#).

#### 6.79.2.6 #define LCD\_LINE6 16

The 6th line of the LCD screen

##### Examples:

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_proto.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_superpro.nxc](#), [ex\\_syslistfiles.nxc](#), and [ex\\_xg1300.nxc](#).

#### 6.79.2.7 #define LCD\_LINE7 8

The 7th line of the LCD screen

##### Examples:

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_proto.nxc](#), [ex\\_string.nxc](#), [ex\\_superpro.nxc](#), and [ex\\_xg1300.nxc](#).

### 6.79.2.8 #define LCD\_LINE8 0

The 8th line of the LCD screen

#### Examples:

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_proto.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_string.nxc](#), [ex\\_superpro.nxc](#), [ex\\_sysmemorymanager.nxc](#), and [ex\\_xg1300.nxc](#).

## 6.80 Time constants

Constants for use with the [Wait\(\)](#) function.

#### Defines

- #define [MS\\_1](#) 1
- #define [MS\\_2](#) 2
- #define [MS\\_3](#) 3
- #define [MS\\_4](#) 4
- #define [MS\\_5](#) 5
- #define [MS\\_6](#) 6
- #define [MS\\_7](#) 7
- #define [MS\\_8](#) 8
- #define [MS\\_9](#) 9
- #define [MS\\_10](#) 10
- #define [MS\\_20](#) 20
- #define [MS\\_30](#) 30
- #define [MS\\_40](#) 40
- #define [MS\\_50](#) 50
- #define [MS\\_60](#) 60
- #define [MS\\_70](#) 70
- #define [MS\\_80](#) 80
- #define [MS\\_90](#) 90
- #define [MS\\_100](#) 100
- #define [MS\\_150](#) 150
- #define [MS\\_200](#) 200
- #define [MS\\_250](#) 250
- #define [MS\\_300](#) 300
- #define [MS\\_350](#) 350
- #define [MS\\_400](#) 400
- #define [MS\\_450](#) 450

- #define [MS\\_500](#) 500
- #define [MS\\_600](#) 600
- #define [MS\\_700](#) 700
- #define [MS\\_800](#) 800
- #define [MS\\_900](#) 900
- #define [SEC\\_1](#) 1000
- #define [SEC\\_2](#) 2000
- #define [SEC\\_3](#) 3000
- #define [SEC\\_4](#) 4000
- #define [SEC\\_5](#) 5000
- #define [SEC\\_6](#) 6000
- #define [SEC\\_7](#) 7000
- #define [SEC\\_8](#) 8000
- #define [SEC\\_9](#) 9000
- #define [SEC\\_10](#) 10000
- #define [SEC\\_15](#) 15000
- #define [SEC\\_20](#) 20000
- #define [SEC\\_30](#) 30000
- #define [MIN\\_1](#) 60000

### 6.80.1 Detailed Description

Constants for use with the [Wait\(\)](#) function.

See also:

[Wait\(\)](#)

### 6.80.2 Define Documentation

#### 6.80.2.1 #define MIN\_1 60000

1 minute

Examples:

[ex\\_SysSetSleepTimeout.nxc](#).

#### 6.80.2.2 #define MS\_1 1

1 millisecond

Examples:

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**6.80.2.3 #define MS\_10 10**

10 milliseconds

**Examples:**[ex\\_diaccl.nxc](#), and [ex\\_PosReg.nxc](#).**6.80.2.4 #define MS\_100 100**

100 milliseconds

**Examples:**[ex\\_joystickmsg.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_sysdrawpolygon.nxc](#), and [ex\\_xg1300.nxc](#).**6.80.2.5 #define MS\_150 150**

150 milliseconds

**6.80.2.6 #define MS\_2 2**

2 milliseconds

**6.80.2.7 #define MS\_20 20**

20 milliseconds

**Examples:**[ex\\_dispgayout.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosp.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).**6.80.2.8 #define MS\_200 200**

200 milliseconds

**Examples:**[ex\\_dispgoutex.nxc](#), and [ex\\_playtones.nxc](#).

**6.80.2.9 #define MS\_250 250**

250 milliseconds

**6.80.2.10 #define MS\_3 3**

3 milliseconds

**6.80.2.11 #define MS\_30 30**

30 milliseconds

**6.80.2.12 #define MS\_300 300**

300 milliseconds

**6.80.2.13 #define MS\_350 350**

350 milliseconds

**6.80.2.14 #define MS\_4 4**

4 milliseconds

**6.80.2.15 #define MS\_40 40**

40 milliseconds

**6.80.2.16 #define MS\_400 400**

400 milliseconds

**6.80.2.17 #define MS\_450 450**

450 milliseconds

**6.80.2.18 #define MS\_5 5**

5 milliseconds

**Examples:**

[ex\\_getchar.nxc](#).

**6.80.2.19 #define MS\_50 50**

50 milliseconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digyro.nxc](#), and [ex\\_playtones.nxc](#).

**6.80.2.20 #define MS\_500 500**

500 milliseconds

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_playsound.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_xg1300.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

**6.80.2.21 #define MS\_6 6**

6 milliseconds

**6.80.2.22 #define MS\_60 60**

60 milliseconds

**6.80.2.23 #define MS\_600 600**

600 milliseconds

**6.80.2.24 #define MS\_7 7**

7 milliseconds

**6.80.2.25 #define MS\_70 70**

70 milliseconds

**6.80.2.26 #define MS\_700 700**

700 milliseconds

**6.80.2.27 #define MS\_8 8**

8 milliseconds

**6.80.2.28 #define MS\_80 80**

80 milliseconds

**6.80.2.29 #define MS\_800 800**

800 milliseconds

**6.80.2.30 #define MS\_9 9**

9 milliseconds

**6.80.2.31 #define MS\_90 90**

90 milliseconds

**6.80.2.32 #define MS\_900 900**

900 milliseconds

**6.80.2.33 #define SEC\_1 1000**

1 second

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_file\\_system.nxc](#),  
[ex\\_getmemoryinfo.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_-  
playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysCommHSRead.nxc](#),  
[ex\\_sysdrawpolygon.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_wait.nxc](#), and [ex\\_-  
yield.nxc](#).

**6.80.2.34 #define SEC\_10 10000**

10 seconds

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_string.nxc](#), [ex\\_syscommmbtconnection.nxc](#), and [ex\\_SysCommHSCControl.nxc](#).

**6.80.2.35 #define SEC\_15 15000**

15 seconds

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_memcmp.nxc](#).

**6.80.2.36 #define SEC\_2 2000**

2 seconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**6.80.2.37 #define SEC\_20 20000**

20 seconds

**6.80.2.38 #define SEC\_3 3000**

3 seconds

**Examples:**

[ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_div.nxc](#), and [ex\\_ldiv.nxc](#).

**6.80.2.39 #define SEC\_30 30000**

30 seconds

**6.80.2.40 #define SEC\_4 4000**

4 seconds

**Examples:**

[ex\\_copy.nxc](#), [ex\\_dispfout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_midstr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_syslistfiles.nxc](#), [util\\_battery\\_1.nxc](#), and [util\\_battery\\_2.nxc](#).

**6.80.2.41 #define SEC\_5 5000**

5 seconds

**Examples:**

[ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_clearline.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_dispfout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_isnan.nxc](#), [ex\\_labs.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_onfwdsyncpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_PFMate.nxc](#), [ex\\_proto.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_UnflattenVar.nxc](#), and [ex\\_wait.nxc](#).

**6.80.2.42 #define SEC\_6 6000**

6 seconds

**Examples:**

[ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), and [ex\\_strtoul.nxc](#).

**6.80.2.43 #define SEC\_7 7000**

7 seconds

**6.80.2.44 #define SEC\_8 8000**

8 seconds

**Examples:**[ex\\_file\\_system.nxc.](#)**6.80.2.45 #define SEC\_9 9000**

9 seconds

**Examples:**[ex\\_SensorHTGyro.nxc.](#)**6.81 Mailbox constants**

Mailbox number constants should be used to avoid confusing NXT-G users.

**Defines**

- #define [MAILBOX1](#) 0
- #define [MAILBOX2](#) 1
- #define [MAILBOX3](#) 2
- #define [MAILBOX4](#) 3
- #define [MAILBOX5](#) 4
- #define [MAILBOX6](#) 5
- #define [MAILBOX7](#) 6
- #define [MAILBOX8](#) 7
- #define [MAILBOX9](#) 8
- #define [MAILBOX10](#) 9

**6.81.1 Detailed Description**

Mailbox number constants should be used to avoid confusing NXT-G users.

**See also:**

[SysMessageWrite\(\)](#), [SysMessageRead\(\)](#), [SendMessage\(\)](#), [ReceiveMessage\(\)](#), [SendRemoteBool\(\)](#), [SendRemoteNumber\(\)](#), [SendRemoteString\(\)](#), [SendResponseBool\(\)](#), [SendResponseNumber\(\)](#), [SendResponseString\(\)](#), [ReceiveRemoteBool\(\)](#), [ReceiveRemoteNumber\(\)](#), [ReceiveRemoteString\(\)](#), [ReceiveRemoteMessageEx\(\)](#), [RemoteMessageRead\(\)](#), [RemoteMessageWrite\(\)](#)

## 6.81.2 Define Documentation

### 6.81.2.1 #define MAILBOX1 0

Mailbox number 1

#### Examples:

[ex\\_joystickmsg.nxc](#), [ex\\_ReceiveMessage.nxc](#), [ex\\_ReceiveRemoteBool.nxc](#),  
[ex\\_ReceiveRemoteMessageEx.nxc](#), [ex\\_ReceiveRemoteNumber.nxc](#), [ex\\_ReceiveRemoteString.nxc](#),  
[ex\\_SendMessage.nxc](#), [ex\\_SendRemoteBool.nxc](#), [ex\\_SendRemoteNumber.nxc](#),  
[ex\\_SendRemoteString.nxc](#), [ex\\_SendResponseBool.nxc](#), [ex\\_SendResponseNumber.nxc](#),  
[ex\\_SendResponseString.nxc](#), [ex\\_SysMessageRead.nxc](#), and [ex\\_SysMessageWrite.nxc](#).

### 6.81.2.2 #define MAILBOX10 9

Mailbox number 10

### 6.81.2.3 #define MAILBOX2 1

Mailbox number 2

### 6.81.2.4 #define MAILBOX3 2

Mailbox number 3

### 6.81.2.5 #define MAILBOX4 3

Mailbox number 4

### 6.81.2.6 #define MAILBOX5 4

Mailbox number 5

### 6.81.2.7 #define MAILBOX6 5

Mailbox number 6

**6.81.2.8 #define MAILBOX7 6**

Mailbox number 7

**6.81.2.9 #define MAILBOX8 7**

Mailbox number 8

**6.81.2.10 #define MAILBOX9 8**

Mailbox number 9

**6.82 VM state constants**

Constants defining possible VM states.

**Defines**

- #define [TIMES\\_UP](#) 6
- #define [ROTATE\\_QUEUE](#) 5
- #define [STOP\\_REQ](#) 4
- #define [BREAKOUT\\_REQ](#) 3
- #define [CLUMP\\_SUSPEND](#) 2
- #define [CLUMP\\_DONE](#) 1

**6.82.1 Detailed Description**

Constants defining possible VM states.

**6.82.2 Define Documentation****6.82.2.1 #define BREAKOUT\_REQ 3**

VM should break out of current thread

**6.82.2.2 #define CLUMP\_DONE 1**

VM has finished executing thread

**6.82.2.3 #define CLUMP\_SUSPEND 2**

VM should suspend thread

**6.82.2.4 #define ROTATE\_QUEUE 5**

VM should rotate queue

**6.82.2.5 #define STOP\_REQ 4**

VM should stop executing program

**6.82.2.6 #define TIMES\_UP 6**

VM time is up

**6.83 Fatal errors**

Constants defining various fatal error conditions.

**Defines**

- #define [ERR\\_ARG](#) -1
- #define [ERR\\_INSTR](#) -2
- #define [ERR\\_FILE](#) -3
- #define [ERR\\_VER](#) -4
- #define [ERR\\_MEM](#) -5
- #define [ERR\\_BAD\\_PTR](#) -6
- #define [ERR\\_CLUMP\\_COUNT](#) -7
- #define [ERR\\_NO\\_CODE](#) -8
- #define [ERR\\_INSANE\\_OFFSET](#) -9
- #define [ERR\\_BAD\\_POOL\\_SIZE](#) -10
- #define [ERR\\_LOADER\\_ERR](#) -11
- #define [ERR\\_SPOTCHECK\\_FAIL](#) -12
- #define [ERR\\_NO\\_ACTIVE\\_CLUMP](#) -13
- #define [ERR\\_DEFAULT\\_OFFSETS](#) -14
- #define [ERR\\_MEMMGR\\_FAIL](#) -15
- #define [ERR\\_NON\\_FATAL](#) -16

### 6.83.1 Detailed Description

Constants defining various fatal error conditions.

### 6.83.2 Define Documentation

#### 6.83.2.1 #define ERR\_ARG -1

0xFF Bad arguments

#### 6.83.2.2 #define ERR\_BAD\_POOL\_SIZE -10

0xF6 VarsCmd.PoolSize > POOL\_MAX\_SIZE

#### 6.83.2.3 #define ERR\_BAD\_PTR -6

0xFA Someone passed us a bad pointer!

#### 6.83.2.4 #define ERR\_CLUMP\_COUNT -7

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT\_A\_CLUMP)

#### 6.83.2.5 #define ERR\_DEFAULT\_OFFSETS -14

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

#### 6.83.2.6 #define ERR\_FILE -3

0xFD Malformed file contents

#### 6.83.2.7 #define ERR\_INSANE\_OFFSET -9

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount \* 2)

#### 6.83.2.8 #define ERR\_INSTR -2

0xFE Illegal bytecode instruction

**6.83.2.9 #define ERR\_LOADER\_ERR -11**

0xF5 LOADER\_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

**6.83.2.10 #define ERR\_MEM -5**

0xFB Insufficient memory available

**6.83.2.11 #define ERR\_MEMMGR\_FAIL -15**

0xF1 (UBYTE \*)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV\_ARRAY[0].Offset

**6.83.2.12 #define ERR\_NO\_ACTIVE\_CLUMP -13**

0xF3 VarsCmd.RunQ.Head == NOT\_A\_CLUMP

**6.83.2.13 #define ERR\_NO\_CODE -8**

0xF8 VarsCmd.CodespaceCount == 0

**6.83.2.14 #define ERR\_NON\_FATAL -16**

Fatal errors are greater than this value

**6.83.2.15 #define ERR\_SPOTCHECK\_FAIL -12**

0xF4 ((UBYTE\*)(VarsCmd.pCodespace) < pData) (c\_cmd.c 1893)

**6.83.2.16 #define ERR\_VER -4**

0xFC Version mismatch between firmware and compiler

**6.84 General errors**

Constants defining general error conditions.

## Defines

- #define [ERR\\_INVALID\\_PORT](#) -16
- #define [ERR\\_INVALID\\_FIELD](#) -17
- #define [ERR\\_INVALID\\_QUEUE](#) -18
- #define [ERR\\_INVALID\\_SIZE](#) -19
- #define [ERR\\_NO\\_PROG](#) -20

### 6.84.1 Detailed Description

Constants defining general error conditions.

### 6.84.2 Define Documentation

#### 6.84.2.1 #define [ERR\\_INVALID\\_FIELD](#) -17

0xEF Attempted to access invalid field of a structure

#### 6.84.2.2 #define [ERR\\_INVALID\\_PORT](#) -16

0xF0 Bad input or output port specified

#### 6.84.2.3 #define [ERR\\_INVALID\\_QUEUE](#) -18

0xEE Illegal queue ID specified

#### 6.84.2.4 #define [ERR\\_INVALID\\_SIZE](#) -19

0xED Illegal size specified

#### 6.84.2.5 #define [ERR\\_NO\\_PROG](#) -20

0xEC No active program

## 6.85 Communications specific errors

Constants defining communication error conditions.

### Defines

- #define [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#) -32
- #define [ERR\\_COMM\\_CHAN\\_INVALID](#) -33
- #define [ERR\\_COMM\\_BUFFER\\_FULL](#) -34
- #define [ERR\\_COMM\\_BUS\\_ERR](#) -35

#### 6.85.1 Detailed Description

Constants defining communication error conditions.

#### 6.85.2 Define Documentation

##### 6.85.2.1 #define [ERR\\_COMM\\_BUFFER\\_FULL](#) -34

0xDE No room in comm buffer

##### 6.85.2.2 #define [ERR\\_COMM\\_BUS\\_ERR](#) -35

0xDD Something went wrong on the communications bus

##### 6.85.2.3 #define [ERR\\_COMM\\_CHAN\\_INVALID](#) -33

0xDF Specified channel/connection is not valid

##### 6.85.2.4 #define [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#) -32

0xE0 Specified channel/connection not configured or busy

## 6.86 Remote control (direct commands) errors

Constants defining errors that can occur during remote control (RC) direct command operations.

### Defines

- #define [ERR\\_RC\\_ILLEGAL\\_VAL](#) -64
- #define [ERR\\_RC\\_BAD\\_PACKET](#) -65
- #define [ERR\\_RC\\_UNKNOWN\\_CMD](#) -66
- #define [ERR\\_RC\\_FAILED](#) -67

### 6.86.1 Detailed Description

Constants defining errors that can occur during remote control (RC) direct command operations.

### 6.86.2 Define Documentation

#### 6.86.2.1 #define ERR\_RC\_BAD\_PACKET -65

0xBF Clearly insane packet

#### 6.86.2.2 #define ERR\_RC\_FAILED -67

0xBD Request failed (i.e. specified file not found)

#### 6.86.2.3 #define ERR\_RC\_ILLEGAL\_VAL -64

0xC0 Data contains out-of-range values

#### 6.86.2.4 #define ERR\_RC\_UNKNOWN\_CMD -66

0xBE Unknown command opcode

## 6.87 Program status constants

Constants defining various states of the command module virtual machine.

### Defines

- #define [PROG\\_IDLE](#) 0
- #define [PROG\\_OK](#) 1
- #define [PROG\\_RUNNING](#) 2
- #define [PROG\\_ERROR](#) 3
- #define [PROG\\_ABORT](#) 4
- #define [PROG\\_RESET](#) 5

### 6.87.1 Detailed Description

Constants defining various states of the command module virtual machine.

### 6.87.2 Define Documentation

#### 6.87.2.1 #define PROG\_ABORT 4

Program has been aborted

#### 6.87.2.2 #define PROG\_ERROR 3

A program error has occurred

#### 6.87.2.3 #define PROG\_IDLE 0

Program state is idle

#### 6.87.2.4 #define PROG\_OK 1

Program state is okay

#### 6.87.2.5 #define PROG\_RESET 5

Program has been reset

#### 6.87.2.6 #define PROG\_RUNNING 2

Program is running

## 6.88 Command module IOMAP offsets

Constant offsets into the Command module IOMAP structure.

### Defines

- #define [CommandOffsetFormatString](#) 0
- #define [CommandOffsetPRCHandler](#) 16
- #define [CommandOffsetTick](#) 20
- #define [CommandOffsetOffsetDS](#) 24
- #define [CommandOffsetOffsetDVA](#) 26
- #define [CommandOffsetProgStatus](#) 28
- #define [CommandOffsetAwake](#) 29
- #define [CommandOffsetActivateFlag](#) 30

- #define [CommandOffsetDeactivateFlag](#) 31
- #define [CommandOffsetFileName](#) 32
- #define [CommandOffsetMemoryPool](#) 52
- #define [CommandOffsetSyncTime](#) 32820
- #define [CommandOffsetSyncTick](#) 32824

### 6.88.1 Detailed Description

Constant offsets into the Command module IOMAP structure.

### 6.88.2 Define Documentation

#### 6.88.2.1 #define [CommandOffsetActivateFlag](#) 30

Offset to the activate flag

#### 6.88.2.2 #define [CommandOffsetAwake](#) 29

Offset to the VM's awake state

#### 6.88.2.3 #define [CommandOffsetDeactivateFlag](#) 31

Offset to the deactivate flag

#### 6.88.2.4 #define [CommandOffsetFileName](#) 32

Offset to the running program's filename

#### 6.88.2.5 #define [CommandOffsetFormatString](#) 0

Offset to the format string

#### 6.88.2.6 #define [CommandOffsetMemoryPool](#) 52

Offset to the VM's memory pool

### Examples:

[ex\\_reladdressof.nxc](#).

**6.88.2.7 #define CommandOffsetOffsetDS 24**

Offset to the running program's data space (DS)

**6.88.2.8 #define CommandOffsetOffsetDVA 26**

Offset to the running program's DOPE vector address (DVA)

**6.88.2.9 #define CommandOffsetPRCHandler 16**

Offset to the RC Handler function pointer

**6.88.2.10 #define CommandOffsetProgStatus 28**

Offset to the running program's status

**Examples:**

[ex\\_RemoteIOMapRead.nxc](#), [ex\\_RemoteIOMapWriteBytes.nxc](#), and [ex\\_RemoteIOMapWriteValue.nxc](#).

**6.88.2.11 #define CommandOffsetSyncTick 32824**

Offset to the VM sync tick

**6.88.2.12 #define CommandOffsetSyncTime 32820**

Offset to the VM sync time

**6.88.2.13 #define CommandOffsetTick 20**

Offset to the VM's current tick

**Examples:**

[ex\\_sysiomapread.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**6.89 IOCtrl module constants**

Constants that are part of the NXT firmware's IOCtrl module.

## Modules

- [PowerOn constants](#)

*Use these constants to power down the NXT or boot it into SAMBA (aka firmware download) mode.*

- [IOCtrl module IOMAP offsets](#)

*Constant offsets into the IOCtrl module IOMAP structure.*

### 6.89.1 Detailed Description

Constants that are part of the NXT firmware's IOCtrl module.

## 6.90 PowerOn constants

Use these constants to power down the NXT or boot it into SAMBA (aka firmware download) mode.

### Defines

- `#define IOCTRL_POWERDOWN 0x5A00`
- `#define IOCTRL_BOOT 0xA55A`

### 6.90.1 Detailed Description

Use these constants to power down the NXT or boot it into SAMBA (aka firmware download) mode.

#### 6.90.2 Define Documentation

##### 6.90.2.1 `#define IOCTRL_BOOT 0xA55A`

Reboot the NXT into SAMBA mode

##### 6.90.2.2 `#define IOCTRL_POWERDOWN 0x5A00`

Power down the NXT

## 6.91 IOCtrl module IOMAP offsets

Constant offsets into the IOCtrl module IOMAP structure.

### Defines

- `#define IOCtrlOffsetPowerOn 0`

### 6.91.1 Detailed Description

Constant offsets into the IOCtrl module IOMAP structure.

### 6.91.2 Define Documentation

#### 6.91.2.1 `#define IOCtrlOffsetPowerOn 0`

Offset to power on field

## 6.92 Loader module constants

Constants that are part of the NXT firmware's Loader module.

### Modules

- [Loader module IOMAP offsets](#)  
*Constant offsets into the Loader module IOMAP structure.*
- [Loader module error codes](#)  
*Error codes returned by functions in the Loader module (file access).*
- [Loader module function constants](#)  
*Constants defining the functions provided by the Loader module.*

### Defines

- `#define EOF -1`
- `#define NULL 0`

### 6.92.1 Detailed Description

Constants that are part of the NXT firmware's Loader module.

### 6.92.2 Define Documentation

#### 6.92.2.1 #define EOF -1

A constant representing end of file

#### 6.92.2.2 #define NULL 0

A constant representing NULL

## 6.93 Loader module IOMAP offsets

Constant offsets into the Loader module IOMAP structure.

### Defines

- #define [LoaderOffsetPFunc](#) 0
- #define [LoaderOffsetFreeUserFlash](#) 4

### 6.93.1 Detailed Description

Constant offsets into the Loader module IOMAP structure.

### 6.93.2 Define Documentation

#### 6.93.2.1 #define LoaderOffsetFreeUserFlash 4

Offset to the amount of free user flash

#### 6.93.2.2 #define LoaderOffsetPFunc 0

Offset to the Loader module function pointer

## 6.94 Loader module error codes

Error codes returned by functions in the Loader module (file access).

## Defines

- #define `LDR_SUCCESS` 0x0000
- #define `LDR_INPROGRESS` 0x0001
- #define `LDR_REQPIN` 0x0002
- #define `LDR_NOMOREHANDLES` 0x8100
- #define `LDR_NOSPACE` 0x8200
- #define `LDR_NOMOREFILES` 0x8300
- #define `LDR_EOFEXPECTED` 0x8400
- #define `LDR_ENDOFFILE` 0x8500
- #define `LDR_NOTLINEARFILE` 0x8600
- #define `LDR_FILENOTFOUND` 0x8700
- #define `LDR_HANDLEALREADYCLOSED` 0x8800
- #define `LDR_NOLINEARSPACE` 0x8900
- #define `LDR_UNDEFINEDERROR` 0x8A00
- #define `LDR_FILEISBUSY` 0x8B00
- #define `LDR_NOWRITEBUFFERS` 0x8C00
- #define `LDR_APPENDNOTPOSSIBLE` 0x8D00
- #define `LDR_FILEISFULL` 0x8E00
- #define `LDR_FILEEXISTS` 0x8F00
- #define `LDR_MODULENOTFOUND` 0x9000
- #define `LDR_OUTOFBOUNDARY` 0x9100
- #define `LDR_ILLEGALFILENAME` 0x9200
- #define `LDR_ILLEGALHANDLE` 0x9300
- #define `LDR_BTBUSY` 0x9400
- #define `LDR_BTCONNECTFAIL` 0x9500
- #define `LDR_BTTIMEOUT` 0x9600
- #define `LDR_FILETX_TIMEOUT` 0x9700
- #define `LDR_FILETX_DSTEXISTS` 0x9800
- #define `LDR_FILETX_SRCMISSING` 0x9900
- #define `LDR_FILETX_STREAMERROR` 0x9A00
- #define `LDR_FILETX_CLOSEERROR` 0x9B00
- #define `LDR_INVALIDSEEK` 0x9C00

### 6.94.1 Detailed Description

Error codes returned by functions in the Loader module (file access).

### 6.94.2 Define Documentation

#### 6.94.2.1 #define `LDR_APPENDNOTPOSSIBLE` 0x8D00

Only datafiles can be appended to.

**6.94.2.2 #define LDR\_BTBUSY 0x9400**

The bluetooth system is busy.

**6.94.2.3 #define LDR\_BTCONNECTFAIL 0x9500**

Bluetooth connection attempt failed.

**6.94.2.4 #define LDR\_BTTIMEOUT 0x9600**

A timeout in the bluetooth system has occurred.

**6.94.2.5 #define LDR\_ENDOFFILE 0x8500**

The end of the file has been reached.

**Examples:**

[ex\\_file\\_system.nxc](#).

**6.94.2.6 #define LDR\_EOFEXPECTED 0x8400**

EOF expected.

**Examples:**

[ex\\_file\\_system.nxc](#).

**6.94.2.7 #define LDR\_FILEEXISTS 0x8F00**

A file with the same name already exists.

**Examples:**

[ex\\_file\\_system.nxc](#).

**6.94.2.8 #define LDR\_FILEISBUSY 0x8B00**

The file is already being used.

**6.94.2.9 #define LDR\_FILEISFULL 0x8E00**

The allocated file size has been filled.

**Examples:**

[ex\\_file\\_system.nxc](#).

**6.94.2.10 #define LDR\_FILENOTFOUND 0x8700**

No files matched the search criteria.

**6.94.2.11 #define LDR\_FILETX\_CLOSEERROR 0x9B00**

Error transmitting file: attempt to close file failed.

**6.94.2.12 #define LDR\_FILETX\_DSTEXISTS 0x9800**

Error transmitting file: destination file exists.

**6.94.2.13 #define LDR\_FILETX\_SRCMISSING 0x9900**

Error transmitting file: source file is missing.

**6.94.2.14 #define LDR\_FILETX\_STREAMERROR 0x9A00**

Error transmitting file: a stream error occurred.

**6.94.2.15 #define LDR\_FILETX\_TIMEOUT 0x9700**

Error transmitting file: a timeout occurred.

**6.94.2.16 #define LDR\_HANDLEALREADYCLOSED 0x8800**

The file handle has already been closed.

**6.94.2.17 #define LDR\_ILLEGALFILENAME 0x9200**

Filename length too long or attempted open a system file (\*.rxe, \*.rtm, or \*.sys) for writing as a datafile.

**6.94.2.18 #define LDR\_ILLEGALHANDLE 0x9300**

Invalid file handle.

**6.94.2.19 #define LDR\_INPROGRESS 0x0001**

The function is executing but has not yet completed.

**6.94.2.20 #define LDR\_INVALIDSEEK 0x9C00**

Invalid file seek operation.

**6.94.2.21 #define LDR\_MODULENOTFOUND 0x9000**

No modules matched the specified search criteria.

**6.94.2.22 #define LDR\_NOLINEARSPACE 0x8900**

Not enough linear flash memory is available.

**6.94.2.23 #define LDR\_NOMOREFILES 0x8300**

The maximum number of files has been reached.

**6.94.2.24 #define LDR\_NOMOREHANDLES 0x8100**

All available file handles are in use.

**6.94.2.25 #define LDR\_NOSPACE 0x8200**

Not enough free flash memory for the specified file size.

**6.94.2.26 #define LDR\_NOTLINEARFILE 0x8600**

The specified file is not linear.

**6.94.2.27 #define LDR\_NOWRITEBUFFERS 0x8C00**

No more write buffers are available.

**6.94.2.28 #define LDR\_OUTOFBOUNDARY 0x9100**

Specified IOMap offset is outside the bounds of the IOMap.

**6.94.2.29 #define LDR\_REQPIN 0x0002**

A PIN exchange request is in progress.

**6.94.2.30 #define LDR\_SUCCESS 0x0000**

The function completed successfully.

**Examples:**

`ex_file_system.nxc`, `ex_findfirstfile.nxc`, `ex_findnextfile.nxc`, `ex_syscommbtcheckstatus.nxc`, `ex_syscommbtconnection.nxc`, `ex_sysfilerename.nxc`, and `ex_sysfileresolvehandle.nxc`.

**6.94.2.31 #define LDR\_UNDEFINEDERROR 0x8A00**

An undefined error has occurred.

**6.95 Loader module function constants**

Constants defining the functions provided by the Loader module.

**Defines**

- #define `LDR_CMD_OPENREAD` 0x80
- #define `LDR_CMD_OPENWRITE` 0x81
- #define `LDR_CMD_READ` 0x82
- #define `LDR_CMD_WRITE` 0x83
- #define `LDR_CMD_CLOSE` 0x84
- #define `LDR_CMD_DELETE` 0x85
- #define `LDR_CMD_FINDFIRST` 0x86
- #define `LDR_CMD_FINDNEXT` 0x87
- #define `LDR_CMD_VERSIONS` 0x88
- #define `LDR_CMD_OPENWRITELINEAR` 0x89
- #define `LDR_CMD_OPENREADLINEAR` 0x8A
- #define `LDR_CMD_OPENWRITEDATA` 0x8B
- #define `LDR_CMD_OPENAPPENDDATA` 0x8C

- #define `LDR_CMD_CROPDATAFILE` 0x8D
- #define `LDR_CMD_FINDFIRSTMODULE` 0x90
- #define `LDR_CMD_FINDNEXTMODULE` 0x91
- #define `LDR_CMD_CLOSEMODHANDLE` 0x92
- #define `LDR_CMD_IOMAPREAD` 0x94
- #define `LDR_CMD_IOMAPWRITE` 0x95
- #define `LDR_CMD_BOOTCMD` 0x97
- #define `LDR_CMD_SETBRICKNAME` 0x98
- #define `LDR_CMD_BTGETADR` 0x9A
- #define `LDR_CMD_DEVICEINFO` 0x9B
- #define `LDR_CMD_DELETEUSERFLASH` 0xA0
- #define `LDR_CMD_POLLCMDLEN` 0xA1
- #define `LDR_CMD_POLLCMD` 0xA2
- #define `LDR_CMD_RENAMEFILE` 0xA3
- #define `LDR_CMD_BTFACTORYRESET` 0xA4
- #define `LDR_CMD_RESIZEDATAFILE` 0xD0
- #define `LDR_CMD_SEEKFROMSTART` 0xD1
- #define `LDR_CMD_SEEKFROMCURRENT` 0xD2
- #define `LDR_CMD_SEEKFROMEND` 0xD3

### 6.95.1 Detailed Description

Constants defining the functions provided by the Loader module.

### 6.95.2 Define Documentation

#### 6.95.2.1 #define `LDR_CMD_BOOTCMD` 0x97

Reboot the NXT into SAMBA mode

#### 6.95.2.2 #define `LDR_CMD_BTFACTORYRESET` 0xA4

Reset bluetooth configuration to factory defaults

#### 6.95.2.3 #define `LDR_CMD_BTGETADR` 0x9A

Get the NXT's bluetooth brick address

#### 6.95.2.4 #define `LDR_CMD_CLOSE` 0x84

Close a file handle

**6.95.2.5 #define LDR\_CMD\_CLOSEMODHANDLE 0x92**

Close a module handle

**6.95.2.6 #define LDR\_CMD\_CROPDATAFILE 0x8D**

Crop a data file to its used space

**6.95.2.7 #define LDR\_CMD\_DELETE 0x85**

Delete a file

**6.95.2.8 #define LDR\_CMD\_DELETEUSERFLASH 0xA0**

Delete all files from user flash memory

**6.95.2.9 #define LDR\_CMD\_DEVICEINFO 0x9B**

Read device information

**6.95.2.10 #define LDR\_CMD\_FINDFIRST 0x86**

Find the first file matching the specified pattern

**6.95.2.11 #define LDR\_CMD\_FINDFIRSTMODULE 0x90**

Find the first module matching the specified pattern

**6.95.2.12 #define LDR\_CMD\_FINDNEXT 0x87**

Find the next file matching the specified pattern

**6.95.2.13 #define LDR\_CMD\_FINDNEXTMODULE 0x91**

Find the next module matching the specified pattern

**6.95.2.14 #define LDR\_CMD\_IOMAPREAD 0x94**

Read data from a module IOMAP

**6.95.2.15 #define LDR\_CMD\_IOMAPWRITE 0x95**

Write data to a module IOMAP

**6.95.2.16 #define LDR\_CMD\_OPENAPPENDDATA 0x8C**

Open a data file for appending

**6.95.2.17 #define LDR\_CMD\_OPENREAD 0x80**

Open a file for reading

**6.95.2.18 #define LDR\_CMD\_OPENREADLINEAR 0x8A**

Open a linear file for reading

**6.95.2.19 #define LDR\_CMD\_OPENWRITE 0x81**

Open a file for writing

**6.95.2.20 #define LDR\_CMD\_OPENWRITEDATA 0x8B**

Open a data file for writing

**6.95.2.21 #define LDR\_CMD\_OPENWRITELINEAR 0x89**

Open a linear file for writing

**6.95.2.22 #define LDR\_CMD\_POLLCMD 0xA2**

Poll command

**6.95.2.23 #define LDR\_CMD\_POLLCMDLEN 0xA1**

Read poll command length

**6.95.2.24 #define LDR\_CMD\_READ 0x82**

Read from a file

**6.95.2.25 #define LDR\_CMD\_RENAMEFILE 0xA3**

Rename a file

**6.95.2.26 #define LDR\_CMD\_RESIZEDATAFILE 0xD0**

Resize a data file

**6.95.2.27 #define LDR\_CMD\_SEEKFROMCURRENT 0xD2**

Seek from the current position

**6.95.2.28 #define LDR\_CMD\_SEEKFROMEND 0xD3**

Seek from the end of the file

**6.95.2.29 #define LDR\_CMD\_SEEKFROMSTART 0xD1**

Seek from the start of the file

**6.95.2.30 #define LDR\_CMD\_SETBRICKNAME 0x98**

Set the NXT's brick name

**6.95.2.31 #define LDR\_CMD\_VERSIONS 0x88**

Read firmware version information

**6.95.2.32 #define LDR\_CMD\_WRITE 0x83**

Write to a file

## 6.96 Sound module constants

Constants that are part of the NXT firmware's Sound module.

## Modules

- [SoundFlags constants](#)  
*Constants for use with the [SoundFlags\(\)](#) function.*
- [SoundState constants](#)  
*Constants for use with the [SoundState\(\)](#) function.*
- [SoundMode constants](#)  
*Constants for use with the [SoundMode\(\)](#) function.*
- [Sound module IOMAP offsets](#)  
*Constant offsets into the Sound module IOMAP structure.*
- [Sound module miscellaneous constants](#)  
*Constants defining miscellaneous sound module aspects.*
- [Tone constants](#)  
*Constants for use in the [SoundPlayTone\(\)](#) API function.*

### 6.96.1 Detailed Description

Constants that are part of the NXT firmware's Sound module.

## 6.97 SoundFlags constants

Constants for use with the [SoundFlags\(\)](#) function.

### Defines

- #define [SOUND\\_FLAGS\\_IDLE](#) 0x00
- #define [SOUND\\_FLAGS\\_UPDATE](#) 0x01
- #define [SOUND\\_FLAGS\\_RUNNING](#) 0x02

### 6.97.1 Detailed Description

Constants for use with the [SoundFlags\(\)](#) function.

See also:

[SoundFlags\(\)](#)

### 6.97.2 Define Documentation

#### 6.97.2.1 #define SOUND\_FLAGS\_IDLE 0x00

R - Sound is idle

#### 6.97.2.2 #define SOUND\_FLAGS\_RUNNING 0x02

R - Currently processing a tone or file

#### 6.97.2.3 #define SOUND\_FLAGS\_UPDATE 0x01

W - Make changes take effect

#### Examples:

[ex\\_SetSoundFlags.nxc](#).

## 6.98 SoundState constants

Constants for use with the [SoundState\(\)](#) function.

#### Defines

- #define [SOUND\\_STATE\\_IDLE](#) 0x00
- #define [SOUND\\_STATE\\_FILE](#) 0x02
- #define [SOUND\\_STATE\\_TONE](#) 0x03
- #define [SOUND\\_STATE\\_STOP](#) 0x04

### 6.98.1 Detailed Description

Constants for use with the [SoundState\(\)](#) function.

#### See also:

[SoundState\(\)](#)

### 6.98.2 Define Documentation

#### 6.98.2.1 #define SOUND\_STATE\_FILE 0x02

R - Processing a file of sound/melody data

**6.98.2.2 #define SOUND\_STATE\_IDLE 0x00**

R - Idle, ready for start sound (SOUND\_UPDATE)

**Examples:**

[ex\\_syssoundgetstate.nxc](#).

**6.98.2.3 #define SOUND\_STATE\_STOP 0x04**

W - Stop sound immediately and close hardware

**Examples:**

[ex\\_SetSoundModuleState.nxc](#), and [ex\\_syssoundsetstate.nxc](#).

**6.98.2.4 #define SOUND\_STATE\_TONE 0x03**

R - Processing a play tone request

**6.99 SoundMode constants**

Constants for use with the [SoundMode\(\)](#) function.

**Defines**

- #define [SOUND\\_MODE\\_ONCE](#) 0x00
- #define [SOUND\\_MODE\\_LOOP](#) 0x01
- #define [SOUND\\_MODE\\_TONE](#) 0x02

**6.99.1 Detailed Description**

Constants for use with the [SoundMode\(\)](#) function.

**See also:**

[SoundMode\(\)](#)

**6.99.2 Define Documentation****6.99.2.1 #define SOUND\_MODE\_LOOP 0x01**

W - Play file until writing SOUND\_STATE\_STOP into SoundState

**6.99.2.2 #define SOUND\_MODE\_ONCE 0x00**

W - Only play file once

**Examples:**

[ex\\_SetSoundMode.nxc](#).

**6.99.2.3 #define SOUND\_MODE\_TONE 0x02**

W - Play tone specified in Frequency for Duration ms

**6.100 Sound module IOMAP offsets**

Constant offsets into the Sound module IOMAP structure.

**Defines**

- #define [SoundOffsetFreq](#) 0
- #define [SoundOffsetDuration](#) 2
- #define [SoundOffsetSampleRate](#) 4
- #define [SoundOffsetSoundFilename](#) 6
- #define [SoundOffsetFlags](#) 26
- #define [SoundOffsetState](#) 27
- #define [SoundOffsetMode](#) 28
- #define [SoundOffsetVolume](#) 29

**6.100.1 Detailed Description**

Constant offsets into the Sound module IOMAP structure.

**6.100.2 Define Documentation****6.100.2.1 #define SoundOffsetDuration 2**

RW - [Tone](#) duration [mS] (2 bytes)

**6.100.2.2 #define SoundOffsetFlags 26**

RW - Play flag - described above (1 byte) [SoundFlags constants](#)

**6.100.2.3 #define SoundOffsetFreq 0**

RW - [Tone](#) frequency [Hz] (2 bytes)

**6.100.2.4 #define SoundOffsetMode 28**

RW - Play mode - described above (1 byte) [SoundMode constants](#)

**6.100.2.5 #define SoundOffsetSampleRate 4**

RW - Sound file sample rate [2000..16000] (2 bytes)

**Examples:**

[ex\\_sysiomapwrite.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**6.100.2.6 #define SoundOffsetSoundFilename 6**

RW - Sound/melody filename (20 bytes)

**6.100.2.7 #define SoundOffsetState 27**

RW - Play state - described above (1 byte) [SoundState constants](#)

**6.100.2.8 #define SoundOffsetVolume 29**

RW - Sound/melody volume [0..4] 0 = off (1 byte)

**6.101 Sound module miscellaneous constants**

Constants defining miscellaneous sound module aspects.

**Defines**

- #define [FREQUENCY\\_MIN](#) 220
- #define [FREQUENCY\\_MAX](#) 14080
- #define [SAMPLERATE\\_MIN](#) 2000
- #define [SAMPLERATE\\_DEFAULT](#) 8000
- #define [SAMPLERATE\\_MAX](#) 16000

### 6.101.1 Detailed Description

Constants defining miscellaneous sound module aspects.

### 6.101.2 Define Documentation

#### 6.101.2.1 #define FREQUENCY\_MAX 14080

Maximum frequency [Hz]

#### 6.101.2.2 #define FREQUENCY\_MIN 220

Minimum frequency [Hz]

#### 6.101.2.3 #define SAMPLERATE\_DEFAULT 8000

Default sample rate [sps]

#### 6.101.2.4 #define SAMPLERATE\_MAX 16000

Max sample rate [sps]

#### 6.101.2.5 #define SAMPLERATE\_MIN 2000

Min sample rate [sps]

## 6.102 Tone constants

Constants for use in the [SoundPlayTone\(\)](#) API function.

### Defines

- #define [TONE\\_A3](#) 220
- #define [TONE\\_AS3](#) 233
- #define [TONE\\_B3](#) 247
- #define [TONE\\_C4](#) 262
- #define [TONE\\_CS4](#) 277
- #define [TONE\\_D4](#) 294
- #define [TONE\\_DS4](#) 311
- #define [TONE\\_E4](#) 330

- #define [TONE\\_F4](#) 349
- #define [TONE\\_FS4](#) 370
- #define [TONE\\_G4](#) 392
- #define [TONE\\_GS4](#) 415
- #define [TONE\\_A4](#) 440
- #define [TONE\\_AS4](#) 466
- #define [TONE\\_B4](#) 494
- #define [TONE\\_C5](#) 523
- #define [TONE\\_CS5](#) 554
- #define [TONE\\_D5](#) 587
- #define [TONE\\_DS5](#) 622
- #define [TONE\\_E5](#) 659
- #define [TONE\\_F5](#) 698
- #define [TONE\\_FS5](#) 740
- #define [TONE\\_G5](#) 784
- #define [TONE\\_GS5](#) 831
- #define [TONE\\_A5](#) 880
- #define [TONE\\_AS5](#) 932
- #define [TONE\\_B5](#) 988
- #define [TONE\\_C6](#) 1047
- #define [TONE\\_CS6](#) 1109
- #define [TONE\\_D6](#) 1175
- #define [TONE\\_DS6](#) 1245
- #define [TONE\\_E6](#) 1319
- #define [TONE\\_F6](#) 1397
- #define [TONE\\_FS6](#) 1480
- #define [TONE\\_G6](#) 1568
- #define [TONE\\_GS6](#) 1661
- #define [TONE\\_A6](#) 1760
- #define [TONE\\_AS6](#) 1865
- #define [TONE\\_B6](#) 1976
- #define [TONE\\_C7](#) 2093
- #define [TONE\\_CS7](#) 2217
- #define [TONE\\_D7](#) 2349
- #define [TONE\\_DS7](#) 2489
- #define [TONE\\_E7](#) 2637
- #define [TONE\\_F7](#) 2794
- #define [TONE\\_FS7](#) 2960
- #define [TONE\\_G7](#) 3136
- #define [TONE\\_GS7](#) 3322
- #define [TONE\\_A7](#) 3520
- #define [TONE\\_AS7](#) 3729
- #define [TONE\\_B7](#) 3951

### 6.102.1 Detailed Description

Constants for use in the [SoundPlayTone\(\)](#) API function.

See also:

[SoundPlayTone\(\)](#)

### 6.102.2 Define Documentation

#### 6.102.2.1 #define TONE\_A3 220

Third octave A

#### 6.102.2.2 #define TONE\_A4 440

Fourth octave A

Examples:

[ex\\_yield.nxc](#).

#### 6.102.2.3 #define TONE\_A5 880

Fifth octave A

#### 6.102.2.4 #define TONE\_A6 1760

Sixth octave A

#### 6.102.2.5 #define TONE\_A7 3520

Seventh octave A

#### 6.102.2.6 #define TONE\_AS3 233

Third octave A sharp

#### 6.102.2.7 #define TONE\_AS4 466

Fourth octave A sharp

**6.102.2.8 #define TONE\_AS5 932**

Fifth octave A sharp

**6.102.2.9 #define TONE\_AS6 1865**

Sixth octave A sharp

**6.102.2.10 #define TONE\_AS7 3729**

Seventh octave A sharp

**6.102.2.11 #define TONE\_B3 247**

Third octave B

**6.102.2.12 #define TONE\_B4 494**

Fourth octave B

**6.102.2.13 #define TONE\_B5 988**

Fifth octave B

**6.102.2.14 #define TONE\_B6 1976**

Sixth octave B

**6.102.2.15 #define TONE\_B7 3951**

Seventh octave B

**6.102.2.16 #define TONE\_C4 262**

Fourth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**6.102.2.17 #define TONE\_C5 523**

Fifth octave C

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_playtones.nxc](#).

**6.102.2.18 #define TONE\_C6 1047**

Sixth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**6.102.2.19 #define TONE\_C7 2093**

Seventh octave C

**6.102.2.20 #define TONE\_CS4 277**

Fourth octave C sharp

**6.102.2.21 #define TONE\_CS5 554**

Fifth octave C sharp

**6.102.2.22 #define TONE\_CS6 1109**

Sixth octave C sharp

**6.102.2.23 #define TONE\_CS7 2217**

Seventh octave C sharp

**6.102.2.24 #define TONE\_D4 294**

Fourth octave D

**6.102.2.25** `#define TONE_D5 587`

Fifth octave D

**6.102.2.26** `#define TONE_D6 1175`

Sixth octave D

**6.102.2.27** `#define TONE_D7 2349`

Seventh octave D

**6.102.2.28** `#define TONE_DS4 311`

Fourth octave D sharp

**6.102.2.29** `#define TONE_DS5 622`

Fifth octave D sharp

**6.102.2.30** `#define TONE_DS6 1245`

Sixth octave D sharp

**6.102.2.31** `#define TONE_DS7 2489`

Seventh octave D sharp

**6.102.2.32** `#define TONE_E4 330`

Fourth octave E

**Examples:**

[ex\\_playtones.nxc](#).

**6.102.2.33** `#define TONE_E5 659`

Fifth octave E

**Examples:**

[ex\\_playtones.nxc](#).

**6.102.2.34 #define TONE\_E6 1319**

Sixth octave E

**6.102.2.35 #define TONE\_E7 2637**

Seventh octave E

**6.102.2.36 #define TONE\_F4 349**

Fourth octave F

**6.102.2.37 #define TONE\_F5 698**

Fifth octave F

**6.102.2.38 #define TONE\_F6 1397**

Sixth octave F

**6.102.2.39 #define TONE\_F7 2794**

Seventh octave F

**6.102.2.40 #define TONE\_FS4 370**

Fourth octave F sharp

**6.102.2.41 #define TONE\_FS5 740**

Fifth octave F sharp

**6.102.2.42 #define TONE\_FS6 1480**

Sixth octave F sharp

**6.102.2.43 #define TONE\_FS7 2960**

Seventh octave F sharp

**6.102.2.44 #define TONE\_G4 392**

Fourth octave G

**Examples:**

[ex\\_playtones.nxc.](#)

**6.102.2.45 #define TONE\_G5 784**

Fifth octave G

**Examples:**

[ex\\_playtones.nxc.](#)

**6.102.2.46 #define TONE\_G6 1568**

Sixth octave G

**6.102.2.47 #define TONE\_G7 3136**

Seventh octave G

**6.102.2.48 #define TONE\_GS4 415**

Fourth octave G sharp

**6.102.2.49 #define TONE\_GS5 831**

Fifth octave G sharp

**6.102.2.50 #define TONE\_GS6 1661**

Sixth octave G sharp

### 6.102.2.51 #define TONE\_GS7 3322

Seventh octave G sharp

## 6.103 Button module constants

Constants that are part of the NXT firmware's Button module.

### Modules

- [Button name constants](#)  
*Constants to specify which button to use with button module functions.*
- [ButtonState constants](#)  
*Constants for use with the [ButtonState\(\)](#) function.*
- [Button module IOMAP offsets](#)  
*Constant offsets into the Button module IOMAP structure.*

### 6.103.1 Detailed Description

Constants that are part of the NXT firmware's Button module.

## 6.104 Button name constants

Constants to specify which button to use with button module functions.

### Defines

- #define [BTN1](#) 0
- #define [BTN2](#) 1
- #define [BTN3](#) 2
- #define [BTN4](#) 3
- #define [BTNEXIT](#) BTN1
- #define [BTNRIGHT](#) BTN2
- #define [BTNLEFT](#) BTN3
- #define [BTNCENTER](#) BTN4
- #define [NO\\_OF\\_BTNS](#) 4

### 6.104.1 Detailed Description

Constants to specify which button to use with button module functions.

See also:

[ButtonPressed\(\)](#), [ButtonState\(\)](#), [ButtonCount\(\)](#), [ReadButtonEx\(\)](#), [SysReadButton\(\)](#), [ReadButtonType](#)

### 6.104.2 Define Documentation

#### 6.104.2.1 #define BTN1 0

The exit button.

Examples:

[ex\\_ButtonCount.nxc](#), [ex\\_ButtonLongPressCount.nxc](#), [ex\\_ButtonLongReleaseCount.nxc](#), [ex\\_ButtonPressCount.nxc](#), [ex\\_ButtonReleaseCount.nxc](#), [ex\\_ButtonShortReleaseCount.nxc](#), [ex\\_ButtonState.nxc](#), [ex\\_ReadButtonEx.nxc](#), [ex\\_SetButtonLongPressCount.nxc](#), [ex\\_SetButtonLongReleaseCount.nxc](#), [ex\\_SetButtonPressCount.nxc](#), [ex\\_SetButtonReleaseCount.nxc](#), [ex\\_SetButtonShortReleaseCount.nxc](#), and [ex\\_SetButtonState.nxc](#).

#### 6.104.2.2 #define BTN2 1

The right button.

#### 6.104.2.3 #define BTN3 2

The left button.

#### 6.104.2.4 #define BTN4 3

The enter button.

#### 6.104.2.5 #define BTNCENTER BTN4

The enter button.

Examples:

[ex\\_buttonpressed.nxc](#), and [ex\\_HTGyroTest.nxc](#).

#### 6.104.2.6 #define BTNEXIT BTN1

The exit button.

##### Examples:

[ex\\_buttonpressed.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

#### 6.104.2.7 #define BTNLEFT BTN3

The left button.

##### Examples:

[ex\\_buttonpressed.nxc](#), and [ex\\_xg1300.nxc](#).

#### 6.104.2.8 #define BTNRIGHT BTN2

The right button.

##### Examples:

[ex\\_buttonpressed.nxc](#), [ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

#### 6.104.2.9 #define NO\_OF\_BTNS 4

The number of NXT buttons.

## 6.105 ButtonState constants

Constants for use with the [ButtonState\(\)](#) function.

### Defines

- #define [BTNSTATE\\_PRESSED\\_EV](#) 0x01
- #define [BTNSTATE\\_SHORT\\_RELEASED\\_EV](#) 0x02
- #define [BTNSTATE\\_LONG\\_PRESSED\\_EV](#) 0x04
- #define [BTNSTATE\\_LONG\\_RELEASED\\_EV](#) 0x08
- #define [BTNSTATE\\_PRESSED\\_STATE](#) 0x80
- #define [BTNSTATE\\_NONE](#) 0x10

### 6.105.1 Detailed Description

Constants for use with the [ButtonState\(\)](#) function. The `_EV` values can be combined together using a bitwise OR operation.

See also:

[ButtonState\(\)](#)

### 6.105.2 Define Documentation

#### 6.105.2.1 `#define BTNSTATE_LONG_PRESSED_EV 0x04`

Button is in the long pressed state.

Examples:

[ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

#### 6.105.2.2 `#define BTNSTATE_LONG_RELEASED_EV 0x08`

Button is in the long released state.

#### 6.105.2.3 `#define BTNSTATE_NONE 0x10`

The default button state.

#### 6.105.2.4 `#define BTNSTATE_PRESSED_EV 0x01`

Button is in the pressed state.

Examples:

[ex\\_SetButtonState.nxc](#).

#### 6.105.2.5 `#define BTNSTATE_PRESSED_STATE 0x80`

A bitmask for the button pressed state

#### 6.105.2.6 `#define BTNSTATE_SHORT_RELEASED_EV 0x02`

Button is in the short released state.

## 6.106 Button module IOMAP offsets

Constant offsets into the Button module IOMAP structure.

### Defines

- #define `ButtonOffsetPressedCnt(b)`  $((b)*8)+0$
- #define `ButtonOffsetLongPressCnt(b)`  $((b)*8)+1$
- #define `ButtonOffsetShortRelCnt(b)`  $((b)*8)+2$
- #define `ButtonOffsetLongRelCnt(b)`  $((b)*8)+3$
- #define `ButtonOffsetRelCnt(b)`  $((b)*8)+4$
- #define `ButtonOffsetState(b)`  $((b)+32)$

### 6.106.1 Detailed Description

Constant offsets into the Button module IOMAP structure.

### 6.106.2 Define Documentation

#### 6.106.2.1 #define `ButtonOffsetLongPressCnt(b)` $((b)*8)+1$

Offset to the LongPressCnt field. This field stores the long press count.

#### 6.106.2.2 #define `ButtonOffsetLongRelCnt(b)` $((b)*8)+3$

Offset to the LongRelCnt field. This field stores the long release count.

#### 6.106.2.3 #define `ButtonOffsetPressedCnt(b)` $((b)*8)+0$

Offset to the PressedCnt field. This field stores the press count.

#### 6.106.2.4 #define `ButtonOffsetRelCnt(b)` $((b)*8)+4$

Offset to the RelCnt field. This field stores the release count.

#### 6.106.2.5 #define `ButtonOffsetShortRelCnt(b)` $((b)*8)+2$

Offset to the ShortRelCnt field. This field stores the short release count.

### 6.106.2.6 #define ButtonOffsetState(b) ((b)+32)

Offset to the State field. This field stores the current button state.

## 6.107 Ui module constants

Constants that are part of the NXT firmware's Ui module.

### Modules

- [CommandFlags constants](#)  
*Constants for use with the [CommandFlags\(\)](#) function.*
- [UIState constants](#)  
*Constants for use with the [UIState\(\)](#) function.*
- [UIButton constants](#)  
*Constants for use with the [UIButton\(\)](#) function.*
- [BluetoothState constants](#)  
*Constants for use with the [BluetoothState\(\)](#) function.*
- [VM run state constants](#)  
*Constants for use with the [VMRunState\(\)](#) function.*
- [Ui module IOMAP offsets](#)  
*Constant offsets into the Ui module IOMAP structure.*

### 6.107.1 Detailed Description

Constants that are part of the NXT firmware's Ui module.

## 6.108 CommandFlags constants

Constants for use with the [CommandFlags\(\)](#) function.

### Defines

- #define [UI\\_FLAGS\\_UPDATE](#) 0x01
- #define [UI\\_FLAGS\\_DISABLE\\_LEFT\\_RIGHT\\_ENTER](#) 0x02

- #define `UI_FLAGS_DISABLE_EXIT` 0x04
- #define `UI_FLAGS_REDRAW_STATUS` 0x08
- #define `UI_FLAGS_RESET_SLEEP_TIMER` 0x10
- #define `UI_FLAGS_EXECUTE_LMS_FILE` 0x20
- #define `UI_FLAGS_BUSY` 0x40
- #define `UI_FLAGS_ENABLE_STATUS_UPDATE` 0x80

### 6.108.1 Detailed Description

Constants for use with the `CommandFlags()` function.

See also:

[CommandFlags\(\)](#)

### 6.108.2 Define Documentation

#### 6.108.2.1 #define `UI_FLAGS_BUSY` 0x40

R - UI busy running or datalogging (popup disabled)

#### 6.108.2.2 #define `UI_FLAGS_DISABLE_EXIT` 0x04

RW - Disable exit button

#### 6.108.2.3 #define `UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER` 0x02

RW - Disable left, right and enter button

#### 6.108.2.4 #define `UI_FLAGS_ENABLE_STATUS_UPDATE` 0x80

W - Enable status line to be updated

#### 6.108.2.5 #define `UI_FLAGS_EXECUTE_LMS_FILE` 0x20

W - Execute LMS file in "LMSfilename" (Try It)

#### 6.108.2.6 #define `UI_FLAGS_REDRAW_STATUS` 0x08

W - Redraw entire status line

**Examples:**

[ex\\_SetCommandFlags.nxc](#).

**6.108.2.7 #define UI\_FLAGS\_RESET\_SLEEP\_TIMER 0x10**

W - Reset sleep timeout timer

**6.108.2.8 #define UI\_FLAGS\_UPDATE 0x01**

W - Make changes take effect

**6.109 UIState constants**

Constants for use with the [UIState\(\)](#) function.

**Defines**

- #define [UI\\_STATE\\_INIT\\_DISPLAY](#) 0
- #define [UI\\_STATE\\_INIT\\_LOW\\_BATTERY](#) 1
- #define [UI\\_STATE\\_INIT\\_INTRO](#) 2
- #define [UI\\_STATE\\_INIT\\_WAIT](#) 3
- #define [UI\\_STATE\\_INIT\\_MENU](#) 4
- #define [UI\\_STATE\\_NEXT\\_MENU](#) 5
- #define [UI\\_STATE\\_DRAW\\_MENU](#) 6
- #define [UI\\_STATE\\_TEST\\_BUTTONS](#) 7
- #define [UI\\_STATE\\_LEFT\\_PRESSED](#) 8
- #define [UI\\_STATE\\_RIGHT\\_PRESSED](#) 9
- #define [UI\\_STATE\\_ENTER\\_PRESSED](#) 10
- #define [UI\\_STATE\\_EXIT\\_PRESSED](#) 11
- #define [UI\\_STATE\\_CONNECT\\_REQUEST](#) 12
- #define [UI\\_STATE\\_EXECUTE\\_FILE](#) 13
- #define [UI\\_STATE\\_EXECUTING\\_FILE](#) 14
- #define [UI\\_STATE\\_LOW\\_BATTERY](#) 15
- #define [UI\\_STATE\\_BT\\_ERROR](#) 16

**6.109.1 Detailed Description**

Constants for use with the [UIState\(\)](#) function.

**See also:**

[UIState\(\)](#)

**6.109.2 Define Documentation****6.109.2.1 #define UI\_STATE\_BT\_ERROR 16**

R - BT error

**6.109.2.2 #define UI\_STATE\_CONNECT\_REQUEST 12**

RW - Request for connection accept

**6.109.2.3 #define UI\_STATE\_DRAW\_MENU 6**

RW - Execute function and draw menu icons

**6.109.2.4 #define UI\_STATE\_ENTER\_PRESSED 10**

RW - Load selected function and next menu id

**6.109.2.5 #define UI\_STATE\_EXECUTE\_FILE 13**

RW - Execute file in "LMSfilename"

**6.109.2.6 #define UI\_STATE\_EXECUTING\_FILE 14**

R - Executing file in "LMSfilename"

**6.109.2.7 #define UI\_STATE\_EXIT\_PRESSED 11**

RW - Load selected function and next menu id

**6.109.2.8 #define UI\_STATE\_INIT\_DISPLAY 0**

RW - Init display and load font, menu etc.

**6.109.2.9 #define UI\_STATE\_INIT\_INTRO 2**

R - Display intro

**6.109.2.10 #define UI\_STATE\_INIT\_LOW\_BATTERY 1**

R - Low battery voltage at power on

**6.109.2.11 #define UI\_STATE\_INIT\_MENU 4**

RW - Init menu system

**6.109.2.12 #define UI\_STATE\_INIT\_WAIT 3**

RW - Wait for initialization end

**6.109.2.13 #define UI\_STATE\_LEFT\_PRESSED 8**

RW - Load selected function and next menu id

**6.109.2.14 #define UI\_STATE\_LOW\_BATTERY 15**

R - Low battery at runtime

**Examples:**[ex\\_SetUIState.nxc](#).**6.109.2.15 #define UI\_STATE\_NEXT\_MENU 5**

RW - Next menu icons ready for drawing

**6.109.2.16 #define UI\_STATE\_RIGHT\_PRESSED 9**

RW - Load selected function and next menu id

**6.109.2.17 #define UI\_STATE\_TEST\_BUTTONS 7**

RW - Wait for buttons to be pressed

**6.110 UIButton constants**Constants for use with the [UIButton\(\)](#) function.

## Defines

- #define [UI\\_BUTTON\\_NONE](#) 0
- #define [UI\\_BUTTON\\_LEFT](#) 1
- #define [UI\\_BUTTON\\_ENTER](#) 2
- #define [UI\\_BUTTON\\_RIGHT](#) 3
- #define [UI\\_BUTTON\\_EXIT](#) 4

### 6.110.1 Detailed Description

Constants for use with the [UIButton\(\)](#) function.

See also:

[UIButton\(\)](#)

### 6.110.2 Define Documentation

#### 6.110.2.1 #define [UI\\_BUTTON\\_ENTER](#) 2

W - Insert enter button

Examples:

[ex\\_SetUIButton.nxc](#).

#### 6.110.2.2 #define [UI\\_BUTTON\\_EXIT](#) 4

W - Insert exit button

#### 6.110.2.3 #define [UI\\_BUTTON\\_LEFT](#) 1

W - Insert left arrow button

#### 6.110.2.4 #define [UI\\_BUTTON\\_NONE](#) 0

R - Button inserted are executed

#### 6.110.2.5 #define [UI\\_BUTTON\\_RIGHT](#) 3

W - Insert right arrow button

## 6.111 BluetoothState constants

Constants for use with the [BluetoothState\(\)](#) function.

### Defines

- #define [UI\\_BT\\_STATE\\_VISIBLE](#) 0x01
- #define [UI\\_BT\\_STATE\\_CONNECTED](#) 0x02
- #define [UI\\_BT\\_STATE\\_OFF](#) 0x04
- #define [UI\\_BT\\_ERROR\\_ATTENTION](#) 0x08
- #define [UI\\_BT\\_CONNECT\\_REQUEST](#) 0x40
- #define [UI\\_BT\\_PIN\\_REQUEST](#) 0x80

### 6.111.1 Detailed Description

Constants for use with the [BluetoothState\(\)](#) function.

See also:

[BluetoothState\(\)](#)

### 6.111.2 Define Documentation

#### 6.111.2.1 #define [UI\\_BT\\_CONNECT\\_REQUEST](#) 0x40

RW - BT get connect accept in progress

#### 6.111.2.2 #define [UI\\_BT\\_ERROR\\_ATTENTION](#) 0x08

W - BT error attention

#### 6.111.2.3 #define [UI\\_BT\\_PIN\\_REQUEST](#) 0x80

RW - BT get pin code

#### 6.111.2.4 #define [UI\\_BT\\_STATE\\_CONNECTED](#) 0x02

RW - BT connected to something

**6.111.2.5 #define UI\_BT\_STATE\_OFF 0x04**

RW - BT power off

**Examples:**[ex\\_SetBluetoothState.nxc](#).**6.111.2.6 #define UI\_BT\_STATE\_VISIBLE 0x01**

RW - BT visible

**6.112 VM run state constants**Constants for use with the [VMRunState\(\)](#) function.**Defines**

- #define [UI\\_VM\\_IDLE](#) 0
- #define [UI\\_VM\\_RUN\\_FREE](#) 1
- #define [UI\\_VM\\_RUN\\_SINGLE](#) 2
- #define [UI\\_VM\\_RUN\\_PAUSE](#) 3
- #define [UI\\_VM\\_RESET1](#) 4
- #define [UI\\_VM\\_RESET2](#) 5

**6.112.1 Detailed Description**Constants for use with the [VMRunState\(\)](#) function.**See also:**[VMRunState\(\)](#)**6.112.2 Define Documentation****6.112.2.1 #define UI\_VM\_IDLE 0**

VM\_IDLE: Just sitting around. Request to run program will lead to ONE of the VM\_RUN\* states.

**6.112.2.2 #define UI\_VM\_RESET1 4**

VM\_RESET1: Initialize state variables and some I/O devices -- executed when programs end

**6.112.2.3 #define UI\_VM\_RESET2 5**

VM\_RESET2: Final clean up and return to IDLE

**6.112.2.4 #define UI\_VM\_RUN\_FREE 1**

VM\_RUN\_FREE: Attempt to run as many instructions as possible within our timeslice

**6.112.2.5 #define UI\_VM\_RUN\_PAUSE 3**

VM\_RUN\_PAUSE: Program still "active", but someone has asked us to pause

**6.112.2.6 #define UI\_VM\_RUN\_SINGLE 2**

VM\_RUN\_SINGLE: Run exactly one instruction per timeslice

**6.113 Ui module IOMAP offsets**

Constant offsets into the Ui module IOMAP structure.

**Defines**

- #define [UIOffsetPMenu](#) 0
- #define [UIOffsetBatteryVoltage](#) 4
- #define [UIOffsetLMSfilename](#) 6
- #define [UIOffsetFlags](#) 26
- #define [UIOffsetState](#) 27
- #define [UIOffsetButton](#) 28
- #define [UIOffsetRunState](#) 29
- #define [UIOffsetBatteryState](#) 30
- #define [UIOffsetBluetoothState](#) 31
- #define [UIOffsetUsbState](#) 32
- #define [UIOffsetSleepTimeout](#) 33
- #define [UIOffsetSleepTimer](#) 34

- #define [UIOffsetRechargeable](#) 35
- #define [UIOffsetVolume](#) 36
- #define [UIOffsetError](#) 37
- #define [UIOffsetOBPPointer](#) 38
- #define [UIOffsetForceOff](#) 39
- #define [UIOffsetAbortFlag](#) 40

### 6.113.1 Detailed Description

Constant offsets into the Ui module IOMAP structure.

### 6.113.2 Define Documentation

#### 6.113.2.1 #define [UIOffsetAbortFlag](#) 40

RW - Long Abort (true == use long press to abort) (1 byte)

#### 6.113.2.2 #define [UIOffsetBatteryState](#) 30

W - Battery state (0..4 capacity) (1 byte)

#### 6.113.2.3 #define [UIOffsetBatteryVoltage](#) 4

R - Battery voltage in millivolts (2 bytes)

#### 6.113.2.4 #define [UIOffsetBluetoothState](#) 31

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)

#### 6.113.2.5 #define [UIOffsetButton](#) 28

RW - Insert button (buttons enumerated above) (1 byte)

#### 6.113.2.6 #define [UIOffsetError](#) 37

W - Error code (1 byte)

#### 6.113.2.7 #define [UIOffsetFlags](#) 26

RW - Update command flags (flags enumerated above) (1 byte)

**6.113.2.8 #define UIOffsetForceOff 39**

W - Force off (&gt; 0 = off) (1 byte)

**6.113.2.9 #define UIOffsetLMSfilename 6**

W - LMS filename to execute (Try It) (20 bytes)

**6.113.2.10 #define UIOffsetOBPPointer 38**

W - Actual OBP step (0 - 4) (1 byte)

**6.113.2.11 #define UIOffsetPMenu 0**

W - Pointer to menu file (4 bytes)

**6.113.2.12 #define UIOffsetRechargeable 35**

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

**6.113.2.13 #define UIOffsetRunState 29**

W - VM Run state (0 = stopped, 1 = running) (1 byte)

**6.113.2.14 #define UIOffsetSleepTimeout 33**

RW - Sleep timeout time (min) (1 byte)

**6.113.2.15 #define UIOffsetSleepTimer 34**

RW - Sleep timer (min) (1 byte)

**6.113.2.16 #define UIOffsetState 27**

RW - UI state (states enumerated above) (1 byte)

**6.113.2.17 #define UIOffsetUsbState 32**

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

**6.113.2.18 #define UIOffsetVolume 36**

RW - Volume used in UI (0 - 4) (1 byte)

**6.114 NBC Input port constants**

Input port constants are used when calling sensor control API functions.

**Defines**

- #define [IN\\_1](#) 0x00
- #define [IN\\_2](#) 0x01
- #define [IN\\_3](#) 0x02
- #define [IN\\_4](#) 0x03

**6.114.1 Detailed Description**

Input port constants are used when calling sensor control API functions. These constants are intended for use in NBC.

**See also:**

[SetSensorType\(\)](#), [SetSensorMode\(\)](#), [S1](#), [S2](#), [S3](#), [S4](#)

**6.114.2 Define Documentation****6.114.2.1 #define IN\_1 0x00**

Input port 1

**6.114.2.2 #define IN\_2 0x01**

Input port 2

**6.114.2.3 #define IN\_3 0x02**

Input port 3

**6.114.2.4 #define IN\_4 0x03**

Input port 4

## 6.115 NBC sensor type constants

Use sensor type constants to configure an input port for a specific type of sensor.

### Defines

- #define [IN\\_TYPE\\_NO\\_SENSOR](#) 0x00
- #define [IN\\_TYPE\\_SWITCH](#) 0x01
- #define [IN\\_TYPE\\_TEMPERATURE](#) 0x02
- #define [IN\\_TYPE\\_REFLECTION](#) 0x03
- #define [IN\\_TYPE\\_ANGLE](#) 0x04
- #define [IN\\_TYPE\\_LIGHT\\_ACTIVE](#) 0x05
- #define [IN\\_TYPE\\_LIGHT\\_INACTIVE](#) 0x06
- #define [IN\\_TYPE\\_SOUND\\_DB](#) 0x07
- #define [IN\\_TYPE\\_SOUND\\_DBA](#) 0x08
- #define [IN\\_TYPE\\_CUSTOM](#) 0x09
- #define [IN\\_TYPE\\_LOWSPEED](#) 0x0A
- #define [IN\\_TYPE\\_LOWSPEED\\_9V](#) 0x0B
- #define [IN\\_TYPE\\_HISPEED](#) 0x0C
- #define [IN\\_TYPE\\_COLORFULL](#) 0x0D
- #define [IN\\_TYPE\\_COLORRED](#) 0x0E
- #define [IN\\_TYPE\\_COLORGREEN](#) 0x0F
- #define [IN\\_TYPE\\_COLORBLUE](#) 0x10
- #define [IN\\_TYPE\\_COLORNONE](#) 0x11
- #define [IN\\_TYPE\\_COLOREXIT](#) 0x12

### 6.115.1 Detailed Description

Use sensor type constants to configure an input port for a specific type of sensor. These constants are intended for use in NBC.

See also:

[SetSensorType\(\)](#)

### 6.115.2 Define Documentation

#### 6.115.2.1 #define [IN\\_TYPE\\_ANGLE](#) 0x04

RCX rotation sensor

**6.115.2.2 #define IN\_TYPE\_COLORBLUE 0x10**

NXT 2.0 color sensor with blue light

**6.115.2.3 #define IN\_TYPE\_COLOREXIT 0x12**

NXT 2.0 color sensor internal state

**6.115.2.4 #define IN\_TYPE\_COLORFULL 0x0D**

NXT 2.0 color sensor in full color mode

**6.115.2.5 #define IN\_TYPE\_COLORGREEN 0x0F**

NXT 2.0 color sensor with green light

**6.115.2.6 #define IN\_TYPE\_COLORNONE 0x11**

NXT 2.0 color sensor with no light

**6.115.2.7 #define IN\_TYPE\_COLORRED 0x0E**

NXT 2.0 color sensor with red light

**6.115.2.8 #define IN\_TYPE\_CUSTOM 0x09**

NXT custom sensor

**6.115.2.9 #define IN\_TYPE\_HISPEED 0x0C**

NXT Hi-speed port (only S4)

**6.115.2.10 #define IN\_TYPE\_LIGHT\_ACTIVE 0x05**

NXT light sensor with light

**6.115.2.11 #define IN\_TYPE\_LIGHT\_INACTIVE 0x06**

NXT light sensor without light

**6.115.2.12 #define IN\_TYPE\_LOWSPEED 0x0A**

NXT I2C digital sensor

**6.115.2.13 #define IN\_TYPE\_LOWSPEED\_9V 0x0B**

NXT I2C digital sensor with 9V power

**6.115.2.14 #define IN\_TYPE\_NO\_SENSOR 0x00**

No sensor configured

**6.115.2.15 #define IN\_TYPE\_REFLECTION 0x03**

RCX light sensor

**6.115.2.16 #define IN\_TYPE\_SOUND\_DB 0x07**

NXT sound sensor with dB scaling

**6.115.2.17 #define IN\_TYPE\_SOUND\_DBA 0x08**

NXT sound sensor with dBA scaling

**6.115.2.18 #define IN\_TYPE\_SWITCH 0x01**

NXT or RCX touch sensor

**6.115.2.19 #define IN\_TYPE\_TEMPERATURE 0x02**

RCX temperature sensor

## 6.116 NBC sensor mode constants

Use sensor mode constants to configure an input port for the desired sensor mode.

## Defines

- #define [IN\\_MODE\\_RAW](#) 0x00
- #define [IN\\_MODE\\_BOOLEAN](#) 0x20
- #define [IN\\_MODE\\_TRANSITIONCNT](#) 0x40
- #define [IN\\_MODE\\_PERIODCOUNTER](#) 0x60
- #define [IN\\_MODE\\_PCTFULLSCALE](#) 0x80
- #define [IN\\_MODE\\_CELSIUS](#) 0xA0
- #define [IN\\_MODE\\_FAHRENHEIT](#) 0xC0
- #define [IN\\_MODE\\_ANGLESTEP](#) 0xE0
- #define [IN\\_MODE\\_SLOPEMASK](#) 0x1F
- #define [IN\\_MODE\\_MODEMASK](#) 0xE0

### 6.116.1 Detailed Description

Use sensor mode constants to configure an input port for the desired sensor mode. The constants are intended for use in NBC.

#### See also:

[SetSensorMode\(\)](#)

### 6.116.2 Define Documentation

#### 6.116.2.1 #define [IN\\_MODE\\_ANGLESTEP](#) 0xE0

RCX rotation sensor (16 ticks per revolution)

#### 6.116.2.2 #define [IN\\_MODE\\_BOOLEAN](#) 0x20

Boolean value (0 or 1)

#### 6.116.2.3 #define [IN\\_MODE\\_CELSIUS](#) 0xA0

RCX temperature sensor value in degrees celcius

#### 6.116.2.4 #define [IN\\_MODE\\_FAHRENHEIT](#) 0xC0

RCX temperature sensor value in degrees fahrenheit

**6.116.2.5 #define IN\_MODE\_MODEMASK 0xE0**

Mask for the mode without any slope value

**6.116.2.6 #define IN\_MODE\_PCTFULLSCALE 0x80**

Scaled value from 0 to 100

**6.116.2.7 #define IN\_MODE\_PERIODCOUNTER 0x60**

Counts the number of boolean periods

**6.116.2.8 #define IN\_MODE\_RAW 0x00**

Raw value from 0 to 1023

**6.116.2.9 #define IN\_MODE\_SLOPEMASK 0x1F**

Mask for slope parameter added to mode

**6.116.2.10 #define IN\_MODE\_TRANSITIONCNT 0x40**

Counts the number of boolean transitions

**6.117 Input field constants**

Constants for use with [SetInput\(\)](#) and [GetInput\(\)](#).

**Defines**

- #define [TypeField](#) 0
- #define [InputModeField](#) 1
- #define [RawValueField](#) 2
- #define [NormalizedValueField](#) 3
- #define [ScaledValueField](#) 4
- #define [InvalidDataField](#) 5

### 6.117.1 Detailed Description

Constants for use with [SetInput\(\)](#) and [GetInput\(\)](#). Each sensor has six fields that are used to define its state.

### 6.117.2 Define Documentation

#### 6.117.2.1 #define InputModeField 1

Input mode field. Contains one of the sensor mode constants. Read/write.

#### 6.117.2.2 #define InvalidDataField 5

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

#### 6.117.2.3 #define NormalizedValueField 3

Normalized value field. Contains the current normalized analog sensor value. Read only.

#### 6.117.2.4 #define RawValueField 2

Raw value field. Contains the current raw analog sensor value. Read only.

#### 6.117.2.5 #define ScaledValueField 4

Scaled value field. Contains the current scaled analog sensor value. Read/write.

#### 6.117.2.6 #define TypeField 0

Type field. Contains one of the sensor type constants. Read/write.

## 6.118 Input port digital pin constants

Constants for use when directly controlling or reading a port's digital pin state.

### Defines

- #define [INPUT\\_DIGIO](#) 0x01

- #define [INPUT\\_DIGI1](#) 0x02

### 6.118.1 Detailed Description

Constants for use when directly controlling or reading a port's digital pin state.

### 6.118.2 Define Documentation

#### 6.118.2.1 #define [INPUT\\_DIGI0](#) 0x01

Digital pin 0

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

#### 6.118.2.2 #define [INPUT\\_DIGI1](#) 0x02

Digital pin 1

## 6.119 Color sensor array indices

Constants for use with color sensor value arrays to index RGB and blank return values.

### Defines

- #define [INPUT\\_RED](#) 0
- #define [INPUT\\_GREEN](#) 1
- #define [INPUT\\_BLUE](#) 2
- #define [INPUT\\_BLANK](#) 3
- #define [INPUT\\_NO\\_OF\\_COLORS](#) 4

### 6.119.1 Detailed Description

Constants for use with color sensor value arrays to index RGB and blank return values.

#### See also:

[ReadSensorColorEx\(\)](#), [ReadSensorColorRaw\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

### 6.119.2 Define Documentation

#### 6.119.2.1 #define INPUT\_BLANK 3

Access the blank value from color sensor value arrays

#### 6.119.2.2 #define INPUT\_BLUE 2

Access the blue value from color sensor value arrays

#### 6.119.2.3 #define INPUT\_GREEN 1

Access the green value from color sensor value arrays

#### 6.119.2.4 #define INPUT\_NO\_OF\_COLORS 4

The number of entries in the color sensor value arrays

#### 6.119.2.5 #define INPUT\_RED 0

Access the red value from color sensor value arrays

#### Examples:

[ex\\_ColorADRaw.nxc](#), [ex\\_ColorBoolean.nxc](#), [ex\\_ColorCalibration.nxc](#), [ex\\_ColorSensorRaw.nxc](#), and [ex\\_ColorSensorValue.nxc](#).

## 6.120 Color values

Constants for use with the ColorValue returned by the color sensor in full color mode.

#### Defines

- #define [INPUT\\_BLACKCOLOR](#) 1
- #define [INPUT\\_BLUECOLOR](#) 2
- #define [INPUT\\_GREENCOLOR](#) 3
- #define [INPUT\\_YELLOWCOLOR](#) 4
- #define [INPUT\\_REDCOLOR](#) 5
- #define [INPUT\\_WHITECOLOR](#) 6

### 6.120.1 Detailed Description

Constants for use with the ColorValue returned by the color sensor in full color mode.

See also:

[SensorValue\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

### 6.120.2 Define Documentation

#### 6.120.2.1 #define INPUT\_BLACKCOLOR 1

The color value is black

#### 6.120.2.2 #define INPUT\_BLUECOLOR 2

The color value is blue

#### 6.120.2.3 #define INPUT\_GREENCOLOR 3

The color value is green

#### 6.120.2.4 #define INPUT\_REDCOLOR 5

The color value is red

#### 6.120.2.5 #define INPUT\_WHITECOLOR 6

The color value is white

#### 6.120.2.6 #define INPUT\_YELLOWCOLOR 4

The color value is yellow

## 6.121 Color calibration state constants

Constants for use with the color calibration state function.

### Defines

- #define [INPUT\\_SENSORCAL](#) 0x01
- #define [INPUT\\_SENSOROFF](#) 0x02
- #define [INPUT\\_RUNNINGCAL](#) 0x20
- #define [INPUT\\_STARTCAL](#) 0x40
- #define [INPUT\\_RESETCAL](#) 0x80

#### 6.121.1 Detailed Description

Constants for use with the color calibration state function.

See also:

[ColorCalibrationState\(\)](#)

#### 6.121.2 Define Documentation

##### 6.121.2.1 #define INPUT\_RESETCAL 0x80

Unused calibration state constant

##### 6.121.2.2 #define INPUT\_RUNNINGCAL 0x20

Unused calibration state constant

##### 6.121.2.3 #define INPUT\_SENSORCAL 0x01

The state returned while the color sensor is calibrating

##### 6.121.2.4 #define INPUT\_SENSOROFF 0x02

The state returned once calibration has completed

##### 6.121.2.5 #define INPUT\_STARTCAL 0x40

Unused calibration state constant

### 6.122 Color calibration constants

Constants for use with the color calibration functions.

**Defines**

- `#define INPUT_CAL_POINT_0 0`
- `#define INPUT_CAL_POINT_1 1`
- `#define INPUT_CAL_POINT_2 2`
- `#define INPUT_NO_OF_POINTS 3`

**6.122.1 Detailed Description**

Constants for use with the color calibration functions.

**See also:**

[ColorCalibration\(\)](#), [ColorCalLimits\(\)](#)

**6.122.2 Define Documentation****6.122.2.1 `#define INPUT_CAL_POINT_0 0`**

Calibration point 0

**Examples:**

[ex\\_ColorCalibration.nxc](#), and [ex\\_ColorCalLimits.nxc](#).

**6.122.2.2 `#define INPUT_CAL_POINT_1 1`**

Calibration point 1

**6.122.2.3 `#define INPUT_CAL_POINT_2 2`**

Calibration point 2

**6.122.2.4 `#define INPUT_NO_OF_POINTS 3`**

The number of calibration points

**6.123 Input module IOMAP offsets**

Constant offsets into the Input module IOMAP structure.

**Defines**

- #define `InputOffsetCustomZeroOffset(p)`  $((p)*20)+0$
- #define `InputOffsetADRaw(p)`  $((p)*20)+2$
- #define `InputOffsetSensorRaw(p)`  $((p)*20)+4$
- #define `InputOffsetSensorValue(p)`  $((p)*20)+6$
- #define `InputOffsetSensorType(p)`  $((p)*20)+8$
- #define `InputOffsetSensorMode(p)`  $((p)*20)+9$
- #define `InputOffsetSensorBoolean(p)`  $((p)*20)+10$
- #define `InputOffsetDigiPinsDir(p)`  $((p)*20)+11$
- #define `InputOffsetDigiPinsIn(p)`  $((p)*20)+12$
- #define `InputOffsetDigiPinsOut(p)`  $((p)*20)+13$
- #define `InputOffsetCustomPctFullScale(p)`  $((p)*20)+14$
- #define `InputOffsetCustomActiveStatus(p)`  $((p)*20)+15$
- #define `InputOffsetInvalidData(p)`  $((p)*20)+16$
- #define `InputOffsetColorCalibration(p, np, nc)`  $(80+((p)*84)+0+((np)*16)+((nc)*4))$
- #define `InputOffsetColorCalLimits(p, np)`  $(80+((p)*84)+48+((np)*2))$
- #define `InputOffsetColorADRaw(p, nc)`  $(80+((p)*84)+52+((nc)*2))$
- #define `InputOffsetColorSensorRaw(p, nc)`  $(80+((p)*84)+60+((nc)*2))$
- #define `InputOffsetColorSensorValue(p, nc)`  $(80+((p)*84)+68+((nc)*2))$
- #define `InputOffsetColorBoolean(p, nc)`  $(80+((p)*84)+76+((nc)*2))$
- #define `InputOffsetColorCalibrationState(p)`  $(80+((p)*84)+80)$

**6.123.1 Detailed Description**

Constant offsets into the Input module IOMAP structure.

**6.123.2 Define Documentation****6.123.2.1 #define InputOffsetADRaw(p)  $((p)*20)+2$** 

Read the AD raw sensor value (2 bytes) uword

**6.123.2.2 #define InputOffsetColorADRaw(p, nc)  $(80+((p)*84)+52+((nc)*2)$** 

Read AD raw color sensor values

**6.123.2.3 #define InputOffsetColorBoolean(p, nc)  $(80+((p)*84)+76+((nc)*2)$** 

Read color sensor boolean values

**6.123.2.4 #define InputOffsetColorCalibration(p, np, nc) (80+((p)\*84)+0+((np)\*16)+((nc)\*4))**

Read/write color calibration point values

**6.123.2.5 #define InputOffsetColorCalibrationState(p) (80+((p)\*84)+80)**

Read color sensor calibration state

**6.123.2.6 #define InputOffsetColorCalLimits(p, np) (80+((p)\*84)+48+((np)\*2))**

Read/write color calibration limits

**6.123.2.7 #define InputOffsetColorSensorRaw(p, nc) (80+((p)\*84)+60+((nc)\*2))**

Read raw color sensor values

**6.123.2.8 #define InputOffsetColorSensorValue(p, nc) (80+((p)\*84)+68+((nc)\*2))**

Read scaled color sensor values

**6.123.2.9 #define InputOffsetCustomActiveStatus(p) (((p)\*20)+15)**

Read/write the active or inactive state of the custom sensor

**6.123.2.10 #define InputOffsetCustomPctFullScale(p) (((p)\*20)+14)**

Read/write the Pct full scale of the custom sensor

**6.123.2.11 #define InputOffsetCustomZeroOffset(p) (((p)\*20)+0)**

Read/write the zero offset of a custom sensor (2 bytes) uword

**6.123.2.12 #define InputOffsetDigiPinsDir(p) (((p)\*20)+11)**

Read/write the direction of the Digital pins (1 is output, 0 is input)

**6.123.2.13 #define InputOffsetDigiPinsIn(p) (((p)\*20)+12)**

Read/write the status of the digital pins

**6.123.2.14 #define InputOffsetDigiPinsOut(p) (((p)\*20)+13)**

Read/write the output level of the digital pins

**6.123.2.15 #define InputOffsetInvalidData(p) (((p)\*20)+16)**

Indicates whether data is invalid (1) or valid (0)

**6.123.2.16 #define InputOffsetSensorBoolean(p) (((p)\*20)+10)**

Read the sensor boolean value

**6.123.2.17 #define InputOffsetSensorMode(p) (((p)\*20)+9)**

Read/write the sensor mode

**6.123.2.18 #define InputOffsetSensorRaw(p) (((p)\*20)+4)**

Read the raw sensor value (2 bytes) uword

**6.123.2.19 #define InputOffsetSensorType(p) (((p)\*20)+8)**

Read/write the sensor type

**6.123.2.20 #define InputOffsetSensorValue(p) (((p)\*20)+6)**

Read/write the scaled sensor value (2 bytes) sword

**6.124 Constants to use with the Input module's Pin function**

Constants for use with the Input module's Pin function.

## Defines

- #define `INPUT_PINCMD_DIR` 0x00
- #define `INPUT_PINCMD_SET` 0x01
- #define `INPUT_PINCMD_CLEAR` 0x02
- #define `INPUT_PINCMD_READ` 0x03
- #define `INPUT_PINCMD_MASK` 0x03
- #define `INPUT_PINCMD_WAIT`(\_usec) ((\_usec)<<2)
- #define `INPUT_PINDIR_OUTPUT` 0x00
- #define `INPUT_PINDIR_INPUT` 0x04

### 6.124.1 Detailed Description

Constants for use with the Input module's Pin function. These are the commands that you can pass into the pin function to change digital pin directions, set or clear pins, or read pin values. Also in this group are mask constants and a macro for ORing a microsecond wait onto the command byte which will occur after the command has been executed.

### 6.124.2 Define Documentation

#### 6.124.2.1 #define `INPUT_PINCMD_CLEAR` 0x02

Clear digital pin(s)

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

#### 6.124.2.2 #define `INPUT_PINCMD_DIR` 0x00

Set digital pin(s) direction

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

#### 6.124.2.3 #define `INPUT_PINCMD_MASK` 0x03

Mask for the two bits used by pin function commands

**6.124.2.4 #define INPUT\_PINCMD\_READ 0x03**

Read digital pin(s)

**6.124.2.5 #define INPUT\_PINCMD\_SET 0x01**

Set digital pin(s)

**Examples:**[ex\\_sysinputpinfunction.nxc.](#)**6.124.2.6 #define INPUT\_PINCMD\_WAIT(\_usec) ((\_usec)<<2)**

A wait value in microseconds that can be added after one of the above commands by ORing with the command

**Examples:**[ex\\_sysinputpinfunction.nxc.](#)**6.124.2.7 #define INPUT\_PINDIR\_INPUT 0x04**

Use with the direction command to set direction to output. OR this with the pin value.

**6.124.2.8 #define INPUT\_PINDIR\_OUTPUT 0x00**

Use with the direction command to set direction to input. OR this with the pin value.

**Examples:**[ex\\_sysinputpinfunction.nxc.](#)**6.125 Output port constants**

Output port constants are used when calling motor control API functions.

**Defines**

- #define `OUT_A` 0x00
- #define `OUT_B` 0x01
- #define `OUT_C` 0x02
- #define `OUT_AB` 0x03
- #define `OUT_AC` 0x04
- #define `OUT_BC` 0x05
- #define `OUT_ABC` 0x06

**6.125.1 Detailed Description**

Output port constants are used when calling motor control API functions.

**6.125.2 Define Documentation****6.125.2.1 #define OUT\_A 0x00**

Output port A

**Examples:**

`ex_coast.nxc`, `ex_coastex.nxc`, `ex_float.nxc`, `ex_getoutput.nxc`, `ex_-motoractualspeed.nxc`, `ex_motorblocktachocount.nxc`, `ex_motormode.nxc`, `ex_motoroutputoptions.nxc`, `ex_motoroverload.nxc`, `ex_motorpower.nxc`, `ex_motorregdvalue.nxc`, `ex_motorregivalue.nxc`, `ex_motorregpvalue.nxc`, `ex_motorregulation.nxc`, `ex_motorrotationcount.nxc`, `ex_motorruntime.nxc`, `ex_motortachocount.nxc`, `ex_motortacholimit.nxc`, `ex_motorturnratio.nxc`, `ex_off.nxc`, `ex_offex.nxc`, `ex_onfwd.nxc`, `ex_onfwdex.nxc`, `ex_onfwdreg.nxc`, `ex_onfwdregex.nxc`, `ex_onfwdregexpid.nxc`, `ex_onfwdregpid.nxc`, `ex_onrev.nxc`, `ex_onrevex.nxc`, `ex_onrevreg.nxc`, `ex_onrevregex.nxc`, `ex_onrevregexpid.nxc`, `ex_onrevregpid.nxc`, `ex_PosReg.nxc`, `ex_RemoteResetMotorPosition.nxc`, `ex_RemoteResetTachoCount.nxc`, `ex_RemoteSetOutputState.nxc`, `ex_-rotatemotor.nxc`, `ex_rotatemotorpid.nxc`, and `ex_yield.nxc`.

**6.125.2.2 #define OUT\_AB 0x03**

Output ports A and B

**Examples:**

`ex_onfwdsync.nxc`, `ex_onfwdsyncex.nxc`, `ex_onfwdsyncexpid.nxc`, `ex_onfwdsyncpid.nxc`, `ex_onrevsync.nxc`, `ex_onrevsyncex.nxc`, `ex_-`

[onrevsyncpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_resetalltachocounts.nxc](#), [ex\\_resetblocktachocount.nxc](#), [ex\\_resetrotationcount.nxc](#), [ex\\_resettachocount.nxc](#), [ex\\_rotatemotorex.nxc](#), [ex\\_rotatemotorexpid.nxc](#), and [ex\\_setoutput.nxc](#).

#### 6.125.2.3 #define OUT\_ABC 0x06

Output ports A, B, and C

#### 6.125.2.4 #define OUT\_AC 0x04

Output ports A and C

#### 6.125.2.5 #define OUT\_B 0x01

Output port B

#### 6.125.2.6 #define OUT\_BC 0x05

Output ports B and C

#### 6.125.2.7 #define OUT\_C 0x02

Output port C

## 6.126 PID constants

PID constants are for adjusting the Proportional, Integral, and Derivative motor controller parameters.

### Defines

- #define [PID\\_0](#) 0
- #define [PID\\_1](#) 32
- #define [PID\\_2](#) 64
- #define [PID\\_3](#) 96
- #define [PID\\_4](#) 128
- #define [PID\\_5](#) 160
- #define [PID\\_6](#) 192
- #define [PID\\_7](#) 224

### 6.126.1 Detailed Description

PID constants are for adjusting the Proportional, Integral, and Derivative motor controller parameters.

See also:

[RotateMotorExPID\(\)](#), [RotateMotorPID\(\)](#), [OnFwdExPID\(\)](#), [OnRevExPID\(\)](#),  
[OnFwdRegExPID\(\)](#), [OnRevRegExPID\(\)](#), [OnFwdRegPID\(\)](#), [OnRevRegPID\(\)](#),  
[OnFwdSyncExPID\(\)](#), [OnRevSyncExPID\(\)](#), [OnFwdSyncPID\(\)](#), [OnRevSyncPID\(\)](#)

### 6.126.2 Define Documentation

#### 6.126.2.1 #define PID\_0 0

PID zero

#### 6.126.2.2 #define PID\_1 32

PID one

#### 6.126.2.3 #define PID\_2 64

PID two

#### 6.126.2.4 #define PID\_3 96

PID three

#### 6.126.2.5 #define PID\_4 128

PID four

#### 6.126.2.6 #define PID\_5 160

PID five

#### 6.126.2.7 #define PID\_6 192

PID six

**6.126.2.8 #define PID\_7 224**

PID seven

**6.127 Output port update flag constants**

Use these constants to specify which motor values need to be updated.

**Defines**

- #define [UF\\_UPDATE\\_MODE](#) 0x01
- #define [UF\\_UPDATE\\_SPEED](#) 0x02
- #define [UF\\_UPDATE\\_TACHO\\_LIMIT](#) 0x04
- #define [UF\\_UPDATE\\_RESET\\_COUNT](#) 0x08
- #define [UF\\_UPDATE\\_PID\\_VALUES](#) 0x10
- #define [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) 0x20
- #define [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) 0x40
- #define [UF\\_PENDING\\_UPDATES](#) 0x80

**6.127.1 Detailed Description**

Use these constants to specify which motor values need to be updated. Update flag constants can be combined with bitwise OR.

**See also:**

[SetOutput\(\)](#)

**6.127.2 Define Documentation****6.127.2.1 #define UF\_PENDING\_UPDATES 0x80**

Are there any pending motor updates?

**6.127.2.2 #define UF\_UPDATE\_MODE 0x01**

Commits changes to the [OutputModeField](#) output property

**6.127.2.3 #define UF\_UPDATE\_PID\_VALUES 0x10**

Commits changes to the PID motor regulation properties

**6.127.2.4 #define UF\_UPDATE\_RESET\_BLOCK\_COUNT 0x20**

Resets the NXT-G block-relative rotation counter

**6.127.2.5 #define UF\_UPDATE\_RESET\_COUNT 0x08**

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

**6.127.2.6 #define UF\_UPDATE\_RESET\_ROTATION\_COUNT 0x40**

Resets the program-relative (user) rotation counter

**6.127.2.7 #define UF\_UPDATE\_SPEED 0x02**

Commits changes to the [PowerField](#) output property

**6.127.2.8 #define UF\_UPDATE\_TACHO\_LIMIT 0x04**

Commits changes to the [TachoLimitField](#) output property

**6.128 Tachometer counter reset flags**

Use these constants to specify which of the three tachometer counters should be reset.

**Defines**

- #define [RESET\\_NONE](#) 0x00
- #define [RESET\\_COUNT](#) 0x08
- #define [RESET\\_BLOCK\\_COUNT](#) 0x20
- #define [RESET\\_ROTATION\\_COUNT](#) 0x40
- #define [RESET\\_BLOCKANDTACHO](#) 0x28
- #define [RESET\\_ALL](#) 0x68

**6.128.1 Detailed Description**

Use these constants to specify which of the three tachometer counters should be reset. Reset constants can be combined with bitwise OR.

**See also:**

[OnFwdEx\(\)](#), [OnRevEx\(\)](#), etc...

### 6.128.2 Define Documentation

#### 6.128.2.1 #define RESET\_ALL 0x68

Reset all three tachometer counters

#### 6.128.2.2 #define RESET\_BLOCK\_COUNT 0x20

Reset the NXT-G block tachometer counter

#### 6.128.2.3 #define RESET\_BLOCKANDTACHO 0x28

Reset both the internal counter and the NXT-G block counter

#### 6.128.2.4 #define RESET\_COUNT 0x08

Reset the internal tachometer counter

#### 6.128.2.5 #define RESET\_NONE 0x00

No counters will be reset

#### Examples:

`ex_coastex.nxc`, `ex_offex.nxc`, `ex_onfwdex.nxc`, `ex_onfwdregex.nxc`, `ex_onfwdregexpid.nxc`, `ex_onfwsyncex.nxc`, `ex_onfwsyncexpid.nxc`, `ex_onrevex.nxc`, `ex_onrevregex.nxc`, `ex_onrevregexpid.nxc`, `ex_onrevsyncex.nxc`, and `ex_onrevsyncexpid.nxc`.

#### 6.128.2.6 #define RESET\_ROTATION\_COUNT 0x40

Reset the rotation counter

## 6.129 Output port mode constants

Use these constants to configure the desired mode for the specified motor(s): coast, motoron, brake, or regulated.

## Defines

- #define `OUT_MODE_COAST` 0x00
- #define `OUT_MODE_MOTORON` 0x01
- #define `OUT_MODE_BRAKE` 0x02
- #define `OUT_MODE_REGULATED` 0x04
- #define `OUT_MODE_REGMETHOD` 0xF0

### 6.129.1 Detailed Description

Use these constants to configure the desired mode for the specified motor(s): coast, motoron, brake, or regulated. Mode constants can be combined with bitwise OR.

#### See also:

[SetOutput\(\)](#)

### 6.129.2 Define Documentation

#### 6.129.2.1 #define `OUT_MODE_BRAKE` 0x02

Uses electronic braking to outputs

#### 6.129.2.2 #define `OUT_MODE_COAST` 0x00

No power and no braking so motors rotate freely.

#### 6.129.2.3 #define `OUT_MODE_MOTORON` 0x01

Enables PWM power to the outputs given the power setting

#### Examples:

[ex\\_RemoteSetOutputState.nxc](#).

#### 6.129.2.4 #define `OUT_MODE_REGMETHOD` 0xF0

Mask for unimplemented regulation mode

#### 6.129.2.5 #define `OUT_MODE_REGULATED` 0x04

Enables active power regulation using the regulation mode value

## 6.130 Output port option constants

Use these constants to configure the desired options for the specified motor(s): hold at limit and ramp down to limit.

### Defines

- #define [OUT\\_OPTION\\_HOLDATLIMIT](#) 0x10
- #define [OUT\\_OPTION\\_RAMPDOWNNTOLIMIT](#) 0x20

### 6.130.1 Detailed Description

Use these constants to configure the desired options for the specified motor(s): hold at limit and ramp down to limit. Option constants can be combined with bitwise OR.

#### See also:

[SetOutput\(\)](#)

### 6.130.2 Define Documentation

#### 6.130.2.1 #define [OUT\\_OPTION\\_HOLDATLIMIT](#) 0x10

Option to have the firmware hold the motor when it reaches the tachometer limit

#### 6.130.2.2 #define [OUT\\_OPTION\\_RAMPDOWNNTOLIMIT](#) 0x20

Option to have the firmware rampdown the motor power as it approaches the tachometer limit

## 6.131 Output regulation option constants

Use these constants to configure the desired options for position regulation.

### Defines

- #define [OUT\\_REGOPTION\\_NO\\_SATURATION](#) 0x01

### 6.131.1 Detailed Description

Use these constants to configure the desired options for position regulation.

### 6.131.2 Define Documentation

#### 6.131.2.1 #define OUT\_REGOPTION\_NO\_SATURATION 0x01

Do not limit intermediary regulation results

#### Examples:

[ex\\_PosReg.nxc](#).

## 6.132 Output port run state constants

Use these constants to configure the desired run state for the specified motor(s): idle, rampup, running, rampdown, or hold.

### Defines

- #define [OUT\\_RUNSTATE\\_IDLE](#) 0x00
- #define [OUT\\_RUNSTATE\\_RAMPUP](#) 0x10
- #define [OUT\\_RUNSTATE\\_RUNNING](#) 0x20
- #define [OUT\\_RUNSTATE\\_RAMPDOWN](#) 0x40
- #define [OUT\\_RUNSTATE\\_HOLD](#) 0x60

### 6.132.1 Detailed Description

Use these constants to configure the desired run state for the specified motor(s): idle, rampup, running, rampdown, or hold.

#### See also:

[SetOutput\(\)](#)

### 6.132.2 Define Documentation

#### 6.132.2.1 #define OUT\_RUNSTATE\_HOLD 0x60

Set motor run state to hold at the current position.

#### 6.132.2.2 #define OUT\_RUNSTATE\_IDLE 0x00

Disable all power to motors.

#### 6.132.2.3 #define OUT\_RUNSTATE\_RAMPDOWN 0x40

Enable ramping down from a current power to a new (lower) power over a specified [TachoLimitField](#) goal.

#### 6.132.2.4 #define OUT\_RUNSTATE\_RAMPUP 0x10

Enable ramping up from a current power to a new (higher) power over a specified [TachoLimitField](#) goal.

#### 6.132.2.5 #define OUT\_RUNSTATE\_RUNNING 0x20

Enable power to motors at the specified power level.

#### Examples:

[ex\\_RemoteSetOutputState.nxc](#).

### 6.133 Output port regulation mode constants

Use these constants to configure the desired regulation mode for the specified motor(s): none, speed regulation, multi-motor synchronization, or position regulation (requires the enhanced NBC/NXC firmware version 1.31+).

#### Defines

- #define [OUT\\_REGMODE\\_IDLE](#) 0
- #define [OUT\\_REGMODE\\_SPEED](#) 1
- #define [OUT\\_REGMODE\\_SYNC](#) 2
- #define [OUT\\_REGMODE\\_POS](#) 4

#### 6.133.1 Detailed Description

Use these constants to configure the desired regulation mode for the specified motor(s): none, speed regulation, multi-motor synchronization, or position regulation (requires the enhanced NBC/NXC firmware version 1.31+).

#### See also:

[SetOutput\(\)](#)

### 6.133.2 Define Documentation

#### 6.133.2.1 #define OUT\_REGMODE\_IDLE 0

No motor regulation.

##### Examples:

[ex\\_RemoteSetOutputState.nxc](#).

#### 6.133.2.2 #define OUT\_REGMODE\_POS 4

Regulate a motor's position.

#### 6.133.2.3 #define OUT\_REGMODE\_SPEED 1

Regulate a motor's speed (aka power).

##### Examples:

[ex\\_onfwdreg.nxc](#), [ex\\_onfwdregex.nxc](#), [ex\\_onfwdregexpid.nxc](#), [ex\\_onfwdregpid.nxc](#), [ex\\_onrevreg.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), and [ex\\_onrevregpid.nxc](#).

#### 6.133.2.4 #define OUT\_REGMODE\_SYNC 2

Synchronize the rotation of two motors.

## 6.134 Output field constants

Constants for use with [SetOutput\(\)](#) and [GetOutput\(\)](#).

### Defines

- #define [UpdateFlagsField](#) 0  
*Update flags field.*
- #define [OutputModeField](#) 1  
*Mode field.*
- #define [PowerField](#) 2

*Power field.*

- #define [ActualSpeedField](#) 3  
*Actual speed field.*
- #define [TachoCountField](#) 4  
*Internal tachometer count field.*
- #define [TachoLimitField](#) 5  
*Tachometer limit field.*
- #define [RunStateField](#) 6  
*Run state field.*
- #define [TurnRatioField](#) 7  
*Turn ratio field.*
- #define [RegModeField](#) 8  
*Regulation mode field.*
- #define [OverloadField](#) 9  
*Overload field.*
- #define [RegPValueField](#) 10  
*Proportional field.*
- #define [RegIValueField](#) 11  
*Integral field.*
- #define [RegDValueField](#) 12  
*Derivative field.*
- #define [BlockTachoCountField](#) 13  
*NXT-G block tachometer count field.*
- #define [RotationCountField](#) 14  
*Rotation counter field.*
- #define [OutputOptionsField](#) 15  
*Options field.*
- #define [MaxSpeedField](#) 16

*MaxSpeed* field.

- #define [MaxAccelerationField](#) 17  
*MaxAcceleration* field.

### 6.134.1 Detailed Description

Constants for use with [SetOutput\(\)](#) and [GetOutput\(\)](#).

See also:

[SetOutput\(\)](#), [GetOutput\(\)](#)

### 6.134.2 Define Documentation

#### 6.134.2.1 #define ActualSpeedField 3

Actual speed field. Contains the actual power level (-100 to 100). Read only. Return the percent of full power the firmware is applying to the output. This may vary from the [PowerField](#) value when auto-regulation code in the firmware responds to a load on the output.

#### 6.134.2.2 #define BlockTachoCountField 13

NXT-G block tachometer count field. Contains the current NXT-G block tachometer count. Read only. Return the block-relative position counter value for the specified port. Refer to the [UpdateFlagsField](#) description for information about how to use block-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) flag in [UpdateFlagsField](#) to request that the firmware reset the [BlockTachoCountField](#). The sign of [BlockTachoCountField](#) indicates the direction of rotation. Positive values indicate forward rotation and negative values indicate reverse rotation. Forward and reverse depend on the orientation of the motor.

#### 6.134.2.3 #define MaxAccelerationField 17

[MaxAcceleration](#) field. Contains the current max acceleration value. Read/write. Set the maximum acceleration to be used during position regulation.

#### 6.134.2.4 #define MaxSpeedField 16

MaxSpeed field. Contains the current max speed value. Read/write. Set the maximum speed to be used during position regulation.

#### 6.134.2.5 #define OutputModeField 1

Mode field. Contains a combination of the output mode constants. Read/write. The [OUT\\_MODE\\_MOTORON](#) bit must be set in order for power to be applied to the motors. Add [OUT\\_MODE\\_BRAKE](#) to enable electronic braking. Braking means that the output voltage is not allowed to float between active PWM pulses. It improves the accuracy of motor output but uses more battery power. To use motor regulation include [OUT\\_MODE\\_REGULATED](#) in the [OutputModeField](#) value. Use [UF\\_UPDATE\\_MODE](#) with [UpdateFlagsField](#) to commit changes to this field.

#### 6.134.2.6 #define OutputOptionsField 15

Options field. Contains a combination of the output options constants. Read/write. Set options for how the output module will act when a tachometer limit is reached. Option constants can be combined with bitwise OR. Use [OUT\\_OPTION\\_HOLDATLIMIT](#) to have the output module hold the motor when it reaches the tachometer limit. Use [OUT\\_OPTION\\_RAMPDOWNTOLIMIT](#) to have the output module ramp down the motor power as it approaches the tachometer limit.

#### 6.134.2.7 #define OverloadField 9

Overload field. Contains a boolean value which is TRUE if the motor is overloaded. Read only. This field will have a value of 1 (true) if the firmware speed regulation cannot overcome a physical load on the motor. In other words, the motor is turning more slowly than expected. If the motor speed can be maintained in spite of loading then this field value is zero (false). In order to use this field the motor must have a non-idle [RunStateField](#), an [OutputModeField](#) which includes [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), and its [RegModeField](#) must be set to [OUT\\_REGMODE\\_SPEED](#).

### 6.134.2.8 #define PowerField 2

Power field. Contains the desired power level (-100 to 100). Read/write. Specify the power level of the output. The absolute value of PowerField is a percentage of the full power of the motor. The sign of PowerField controls the rotation direction. Positive values tell the firmware to turn the motor forward, while negative values turn the motor backward. Use [UF\\_UPDATE\\_SPEED](#) with [UpdateFlagsField](#) to commit changes to this field.

### 6.134.2.9 #define RegDValueField 12

Derivative field. Contains the derivative constant for the PID motor controller. Read/write. This field specifies the derivative term used in the internal proportional-integral-derivative (PID) control algorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

### 6.134.2.10 #define RegIValueField 11

Integral field. Contains the integral constant for the PID motor controller. Read/write. This field specifies the integral term used in the internal proportional-integral-derivative (PID) control algorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

### 6.134.2.11 #define RegModeField 8

Regulation mode field. Contains one of the regulation mode constants. Read/write. This field specifies the regulation mode to use with the specified port(s). It is ignored if the [OUT\\_MODE\\_REGULATED](#) bit is not set in the [OutputModeField](#) field. Unlike [OutputModeField](#), [RegModeField](#) is not a bitfield. Only one regulation mode value can be set at a time. Speed regulation means that the firmware tries to maintain a certain speed based on the [PowerField](#) setting. The firmware adjusts the PWM duty cycle if the motor is affected by a physical load. This adjustment is reflected by the value of the [ActualSpeedField](#) property. When using speed regulation, do not set [PowerField](#) to its maximum value since the firmware cannot adjust to higher power levels in that situation. Synchronization means the firmware tries to keep two motors in sync regardless of physical loads. Use this mode to maintain a straight path for a mobile robot automatically. Also use this mode with the [TurnRatioField](#) property to provide proportional turning. Set [OUT\\_REGMODE\\_SYNC](#) on at least two motor ports in order

for synchronization to function. Setting [OUT\\_REGMODE\\_SYNC](#) on all three motor ports will result in only the first two ([OUT\\_A](#) and [OUT\\_B](#)) being synchronized.

#### 6.134.2.12 #define RegPValueField 10

Proportional field. Contains the proportional constant for the PID motor controller. Read/write. This field specifies the proportional term used in the internal proportional-integral-derivative (PID) control algorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

#### 6.134.2.13 #define RotationCountField 14

Rotation counter field. Contains the current rotation count. Read only. Return the program-relative position counter value for the specified port. Refer to the [UpdateFlagsField](#) description for information about how to use program-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) flag in [UpdateFlagsField](#) to request that the firmware reset the RotationCountField. The sign of RotationCountField indicates the direction of rotation. Positive values indicate forward rotation and negative values indicate reverse rotation. Forward and reverse depend on the orientation of the motor.

#### 6.134.2.14 #define RunStateField 6

Run state field. Contains one of the run state constants. Read/write. Use this field to specify the running state of an output. Set the RunStateField to [OUT\\_RUNSTATE\\_RUNNING](#) to enable power to any output. Use [OUT\\_RUNSTATE\\_RAMPUP](#) to enable automatic ramping to a new [PowerField](#) level greater than the current [PowerField](#) level. Use [OUT\\_RUNSTATE\\_RAMPDOWN](#) to enable automatic ramping to a new [PowerField](#) level less than the current [PowerField](#) level. Both the rampup and ramp-down bits must be used in conjunction with appropriate [TachoLimitField](#) and [PowerField](#) values. In this case the firmware smoothly increases or decreases the actual power to the new [PowerField](#) level over the total number of degrees of rotation specified in [TachoLimitField](#).

#### 6.134.2.15 #define TachoCountField 4

Internal tachometer count field. Contains the current internal tachometer count. Read only. Return the internal position counter value for the specified output. The internal count is reset automatically when a new goal is set using the [TachoLimitField](#) and the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag. Set the [UF\\_UPDATE\\_RESET\\_COUNT](#) flag in [UpdateFlagsField](#) to reset [TachoCountField](#) and cancel any [TachoLimitField](#). The sign of [TachoCountField](#) indicates the motor rotation direction.

#### 6.134.2.16 #define TachoLimitField 5

Tachometer limit field. Contains the current tachometer limit. Read/write. Specify the number of degrees the motor should rotate. Use [UF\\_UPDATE\\_TACHO\\_LIMIT](#) with the [UpdateFlagsField](#) field to commit changes to the [TachoLimitField](#). The value of this field is a relative distance from the current motor position at the moment when the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag is processed.

#### 6.134.2.17 #define TurnRatioField 7

Turn ratio field. Contains the current turn ratio. Only applicable when synchronizing multiple motors. Read/write. Use this field to specify a proportional turning ratio. This field must be used in conjunction with other field values: [OutputModeField](#) must include [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), [RegModeField](#) must be set to [OUT\\_REGMODE\\_SYNC](#), [RunStateField](#) must not be [OUT\\_RUNSTATE\\_IDLE](#), and [PowerField](#) must be non-zero. There are only three valid combinations of left and right motors for use with [TurnRatioField](#): [OUT\\_AB](#), [OUT\\_BC](#), and [OUT\\_AC](#). In each of these three options the first motor listed is considered to be the left motor and the second motor is the right motor, regardless of the physical configuration of the robot. Negative turn ratio values shift power toward the left motor while positive values shift power toward the right motor. An absolute value of 50 usually results in one motor stopping. An absolute value of 100 usually results in two motors turning in opposite directions at equal power.

#### 6.134.2.18 #define UpdateFlagsField 0

Update flags field. Contains a combination of the update flag constants. Read/write. Use [UF\\_UPDATE\\_MODE](#), [UF\\_UPDATE\\_SPEED](#), [UF\\_UPDATE\\_TACHO\\_LIMIT](#), and [UF\\_UPDATE\\_PID\\_VALUES](#) along with other fields to commit changes to the state of outputs. Set the appropriate flags after setting one or more of the output fields in order for the changes to actually go into affect.

## 6.135 Output module IOMAP offsets

Constant offsets into the Output module IOMAP structure.

### Defines

- #define `OutputOffsetTachoCount(p)`  $((p)*32)+0$
- #define `OutputOffsetBlockTachoCount(p)`  $((p)*32)+4$
- #define `OutputOffsetRotationCount(p)`  $((p)*32)+8$
- #define `OutputOffsetTachoLimit(p)`  $((p)*32)+12$
- #define `OutputOffsetMotorRPM(p)`  $((p)*32)+16$
- #define `OutputOffsetFlags(p)`  $((p)*32)+18$
- #define `OutputOffsetMode(p)`  $((p)*32)+19$
- #define `OutputOffsetSpeed(p)`  $((p)*32)+20$
- #define `OutputOffsetActualSpeed(p)`  $((p)*32)+21$
- #define `OutputOffsetRegPParameter(p)`  $((p)*32)+22$
- #define `OutputOffsetRegIParameter(p)`  $((p)*32)+23$
- #define `OutputOffsetRegDParameter(p)`  $((p)*32)+24$
- #define `OutputOffsetRunState(p)`  $((p)*32)+25$
- #define `OutputOffsetRegMode(p)`  $((p)*32)+26$
- #define `OutputOffsetOverloaded(p)`  $((p)*32)+27$
- #define `OutputOffsetSyncTurnParameter(p)`  $((p)*32)+28$
- #define `OutputOffsetOptions(p)`  $((p)*32)+29$
- #define `OutputOffsetMaxSpeed(p)`  $((p)*32)+30$
- #define `OutputOffsetMaxAccel(p)`  $((p)*32)+31$
- #define `OutputOffsetRegulationTime` 96
- #define `OutputOffsetRegulationOptions` 97

### 6.135.1 Detailed Description

Constant offsets into the Output module IOMAP structure.

### 6.135.2 Define Documentation

#### 6.135.2.1 #define `OutputOffsetActualSpeed(p)` $((p)*32)+21$

R - Holds the current motor speed (1 byte) sbyte

#### 6.135.2.2 #define `OutputOffsetBlockTachoCount(p)` $((p)*32)+4$

R - Holds current number of counts for the current output block (4 bytes) slong

**6.135.2.3 #define OutputOffsetFlags(p) (((p)\*32)+18)**

RW - Holds flags for which data should be updated (1 byte) ubyte

**6.135.2.4 #define OutputOffsetMaxAccel(p) (((p)\*32)+31)**

RW - holds the maximum acceleration for position regulation (1 byte) sbyte (NBC/NXC)

**6.135.2.5 #define OutputOffsetMaxSpeed(p) (((p)\*32)+30)**

RW - holds the maximum speed for position regulation (1 byte) sbyte (NBC/NXC)

**6.135.2.6 #define OutputOffsetMode(p) (((p)\*32)+19)**

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

**6.135.2.7 #define OutputOffsetMotorRPM(p) (((p)\*32)+16)**

Not updated, will be removed later !! (2 bytes) sword

**6.135.2.8 #define OutputOffsetOptions(p) (((p)\*32)+29)**

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

**6.135.2.9 #define OutputOffsetOverloaded(p) (((p)\*32)+27)**

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

**6.135.2.10 #define OutputOffsetRegDParameter(p) (((p)\*32)+24)**

RW - Holds the D-constant used in the regulation (1 byte) ubyte

**6.135.2.11 #define OutputOffsetRegIParameter(p) (((p)\*32)+23)**

RW - Holds the I-constant used in the regulation (1 byte) ubyte

**6.135.2.12 #define OutputOffsetRegMode(p) (((p)\*32)+26)**

RW - Tells which regulation mode should be used (1 byte) ubyte

**6.135.2.13 #define OutputOffsetRegPParameter(p) (((p)\*32)+22)**

RW - Holds the P-constant used in the regulation (1 byte) ubyte

**6.135.2.14 #define OutputOffsetRegulationOptions 97**

use for position regulation options (1 byte) ubyte (NBC/NXC)

**6.135.2.15 #define OutputOffsetRegulationTime 96**

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

**6.135.2.16 #define OutputOffsetRotationCount(p) (((p)\*32)+8)**

R - Holds current number of counts for the rotation counter to the output (4 bytes)  
slong

**6.135.2.17 #define OutputOffsetRunState(p) (((p)\*32)+25)**

RW - Holds the current motor run state in the output module (1 byte) ubyte

**6.135.2.18 #define OutputOffsetSpeed(p) (((p)\*32)+20)**

RW - Holds the wanted speed (1 byte) sbyte

**6.135.2.19 #define OutputOffsetSyncTurnParameter(p) (((p)\*32)+28)**

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

**6.135.2.20 #define OutputOffsetTachoCount(p) (((p)\*32)+0)**

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes)  
slong

**6.135.2.21 #define OutputOffsetTachoLimit(p) (((p)\*32)+12)**

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

**6.136 LowSpeed module constants**

Constants that are part of the NXT firmware's LowSpeed module.

**Modules**

- [LSState constants](#)  
*Constants for the low speed module LSState function.*
- [LSChannelState constants](#)  
*Constants for the low speed module LSChannelState function.*
- [LSMode constants](#)  
*Constants for the low speed module LSMODE function.*
- [LSErrorType constants](#)  
*Constants for the low speed module LSErrorType function.*
- [Low speed module IOMAP offsets](#)  
*Constant offsets into the low speed module IOMAP structure.*
- [LSNoRestartOnRead constants](#)  
*Constants for the low speed module LSNoRestartOnRead and SetLSNoRestartOnRead functions.*
- [Standard I2C constants](#)  
*Constants for use with standard I2C devices.*
- [LEGO I2C address constants](#)  
*Constants for LEGO I2C device addresses.*
- [Ultrasonic sensor constants](#)  
*Constants for use with the ultrasonic sensor.*
- [LEGO temperature sensor constants](#)  
*Constants for use with the LEGO temperature sensor.*
- [E-Meter sensor constants](#)

*Constants for use with the e-meter sensor.*

- [I2C option constants](#)

*Constants for the SetI2COptions function.*

### 6.136.1 Detailed Description

Constants that are part of the NXT firmware's LowSpeed module.

## 6.137 LSState constants

Constants for the low speed module LSState function.

### Defines

- #define [COM\\_CHANNEL\\_NONE\\_ACTIVE](#) 0x00
- #define [COM\\_CHANNEL\\_ONE\\_ACTIVE](#) 0x01
- #define [COM\\_CHANNEL\\_TWO\\_ACTIVE](#) 0x02
- #define [COM\\_CHANNEL\\_THREE\\_ACTIVE](#) 0x04
- #define [COM\\_CHANNEL\\_FOUR\\_ACTIVE](#) 0x08

### 6.137.1 Detailed Description

Constants for the low speed module LSState function. These values are combined together using a bitwise OR operation.

#### See also:

[LSState\(\)](#)

### 6.137.2 Define Documentation

#### 6.137.2.1 #define COM\_CHANNEL\_FOUR\_ACTIVE 0x08

Low speed channel 4 is active

#### 6.137.2.2 #define COM\_CHANNEL\_NONE\_ACTIVE 0x00

None of the low speed channels are active

**6.137.2.3 #define COM\_CHANNEL\_ONE\_ACTIVE 0x01**

Low speed channel 1 is active

**6.137.2.4 #define COM\_CHANNEL\_THREE\_ACTIVE 0x04**

Low speed channel 3 is active

**6.137.2.5 #define COM\_CHANNEL\_TWO\_ACTIVE 0x02**

Low speed channel 2 is active

**6.138 LSChannelState constants**

Constants for the low speed module LSChannelState function.

**Defines**

- #define [LOWSPEED\\_IDLE](#) 0
- #define [LOWSPEED\\_INIT](#) 1
- #define [LOWSPEED\\_LOAD\\_BUFFER](#) 2
- #define [LOWSPEED\\_COMMUNICATING](#) 3
- #define [LOWSPEED\\_ERROR](#) 4
- #define [LOWSPEED\\_DONE](#) 5

**6.138.1 Detailed Description**

Constants for the low speed module LSChannelState function.

See also:

[LSChannelState\(\)](#)

**6.138.2 Define Documentation****6.138.2.1 #define LOWSPEED\_COMMUNICATING 3**

Channel is actively communicating

**6.138.2.2 #define LOWSPEED\_DONE 5**

Channel is done communicating

**6.138.2.3 #define LOWSPEED\_ERROR 4**

Channel is in an error state

**6.138.2.4 #define LOWSPEED\_IDLE 0**

Channel is idle

**Examples:**

[ex\\_syscommlscheckstatus.nxc](#).

**6.138.2.5 #define LOWSPEED\_INIT 1**

Channel is being initialized

**6.138.2.6 #define LOWSPEED\_LOAD\_BUFFER 2**

Channel buffer is loading

**6.139 LSMode constants**

Constants for the low speed module LSMode function.

**Defines**

- #define [LOWSPEED\\_TRANSMITTING](#) 1
- #define [LOWSPEED\\_RECEIVING](#) 2
- #define [LOWSPEED\\_DATA\\_RECEIVED](#) 3

**6.139.1 Detailed Description**

Constants for the low speed module LSMode function.

**See also:**

[LSMode\(\)](#)

### 6.139.2 Define Documentation

#### 6.139.2.1 #define LOWSPEED\_DATA\_RECEIVED 3

Lowspeed port is in data received mode

#### 6.139.2.2 #define LOWSPEED\_RECEIVING 2

Lowspeed port is in receiving mode

#### 6.139.2.3 #define LOWSPEED\_TRANSMITTING 1

Lowspeed port is in transmitting mode

## 6.140 LSErrorType constants

Constants for the low speed module LSErrorType function.

### Defines

- #define [LOWSPEED\\_NO\\_ERROR](#) 0
- #define [LOWSPEED\\_CH\\_NOT\\_READY](#) 1
- #define [LOWSPEED\\_TX\\_ERROR](#) 2
- #define [LOWSPEED\\_RX\\_ERROR](#) 3

### 6.140.1 Detailed Description

Constants for the low speed module LSErrorType function.

See also:

[LSErrorType\(\)](#)

### 6.140.2 Define Documentation

#### 6.140.2.1 #define LOWSPEED\_CH\_NOT\_READY 1

Lowspeed port is not ready

#### 6.140.2.2 #define LOWSPEED\_NO\_ERROR 0

Lowspeed port has no error

**6.140.2.3 #define LOWSPEED\_RX\_ERROR 3**

Lowspeed port encountered an error while receiving data

**6.140.2.4 #define LOWSPEED\_TX\_ERROR 2**

Lowspeed port encountered an error while transmitting data

**6.141 Low speed module IOMAP offsets**

Constant offsets into the low speed module IOMAP structure.

**Defines**

- #define `LowSpeedOffsetInBufBuf(p)`  $((p)*19)+0$
- #define `LowSpeedOffsetInBufInPtr(p)`  $((p)*19)+16$
- #define `LowSpeedOffsetInBufOutPtr(p)`  $((p)*19)+17$
- #define `LowSpeedOffsetInBufBytesToRx(p)`  $((p)*19)+18$
- #define `LowSpeedOffsetOutBufBuf(p)`  $((p)*19)+76$
- #define `LowSpeedOffsetOutBufInPtr(p)`  $((p)*19)+92$
- #define `LowSpeedOffsetOutBufOutPtr(p)`  $((p)*19)+93$
- #define `LowSpeedOffsetOutBufBytesToRx(p)`  $((p)*19)+94$
- #define `LowSpeedOffsetMode(p)`  $((p)+152)$
- #define `LowSpeedOffsetChannelState(p)`  $((p)+156)$
- #define `LowSpeedOffsetErrorType(p)`  $((p)+160)$
- #define `LowSpeedOffsetState` 164
- #define `LowSpeedOffsetSpeed` 165
- #define `LowSpeedOffsetNoRestartOnRead` 166

**6.141.1 Detailed Description**

Constant offsets into the low speed module IOMAP structure.

**6.141.2 Define Documentation****6.141.2.1 #define LowSpeedOffsetChannelState(p) ((p)+156)**

R - Lowspeed channgel state (1 byte)

**6.141.2.2 #define LowSpeedOffsetErrorType(p) ((p)+160)**

R - Lowspeed port error type (1 byte)

**6.141.2.3 #define LowSpeedOffsetInBufBuf(p) (((p)\*19)+0)**

RW - Input buffer data buffer field offset (16 bytes)

**6.141.2.4 #define LowSpeedOffsetInBufBytesToRx(p) (((p)\*19)+18)**

RW - Input buffer bytes to receive field offset (1 byte)

**6.141.2.5 #define LowSpeedOffsetInBufInPtr(p) (((p)\*19)+16)**

RW - Input buffer in pointer field offset (1 byte)

**6.141.2.6 #define LowSpeedOffsetInBufOutPtr(p) (((p)\*19)+17)**

RW - Input buffer out pointer field offset (1 byte)

**6.141.2.7 #define LowSpeedOffsetMode(p) ((p)+152)**

R - Lowspeed port mode (1 byte)

**6.141.2.8 #define LowSpeedOffsetNoRestartOnRead 166**

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

**6.141.2.9 #define LowSpeedOffsetOutBufBuf(p) (((p)\*19)+76)**

RW - Output buffer data buffer field offset (16 bytes)

**6.141.2.10 #define LowSpeedOffsetOutBufBytesToRx(p) (((p)\*19)+94)**

RW - Output buffer bytes to receive field offset (1 byte)

**6.141.2.11 #define LowSpeedOffsetOutBufInPtr(p) (((p)\*19)+92)**

RW - Output buffer in pointer field offset (1 byte)

**6.141.2.12 #define LowSpeedOffsetOutBufOutPtr(p) (((p)\*19)+93)**

RW - Output buffer out pointer field offset (1 byte)

**6.141.2.13 #define LowSpeedOffsetSpeed 165**

R - Lowspeed speed (unused)

**6.141.2.14 #define LowSpeedOffsetState 164**

R - Lowspeed state (all channels)

**6.142 LSNoRestartOnRead constants**

Constants for the low speed module LSNoRestartOnRead and SetLSNoRestartOnRead functions.

**Defines**

- #define [LSREAD\\_RESTART\\_ALL](#) 0x00
- #define [LSREAD\\_NO\\_RESTART\\_1](#) 0x01
- #define [LSREAD\\_NO\\_RESTART\\_2](#) 0x02
- #define [LSREAD\\_NO\\_RESTART\\_3](#) 0x04
- #define [LSREAD\\_NO\\_RESTART\\_4](#) 0x08
- #define [LSREAD\\_RESTART\\_NONE](#) 0x0F
- #define [LSREAD\\_NO\\_RESTART\\_MASK](#) 0x10

**6.142.1 Detailed Description**

Constants for the low speed module LSNoRestartOnRead and SetLSNoRestartOnRead functions. These values are combined with a bitwise OR operation.

**See also:**

[LSNoRestartOnRead\(\)](#), [SetLSNoRestartOnRead\(\)](#)

**6.142.2 Define Documentation****6.142.2.1 #define LSREAD\_NO\_RESTART\_1 0x01**

No restart on read for channel 1

**6.142.2.2 #define LSREAD\_NO\_RESTART\_2 0x02**

No restart on read for channel 2

**6.142.2.3 #define LSREAD\_NO\_RESTART\_3 0x04**

No restart on read for channel 3

**6.142.2.4 #define LSREAD\_NO\_RESTART\_4 0x08**

No restart on read for channel 4

**6.142.2.5 #define LSREAD\_NO\_RESTART\_MASK 0x10**

No restart mask

**6.142.2.6 #define LSREAD\_RESTART\_ALL 0x00**

Restart on read for all channels (default)

**6.142.2.7 #define LSREAD\_RESTART\_NONE 0x0F**

No restart on read for all channels

**6.143 Standard I2C constants**

Constants for use with standard I2C devices.

**Defines**

- #define [I2C\\_ADDR\\_DEFAULT](#) 0x02
- #define [I2C\\_REG\\_VERSION](#) 0x00
- #define [I2C\\_REG\\_VENDOR\\_ID](#) 0x08
- #define [I2C\\_REG\\_DEVICE\\_ID](#) 0x10
- #define [I2C\\_REG\\_CMD](#) 0x41

**6.143.1 Detailed Description**

Constants for use with standard I2C devices.

### 6.143.2 Define Documentation

#### 6.143.2.1 #define I2C\_ADDR\_DEFAULT 0x02

Standard NXT I2C device address

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_I2CSendCommand.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_MSDeenergize.nxc](#), [ex\\_MSEnergize.nxc](#), [ex\\_MSIRTrain.nxc](#), [ex\\_MSPFComboDirect.nxc](#), [ex\\_MSPFComboPWM.nxc](#), [ex\\_MSPFRawOutput.nxc](#), [ex\\_MSPFRepeat.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSinglePin.nxc](#), [ex\\_MSPFTrain.nxc](#), [ex\\_MSReadValue.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

#### 6.143.2.2 #define I2C\_REG\_CMD 0x41

Standard NXT I2C device command register

**Examples:**

[ex\\_MSReadValue.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

#### 6.143.2.3 #define I2C\_REG\_DEVICE\_ID 0x10

Standard NXT I2C device ID register

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

#### 6.143.2.4 #define I2C\_REG\_VENDOR\_ID 0x08

Standard NXT I2C vendor ID register

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

#### 6.143.2.5 #define I2C\_REG\_VERSION 0x00

Standard NXT I2C version register

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

**6.144 LEGO I2C address constants**

Constants for LEGO I2C device addresses.

**Defines**

- #define [LEGO\\_ADDR\\_US](#) 0x02
- #define [LEGO\\_ADDR\\_TEMP](#) 0x98
- #define [LEGO\\_ADDR\\_EMETER](#) 0x04

**6.144.1 Detailed Description**

Constants for LEGO I2C device addresses.

**6.144.2 Define Documentation****6.144.2.1 #define LEGO\_ADDR\_EMETER 0x04**

The LEGO e-meter sensor's I2C address

**6.144.2.2 #define LEGO\_ADDR\_TEMP 0x98**

The LEGO temperature sensor's I2C address

**6.144.2.3 #define LEGO\_ADDR\_US 0x02**

The LEGO ultrasonic sensor's I2C address

**6.145 Ultrasonic sensor constants**

Constants for use with the ultrasonic sensor.

**Defines**

- #define [US\\_CMD\\_OFF](#) 0x00
- #define [US\\_CMD\\_SINGLESHOT](#) 0x01

- #define `US_CMD_CONTINUOUS` 0x02
- #define `US_CMD_EVENTCAPTURE` 0x03
- #define `US_CMD_WARMRESET` 0x04
- #define `US_REG_CM_INTERVAL` 0x40
- #define `US_REG_ACTUAL_ZERO` 0x50
- #define `US_REG_SCALE_FACTOR` 0x51
- #define `US_REG_SCALE_DIVISOR` 0x52
- #define `US_REG_FACTORY_ACTUAL_ZERO` 0x11
- #define `US_REG_FACTORY_SCALE_FACTOR` 0x12
- #define `US_REG_FACTORY_SCALE_DIVISOR` 0x13
- #define `US_REG_MEASUREMENT_UNITS` 0x14

### 6.145.1 Detailed Description

Constants for use with the ultrasonic sensor.

### 6.145.2 Define Documentation

#### 6.145.2.1 #define `US_CMD_CONTINUOUS` 0x02

Command to put the ultrasonic sensor into continuous polling mode (default)

#### 6.145.2.2 #define `US_CMD_EVENTCAPTURE` 0x03

Command to put the ultrasonic sensor into event capture mode

#### 6.145.2.3 #define `US_CMD_OFF` 0x00

Command to turn off the ultrasonic sensor

#### Examples:

[ex\\_writei2cregister.nxc](#).

#### 6.145.2.4 #define `US_CMD_SINGLESHOT` 0x01

Command to put the ultrasonic sensor into single shot mode

#### 6.145.2.5 #define `US_CMD_WARMRESET` 0x04

Command to warm reset the ultrasonic sensor

**6.145.2.6 #define US\_REG\_ACTUAL\_ZERO 0x50**

The register address used to store the actual zero value

**6.145.2.7 #define US\_REG\_CM\_INTERVAL 0x40**

The register address used to store the CM interval

**6.145.2.8 #define US\_REG\_FACTORY\_ACTUAL\_ZERO 0x11**

The register address containing the factory setting for the actual zero value

**6.145.2.9 #define US\_REG\_FACTORY\_SCALE\_DIVISOR 0x13**

The register address containing the factory setting for the scale divisor value

**6.145.2.10 #define US\_REG\_FACTORY\_SCALE\_FACTOR 0x12**

The register address containing the factory setting for the scale factor value

**6.145.2.11 #define US\_REG\_MEASUREMENT\_UNITS 0x14**

The register address containing the measurement units (degrees C or F)

**6.145.2.12 #define US\_REG\_SCALE\_DIVISOR 0x52**

The register address used to store the scale divisor value

**6.145.2.13 #define US\_REG\_SCALE\_FACTOR 0x51**

The register address used to store the scale factor value

**6.146 LEGO temperature sensor constants**

Constants for use with the LEGO temperature sensor.

## Defines

- #define `TEMP_RES_9BIT` 0x00
- #define `TEMP_RES_10BIT` 0x20
- #define `TEMP_RES_11BIT` 0x40
- #define `TEMP_RES_12BIT` 0x60
- #define `TEMP_SD_CONTINUOUS` 0x00
- #define `TEMP_SD_SHUTDOWN` 0x01
- #define `TEMP_TM_COMPARATOR` 0x00
- #define `TEMP_TM_INTERRUPT` 0x02
- #define `TEMP_OS_ONESHOT` 0x80
- #define `TEMP_FQ_1` 0x00
- #define `TEMP_FQ_2` 0x08
- #define `TEMP_FQ_4` 0x10
- #define `TEMP_FQ_6` 0x18
- #define `TEMP_POL_LOW` 0x00
- #define `TEMP_POL_HIGH` 0x04
- #define `TEMP_REG_TEMP` 0x00
- #define `TEMP_REG_CONFIG` 0x01
- #define `TEMP_REG_TLOW` 0x02
- #define `TEMP_REG_THIGH` 0x03

### 6.146.1 Detailed Description

Constants for use with the LEGO temperature sensor.

### 6.146.2 Define Documentation

#### 6.146.2.1 #define `TEMP_FQ_1` 0x00

Set fault queue to 1 fault before alert

#### 6.146.2.2 #define `TEMP_FQ_2` 0x08

Set fault queue to 2 faults before alert

#### 6.146.2.3 #define `TEMP_FQ_4` 0x10

Set fault queue to 4 faults before alert

**6.146.2.4 #define TEMP\_FQ\_6 0x18**

Set fault queue to 6 faults before alert

**6.146.2.5 #define TEMP\_OS\_ONESHOT 0x80**

Set the sensor into oneshot mode. When the device is in shutdown mode this will start a single temperature conversion. The device returns to shutdown mode when it completes.

**6.146.2.6 #define TEMP\_POL\_HIGH 0x04**

Set polarity of ALERT pin to be active HIGH

**6.146.2.7 #define TEMP\_POL\_LOW 0x00**

Set polarity of ALERT pin to be active LOW

**6.146.2.8 #define TEMP\_REG\_CONFIG 0x01**

The register for reading/writing sensor configuration values

**6.146.2.9 #define TEMP\_REG\_TEMP 0x00**

The register where temperature values can be read

**6.146.2.10 #define TEMP\_REG\_THIGH 0x03**

The register for reading/writing a user-defined high temperature limit

**6.146.2.11 #define TEMP\_REG\_TLOW 0x02**

The register for reading/writing a user-defined low temperature limit

**6.146.2.12 #define TEMP\_RES\_10BIT 0x20**

Set the temperature conversion resolution to 10 bit

**6.146.2.13 #define TEMP\_RES\_11BIT 0x40**

Set the temperature conversion resolution to 11 bit

**6.146.2.14 #define TEMP\_RES\_12BIT 0x60**

Set the temperature conversion resolution to 12 bit

**Examples:**

[ex\\_ConfigureTemperatureSensor.nxc](#).

**6.146.2.15 #define TEMP\_RES\_9BIT 0x00**

Set the temperature conversion resolution to 9 bit

**6.146.2.16 #define TEMP\_SD\_CONTINUOUS 0x00**

Set the sensor mode to continuous

**6.146.2.17 #define TEMP\_SD\_SHUTDOWN 0x01**

Set the sensor mode to shutdown. The device will shut down after the current conversion is completed.

**6.146.2.18 #define TEMP\_TM\_COMPARATOR 0x00**

Set the thermostat mode to comparator

**6.146.2.19 #define TEMP\_TM\_INTERRUPT 0x02**

Set the thermostat mode to interrupt

**6.147 E-Meter sensor constants**

Constants for use with the e-meter sensor.

## Defines

- #define EMETER\_REG\_VIN 0x0a
- #define EMETER\_REG\_AIN 0x0c
- #define EMETER\_REG\_VOUT 0x0e
- #define EMETER\_REG\_AOUT 0x10
- #define EMETER\_REG\_JOULES 0x12
- #define EMETER\_REG\_WIN 0x14
- #define EMETER\_REG\_WOUT 0x16

### 6.147.1 Detailed Description

Constants for use with the e-meter sensor.

### 6.147.2 Define Documentation

#### 6.147.2.1 #define EMETER\_REG\_AIN 0x0c

The register address for amps in

#### 6.147.2.2 #define EMETER\_REG\_AOUT 0x10

The register address for amps out

#### 6.147.2.3 #define EMETER\_REG\_JOULES 0x12

The register address for joules

#### 6.147.2.4 #define EMETER\_REG\_VIN 0x0a

The register address for voltage in

#### 6.147.2.5 #define EMETER\_REG\_VOUT 0x0e

The register address for voltage out

#### 6.147.2.6 #define EMETER\_REG\_WIN 0x14

The register address for watts in

**6.147.2.7 #define EMETER\_REG\_WOUT 0x16**

The register address for watts out

**6.148 I2C option constants**

Constants for the SetI2COptions function.

**Defines**

- #define [I2C\\_OPTION\\_STANDARD](#) 0x00
- #define [I2C\\_OPTION\\_NORESTART](#) 0x04
- #define [I2C\\_OPTION\\_FAST](#) 0x08

**6.148.1 Detailed Description**

Constants for the SetI2COptions function. These values are combined with a bitwise OR operation.

**See also:**

[SetI2COptions\(\)](#)

**6.148.2 Define Documentation****6.148.2.1 #define I2C\_OPTION\_FAST 0x08**

Fast I2C speed

**6.148.2.2 #define I2C\_OPTION\_NORESTART 0x04**

Use no restart on I2C read

**6.148.2.3 #define I2C\_OPTION\_STANDARD 0x00**

Standard I2C speed

**6.149 Display module constants**

Constants that are part of the NXT firmware's Display module.

## Modules

- [Line number constants](#)  
*Line numbers for use with DrawText system function.*
- [DisplayExecuteFunction constants](#)  
*Constants that are for use with the DisplayExecuteFunction system call.*
- [Drawing option constants](#)  
*Constants that are for specifying drawing options in several display module API functions.*
- [Display flags](#)  
*Constants that are for use with the display flags functions.*
- [Display contrast constants](#)  
*Constants that are for use with the display contrast API functions.*
- [Text line constants](#)  
*Constants that are for use with getting/setting display data.*
- [Display module IOMAP offsets](#)  
*Constant offsets into the display module IOMAP structure.*

## Defines

- #define [SCREEN\\_MODE\\_RESTORE](#) 0x00
- #define [SCREEN\\_MODE\\_CLEAR](#) 0x01
- #define [DISPLAY\\_HEIGHT](#) 64
- #define [DISPLAY\\_WIDTH](#) 100
- #define [DISPLAY\\_MENUICONS\\_Y](#) 40
- #define [DISPLAY\\_MENUICONS\\_X\\_OFFS](#) 7
- #define [DISPLAY\\_MENUICONS\\_X\\_DIFF](#) 31
- #define [MENUICON\\_LEFT](#) 0
- #define [MENUICON\\_CENTER](#) 1
- #define [MENUICON\\_RIGHT](#) 2
- #define [MENUICONS](#) 3
- #define [FRAME\\_SELECT](#) 0
- #define [STATUSTEXT](#) 1
- #define [MENUTEXT](#) 2
- #define [STEPLINE](#) 3
- #define [TOPLINE](#) 4

- #define `SPECIALS` 5
- #define `STATUSICON_BLUETOOTH` 0
- #define `STATUSICON_USB` 1
- #define `STATUSICON_VM` 2
- #define `STATUSICON_BATTERY` 3
- #define `STATUSICONS` 4
- #define `SCREEN_BACKGROUND` 0
- #define `SCREEN_LARGE` 1
- #define `SCREEN_SMALL` 2
- #define `SCREENS` 3
- #define `BITMAP_1` 0
- #define `BITMAP_2` 1
- #define `BITMAP_3` 2
- #define `BITMAP_4` 3
- #define `BITMAPS` 4
- #define `STEPICON_1` 0
- #define `STEPICON_2` 1
- #define `STEPICON_3` 2
- #define `STEPICON_4` 3
- #define `STEPICON_5` 4
- #define `STEPICONS` 5

### 6.149.1 Detailed Description

Constants that are part of the NXT firmware's Display module.

### 6.149.2 Define Documentation

#### 6.149.2.1 #define `BITMAP_1` 0

Bitmap 1

#### 6.149.2.2 #define `BITMAP_2` 1

Bitmap 2

#### 6.149.2.3 #define `BITMAP_3` 2

Bitmap 3

**6.149.2.4 #define BITMAP\_4 3**

Bitmap 4

**6.149.2.5 #define BITMAPS 4**

The number of bitmap bits

**6.149.2.6 #define DISPLAY\_HEIGHT 64**

The height of the LCD screen in pixels

**Examples:**[ex\\_LineOut.nxc.](#)**6.149.2.7 #define DISPLAY\_MENUICONS\_X\_DIFF 31****6.149.2.8 #define DISPLAY\_MENUICONS\_X\_OFFS 7****6.149.2.9 #define DISPLAY\_MENUICONS\_Y 40****6.149.2.10 #define DISPLAY\_WIDTH 100**

The width of the LCD screen in pixels

**Examples:**[ex\\_LineOut.nxc.](#)**6.149.2.11 #define FRAME\_SELECT 0**

Center icon select frame

**6.149.2.12 #define MENUICON\_CENTER 1**

Center icon

**6.149.2.13 #define MENUICON\_LEFT 0**

Left icon

**6.149.2.14 #define MENUICON\_RIGHT 2**

Right icon

**6.149.2.15 #define MENUICONS 3**

The number of menu icons

**6.149.2.16 #define MENUTEXT 2**

Center icon text

**6.149.2.17 #define SCREEN\_BACKGROUND 0**

Entire screen

**6.149.2.18 #define SCREEN\_LARGE 1**

Entire screen except status line

**6.149.2.19 #define SCREEN\_MODE\_CLEAR 0x01**

Clear the screen

See also:

[SetScreenMode\(\)](#)

**6.149.2.20 #define SCREEN\_MODE\_RESTORE 0x00**

Restore the screen

See also:

[SetScreenMode\(\)](#)

**6.149.2.21 #define SCREEN\_SMALL 2**

Screen between menu icons and status line

**6.149.2.22 #define SCREENS 3**

The number of screen bits

**6.149.2.23 #define SPECIALS 5**

The number of special bit values

**6.149.2.24 #define STATUSICON\_BATTERY 3**

Battery status icon collection

**6.149.2.25 #define STATUSICON\_BLUETOOTH 0**

BlueTooth status icon collection

**6.149.2.26 #define STATUSICON\_USB 1**

USB status icon collection

**6.149.2.27 #define STATUSICON\_VM 2**

VM status icon collection

**6.149.2.28 #define STATUSICONS 4**

The number of status icons

**6.149.2.29 #define STATUSTEXT 1**

Status text (BT name)

6.149.2.30 #define STEPICON\_1 0

Left most step icon

6.149.2.31 #define STEPICON\_2 1

6.149.2.32 #define STEPICON\_3 2

6.149.2.33 #define STEPICON\_4 3

6.149.2.34 #define STEPICON\_5 4

Right most step icon

6.149.2.35 #define STEPICONS 5

6.149.2.36 #define STEPLINE 3

Step collection lines

6.149.2.37 #define TOPLINE 4

Top status underline

## 6.150 DisplayExecuteFunction constants

Constants that are for use with the DisplayExecuteFunction system call.

## Defines

- #define [DISPLAY\\_ERASE\\_ALL](#) 0x00
- #define [DISPLAY\\_PIXEL](#) 0x01
- #define [DISPLAY\\_HORIZONTAL\\_LINE](#) 0x02
- #define [DISPLAY\\_VERTICAL\\_LINE](#) 0x03
- #define [DISPLAY\\_CHAR](#) 0x04
- #define [DISPLAY\\_ERASE\\_LINE](#) 0x05
- #define [DISPLAY\\_FILL\\_REGION](#) 0x06
- #define [DISPLAY\\_FRAME](#) 0x07

### 6.150.1 Detailed Description

Constants that are for use with the DisplayExecuteFunction system call.

### 6.150.2 Define Documentation

#### 6.150.2.1 #define DISPLAY\_CHAR 0x04

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

#### 6.150.2.2 #define DISPLAY\_ERASE\_ALL 0x00

W - erase entire screen (CMD,x,x,x,x,x)

### Examples:

[ex\\_sysdisplayexecutefunction.nxc](#).

#### 6.150.2.3 #define DISPLAY\_ERASE\_LINE 0x05

W - erase a single line (CMD,x,LINE,x,x,x)

#### 6.150.2.4 #define DISPLAY\_FILL\_REGION 0x06

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

#### 6.150.2.5 #define DISPLAY\_FRAME 0x07

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**6.150.2.6 #define DISPLAY\_HORIZONTAL\_LINE 0x02**

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

**Examples:**

[ex\\_dispfunc.nxc](#).

**6.150.2.7 #define DISPLAY\_PIXEL 0x01**

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

**6.150.2.8 #define DISPLAY\_VERTICAL\_LINE 0x03**

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

**6.151 Drawing option constants**

Constants that are for specifying drawing options in several display module API functions.

**Modules**

- [Font drawing option constants](#)

*These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.*

**Defines**

- #define [DRAW\\_OPT\\_NORMAL](#) (0x0000)
- #define [DRAW\\_OPT\\_CLEAR\\_WHOLE\\_SCREEN](#) (0x0001)
- #define [DRAW\\_OPT\\_CLEAR\\_EXCEPT\\_STATUS\\_SCREEN](#) (0x0002)
- #define [DRAW\\_OPT\\_CLEAR\\_PIXELS](#) (0x0004)
- #define [DRAW\\_OPT\\_CLEAR](#) (0x0004)
- #define [DRAW\\_OPT\\_INVERT](#) (0x0004)
- #define [DRAW\\_OPT\\_LOGICAL\\_COPY](#) (0x0000)
- #define [DRAW\\_OPT\\_LOGICAL\\_AND](#) (0x0008)
- #define [DRAW\\_OPT\\_LOGICAL\\_OR](#) (0x0010)
- #define [DRAW\\_OPT\\_LOGICAL\\_XOR](#) (0x0018)
- #define [DRAW\\_OPT\\_FILL\\_SHAPE](#) (0x0020)

- #define [DRAW\\_OPT\\_CLEAR\\_SCREEN\\_MODES](#) (0x0003)
- #define [DRAW\\_OPT\\_LOGICAL\\_OPERATIONS](#) (0x0018)
- #define [DRAW\\_OPT\\_POLYGON\\_POLYLINE](#) (0x0400)

### 6.151.1 Detailed Description

Constants that are for specifying drawing options in several display module API functions. Bits 0 & 1 (values 0,1,2,3) control screen clearing behaviour (Not within RIC files). Bit 2 (value 4) controls the NOT operation, i.e. draw in white or invert text/graphics. Bits 3 & 4 (values 0,8,16,24) control pixel logical combinations (COPY/AND/OR/XOR). Bit 5 (value 32) controls shape filling, or overrides text/graphic bitmaps with set pixels. These may be ORed together for the full instruction (e.g., [DRAW\\_OPT\\_NORMAL](#)|[DRAW\\_OPT\\_LOGICAL\\_XOR](#)) These operations are resolved into the separate, common parameters defined in 'c\_display.iom' before any drawing function is called. Note that when drawing a RIC file, the initial 'DrawingOptions' parameter supplied in the drawing instruction controls screen clearing, but nothing else. The 'CopyOptions' parameter from each instruction in the RIC file then controls graphic operations, but the screen-clearing bits are ignored.

See also:

[TextOut\(\)](#), [NumOut\(\)](#), [PointOut\(\)](#), [LineOut\(\)](#), [CircleOut\(\)](#), [RectOut\(\)](#), [PolyOut\(\)](#), [EllipseOut\(\)](#), [FontTextOut\(\)](#), [FontNumOut\(\)](#), [GraphicOut\(\)](#), [GraphicArrayOut\(\)](#)

### 6.151.2 Define Documentation

#### 6.151.2.1 #define [DRAW\\_OPT\\_CLEAR](#) (0x0004)

Clear pixels while drawing (aka draw in white)

#### 6.151.2.2 #define [DRAW\\_OPT\\_CLEAR\\_EXCEPT\\_STATUS\\_SCREEN](#) (0x0002)

Clear the screen except for the status line before drawing

#### 6.151.2.3 #define [DRAW\\_OPT\\_CLEAR\\_PIXELS](#) (0x0004)

Clear pixels while drawing (aka draw in white)

#### 6.151.2.4 #define [DRAW\\_OPT\\_CLEAR\\_SCREEN\\_MODES](#) (0x0003)

Bit mask for the clear screen modes

**6.151.2.5 #define DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN (0x0001)**

Clear the entire screen before drawing

**Examples:**

[ex\\_dispgoutex.nxc](#).

**6.151.2.6 #define DRAW\_OPT\_FILL\_SHAPE (0x0020)**

Fill the shape while drawing (rectangle, circle, ellipses, and polygon)

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysDrawEllipse.nxc](#),  
and [ex\\_sysdrawpolygon.nxc](#).

**6.151.2.7 #define DRAW\_OPT\_INVERT (0x0004)**

Invert text or graphics

**Examples:**

[ex\\_dispftout.nxc](#).

**6.151.2.8 #define DRAW\_OPT\_LOGICAL\_AND (0x0008)**

Draw pixels using a logical AND operation

**Examples:**

[ex\\_dispftout.nxc](#).

**6.151.2.9 #define DRAW\_OPT\_LOGICAL\_COPY (0x0000)**

Draw pixels using a logical copy operation

**6.151.2.10 #define DRAW\_OPT\_LOGICAL\_OPERATIONS (0x0018)**

Bit mask for the logical drawing operations

**6.151.2.11 #define DRAW\_OPT\_LOGICAL\_OR (0x0010)**

Draw pixels using a logical OR operation

**Examples:**

[ex\\_dispftout.nxc](#).

**6.151.2.12 #define DRAW\_OPT\_LOGICAL\_XOR (0x0018)**

Draw pixels using a logical XOR operation

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysDrawEllipse.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**6.151.2.13 #define DRAW\_OPT\_NORMAL (0x0000)**

Normal drawing

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispfunc.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**6.151.2.14 #define DRAW\_OPT\_POLYGON\_POLYLINE (0x0400)**

When drawing polygons, do not close (i.e., draw a polyline instead)

**6.152 Font drawing option constants**

These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.

**Defines**

- #define [DRAW\\_OPT\\_FONT\\_DIRECTIONS](#) (0x01C0)
- #define [DRAW\\_OPT\\_FONT\\_WRAP](#) (0x0200)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RB](#) (0x0000)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RT](#) (0x0040)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_R2LB](#) (0x0080)

- #define [DRAW\\_OPT\\_FONT\\_DIR\\_R2LT](#) (0x00C0)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TL](#) (0x0100)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TR](#) (0x0140)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_T2BL](#) (0x0180)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_T2BR](#) (0x01C0)

### 6.152.1 Detailed Description

These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.

See also:

[FontTextOut\(\)](#), [FontNumOut\(\)](#)

### 6.152.2 Define Documentation

#### 6.152.2.1 #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TL](#) (0x0100)

Font bottom to top left align

#### 6.152.2.2 #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TR](#) (0x0140)

Font bottom to top right align

#### 6.152.2.3 #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RB](#) (0x0000)

Font left to right bottom align

Examples:

[ex\\_dispftout.nxc](#).

#### 6.152.2.4 #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RT](#) (0x0040)

Font left to right top align

Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**6.152.2.5 #define DRAW\_OPT\_FONT\_DIR\_R2LB (0x0080)**

Font right to left bottom align

**6.152.2.6 #define DRAW\_OPT\_FONT\_DIR\_R2LT (0x00C0)**

Font right to left top align

**6.152.2.7 #define DRAW\_OPT\_FONT\_DIR\_T2BL (0x0180)**

Font top to bottom left align

**Examples:**

[ex\\_dispftout.nxc](#).

**6.152.2.8 #define DRAW\_OPT\_FONT\_DIR\_T2BR (0x01C0)**

Font top to bottom right align

**6.152.2.9 #define DRAW\_OPT\_FONT DIRECTIONS (0x01C0)**

Bit mask for the font direction bits

**6.152.2.10 #define DRAW\_OPT\_FONT\_WRAP (0x0200)**

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

**Examples:**

[ex\\_dispftout.nxc](#).

**6.153 Display flags**

Constants that are for use with the display flags functions.

**Defines**

- #define [DISPLAY\\_ON](#) 0x01
- #define [DISPLAY\\_REFRESH](#) 0x02
- #define [DISPLAY\\_POPUP](#) 0x08

- #define [DISPLAY\\_REFRESH\\_DISABLED](#) 0x40
- #define [DISPLAY\\_BUSY](#) 0x80

### 6.153.1 Detailed Description

Constants that are for use with the display flags functions.

See also:

[SetDisplayFlags\(\)](#), [DisplayFlags\(\)](#)

### 6.153.2 Define Documentation

#### 6.153.2.1 #define DISPLAY\_BUSY 0x80

R - Refresh in progress

#### 6.153.2.2 #define DISPLAY\_ON 0x01

W - Display on

#### 6.153.2.3 #define DISPLAY\_POPUP 0x08

W - Use popup display memory

Examples:

[ex\\_dispmisc.nxc](#).

#### 6.153.2.4 #define DISPLAY\_REFRESH 0x02

W - Enable refresh

#### 6.153.2.5 #define DISPLAY\_REFRESH\_DISABLED 0x40

R - Refresh disabled

## 6.154 Display contrast constants

Constants that are for use with the display contrast API functions.

**Defines**

- #define [DISPLAY\\_CONTRAST\\_DEFAULT](#) 0x5A
- #define [DISPLAY\\_CONTRAST\\_MAX](#) 0x7F

**6.154.1 Detailed Description**

Constants that are for use with the display contrast API functions.

**See also:**

[SetDisplayContrast\(\)](#), [DisplayContrast\(\)](#)

**6.154.2 Define Documentation****6.154.2.1 #define DISPLAY\_CONTRAST\_DEFAULT 0x5A**

Default display contrast value

**Examples:**

[ex\\_contrast.nxc](#), and [ex\\_setdisplaycontrast.nxc](#).

**6.154.2.2 #define DISPLAY\_CONTRAST\_MAX 0x7F**

Maximum display contrast value

**Examples:**

[ex\\_contrast.nxc](#).

**6.155 Text line constants**

Constants that are for use with getting/setting display data.

**Defines**

- #define [TEXTLINE\\_1](#) 0
- #define [TEXTLINE\\_2](#) 1
- #define [TEXTLINE\\_3](#) 2
- #define [TEXTLINE\\_4](#) 3
- #define [TEXTLINE\\_5](#) 4

- #define [TEXTLINE\\_6](#) 5
- #define [TEXTLINE\\_7](#) 6
- #define [TEXTLINE\\_8](#) 7
- #define [TEXTLINES](#) 8

### 6.155.1 Detailed Description

Constants that are for use with getting/setting display data.

See also:

[SetDisplayNormal\(\)](#), [GetDisplayNormal\(\)](#), [SetDisplayPopup\(\)](#), [GetDisplayPopup\(\)](#)

### 6.155.2 Define Documentation

#### 6.155.2.1 #define TEXTLINE\_1 0

Text line 1

Examples:

[ex\\_GetDisplayNormal.nxc](#), [ex\\_GetDisplayPopup.nxc](#), [ex\\_SetDisplayNormal.nxc](#),  
and [ex\\_SetDisplayPopup.nxc](#).

#### 6.155.2.2 #define TEXTLINE\_2 1

Text line 2

#### 6.155.2.3 #define TEXTLINE\_3 2

Text line 3

#### 6.155.2.4 #define TEXTLINE\_4 3

Text line 4

#### 6.155.2.5 #define TEXTLINE\_5 4

Text line 5

**6.155.2.6 #define TEXTLINE\_6 5**

Text line 6

**6.155.2.7 #define TEXTLINE\_7 6**

Text line 7

**6.155.2.8 #define TEXTLINE\_8 7**

Text line 8

**6.155.2.9 #define TEXTLINES 8**

The number of text lines on the LCD

**6.156 Display module IOMAP offsets**

Constant offsets into the display module IOMAP structure.

**Defines**

- #define [DisplayOffsetPFunc](#) 0
- #define [DisplayOffsetEraseMask](#) 4
- #define [DisplayOffsetUpdateMask](#) 8
- #define [DisplayOffsetPFont](#) 12
- #define [DisplayOffsetPTextLines](#)(p) (((p)\*4)+16)
- #define [DisplayOffsetPStatusText](#) 48
- #define [DisplayOffsetPStatusIcons](#) 52
- #define [DisplayOffsetPScreens](#)(p) (((p)\*4)+56)
- #define [DisplayOffsetPBitmaps](#)(p) (((p)\*4)+68)
- #define [DisplayOffsetPMenuText](#) 84
- #define [DisplayOffsetPMenuIcons](#)(p) (((p)\*4)+88)
- #define [DisplayOffsetPStepIcons](#) 100
- #define [DisplayOffsetDisplay](#) 104
- #define [DisplayOffsetStatusIcons](#)(p) ((p)+108)
- #define [DisplayOffsetStepIcons](#)(p) ((p)+112)
- #define [DisplayOffsetFlags](#) 117
- #define [DisplayOffsetTextLinesCenterFlags](#) 118
- #define [DisplayOffsetNormal](#)(l, w) (((l)\*100)+(w)+119)
- #define [DisplayOffsetPopup](#)(l, w) (((l)\*100)+(w)+919)
- #define [DisplayOffsetContrast](#) 1719

### 6.156.1 Detailed Description

Constant offsets into the display module IOMAP structure.

### 6.156.2 Define Documentation

#### 6.156.2.1 #define DisplayOffsetContrast 1719

Adjust the display contrast with this field

#### 6.156.2.2 #define DisplayOffsetDisplay 104

Display content copied to physical display every 17 mS

#### 6.156.2.3 #define DisplayOffsetEraseMask 4

Section erase mask (executed first)

#### 6.156.2.4 #define DisplayOffsetFlags 117

Update flags enumerated above

#### 6.156.2.5 #define DisplayOffsetNormal(l, w) (((l)\*100)+(w)+119)

Raw display memory for normal screen

#### 6.156.2.6 #define DisplayOffsetPBitmaps(p) (((p)\*4)+68)

Pointer to free bitmap files

#### 6.156.2.7 #define DisplayOffsetPFont 12

Pointer to font file

#### 6.156.2.8 #define DisplayOffsetPFunc 0

Simple draw entry

**6.156.2.9 #define DisplayOffsetPMenuIcons(p) (((p)\*4)+88)**

Pointer to menu icon images (NULL == none)

**6.156.2.10 #define DisplayOffsetPMenuText 84**

Pointer to menu icon text (NULL == none)

**6.156.2.11 #define DisplayOffsetPopup(l, w) (((l)\*100)+(w)+919)**

Raw display memory for popup screen

**6.156.2.12 #define DisplayOffsetPScreens(p) (((p)\*4)+56)**

Pointer to screen bitmap file

**6.156.2.13 #define DisplayOffsetPStatusIcons 52**

Pointer to status icon collection file

**6.156.2.14 #define DisplayOffsetPStatusText 48**

Pointer to status text string

**6.156.2.15 #define DisplayOffsetPStepIcons 100**

Pointer to step icon collection file

**6.156.2.16 #define DisplayOffsetPTextLines(p) (((p)\*4)+16)**

Pointer to text strings

**6.156.2.17 #define DisplayOffsetStatusIcons(p) ((p)+108)**

Index in status icon collection file (index = 0 -> none)

**6.156.2.18 #define DisplayOffsetStepIcons(p) ((p)+112)**

Index in step icon collection file (index = 0 -> none)

**6.156.2.19 #define DisplayOffsetTextLinesCenterFlags 118**

Mask to center TextLines

**6.156.2.20 #define DisplayOffsetUpdateMask 8**

Section update mask (executed next)

**6.157 Comm module constants**

Constants that are part of the NXT firmware's Comm module.

**Modules**

- [Mailbox constants](#)  
*Mailbox number constants should be used to avoid confusing NXT-G users.*
- [Miscellaneous Comm module constants](#)  
*Miscellaneous constants related to the Comm module.*
- [Bluetooth State constants](#)  
*Constants related to the bluetooth state.*
- [Data mode constants](#)  
*Constants related to the bluetooth and hi-speed data modes.*
- [Bluetooth state status constants](#)  
*Constants related to the bluetooth state status.*
- [Remote connection constants](#)  
*Constants for specifying remote connection slots.*
- [Bluetooth hardware status constants](#)  
*Constants related to the bluetooth hardware status.*
- [Hi-speed port constants](#)  
*Constants related to the hi-speed port.*
- [Device status constants](#)  
*Constants referring to DeviceStatus within DeviceTable.*

- [Comm module interface function constants](#)

*Constants for all the Comm module interface functions executable via SysCommExecuteFunction.*

- [Comm module status code constants](#)

*Constants for Comm module status codes.*

- [Comm module IOMAP offsets](#)

*Constant offsets into the Comm module IOMAP structure.*

### 6.157.1 Detailed Description

Constants that are part of the NXT firmware's Comm module.

## 6.158 Miscellaneous Comm module constants

Miscellaneous constants related to the Comm module.

### Defines

- #define [SIZE\\_OF\\_USBBUF](#) 64
- #define [USB\\_PROTOCOL\\_OVERHEAD](#) 2
- #define [SIZE\\_OF\\_USBDATA](#) 62
- #define [SIZE\\_OF\\_HSBUFF](#) 128
- #define [SIZE\\_OF\\_BTBUFF](#) 128
- #define [BT\\_CMD\\_BYTE](#) 1
- #define [SIZE\\_OF\\_BT\\_DEVICE\\_TABLE](#) 30
- #define [SIZE\\_OF\\_BT\\_CONNECT\\_TABLE](#) 4
- #define [SIZE\\_OF\\_BT\\_NAME](#) 16
- #define [SIZE\\_OF\\_BRICK\\_NAME](#) 8
- #define [SIZE\\_OF\\_CLASS\\_OF\\_DEVICE](#) 4
- #define [SIZE\\_OF\\_BT\\_PINCODE](#) 16
- #define [SIZE\\_OF\\_BDADDR](#) 7
- #define [MAX\\_BT\\_MSG\\_SIZE](#) 60000
- #define [BT\\_DEFAULT\\_INQUIRY\\_MAX](#) 0
- #define [BT\\_DEFAULT\\_INQUIRY\\_TIMEOUT\\_LO](#) 15

### 6.158.1 Detailed Description

Miscellaneous constants related to the Comm module.

**6.158.2 Define Documentation****6.158.2.1 #define BT\_CMD\_BYTE 1**

Size of Bluetooth command

**6.158.2.2 #define BT\_DEFAULT\_INQUIRY\_MAX 0**

Bluetooth default inquiry Max (0 == unlimited)

**6.158.2.3 #define BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO 15**

Bluetooth inquiry timeout (15\*1.28 sec = 19.2 sec)

**6.158.2.4 #define MAX\_BT\_MSG\_SIZE 60000**

Max Bluetooth Message Size

**6.158.2.5 #define SIZE\_OF\_BDADDR 7**

Size of Bluetooth Address

**6.158.2.6 #define SIZE\_OF\_BRICK\_NAME 8**

Size of NXT Brick name

**6.158.2.7 #define SIZE\_OF\_BT\_CONNECT\_TABLE 4**

Size of Bluetooth connection table -- Index 0 is always incoming connection

**6.158.2.8 #define SIZE\_OF\_BT\_DEVICE\_TABLE 30**

Size of Bluetooth device table

**6.158.2.9 #define SIZE\_OF\_BT\_NAME 16**

Size of Bluetooth name

**6.158.2.10 #define SIZE\_OF\_BT\_PINCODE 16**

Size of Bluetooth PIN

**6.158.2.11 #define SIZE\_OF\_BTBUF 128**

Size of Bluetooth buffer

**6.158.2.12 #define SIZE\_OF\_CLASS\_OF\_DEVICE 4**

Size of class of device

**6.158.2.13 #define SIZE\_OF\_HSBUF 128**

Size of High Speed Port 4 buffer

**6.158.2.14 #define SIZE\_OF\_USBBUF 64**

Size of USB Buffer in bytes

**6.158.2.15 #define SIZE\_OF\_USBDATA 62**

Size of USB Buffer available for data

**6.158.2.16 #define USB\_PROTOCOL\_OVERHEAD 2**

Size of USB Overhead in bytes -- Command type byte + Command

## 6.159 Bluetooth State constants

Constants related to the bluetooth state.

### Defines

- #define [BT\\_ARM\\_OFF](#) 0
- #define [BT\\_ARM\\_CMD\\_MODE](#) 1
- #define [BT\\_ARM\\_DATA\\_MODE](#) 2

### 6.159.1 Detailed Description

Constants related to the bluetooth state.

### 6.159.2 Define Documentation

#### 6.159.2.1 #define BT\_ARM\_CMD\_MODE 1

BtState constant bluetooth command mode

#### 6.159.2.2 #define BT\_ARM\_DATA\_MODE 2

BtState constant bluetooth data mode

#### 6.159.2.3 #define BT\_ARM\_OFF 0

BtState constant bluetooth off

## 6.160 Data mode constants

Constants related to the bluetooth and hi-speed data modes.

### Defines

- #define [DATA\\_MODE\\_NXT](#) 0x00
- #define [DATA\\_MODE\\_GPS](#) 0x01
- #define [DATA\\_MODE\\_RAW](#) 0x02
- #define [DATA\\_MODE\\_MASK](#) 0x07
- #define [DATA\\_MODE\\_UPDATE](#) 0x08

### 6.160.1 Detailed Description

Constants related to the bluetooth and hi-speed data modes.

### 6.160.2 Define Documentation

#### 6.160.2.1 #define DATA\_MODE\_GPS 0x01

Use GPS data mode

**Examples:**

[ex\\_DataMode.nxc](#).

**6.160.2.2 #define DATA\_MODE\_MASK 0x07**

A mask for the data mode bits.

**6.160.2.3 #define DATA\_MODE\_NXT 0x00**

Use NXT data mode

**Examples:**

[ex\\_DataMode.nxc](#).

**6.160.2.4 #define DATA\_MODE\_RAW 0x02**

Use RAW data mode

**6.160.2.5 #define DATA\_MODE\_UPDATE 0x08**

Indicates that the data mode has been changed.

**6.161 Bluetooth state status constants**

Constants related to the bluetooth state status.

**Defines**

- #define [BT\\_BRICK\\_VISIBILITY](#) 0x01
- #define [BT\\_BRICK\\_PORT\\_OPEN](#) 0x02
- #define [BT\\_CONNECTION\\_0\\_ENABLE](#) 0x10
- #define [BT\\_CONNECTION\\_1\\_ENABLE](#) 0x20
- #define [BT\\_CONNECTION\\_2\\_ENABLE](#) 0x40
- #define [BT\\_CONNECTION\\_3\\_ENABLE](#) 0x80

**6.161.1 Detailed Description**

Constants related to the bluetooth state status.

### 6.161.2 Define Documentation

#### 6.161.2.1 #define BT\_BRICK\_PORT\_OPEN 0x02

BtStateStatus port open bit

#### 6.161.2.2 #define BT\_BRICK\_VISIBILITY 0x01

BtStateStatus brick visibility bit

#### 6.161.2.3 #define BT\_CONNECTION\_0\_ENABLE 0x10

BtStateStatus connection 0 enable/disable bit

#### 6.161.2.4 #define BT\_CONNECTION\_1\_ENABLE 0x20

BtStateStatus connection 1 enable/disable bit

#### 6.161.2.5 #define BT\_CONNECTION\_2\_ENABLE 0x40

BtStateStatus connection 2 enable/disable bit

#### 6.161.2.6 #define BT\_CONNECTION\_3\_ENABLE 0x80

BtStateStatus connection 3 enable/disable bit

## 6.162 Remote connection constants

Constants for specifying remote connection slots.

### Defines

- #define [CONN\\_BT0](#) 0x0
- #define [CONN\\_BT1](#) 0x1
- #define [CONN\\_BT2](#) 0x2
- #define [CONN\\_BT3](#) 0x3
- #define [CONN\\_HS4](#) 0x4
- #define [CONN\\_HS\\_ALL](#) 0x4
- #define [CONN\\_HS\\_1](#) 0x5
- #define [CONN\\_HS\\_2](#) 0x6

- #define `CONN_HS_3` 0x7
- #define `CONN_HS_4` 0x8
- #define `CONN_HS_5` 0x9
- #define `CONN_HS_6` 0xa
- #define `CONN_HS_7` 0xb
- #define `CONN_HS_8` 0xc

### 6.162.1 Detailed Description

Constants for specifying remote connection slots.

### 6.162.2 Define Documentation

#### 6.162.2.1 #define `CONN_BT0` 0x0

Bluetooth connection 0

#### 6.162.2.2 #define `CONN_BT1` 0x1

Bluetooth connection 1

### Examples:

```

ex_RemoteCloseFile.nxc,      ex_RemoteConnectionIdle.nxc,      ex_-
RemoteConnectionWrite.nxc,   ex_RemoteDatalogRead.nxc,        ex_-
RemoteDatalogSetTimes.nxc,   ex_RemoteDeleteFile.nxc,         ex_-
RemoteDeleteUserFlash.nxc,   ex_RemoteFindFirstFile.nxc,      ex_-
RemoteFindNextFile.nxc,     ex_RemoteGetBatteryLevel.nxc,     ex_-
RemoteGetBluetoothAddress.nxc, ex_RemoteGetConnectionCount.nxc,
ex_RemoteGetConnectionName.nxc, ex_RemoteGetContactCount.nxc,    ex_-
RemoteGetContactName.nxc,    ex_RemoteGetCurrentProgramName.nxc,
ex_RemoteGetDeviceInfo.nxc,  ex_RemoteGetFirmwareVersion.nxc,
ex_RemoteGetInputValues.nxc, ex_RemoteGetOutputState.nxc,
ex_RemoteGetProperty.nxc,    ex_RemoteIOMapRead.nxc,          ex_-
RemoteIOMapWriteBytes.nxc,   ex_RemoteIOMapWriteValue.nxc,
ex_RemoteLowSpeedGetStatus.nxc, ex_RemoteLowSpeedRead.nxc,
ex_RemoteLowSpeedWrite.nxc,  ex_RemoteOpenAppendData.nxc,
ex_RemoteOpenRead.nxc,      ex_RemoteOpenWrite.nxc,          ex_-
RemoteOpenWriteData.nxc,    ex_RemoteOpenWriteLinear.nxc,    ex_-
RemotePollCommand.nxc,      ex_RemotePollCommandLength.nxc,  ex_-
RemoteRead.nxc, ex_RemoteRenameFile.nxc, ex_RemoteResetTachoCount.nxc,
ex_RemoteSetProperty.nxc, and ex_RemoteWrite.nxc.

```

**6.162.2.3 #define CONN\_BT2 0x2**

Bluetooth connection 2

**6.162.2.4 #define CONN\_BT3 0x3**

Bluetooth connection 3

**6.162.2.5 #define CONN\_HS4 0x4**

RS485 (hi-speed) connection (port 4, all devices)

**6.162.2.6 #define CONN\_HS\_1 0x5**

RS485 (hi-speed) connection (port 4, device address 1)

**6.162.2.7 #define CONN\_HS\_2 0x6**

RS485 (hi-speed) connection (port 4, device address 2)

**6.162.2.8 #define CONN\_HS\_3 0x7**

RS485 (hi-speed) connection (port 4, device address 3)

**6.162.2.9 #define CONN\_HS\_4 0x8**

RS485 (hi-speed) connection (port 4, device address 4)

**6.162.2.10 #define CONN\_HS\_5 0x9**

RS485 (hi-speed) connection (port 4, device address 5)

**6.162.2.11 #define CONN\_HS\_6 0xa**

RS485 (hi-speed) connection (port 4, device address 6)

**6.162.2.12 #define CONN\_HS\_7 0xb**

RS485 (hi-speed) connection (port 4, device address 7)

**6.162.2.13 #define CONN\_HS\_8 0xc**

RS485 (hi-speed) connection (port 4, device address 8)

**6.162.2.14 #define CONN\_HS\_ALL 0x4**

RS485 (hi-speed) connection (port 4, all devices)

**6.163 Bluetooth hardware status constants**

Constants related to the bluetooth hardware status.

**Defines**

- #define [BT\\_ENABLE](#) 0x00
- #define [BT\\_DISABLE](#) 0x01

**6.163.1 Detailed Description**

Constants related to the bluetooth hardware status.

**6.163.2 Define Documentation****6.163.2.1 #define BT\_DISABLE 0x01**

BtHwStatus bluetooth disable

**6.163.2.2 #define BT\_ENABLE 0x00**

BtHwStatus bluetooth enable

**6.164 Hi-speed port constants**

Constants related to the hi-speed port.

**Modules**

- [Hi-speed port flags constants](#)  
*Constants related to the hi-speed port flags.*

- [Hi-speed port state constants](#)  
*Constants related to the hi-speed port state.*
- [Hi-speed port SysCommHSControl constants](#)  
*Constants for use with the SysCommHSControl API function.*
- [Hi-speed port baud rate constants](#)  
*Constants for configuring the hi-speed port baud rate (HsSpeed).*
- [Hi-speed port UART mode constants](#)  
*Constants referring to HsMode UART configuration settings.*
- [Hi-speed port address constants](#)  
*Constants that are used to specify the Hi-speed (RS-485) port device address.*

#### 6.164.1 Detailed Description

Constants related to the hi-speed port.

### 6.165 Hi-speed port flags constants

Constants related to the hi-speed port flags.

#### Defines

- #define [HS\\_UPDATE](#) 1

#### 6.165.1 Detailed Description

Constants related to the hi-speed port flags.

#### 6.165.2 Define Documentation

##### 6.165.2.1 #define HS\_UPDATE 1

HsFlags high speed update required

## 6.166 Hi-speed port state constants

Constants related to the hi-speed port state.

### Defines

- #define [HS\\_INITIALISE](#) 1
- #define [HS\\_INIT\\_RECEIVER](#) 2
- #define [HS\\_SEND\\_DATA](#) 3
- #define [HS\\_DISABLE](#) 4
- #define [HS\\_ENABLE](#) 5
- #define [HS\\_DEFAULT](#) 6
- #define [HS\\_BYTES\\_REMAINING](#) 16

### 6.166.1 Detailed Description

Constants related to the hi-speed port state.

### 6.166.2 Define Documentation

#### 6.166.2.1 #define [HS\\_BYTES\\_REMAINING](#) 16

HsState bytes remaining to be sent

#### 6.166.2.2 #define [HS\\_DEFAULT](#) 6

HsState default

#### 6.166.2.3 #define [HS\\_DISABLE](#) 4

HsState disable

#### 6.166.2.4 #define [HS\\_ENABLE](#) 5

HsState enable

#### 6.166.2.5 #define [HS\\_INIT\\_RECEIVER](#) 2

HsState initialize receiver

**6.166.2.6 #define HS\_INITIALISE 1**

HsState initialize

**6.166.2.7 #define HS\_SEND\_DATA 3**

HsState send data

**6.167 Hi-speed port SysCommHSControl constants**

Constants for use with the SysCommHSControl API function.

**Defines**

- #define [HS\\_CTRL\\_INIT](#) 0
- #define [HS\\_CTRL\\_UART](#) 1
- #define [HS\\_CTRL\\_EXIT](#) 2

**6.167.1 Detailed Description**

Constants for use with the SysCommHSControl API function.

**See also:**

[SysCommHSControl\(\)](#)

**6.167.2 Define Documentation****6.167.2.1 #define HS\_CTRL\_EXIT 2**

Ddisable the high speed port

**6.167.2.2 #define HS\_CTRL\_INIT 0**

Enable the high speed port

**Examples:**

[ex\\_SysCommHSControl.nxc](#).

### 6.167.2.3 #define HS\_CTRL\_UART 1

Setup the high speed port UART configuration

## 6.168 Hi-speed port baud rate constants

Constants for configuring the hi-speed port baud rate (HsSpeed).

### Defines

- #define HS\_BAUD\_1200 0
- #define HS\_BAUD\_2400 1
- #define HS\_BAUD\_3600 2
- #define HS\_BAUD\_4800 3
- #define HS\_BAUD\_7200 4
- #define HS\_BAUD\_9600 5
- #define HS\_BAUD\_14400 6
- #define HS\_BAUD\_19200 7
- #define HS\_BAUD\_28800 8
- #define HS\_BAUD\_38400 9
- #define HS\_BAUD\_57600 10
- #define HS\_BAUD\_76800 11
- #define HS\_BAUD\_115200 12
- #define HS\_BAUD\_230400 13
- #define HS\_BAUD\_460800 14
- #define HS\_BAUD\_921600 15
- #define HS\_BAUD\_DEFAULT 15

### 6.168.1 Detailed Description

Constants for configuring the hi-speed port baud rate (HsSpeed).

### 6.168.2 Define Documentation

#### 6.168.2.1 #define HS\_BAUD\_115200 12

HsSpeed 115200 Baud

#### 6.168.2.2 #define HS\_BAUD\_1200 0

HsSpeed 1200 Baud

6.168.2.3 #define HS\_BAUD\_14400 6

HsSpeed 14400 Baud

6.168.2.4 #define HS\_BAUD\_19200 7

HsSpeed 19200 Baud

6.168.2.5 #define HS\_BAUD\_230400 13

HsSpeed 230400 Baud

6.168.2.6 #define HS\_BAUD\_2400 1

HsSpeed 2400 Baud

6.168.2.7 #define HS\_BAUD\_28800 8

HsSpeed 28800 Baud

6.168.2.8 #define HS\_BAUD\_3600 2

HsSpeed 3600 Baud

6.168.2.9 #define HS\_BAUD\_38400 9

HsSpeed 38400 Baud

6.168.2.10 #define HS\_BAUD\_460800 14

HsSpeed 460800 Baud

6.168.2.11 #define HS\_BAUD\_4800 3

HsSpeed 4800 Baud

6.168.2.12 #define HS\_BAUD\_57600 10

HsSpeed 57600 Baud

**6.168.2.13 #define HS\_BAUD\_7200 4**

HsSpeed 7200 Baud

**6.168.2.14 #define HS\_BAUD\_76800 11**

HsSpeed 76800 Baud

**6.168.2.15 #define HS\_BAUD\_921600 15**

HsSpeed 921600 Baud

**6.168.2.16 #define HS\_BAUD\_9600 5**

HsSpeed 9600 Baud

**6.168.2.17 #define HS\_BAUD\_DEFAULT 15**

HsSpeed default Baud (921600)

**Examples:**[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).**6.169 Hi-speed port UART mode constants**

Constants referring to HsMode UART configuration settings.

**Modules**

- [Hi-speed port data bits constants](#)  
*Constants referring to HsMode (number of data bits).*
- [Hi-speed port stop bits constants](#)  
*Constants referring to HsMode (number of stop bits).*
- [Hi-speed port parity constants](#)  
*Constants referring to HsMode (parity).*
- [Hi-speed port combined UART constants](#)  
*Constants that combine data bits, parity, and stop bits into a single value.*

## Defines

- #define [HS\\_MODE\\_UART\\_RS485](#) 0x0
- #define [HS\\_MODE\\_UART\\_RS232](#) 0x1
- #define [HS\\_MODE\\_MASK](#) 0xFFFF0
- #define [HS\\_UART\\_MASK](#) 0x000F
- #define [HS\\_MODE\\_DEFAULT](#) HS\_MODE\_8N1

### 6.169.1 Detailed Description

Constants referring to HsMode UART configuration settings.

### 6.169.2 Define Documentation

#### 6.169.2.1 #define [HS\\_MODE\\_DEFAULT](#) [HS\\_MODE\\_8N1](#)

HsMode default mode (8 data bits, no parity, 1 stop bit)

#### Examples:

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

#### 6.169.2.2 #define [HS\\_MODE\\_MASK](#) [0xFFFF0](#)

HsMode mode mask

#### 6.169.2.3 #define [HS\\_MODE\\_UART\\_RS232](#) [0x1](#)

HsMode UART in normal or RS232 mode

#### 6.169.2.4 #define [HS\\_MODE\\_UART\\_RS485](#) [0x0](#)

HsMode UART in default or RS485 mode

#### 6.169.2.5 #define [HS\\_UART\\_MASK](#) [0x000F](#)

HsMode UART mask

## 6.170 Hi-speed port data bits constants

Constants referring to HsMode (number of data bits).

**Defines**

- #define [HS\\_MODE\\_5\\_DATA](#) 0x0000
- #define [HS\\_MODE\\_6\\_DATA](#) 0x0040
- #define [HS\\_MODE\\_7\\_DATA](#) 0x0080
- #define [HS\\_MODE\\_8\\_DATA](#) 0x00C0

**6.170.1 Detailed Description**

Constants referring to HsMode (number of data bits).

**6.170.2 Define Documentation****6.170.2.1 #define HS\_MODE\_5\_DATA 0x0000**

HsMode 5 data bits

**6.170.2.2 #define HS\_MODE\_6\_DATA 0x0040**

HsMode 6 data bits

**6.170.2.3 #define HS\_MODE\_7\_DATA 0x0080**

HsMode 7 data bits

**6.170.2.4 #define HS\_MODE\_8\_DATA 0x00C0**

HsMode 8 data bits

**6.171 Hi-speed port stop bits constants**

Constants referring to HsMode (number of stop bits).

**Defines**

- #define [HS\\_MODE\\_10\\_STOP](#) 0x0000
- #define [HS\\_MODE\\_15\\_STOP](#) 0x1000
- #define [HS\\_MODE\\_20\\_STOP](#) 0x2000

### 6.171.1 Detailed Description

Constants referring to HsMode (number of stop bits).

### 6.171.2 Define Documentation

#### 6.171.2.1 #define HS\_MODE\_10\_STOP 0x0000

HsMode 1 stop bit

#### 6.171.2.2 #define HS\_MODE\_15\_STOP 0x1000

HsMode 1.5 stop bits

#### 6.171.2.3 #define HS\_MODE\_20\_STOP 0x2000

HsMode 2 stop bits

## 6.172 Hi-speed port parity constants

Constants referring to HsMode (parity).

### Defines

- #define [HS\\_MODE\\_E\\_PARITY](#) 0x0000
- #define [HS\\_MODE\\_O\\_PARITY](#) 0x0200
- #define [HS\\_MODE\\_S\\_PARITY](#) 0x0400
- #define [HS\\_MODE\\_M\\_PARITY](#) 0x0600
- #define [HS\\_MODE\\_N\\_PARITY](#) 0x0800

### 6.172.1 Detailed Description

Constants referring to HsMode (parity).

### 6.172.2 Define Documentation

#### 6.172.2.1 #define HS\_MODE\_E\_PARITY 0x0000

HsMode Even parity

**6.172.2.2 #define HS\_MODE\_M\_PARITY 0x0600**

HsMode Mark parity

**6.172.2.3 #define HS\_MODE\_N\_PARITY 0x0800**

HsMode No parity

**6.172.2.4 #define HS\_MODE\_O\_PARITY 0x0200**

HsMode Odd parity

**6.172.2.5 #define HS\_MODE\_S\_PARITY 0x0400**

HsMode Space parity

## 6.173 Hi-speed port combined UART constants

Constants that combine data bits, parity, and stop bits into a single value.

### Defines

- #define [HS\\_MODE\\_8N1](#) (HS\_MODE\_8\_DATA|HS\_MODE\_N\_PARITY|HS\_MODE\_10\_STOP)
- #define [HS\\_MODE\\_7E1](#) (HS\_MODE\_7\_DATA|HS\_MODE\_E\_PARITY|HS\_MODE\_10\_STOP)

### 6.173.1 Detailed Description

Constants that combine data bits, parity, and stop bits into a single value.

### 6.173.2 Define Documentation

**6.173.2.1 #define HS\_MODE\_7E1 (HS\_MODE\_7\_DATA|HS\_MODE\_E\_PARITY|HS\_MODE\_10\_STOP)**

HsMode 7 data bits, even parity, 1 stop bit

**6.173.2.2** `#define HS_MODE_8N1 (HS_MODE_8_DATA|HS_MODE_N_-  
PARITY|HS_MODE_10_STOP)`

HsMode 8 data bits, no parity, 1 stop bit

**Examples:**

[ex\\_sethsmode.nxc](#).

## 6.174 Hi-speed port address constants

Constants that are used to specify the Hi-speed (RS-485) port device address.

### Defines

- `#define HS_ADDRESS_ALL 0`
- `#define HS_ADDRESS_1 1`
- `#define HS_ADDRESS_2 2`
- `#define HS_ADDRESS_3 3`
- `#define HS_ADDRESS_4 4`
- `#define HS_ADDRESS_5 5`
- `#define HS_ADDRESS_6 6`
- `#define HS_ADDRESS_7 7`
- `#define HS_ADDRESS_8 8`

### 6.174.1 Detailed Description

Constants that are used to specify the Hi-speed (RS-485) port device address.

### 6.174.2 Define Documentation

#### 6.174.2.1 `#define HS_ADDRESS_1 1`

HsAddress device address 1

#### 6.174.2.2 `#define HS_ADDRESS_2 2`

HsAddress device address 2

#### 6.174.2.3 `#define HS_ADDRESS_3 3`

HsAddress device address 3

**6.174.2.4 #define HS\_ADDRESS\_4 4**

HsAddress device address 4

**6.174.2.5 #define HS\_ADDRESS\_5 5**

HsAddress device address 5

**6.174.2.6 #define HS\_ADDRESS\_6 6**

HsAddress device address 6

**6.174.2.7 #define HS\_ADDRESS\_7 7**

HsAddress device address 7

**6.174.2.8 #define HS\_ADDRESS\_8 8**

HsAddress device address 8

**6.174.2.9 #define HS\_ADDRESS\_ALL 0**

HsAddress all devices

**6.175 Device status constants**

Constants referring to DeviceStatus within DeviceTable.

**Defines**

- #define [BT\\_DEVICE\\_EMPTY](#) 0x00
- #define [BT\\_DEVICE\\_UNKNOWN](#) 0x01
- #define [BT\\_DEVICE\\_KNOWN](#) 0x02
- #define [BT\\_DEVICE\\_NAME](#) 0x40
- #define [BT\\_DEVICE\\_AWAY](#) 0x80

**6.175.1 Detailed Description**

Constants referring to DeviceStatus within DeviceTable.

### 6.175.2 Define Documentation

#### 6.175.2.1 #define BT\_DEVICE\_AWAY 0x80

Bluetooth device away

#### 6.175.2.2 #define BT\_DEVICE\_EMPTY 0x00

Bluetooth device table empty

#### 6.175.2.3 #define BT\_DEVICE\_KNOWN 0x02

Bluetooth device known

#### 6.175.2.4 #define BT\_DEVICE\_NAME 0x40

Bluetooth device name

#### 6.175.2.5 #define BT\_DEVICE\_UNKNOWN 0x01

Bluetooth device unknown

## 6.176 Comm module interface function constants

Constants for all the Comm module interface functions executable via SysCommExecuteFunction.

### Defines

- #define [INTF\\_SENDFILE](#) 0
- #define [INTF\\_SEARCH](#) 1
- #define [INTF\\_STOPSEARCH](#) 2
- #define [INTF\\_CONNECT](#) 3
- #define [INTF\\_DISCONNECT](#) 4
- #define [INTF\\_DISCONNECTALL](#) 5
- #define [INTF\\_REMOVEDEVICE](#) 6
- #define [INTF\\_VISIBILITY](#) 7
- #define [INTF\\_SETCMDMODE](#) 8
- #define [INTF\\_OPENSTREAM](#) 9
- #define [INTF\\_SENDDATA](#) 10

- #define [INTF\\_FACTORYRESET](#) 11
- #define [INTF\\_BTON](#) 12
- #define [INTF\\_BTOFF](#) 13
- #define [INTF\\_SETBTNAME](#) 14
- #define [INTF\\_EXTREAD](#) 15
- #define [INTF\\_PINREQ](#) 16
- #define [INTF\\_CONNECTREQ](#) 17
- #define [INTF\\_CONNECTBYNAME](#) 18

### 6.176.1 Detailed Description

Constants for all the Comm module interface functions executable via SysCommExecuteFunction.

See also:

[SysCommExecuteFunction\(\)](#)

### 6.176.2 Define Documentation

#### 6.176.2.1 #define INTF\_BTOFF 13

Turn off the bluetooth radio

Examples:

[ex\\_syscommexecutefunction.nxc](#).

#### 6.176.2.2 #define INTF\_BTON 12

Turn on the bluetooth radio

#### 6.176.2.3 #define INTF\_CONNECT 3

Connect to one of the known devices

#### 6.176.2.4 #define INTF\_CONNECTBYNAME 18

Connect to a bluetooth device by name

#### 6.176.2.5 #define INTF\_CONNECTREQ 17

Connection request from another device

**6.176.2.6 #define INTF\_DISCONNECT 4**

Disconnect from one of the connected devices

**6.176.2.7 #define INTF\_DISCONNECTALL 5**

Disconnect all devices

**6.176.2.8 #define INTF\_EXTREAD 15**

External read request

**6.176.2.9 #define INTF\_FACTORYRESET 11**

Reset bluetooth settings to factory values

**6.176.2.10 #define INTF\_OPENSTREAM 9**

Open a bluetooth stream

**6.176.2.11 #define INTF\_PINREQ 16**

Bluetooth PIN request

**6.176.2.12 #define INTF\_REMOVEDEVICE 6**

Remove a device from the known devices table

**6.176.2.13 #define INTF\_SEARCH 1**

Search for bluetooth devices

**6.176.2.14 #define INTF\_SENDDATA 10**

Send data over a bluetooth connection

**6.176.2.15 #define INTF\_SENDFILE 0**

Send a file via bluetooth to another device

**6.176.2.16 #define INTF\_SETBTNAME 14**

Set the bluetooth name

**6.176.2.17 #define INTF\_SETCMDMODE 8**

Set bluetooth into command mode

**6.176.2.18 #define INTF\_STOPSEARCH 2**

Stop searching for bluetooth devices

**6.176.2.19 #define INTF\_VISIBILITY 7**

Set the bluetooth visibility on or off

**6.177 Comm module status code constants**

Constants for Comm module status codes.

**Defines**

- #define [LR\\_SUCCESS](#) 0x50
- #define [LR\\_COULD\\_NOT\\_SAVE](#) 0x51
- #define [LR\\_STORE\\_IS\\_FULL](#) 0x52
- #define [LR\\_ENTRY\\_REMOVED](#) 0x53
- #define [LR\\_UNKNOWN\\_ADDR](#) 0x54
- #define [USB\\_CMD\\_READY](#) 0x01
- #define [BT\\_CMD\\_READY](#) 0x02
- #define [HS\\_CMD\\_READY](#) 0x04

**6.177.1 Detailed Description**

Constants for Comm module status codes.

**6.177.2 Define Documentation****6.177.2.1 #define BT\_CMD\_READY 0x02**

A constant representing bluetooth direct command

**6.177.2.2 #define HS\_CMD\_READY 0x04**

A constant representing high speed direct command

**6.177.2.3 #define LR\_COULD\_NOT\_SAVE 0x51**

Bluetooth list result could not save

**6.177.2.4 #define LR\_ENTRY\_REMOVED 0x53**

Bluetooth list result entry removed

**6.177.2.5 #define LR\_STORE\_IS\_FULL 0x52**

Bluetooth list result store is full

**6.177.2.6 #define LR\_SUCCESS 0x50**

Bluetooth list result success

**6.177.2.7 #define LR\_UNKNOWN\_ADDR 0x54**

Bluetooth list result unknown address

**6.177.2.8 #define USB\_CMD\_READY 0x01**

A constant representing usb direct command

**6.178 Comm module IOMAP offsets**

Constant offsets into the Comm module IOMAP structure.

**Defines**

- #define [CommOffsetPFunc](#) 0
- #define [CommOffsetPFuncTwo](#) 4
- #define [CommOffsetBtDeviceTableName\(p\)](#) (((p)\*31)+8)
- #define [CommOffsetBtDeviceTableClassOfDevice\(p\)](#) (((p)\*31)+24)
- #define [CommOffsetBtDeviceTableBdAddr\(p\)](#) (((p)\*31)+28)

- #define `CommOffsetBtDeviceTableDeviceStatus(p)` (((p)\*31)+35)
- #define `CommOffsetBtConnectTableName(p)` (((p)\*47)+938)
- #define `CommOffsetBtConnectTableClassOfDevice(p)` (((p)\*47)+954)
- #define `CommOffsetBtConnectTablePinCode(p)` (((p)\*47)+958)
- #define `CommOffsetBtConnectTableBdAddr(p)` (((p)\*47)+974)
- #define `CommOffsetBtConnectTableHandleNr(p)` (((p)\*47)+981)
- #define `CommOffsetBtConnectTableStreamStatus(p)` (((p)\*47)+982)
- #define `CommOffsetBtConnectTableLinkQuality(p)` (((p)\*47)+983)
- #define `CommOffsetBrickDataName` 1126
- #define `CommOffsetBrickDataBluecoreVersion` 1142
- #define `CommOffsetBrickDataBdAddr` 1144
- #define `CommOffsetBrickDataBtStateStatus` 1151
- #define `CommOffsetBrickDataBtHwStatus` 1152
- #define `CommOffsetBrickDataTimeOutValue` 1153
- #define `CommOffsetBtInBufBuf` 1157
- #define `CommOffsetBtInBufInPtr` 1285
- #define `CommOffsetBtInBufOutPtr` 1286
- #define `CommOffsetBtOutBufBuf` 1289
- #define `CommOffsetBtOutBufInPtr` 1417
- #define `CommOffsetBtOutBufOutPtr` 1418
- #define `CommOffsetHsInBufBuf` 1421
- #define `CommOffsetHsInBufInPtr` 1549
- #define `CommOffsetHsInBufOutPtr` 1550
- #define `CommOffsetHsOutBufBuf` 1553
- #define `CommOffsetHsOutBufInPtr` 1681
- #define `CommOffsetHsOutBufOutPtr` 1682
- #define `CommOffsetUsbInBufBuf` 1685
- #define `CommOffsetUsbInBufInPtr` 1749
- #define `CommOffsetUsbInBufOutPtr` 1750
- #define `CommOffsetUsbOutBufBuf` 1753
- #define `CommOffsetUsbOutBufInPtr` 1817
- #define `CommOffsetUsbOutBufOutPtr` 1818
- #define `CommOffsetUsbPollBufBuf` 1821
- #define `CommOffsetUsbPollBufInPtr` 1885
- #define `CommOffsetUsbPollBufOutPtr` 1886
- #define `CommOffsetBtDeviceCnt` 1889
- #define `CommOffsetBtDeviceNameCnt` 1890
- #define `CommOffsetHsFlags` 1891
- #define `CommOffsetHsSpeed` 1892
- #define `CommOffsetHsState` 1893
- #define `CommOffsetUsbState` 1894
- #define `CommOffsetHsAddress` 1895
- #define `CommOffsetHsMode` 1896
- #define `CommOffsetBtDataMode` 1898
- #define `CommOffsetHsDataMode` 1899

### 6.178.1 Detailed Description

Constant offsets into the Comm module IOMAP structure.

### 6.178.2 Define Documentation

#### 6.178.2.1 #define CommOffsetBrickDataBdAddr 1144

Offset to Bluetooth address (7 bytes)

#### 6.178.2.2 #define CommOffsetBrickDataBluecoreVersion 1142

Offset to Bluecore version (2 bytes)

#### 6.178.2.3 #define CommOffsetBrickDataBtHwStatus 1152

Offset to BtHwStatus (1 byte)

#### 6.178.2.4 #define CommOffsetBrickDataBtStateStatus 1151

Offset to BtStateStatus (1 byte)

#### 6.178.2.5 #define CommOffsetBrickDataName 1126

Offset to brick name (16 bytes)

#### 6.178.2.6 #define CommOffsetBrickDataTimeOutValue 1153

Offset to data timeout value (1 byte)

#### 6.178.2.7 #define CommOffsetBtConnectTableBdAddr(p) (((p)\*47)+974)

Offset to Bluetooth connect table address (7 bytes)

#### 6.178.2.8 #define CommOffsetBtConnectTableClassOfDevice(p) (((p)\*47)+954)

Offset to Bluetooth connect table device class (4 bytes)

**6.178.2.9 #define CommOffsetBtConnectTableHandleNr(p) (((p)\*47)+981)**

Offset to Bluetooth connect table handle (1 byte)

**6.178.2.10 #define CommOffsetBtConnectTableLinkQuality(p) (((p)\*47)+983)**

Offset to Bluetooth connect table link quality (1 byte)

**6.178.2.11 #define CommOffsetBtConnectTableName(p) (((p)\*47)+938)**

Offset to Bluetooth connect table name (16 bytes)

**6.178.2.12 #define CommOffsetBtConnectTablePinCode(p) (((p)\*47)+958)**

Offset to Bluetooth connect table pin code (16 bytes)

**6.178.2.13 #define CommOffsetBtConnectTableStreamStatus(p) (((p)\*47)+982)**

Offset to Bluetooth connect table stream status (1 byte)

**6.178.2.14 #define CommOffsetBtDataMode 1898**

Offset to Bluetooth data mode (1 byte)

**6.178.2.15 #define CommOffsetBtDeviceCnt 1889**

Offset to Bluetooth device count (1 byte)

**6.178.2.16 #define CommOffsetBtDeviceNameCnt 1890**

Offset to Bluetooth device name count (1 byte)

**6.178.2.17 #define CommOffsetBtDeviceTableBdAddr(p) (((p)\*31)+28)**

Offset to Bluetooth device table address (7 bytes)

**6.178.2.18 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)\*31)+24)**

Offset to Bluetooth device table device class (4 bytes)

**6.178.2.19 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)\*31)+35)**

Offset to Bluetooth device table status (1 byte)

**6.178.2.20 #define CommOffsetBtDeviceTableName(p) (((p)\*31)+8)**

Offset to BT device table name (16 bytes)

**6.178.2.21 #define CommOffsetBtInBufBuf 1157**

Offset to Bluetooth input buffer data (128 bytes)

**6.178.2.22 #define CommOffsetBtInBufInPtr 1285**

Offset to Bluetooth input buffer front pointer (1 byte)

**6.178.2.23 #define CommOffsetBtInBufOutPtr 1286**

Offset to Bluetooth output buffer back pointer (1 byte)

**6.178.2.24 #define CommOffsetBtOutBufBuf 1289**

Offset to Bluetooth output buffer offset data (128 bytes)

**6.178.2.25 #define CommOffsetBtOutBufInPtr 1417**

Offset to Bluetooth output buffer front pointer (1 byte)

**6.178.2.26 #define CommOffsetBtOutBufOutPtr 1418**

Offset to Bluetooth output buffer back pointer (1 byte)

**6.178.2.27 #define CommOffsetHsAddress 1895**

Offset to High Speed address (1 byte)

**6.178.2.28 #define CommOffsetHsDataMode 1899**

Offset to High Speed data mode (1 byte)

**6.178.2.29 #define CommOffsetHsFlags 1891**

Offset to High Speed flags (1 byte)

**6.178.2.30 #define CommOffsetHsInBufBuf 1421**

Offset to High Speed input buffer data (128 bytes)

**6.178.2.31 #define CommOffsetHsInBufInPtr 1549**

Offset to High Speed input buffer front pointer (1 byte)

**6.178.2.32 #define CommOffsetHsInBufOutPtr 1550**

Offset to High Speed input buffer back pointer (1 byte)

**6.178.2.33 #define CommOffsetHsMode 1896**

Offset to High Speed mode (2 bytes)

**6.178.2.34 #define CommOffsetHsOutBufBuf 1553**

Offset to High Speed output buffer data (128 bytes)

**6.178.2.35 #define CommOffsetHsOutBufInPtr 1681**

Offset to High Speed output buffer front pointer (1 byte)

**6.178.2.36 #define CommOffsetHsOutBufOutPtr 1682**

Offset to High Speed output buffer back pointer (1 byte)

**6.178.2.37 #define CommOffsetHsSpeed 1892**

Offset to High Speed speed (1 byte)

**6.178.2.38 #define CommOffsetHsState 1893**

Offset to High Speed state (1 byte)

**6.178.2.39 #define CommOffsetPFunc 0**

Offset to the Comm module first function pointer (4 bytes)

**6.178.2.40 #define CommOffsetPFuncTwo 4**

Offset to the Comm module second function pointer (4 bytes)

**6.178.2.41 #define CommOffsetUsbInBufBuf 1685**

Offset to Usb input buffer data (64 bytes)

**6.178.2.42 #define CommOffsetUsbInBufInPtr 1749**

Offset to Usb input buffer front pointer (1 byte)

**6.178.2.43 #define CommOffsetUsbInBufOutPtr 1750**

Offset to Usb input buffer back pointer (1 byte)

**6.178.2.44 #define CommOffsetUsbOutBufBuf 1753**

Offset to Usb output buffer data (64 bytes)

**6.178.2.45 #define CommOffsetUsbOutBufInPtr 1817**

Offset to Usb output buffer front pointer (1 byte)

**6.178.2.46 #define CommOffsetUsbOutBufOutPtr 1818**

Offset to Usb output buffer back pointer (1 byte)

**6.178.2.47 #define CommOffsetUsbPollBufBuf 1821**

Offset to Usb Poll buffer data (64 bytes)

**6.178.2.48 #define CommOffsetUsbPollBufInPtr 1885**

Offset to Usb Poll buffer front pointer (1 byte)

**6.178.2.49 #define CommOffsetUsbPollBufOutPtr 1886**

Offset to Usb Poll buffer back pointer (1 byte)

**6.178.2.50 #define CommOffsetUsbState 1894**

Offset to Usb State (1 byte)

**6.179 RCX constants**

Constants that are for use with devices that communicate with the RCX or Scout programmable bricks via IR such as the HiTechnic IRLink or the MindSensors nRLink.

**Modules**

- [RCX output constants](#)  
*Constants for use when choosing RCX outputs.*
- [RCX output mode constants](#)  
*Constants for use when configuring RCX output mode.*
- [RCX output direction constants](#)  
*Constants for use when configuring RCX output direction.*
- [RCX output power constants](#)  
*Constants for use when configuring RCX output power.*
- [RCX IR remote constants](#)  
*Constants for use when simulating RCX IR remote messages.*
- [RCX and Scout sound constants](#)  
*Constants for use when playing standard RCX and Scout sounds.*
- [Scout constants](#)  
*Constants for use when controlling the Scout brick.*
- [RCX and Scout source constants](#)  
*Constants for use when specifying RCX and Scout sources.*
- [RCX and Scout opcode constants](#)  
*Constants for use when specifying RCX and Scout opcodes.*

### 6.179.1 Detailed Description

Constants that are for use with devices that communicate with the RCX or Scout programmable bricks via IR such as the HiTechnic IRLink or the MindSensors nRLink.

## 6.180 RCX output constants

Constants for use when choosing RCX outputs.

### Defines

- #define [RCX\\_OUT\\_A](#) 0x01
- #define [RCX\\_OUT\\_B](#) 0x02
- #define [RCX\\_OUT\\_C](#) 0x04
- #define [RCX\\_OUT\\_AB](#) 0x03
- #define [RCX\\_OUT\\_AC](#) 0x05
- #define [RCX\\_OUT\\_BC](#) 0x06
- #define [RCX\\_OUT\\_ABC](#) 0x07

### 6.180.1 Detailed Description

Constants for use when choosing RCX outputs.

### 6.180.2 Define Documentation

#### 6.180.2.1 #define RCX\_OUT\_A 0x01

RCX Output A

### Examples:

```

ex_HTRCXDisableOutput.nxc,      ex_HTRCXEnableOutput.nxc,      ex_
HTRCXFloat.nxc,                ex_HTRCXFwd.nxc,                ex_HTRCXInvertOutput.nxc,
ex_HTRCXObvertOutput.nxc,      ex_HTRCXOff.nxc,                ex_
HTRCXOn.nxc,                  ex_HTRCXOnFor.nxc,              ex_HTRCXOnFwd.nxc,          ex_
HTRCXOnRev.nxc,               ex_HTRCXRev.nxc,                ex_HTRCXSetDirection.nxc,
ex_HTRCXSetGlobalDirection.nxc, ex_HTRCXSetGlobalOutput.nxc,
ex_HTRCXSetMaxPower.nxc,       ex_HTRCXSetOutput.nxc,          ex_
HTRCXSetPower.nxc,            ex_HTRCXToggle.nxc,             ex_MSRCXDisableOutput.nxc,
ex_MSRCXEnableOutput.nxc,      ex_MSRCXFloat.nxc,              ex_MSRCXFwd.nxc,          ex_
MSRCXInvertOutput.nxc,        ex_MSRCXObvertOutput.nxc,       ex_MSRCXOff.nxc,

```

[ex\\_MSRCXOn.nxc](#), [ex\\_MSRCXOnFor.nxc](#), [ex\\_MSRCXOnFwd.nxc](#), [ex\\_MSRCXOnRev.nxc](#), [ex\\_MSRCXRev.nxc](#), [ex\\_MSRCXSetDirection.nxc](#), [ex\\_MSRCXSetGlobalDirection.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), [ex\\_MSRCXSetMaxPower.nxc](#), [ex\\_MSRCXSetOutput.nxc](#), [ex\\_MSRCXSetPower.nxc](#), and [ex\\_MSRCXToggle.nxc](#).

#### 6.180.2.2 `#define RCX_OUT_AB 0x03`

RCX Outputs A and B

#### 6.180.2.3 `#define RCX_OUT_ABC 0x07`

RCX Outputs A, B, and C

#### 6.180.2.4 `#define RCX_OUT_AC 0x05`

RCX Outputs A and C

#### 6.180.2.5 `#define RCX_OUT_B 0x02`

RCX Output B

#### 6.180.2.6 `#define RCX_OUT_BC 0x06`

RCX Outputs B and C

#### 6.180.2.7 `#define RCX_OUT_C 0x04`

RCX Output C

## 6.181 RCX output mode constants

Constants for use when configuring RCX output mode.

### Defines

- `#define RCX_OUT_FLOAT 0`
- `#define RCX_OUT_OFF 0x40`
- `#define RCX_OUT_ON 0x80`

### 6.181.1 Detailed Description

Constants for use when configuring RCX output mode.

### 6.181.2 Define Documentation

#### 6.181.2.1 #define RCX\_OUT\_FLOAT 0

Set RCX output to float

#### 6.181.2.2 #define RCX\_OUT\_OFF 0x40

Set RCX output to off

#### 6.181.2.3 #define RCX\_OUT\_ON 0x80

Set RCX output to on

### Examples:

[ex\\_HTRCXSetGlobalOutput.nxc](#), [ex\\_HTRCXSetOutput.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), and [ex\\_MSRCXSetOutput.nxc](#).

## 6.182 RCX output direction constants

Constants for use when configuring RCX output direction.

### Defines

- #define [RCX\\_OUT\\_REV](#) 0
- #define [RCX\\_OUT\\_TOGGLE](#) 0x40
- #define [RCX\\_OUT\\_FWD](#) 0x80

### 6.182.1 Detailed Description

Constants for use when configuring RCX output direction.

### 6.182.2 Define Documentation

#### 6.182.2.1 #define RCX\_OUT\_FWD 0x80

Set RCX output direction to forward

**Examples:**

[ex\\_HTRCXSetDirection.nxc](#), [ex\\_HTRCXSetGlobalDirection.nxc](#), [ex\\_MSRCXSetDirection.nxc](#), and [ex\\_MSRCXSetGlobalDirection.nxc](#).

**6.182.2.2 #define RCX\_OUT\_REV 0**

Set RCX output direction to reverse

**6.182.2.3 #define RCX\_OUT\_TOGGLE 0x40**

Set RCX output direction to toggle

**6.183 RCX output power constants**

Constants for use when configuring RCX output power.

**Defines**

- #define [RCX\\_OUT\\_LOW](#) 0
- #define [RCX\\_OUT\\_HALF](#) 3
- #define [RCX\\_OUT\\_FULL](#) 7

**6.183.1 Detailed Description**

Constants for use when configuring RCX output power.

**6.183.2 Define Documentation****6.183.2.1 #define RCX\_OUT\_FULL 7**

Set RCX output power level to full

**Examples:**

[ex\\_HTRCXSetPower.nxc](#), and [ex\\_MSRCXSetPower.nxc](#).

**6.183.2.2 #define RCX\_OUT\_HALF 3**

Set RCX output power level to half

### 6.183.2.3 #define RCX\_OUT\_LOW 0

Set RCX output power level to low

## 6.184 RCX IR remote constants

Constants for use when simulating RCX IR remote messages.

### Defines

- #define [RCX\\_RemoteKeysReleased](#) 0x0000
- #define [RCX\\_RemotePBMessage1](#) 0x0100
- #define [RCX\\_RemotePBMessage2](#) 0x0200
- #define [RCX\\_RemotePBMessage3](#) 0x0400
- #define [RCX\\_RemoteOutAForward](#) 0x0800
- #define [RCX\\_RemoteOutBForward](#) 0x1000
- #define [RCX\\_RemoteOutCForward](#) 0x2000
- #define [RCX\\_RemoteOutABackward](#) 0x4000
- #define [RCX\\_RemoteOutBBackward](#) 0x8000
- #define [RCX\\_RemoteOutCBackward](#) 0x0001
- #define [RCX\\_RemoteSelProgram1](#) 0x0002
- #define [RCX\\_RemoteSelProgram2](#) 0x0004
- #define [RCX\\_RemoteSelProgram3](#) 0x0008
- #define [RCX\\_RemoteSelProgram4](#) 0x0010
- #define [RCX\\_RemoteSelProgram5](#) 0x0020
- #define [RCX\\_RemoteStopOutOff](#) 0x0040
- #define [RCX\\_RemotePlayASound](#) 0x0080

### 6.184.1 Detailed Description

Constants for use when simulating RCX IR remote messages.

### 6.184.2 Define Documentation

#### 6.184.2.1 #define RCX\_RemoteKeysReleased 0x0000

All remote keys have been released

#### 6.184.2.2 #define RCX\_RemoteOutABackward 0x4000

Set output A backward

**6.184.2.3 #define RCX\_RemoteOutAForward 0x0800**

Set output A forward

**6.184.2.4 #define RCX\_RemoteOutBBackward 0x8000**

Set output B backward

**6.184.2.5 #define RCX\_RemoteOutBForward 0x1000**

Set output B forward

**6.184.2.6 #define RCX\_RemoteOutCBackward 0x0001**

Set output C backward

**6.184.2.7 #define RCX\_RemoteOutCForward 0x2000**

Set output C forward

**6.184.2.8 #define RCX\_RemotePBMessage1 0x0100**

Send PB message 1

**6.184.2.9 #define RCX\_RemotePBMessage2 0x0200**

Send PB message 2

**6.184.2.10 #define RCX\_RemotePBMessage3 0x0400**

Send PB message 3

**6.184.2.11 #define RCX\_RemotePlayASound 0x0080**

Play a sound

**Examples:**

[ex\\_HTRCXRemote.nxc](#), and [ex\\_MSRCXRemote.nxc](#).

- 6.184.2.12** `#define RCX_RemoteSelProgram1 0x0002`  
Select program 1
- 6.184.2.13** `#define RCX_RemoteSelProgram2 0x0004`  
Select program 2
- 6.184.2.14** `#define RCX_RemoteSelProgram3 0x0008`  
Select program 3
- 6.184.2.15** `#define RCX_RemoteSelProgram4 0x0010`  
Select program 4
- 6.184.2.16** `#define RCX_RemoteSelProgram5 0x0020`  
Select program 5
- 6.184.2.17** `#define RCX_RemoteStopOutOff 0x0040`  
Stop and turn off outputs

## 6.185 RCX and Scout sound constants

Constants for use when playing standard RCX and Scout sounds.

### Defines

- `#define SOUND_CLICK 0`
- `#define SOUND_DOUBLE_BEEP 1`
- `#define SOUND_DOWN 2`
- `#define SOUND_UP 3`
- `#define SOUND_LOW_BEEP 4`
- `#define SOUND_FAST_UP 5`

### 6.185.1 Detailed Description

Constants for use when playing standard RCX and Scout sounds.

**6.185.2 Define Documentation****6.185.2.1 #define SOUND\_CLICK 0**

Play the standard key click sound

**6.185.2.2 #define SOUND\_DOUBLE\_BEEP 1**

Play the standard double beep sound

**6.185.2.3 #define SOUND\_DOWN 2**

Play the standard sweep down sound

**Examples:**

[ex\\_playsound.nxc](#).

**6.185.2.4 #define SOUND\_FAST\_UP 5**

Play the standard fast up sound

**Examples:**

[ex\\_playsound.nxc](#).

**6.185.2.5 #define SOUND\_LOW\_BEEP 4**

Play the standard low beep sound

**Examples:**

[ex\\_playsound.nxc](#).

**6.185.2.6 #define SOUND\_UP 3**

Play the standard sweep up sound

**Examples:**

[ex\\_playsound.nxc](#).

## 6.186 Scout constants

Constants for use when controlling the Scout brick.

### Modules

- [Scout light constants](#)  
*Constants for use when controlling the Scout light settings.*
- [Scout sound constants](#)  
*Constants for use when playing standard Scout sounds.*
- [Scout sound set constants](#)  
*Constants for use when choosing standard Scout sound sets.*
- [Scout mode constants](#)  
*Constants for use when setting the scout mode.*
- [Scout motion rule constants](#)  
*Constants for use when setting the scout motion rule.*
- [Scout touch rule constants](#)  
*Constants for use when setting the scout touch rule.*
- [Scout light rule constants](#)  
*Constants for use when setting the scout light rule.*
- [Scout transmit rule constants](#)  
*Constants for use when setting the scout transmit rule.*
- [Scout special effect constants](#)  
*Constants for use when setting the scout special effect.*

### 6.186.1 Detailed Description

Constants for use when controlling the Scout brick.

## 6.187 Scout light constants

Constants for use when controlling the Scout light settings.

**Defines**

- #define [SCOUT\\_LIGHT\\_ON](#) 0x80
- #define [SCOUT\\_LIGHT\\_OFF](#) 0

**6.187.1 Detailed Description**

Constants for use when controlling the Scout light settings.

**6.187.2 Define Documentation****6.187.2.1 #define SCOUT\_LIGHT\_OFF 0**

Turn off the scout light

**6.187.2.2 #define SCOUT\_LIGHT\_ON 0x80**

Turn on the scout light

**Examples:**

[ex\\_HTScoutSetLight.nxc](#).

**6.188 Scout sound constants**

Constants for use when playing standard Scout sounds.

**Defines**

- #define [SCOUT\\_SOUND\\_REMOTE](#) 6
- #define [SCOUT\\_SOUND\\_ENTERSA](#) 7
- #define [SCOUT\\_SOUND\\_KEYERROR](#) 8
- #define [SCOUT\\_SOUND\\_NONE](#) 9
- #define [SCOUT\\_SOUND\\_TOUCH1\\_PRES](#) 10
- #define [SCOUT\\_SOUND\\_TOUCH1\\_REL](#) 11
- #define [SCOUT\\_SOUND\\_TOUCH2\\_PRES](#) 12
- #define [SCOUT\\_SOUND\\_TOUCH2\\_REL](#) 13
- #define [SCOUT\\_SOUND\\_ENTER\\_BRIGHT](#) 14
- #define [SCOUT\\_SOUND\\_ENTER\\_NORMAL](#) 15
- #define [SCOUT\\_SOUND\\_ENTER\\_DARK](#) 16
- #define [SCOUT\\_SOUND\\_1\\_BLINK](#) 17

- #define `SCOUT_SOUND_2_BLINK` 18
- #define `SCOUT_SOUND_COUNTER1` 19
- #define `SCOUT_SOUND_COUNTER2` 20
- #define `SCOUT_SOUND_TIMER1` 21
- #define `SCOUT_SOUND_TIMER2` 22
- #define `SCOUT_SOUND_TIMER3` 23
- #define `SCOUT_SOUND_MAIL_RECEIVED` 24
- #define `SCOUT_SOUND_SPECIAL1` 25
- #define `SCOUT_SOUND_SPECIAL2` 26
- #define `SCOUT_SOUND_SPECIAL3` 27

### 6.188.1 Detailed Description

Constants for use when playing standard Scout sounds.

### 6.188.2 Define Documentation

#### 6.188.2.1 #define `SCOUT_SOUND_1_BLINK` 17

Play the Scout 1 blink sound

#### 6.188.2.2 #define `SCOUT_SOUND_2_BLINK` 18

Play the Scout 2 blink sound

#### 6.188.2.3 #define `SCOUT_SOUND_COUNTER1` 19

Play the Scout counter 1 sound

#### 6.188.2.4 #define `SCOUT_SOUND_COUNTER2` 20

Play the Scout counter 2 sound

#### 6.188.2.5 #define `SCOUT_SOUND_ENTER_BRIGHT` 14

Play the Scout enter bright sound

#### 6.188.2.6 #define `SCOUT_SOUND_ENTER_DARK` 16

Play the Scout enter dark sound

**6.188.2.7 #define SCOUT\_SOUND\_ENTER\_NORMAL 15**

Play the Scout enter normal sound

**6.188.2.8 #define SCOUT\_SOUND\_ENTERSA 7**

Play the Scout enter standalone sound

**6.188.2.9 #define SCOUT\_SOUND\_KEYERROR 8**

Play the Scout key error sound

**6.188.2.10 #define SCOUT\_SOUND\_MAIL\_RECEIVED 24**

Play the Scout mail received sound

**6.188.2.11 #define SCOUT\_SOUND\_NONE 9**

Play the Scout none sound

**6.188.2.12 #define SCOUT\_SOUND\_REMOTE 6**

Play the Scout remote sound

**6.188.2.13 #define SCOUT\_SOUND\_SPECIAL1 25**

Play the Scout special 1 sound

**6.188.2.14 #define SCOUT\_SOUND\_SPECIAL2 26**

Play the Scout special 2 sound

**6.188.2.15 #define SCOUT\_SOUND\_SPECIAL3 27**

Play the Scout special 3 sound

**6.188.2.16 #define SCOUT\_SOUND\_TIMER1 21**

Play the Scout timer 1 sound

**6.188.2.17 #define SCOUT\_SOUND\_TIMER2 22**

Play the Scout timer 2 sound

**6.188.2.18 #define SCOUT\_SOUND\_TIMER3 23**

Play the Scout timer 3 sound

**6.188.2.19 #define SCOUT\_SOUND\_TOUCH1\_PRES 10**

Play the Scout touch 1 pressed sound

**6.188.2.20 #define SCOUT\_SOUND\_TOUCH1\_REL 11**

Play the Scout touch 1 released sound

**6.188.2.21 #define SCOUT\_SOUND\_TOUCH2\_PRES 12**

Play the Scout touch 2 pressed sound

**6.188.2.22 #define SCOUT\_SOUND\_TOUCH2\_REL 13**

Play the Scout touch 2 released sound

**6.189 Scout sound set constants**

Constants for use when choosing standard Scout sound sets.

**Defines**

- #define [SCOUT\\_SNDSET\\_NONE](#) 0
- #define [SCOUT\\_SNDSET\\_BASIC](#) 1
- #define [SCOUT\\_SNDSET\\_BUG](#) 2
- #define [SCOUT\\_SNDSET\\_ALARM](#) 3
- #define [SCOUT\\_SNDSET\\_RANDOM](#) 4
- #define [SCOUT\\_SNDSET\\_SCIENCE](#) 5

**6.189.1 Detailed Description**

Constants for use when choosing standard Scout sound sets.

## 6.189.2 Define Documentation

### 6.189.2.1 #define SCOUT\_SNDSET\_ALARM 3

Set sound set to alarm

### 6.189.2.2 #define SCOUT\_SNDSET\_BASIC 1

Set sound set to basic

### 6.189.2.3 #define SCOUT\_SNDSET\_BUG 2

Set sound set to bug

### 6.189.2.4 #define SCOUT\_SNDSET\_NONE 0

Set sound set to none

### 6.189.2.5 #define SCOUT\_SNDSET\_RANDOM 4

Set sound set to random

### 6.189.2.6 #define SCOUT\_SNDSET\_SCIENCE 5

Set sound set to science

## 6.190 Scout mode constants

Constants for use when setting the scout mode.

### Defines

- #define [SCOUT\\_MODE\\_STANDALONE](#) 0
- #define [SCOUT\\_MODE\\_POWER](#) 1

### 6.190.1 Detailed Description

Constants for use when setting the scout mode.

## 6.190.2 Define Documentation

### 6.190.2.1 #define SCOUT\_MODE\_POWER 1

Enter power mode

#### Examples:

[ex\\_HTScoutSetScoutMode.nxc](#), and [ex\\_MSScoutSetScoutMode.nxc](#).

### 6.190.2.2 #define SCOUT\_MODE\_STANDALONE 0

Enter stand alone mode

## 6.191 Scout motion rule constants

Constants for use when setting the scout motion rule.

### Defines

- #define [SCOUT\\_MR\\_NO\\_MOTION](#) 0
- #define [SCOUT\\_MR\\_FORWARD](#) 1
- #define [SCOUT\\_MR\\_ZIGZAG](#) 2
- #define [SCOUT\\_MR\\_CIRCLE\\_RIGHT](#) 3
- #define [SCOUT\\_MR\\_CIRCLE\\_LEFT](#) 4
- #define [SCOUT\\_MR\\_LOOP\\_A](#) 5
- #define [SCOUT\\_MR\\_LOOP\\_B](#) 6
- #define [SCOUT\\_MR\\_LOOP\\_AB](#) 7

### 6.191.1 Detailed Description

Constants for use when setting the scout motion rule.

## 6.191.2 Define Documentation

### 6.191.2.1 #define SCOUT\_MR\_CIRCLE\_LEFT 4

Motion rule circle left

### 6.191.2.2 #define SCOUT\_MR\_CIRCLE\_RIGHT 3

Motion rule circle right

**6.191.2.3 #define SCOUT\_MR\_FORWARD 1**

Motion rule forward

**Examples:**[ex\\_MSScoutSetScoutRules.nxc](#).**6.191.2.4 #define SCOUT\_MR\_LOOP\_A 5**

Motion rule loop A

**6.191.2.5 #define SCOUT\_MR\_LOOP\_AB 7**

Motion rule loop A then B

**6.191.2.6 #define SCOUT\_MR\_LOOP\_B 6**

Motion rule loop B

**6.191.2.7 #define SCOUT\_MR\_NO\_MOTION 0**

Motion rule none

**6.191.2.8 #define SCOUT\_MR\_ZIGZAG 2**

Motion rule zigzag

**6.192 Scout touch rule constants**

Constants for use when setting the scout touch rule.

**Defines**

- #define [SCOUT\\_TR\\_IGNORE](#) 0
- #define [SCOUT\\_TR\\_REVERSE](#) 1
- #define [SCOUT\\_TR\\_AVOID](#) 2
- #define [SCOUT\\_TR\\_WAIT\\_FOR](#) 3
- #define [SCOUT\\_TR\\_OFF\\_WHEN](#) 4

### 6.192.1 Detailed Description

Constants for use when setting the scout touch rule.

### 6.192.2 Define Documentation

#### 6.192.2.1 #define SCOUT\_TR\_AVOID 2

Touch rule avoid

#### 6.192.2.2 #define SCOUT\_TR\_IGNORE 0

Touch rule ignore

#### 6.192.2.3 #define SCOUT\_TR\_OFF\_WHEN 4

Touch rule off when

#### 6.192.2.4 #define SCOUT\_TR\_REVERSE 1

Touch rule reverse

### Examples:

[ex\\_MSScoutSetScoutRules.nxc](#).

#### 6.192.2.5 #define SCOUT\_TR\_WAIT\_FOR 3

Touch rule wait for

## 6.193 Scout light rule constants

Constants for use when setting the scout light rule.

### Defines

- #define [SCOUT\\_LR\\_IGNORE](#) 0
- #define [SCOUT\\_LR\\_SEEK\\_LIGHT](#) 1
- #define [SCOUT\\_LR\\_SEEK\\_DARK](#) 2
- #define [SCOUT\\_LR\\_AVOID](#) 3

- #define `SCOUT_LR_WAIT_FOR` 4
- #define `SCOUT_LR_OFF_WHEN` 5

### 6.193.1 Detailed Description

Constants for use when setting the scout light rule.

### 6.193.2 Define Documentation

#### 6.193.2.1 #define `SCOUT_LR_AVOID` 3

Light rule avoid

#### 6.193.2.2 #define `SCOUT_LR_IGNORE` 0

Light rule ignore

#### Examples:

[ex\\_MSScoutSetScoutRules.nxc](#).

#### 6.193.2.3 #define `SCOUT_LR_OFF_WHEN` 5

Light rule off when

#### 6.193.2.4 #define `SCOUT_LR_SEEK_DARK` 2

Light rule seek dark

#### 6.193.2.5 #define `SCOUT_LR_SEEK_LIGHT` 1

Light rule seek light

#### 6.193.2.6 #define `SCOUT_LR_WAIT_FOR` 4

Light rule wait for

## 6.194 Scout transmit rule constants

Constants for use when setting the scout transmit rule.

**Defines**

- #define [SCOUT\\_TGS\\_SHORT](#) 0
- #define [SCOUT\\_TGS\\_MEDIUM](#) 1
- #define [SCOUT\\_TGS\\_LONG](#) 2

**6.194.1 Detailed Description**

Constants for use when setting the scout transmit rule.

**6.194.2 Define Documentation****6.194.2.1 #define SCOUT\_TGS\_LONG 2**

Transmit level long

**6.194.2.2 #define SCOUT\_TGS\_MEDIUM 1**

Transmit level medium

**6.194.2.3 #define SCOUT\_TGS\_SHORT 0**

Transmit level short

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**6.195 Scout special effect constants**

Constants for use when setting the scout special effect.

**Defines**

- #define [SCOUT\\_FXR\\_NONE](#) 0
- #define [SCOUT\\_FXR\\_BUG](#) 1
- #define [SCOUT\\_FXR\\_ALARM](#) 2
- #define [SCOUT\\_FXR\\_RANDOM](#) 3
- #define [SCOUT\\_FXR\\_SCIENCE](#) 4

### 6.195.1 Detailed Description

Constants for use when setting the scout special effect.

### 6.195.2 Define Documentation

#### 6.195.2.1 #define SCOUT\_FXR\_ALARM 2

Alarm special effects

#### 6.195.2.2 #define SCOUT\_FXR\_BUG 1

Bug special effects

#### Examples:

[ex\\_MSScoutSetScoutRules.nxc](#).

#### 6.195.2.3 #define SCOUT\_FXR\_NONE 0

No special effects

#### 6.195.2.4 #define SCOUT\_FXR\_RANDOM 3

Random special effects

#### 6.195.2.5 #define SCOUT\_FXR\_SCIENCE 4

Science special effects

## 6.196 RCX and Scout source constants

Constants for use when specifying RCX and Scout sources.

### Defines

- #define [RCX\\_VariableSrc](#) 0
- #define [RCX\\_TimerSrc](#) 1
- #define [RCX\\_ConstantSrc](#) 2
- #define [RCX\\_OutputStatusSrc](#) 3

- #define [RCX\\_RandomSrc](#) 4
- #define [RCX\\_ProgramSlotSrc](#) 8
- #define [RCX\\_InputValueSrc](#) 9
- #define [RCX\\_InputTypeSrc](#) 10
- #define [RCX\\_InputModeSrc](#) 11
- #define [RCX\\_InputRawSrc](#) 12
- #define [RCX\\_InputBooleanSrc](#) 13
- #define [RCX\\_WatchSrc](#) 14
- #define [RCX\\_MessageSrc](#) 15
- #define [RCX\\_GlobalMotorStatusSrc](#) 17
- #define [RCX\\_ScoutRulesSrc](#) 18
- #define [RCX\\_ScoutLightParamsSrc](#) 19
- #define [RCX\\_ScoutTimerLimitSrc](#) 20
- #define [RCX\\_CounterSrc](#) 21
- #define [RCX\\_ScoutCounterLimitSrc](#) 22
- #define [RCX\\_TaskEventsSrc](#) 23
- #define [RCX\\_ScoutEventFBSrc](#) 24
- #define [RCX\\_EventStateSrc](#) 25
- #define [RCX\\_TenMSTimerSrc](#) 26
- #define [RCX\\_ClickCounterSrc](#) 27
- #define [RCX\\_UpperThresholdSrc](#) 28
- #define [RCX\\_LowerThresholdSrc](#) 29
- #define [RCX\\_HysteresisSrc](#) 30
- #define [RCX\\_DurationSrc](#) 31
- #define [RCX\\_UARTSetupSrc](#) 33
- #define [RCX\\_BatteryLevelSrc](#) 34
- #define [RCX\\_FirmwareVersionSrc](#) 35
- #define [RCX\\_IndirectVarSrc](#) 36
- #define [RCX\\_DatalogSrcIndirectSrc](#) 37
- #define [RCX\\_DatalogSrcDirectSrc](#) 38
- #define [RCX\\_DatalogValueIndirectSrc](#) 39
- #define [RCX\\_DatalogValueDirectSrc](#) 40
- #define [RCX\\_DatalogRawIndirectSrc](#) 41
- #define [RCX\\_DatalogRawDirectSrc](#) 42

### 6.196.1 Detailed Description

Constants for use when specifying RCX and Scout sources.

## 6.196.2 Define Documentation

### 6.196.2.1 #define RCX\_BatteryLevelSrc 34

The RCX battery level source

### 6.196.2.2 #define RCX\_ClickCounterSrc 27

The RCX event click counter source

### 6.196.2.3 #define RCX\_ConstantSrc 2

The RCX constant value source

#### Examples:

[ex\\_HTRCXEvent.nxc](#), [ex\\_HTRCXSetEvent.nxc](#), [ex\\_HTRCXSetMaxPower.nxc](#),  
[ex\\_HTRCXSetPower.nxc](#), [ex\\_HTSoutSendVLL.nxc](#), [ex\\_-  
HTScoutSetEventFeedback.nxc](#), [ex\\_HTSoutSetSensorClickTime.nxc](#),  
[ex\\_HTSoutSetSensorHysteresis.nxc](#), [ex\\_MSRCXAndVar.nxc](#), [ex\\_-  
MSRCXDivVar.nxc](#), [ex\\_MSRCXEvent.nxc](#), [ex\\_MSRCXOrVar.nxc](#), [ex\\_-  
MSRCXSetEvent.nxc](#), [ex\\_MSRCXSetMaxPower.nxc](#), [ex\\_MSRCXSetPower.nxc](#),  
[ex\\_MSScoutSendVLL.nxc](#), [ex\\_MSScoutSetCounterLimit.nxc](#), [ex\\_-  
MSScoutSetEventFeedback.nxc](#), [ex\\_MSScoutSetSensorClickTime.nxc](#), [ex\\_-  
MSScoutSetSensorHysteresis.nxc](#), and [ex\\_MSScoutSetTimerLimit.nxc](#).

### 6.196.2.4 #define RCX\_CounterSrc 21

The RCX counter source

### 6.196.2.5 #define RCX\_DatalogRawDirectSrc 42

The RCX direct datalog raw source

### 6.196.2.6 #define RCX\_DatalogRawIndirectSrc 41

The RCX indirect datalog raw source

### 6.196.2.7 #define RCX\_DatalogSrcDirectSrc 38

The RCX direct datalog source source

**6.196.2.8 #define RCX\_DatalogSrcIndirectSrc 37**

The RCX indirect datalog source source

**6.196.2.9 #define RCX\_DatalogValueDirectSrc 40**

The RCX direct datalog value source

**6.196.2.10 #define RCX\_DatalogValueIndirectSrc 39**

The RCX indirect datalog value source

**6.196.2.11 #define RCX\_DurationSrc 31**

The RCX event duration source

**6.196.2.12 #define RCX\_EventStateSrc 25**

The RCX event static source

**6.196.2.13 #define RCX\_FirmwareVersionSrc 35**

The RCX firmware version source

**6.196.2.14 #define RCX\_GlobalMotorStatusSrc 17**

The RCX global motor status source

**6.196.2.15 #define RCX\_HysteresisSrc 30**

The RCX event hysteresis source

**6.196.2.16 #define RCX\_IndirectVarSrc 36**

The RCX indirect variable source

**6.196.2.17 #define RCX\_InputBooleanSrc 13**

The RCX input boolean source

**6.196.2.18 #define RCX\_InputModeSrc 11**

The RCX input mode source

**6.196.2.19 #define RCX\_InputRawSrc 12**

The RCX input raw source

**6.196.2.20 #define RCX\_InputTypeSrc 10**

The RCX input type source

**6.196.2.21 #define RCX\_InputValueSrc 9**

The RCX input value source

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#), [ex\\_MSRCXAddToDatalog.nxc](#), and [ex\\_MSRCXSumVar.nxc](#).

**6.196.2.22 #define RCX\_LowerThresholdSrc 29**

The RCX event lower threshold source

**6.196.2.23 #define RCX\_MessageSrc 15**

The RCX message source

**6.196.2.24 #define RCX\_OutputStatusSrc 3**

The RCX output status source

**6.196.2.25 #define RCX\_ProgramSlotSrc 8**

The RCX program slot source

**6.196.2.26 #define RCX\_RandomSrc 4**

The RCX random number source

**Examples:**[ex\\_MSRCXSet.nxc](#), and [ex\\_MSRCXSubVar.nxc](#).**6.196.2.27 #define RCX\_ScoutCounterLimitSrc 22**

The Scout counter limit source

**6.196.2.28 #define RCX\_ScoutEventFBSrc 24**

The Scout event feedback source

**6.196.2.29 #define RCX\_ScoutLightParamsSrc 19**

The Scout light parameters source

**6.196.2.30 #define RCX\_ScoutRulesSrc 18**

The Scout rules source

**6.196.2.31 #define RCX\_ScoutTimerLimitSrc 20**

The Scout timer limit source

**6.196.2.32 #define RCX\_TaskEventsSrc 23**

The RCX task events source

**6.196.2.33 #define RCX\_TenMSTimerSrc 26**

The RCX 10ms timer source

**6.196.2.34 #define RCX\_TimerSrc 1**

The RCX timer source

**6.196.2.35 #define RCX\_UARTSetupSrc 33**

The RCX UART setup source

**6.196.2.36 #define RCX\_UpperThresholdSrc 28**

The RCX event upper threshold source

**6.196.2.37 #define RCX\_VariableSrc 0**

The RCX variable source

**Examples:**

ex\_HTRCXPoll.nxc, ex\_HTRCXSelectDisplay.nxc, ex-HTScoutSetSensorLowerLimit.nxc, ex-HTScoutSetSensorUpperLimit.nxc, ex\_MSRCXAbsVar.nxc, ex\_MSRCXMulVar.nxc, ex\_MSRCXPoll.nxc, ex\_MSRCXSelectDisplay.nxc, ex\_MSRCXSet.nxc, ex\_MSRCXSetUserDisplay.nxc, ex\_MSRCXSetVar.nxc, MSRCXSgnVar.nxc, ex\_MSScoutSetSensorLowerLimit.nxc, and ex\_MSScoutSetSensorUpperLimit.nxc.

**6.196.2.38 #define RCX\_WatchSrc 14**

The RCX watch source

**6.197 RCX and Scout opcode constants**

Constants for use when specifying RCX and Scout opcodes.

**Defines**

- #define [RCX\\_PingOp](#) 0x10
- #define [RCX\\_BatteryLevelOp](#) 0x30
- #define [RCX\\_DeleteTasksOp](#) 0x40
- #define [RCX\\_StopAllTasksOp](#) 0x50
- #define [RCX\\_PBTurnOffOp](#) 0x60
- #define [RCX\\_DeleteSubsOp](#) 0x70
- #define [RCX\\_ClearSoundOp](#) 0x80
- #define [RCX\\_ClearMsgOp](#) 0x90
- #define [RCX\\_LSCalibrateOp](#) 0xc0

- #define `RCX_MuteSoundOp` 0xd0
- #define `RCX_UnmuteSoundOp` 0xe0
- #define `RCX_ClearAllEventsOp` 0x06
- #define `RCX_OnOffFloatOp` 0x21
- #define `RCX_IRModeOp` 0x31
- #define `RCX_PlaySoundOp` 0x51
- #define `RCX_DeleteTaskOp` 0x61
- #define `RCX_StartTaskOp` 0x71
- #define `RCX_StopTaskOp` 0x81
- #define `RCX_SelectProgramOp` 0x91
- #define `RCX_ClearTimerOp` 0xa1
- #define `RCX_AutoOffOp` 0xb1
- #define `RCX_DeleteSubOp` 0xc1
- #define `RCX_ClearSensorOp` 0xd1
- #define `RCX_OutputDirOp` 0xe1
- #define `RCX_PlayToneVarOp` 0x02
- #define `RCX_PollOp` 0x12
- #define `RCX_SetWatchOp` 0x22
- #define `RCX_InputTypeOp` 0x32
- #define `RCX_InputModeOp` 0x42
- #define `RCX_SetDatalogOp` 0x52
- #define `RCX_DatalogOp` 0x62
- #define `RCX_SendUARTDataOp` 0xc2
- #define `RCX_RemoteOp` 0xd2
- #define `RCX_VLLOp` 0xe2
- #define `RCX_DirectEventOp` 0x03
- #define `RCX_OutputPowerOp` 0x13
- #define `RCX_PlayToneOp` 0x23
- #define `RCX_DisplayOp` 0x33
- #define `RCX_PollMemoryOp` 0x63
- #define `RCX_SetFeedbackOp` 0x83
- #define `RCX_SetEventOp` 0x93
- #define `RCX_GOutputPowerOp` 0xa3
- #define `RCX_LSUpperThreshOp` 0xb3
- #define `RCX_LSLowerThreshOp` 0xc3
- #define `RCX_LSHysteresisOp` 0xd3
- #define `RCX_LSBlinkTimeOp` 0xe3
- #define `RCX_CalibrateEventOp` 0x04
- #define `RCX_SetVarOp` 0x14
- #define `RCX_SumVarOp` 0x24
- #define `RCX_SubVarOp` 0x34
- #define `RCX_DivVarOp` 0x44
- #define `RCX_MulVarOp` 0x54

- #define `RCX_SgnVarOp` 0x64
- #define `RCX_AbsVarOp` 0x74
- #define `RCX_AndVarOp` 0x84
- #define `RCX_OrVarOp` 0x94
- #define `RCX_UploadDatalogOp` 0xa4
- #define `RCX_SetTimerLimitOp` 0xc4
- #define `RCX_SetCounterOp` 0xd4
- #define `RCX_SetSourceValueOp` 0x05
- #define `RCX_UnlockOp` 0x15
- #define `RCX_BootModeOp` 0x65
- #define `RCX_UnlockFirmOp` 0xa5
- #define `RCX_ScoutRulesOp` 0xd5
- #define `RCX_ViewSourceValOp` 0xe5
- #define `RCX_ScoutOp` 0x47
- #define `RCX_SoundOp` 0x57
- #define `RCX_GOutputModeOp` 0x67
- #define `RCX_GOutputDirOp` 0x77
- #define `RCX_LightOp` 0x87
- #define `RCX_IncCounterOp` 0x97
- #define `RCX_DecCounterOp` 0xa7
- #define `RCX_ClearCounterOp` 0xb7
- #define `RCX_SetPriorityOp` 0xd7
- #define `RCX_MessageOp` 0xf7

### 6.197.1 Detailed Description

Constants for use when specifying RCX and Scout opcodes.

### 6.197.2 Define Documentation

#### 6.197.2.1 #define `RCX_AbsVarOp` 0x74

Absolute value function

#### 6.197.2.2 #define `RCX_AndVarOp` 0x84

AND function

#### 6.197.2.3 #define `RCX_AutoOffOp` 0xb1

Set auto off timer

**6.197.2.4 #define RCX\_BatteryLevelOp 0x30**

Read the battery level

**6.197.2.5 #define RCX\_BootModeOp 0x65**

Set into book mode

**6.197.2.6 #define RCX\_CalibrateEventOp 0x04**

Calibrate event

**6.197.2.7 #define RCX\_ClearAllEventsOp 0x06**

Clear all events

**6.197.2.8 #define RCX\_ClearCounterOp 0xb7**

Clear a counter

**6.197.2.9 #define RCX\_ClearMsgOp 0x90**

Clear message

**6.197.2.10 #define RCX\_ClearSensorOp 0xd1**

Clear a sensor

**6.197.2.11 #define RCX\_ClearSoundOp 0x80**

Clear sound

**6.197.2.12 #define RCX\_ClearTimerOp 0xa1**

Clear a timer

**6.197.2.13 #define RCX\_DatalogOp 0x62**

Datalog the specified source/value

---

|                   |                                       |                            |
|-------------------|---------------------------------------|----------------------------|
| <b>6.197.2.14</b> | <b>#define RCX_DecCounterOp 0xa7</b>  | Decrement a counter        |
| <b>6.197.2.15</b> | <b>#define RCX_DeleteSubOp 0xc1</b>   | Delete a subroutine        |
| <b>6.197.2.16</b> | <b>#define RCX_DeleteSubsOp 0x70</b>  | Delete subroutines         |
| <b>6.197.2.17</b> | <b>#define RCX_DeleteTaskOp 0x61</b>  | Delete a task              |
| <b>6.197.2.18</b> | <b>#define RCX_DeleteTasksOp 0x40</b> | Delete tasks               |
| <b>6.197.2.19</b> | <b>#define RCX_DirectEventOp 0x03</b> | Fire an event              |
| <b>6.197.2.20</b> | <b>#define RCX_DisplayOp 0x33</b>     | Set LCD display value      |
| <b>6.197.2.21</b> | <b>#define RCX_DivVarOp 0x44</b>      | Divide function            |
| <b>6.197.2.22</b> | <b>#define RCX_GOutputDirOp 0x77</b>  | Set global motor direction |
| <b>6.197.2.23</b> | <b>#define RCX_GOutputModeOp 0x67</b> | Set global motor mode      |

**6.197.2.24 #define RCX\_GOutputPowerOp 0xa3**

Set global motor power levels

**6.197.2.25 #define RCX\_IncCounterOp 0x97**

Increment a counter

**6.197.2.26 #define RCX\_InputModeOp 0x42**

Set the input mode

**6.197.2.27 #define RCX\_InputTypeOp 0x32**

Set the input type

**6.197.2.28 #define RCX\_IRModeOp 0x31**

Set the IR transmit mode

**6.197.2.29 #define RCX\_LightOp 0x87**

Light opcode

**6.197.2.30 #define RCX\_LSblinkTimeOp 0xe3**

Set the light sensor blink time

**6.197.2.31 #define RCX\_LSCalibrateOp 0xc0**

Calibrate the light sensor

**6.197.2.32 #define RCX\_LSHysteresisOp 0xd3**

Set the light sensor hysteresis

**6.197.2.33 #define RCX\_LSLowerThreshOp 0xc3**

Set the light sensor lower threshold

**6.197.2.34 #define RCX\_LSUpperThreshOp 0xb3**

Set the light sensor upper threshold

**6.197.2.35 #define RCX\_MessageOp 0xf7**

Set message

**6.197.2.36 #define RCX\_MulVarOp 0x54**

Multiply function

**6.197.2.37 #define RCX\_MuteSoundOp 0xd0**

Mute sound

**6.197.2.38 #define RCX\_OnOffFloatOp 0x21**

Control motor state - on, off, float

**6.197.2.39 #define RCX\_OrVarOp 0x94**

OR function

**6.197.2.40 #define RCX\_OutputDirOp 0xe1**

Set the motor direction

**6.197.2.41 #define RCX\_OutputPowerOp 0x13**

Set the motor power level

**6.197.2.42 #define RCX\_PBTurnOffOp 0x60**

Turn off the brick

**6.197.2.43 #define RCX\_PingOp 0x10**

Ping the brick

**6.197.2.44 #define RCX\_PlaySoundOp 0x51**

Play a sound

**6.197.2.45 #define RCX\_PlayToneOp 0x23**

Play a tone

**6.197.2.46 #define RCX\_PlayToneVarOp 0x02**

Play a tone using a variable

**6.197.2.47 #define RCX\_PollMemoryOp 0x63**

Poll a memory location

**6.197.2.48 #define RCX\_PollOp 0x12**

Poll a source/value combination

**6.197.2.49 #define RCX\_RemoteOp 0xd2**

Execute simulated remote control buttons

**6.197.2.50 #define RCX\_ScoutOp 0x47**

Scout opcode

**6.197.2.51 #define RCX\_ScoutRulesOp 0xd5**

Set Scout rules

**6.197.2.52 #define RCX\_SelectProgramOp 0x91**

Select a program slot

**6.197.2.53 #define RCX\_SendUARTDataOp 0xc2**

Send data via IR using UART settings

**6.197.2.54 #define RCX\_SetCounterOp 0xd4**

Set counter value

**6.197.2.55 #define RCX\_SetDatalogOp 0x52**

Set the datalog size

**6.197.2.56 #define RCX\_SetEventOp 0x93**

Set an event

**6.197.2.57 #define RCX\_SetFeedbackOp 0x83**

Set Scout feedback

**6.197.2.58 #define RCX\_SetPriorityOp 0xd7**

Set task priority

**6.197.2.59 #define RCX\_SetSourceValueOp 0x05**

Set a source/value

**6.197.2.60 #define RCX\_SetTimerLimitOp 0xc4**

Set timer limit

**6.197.2.61 #define RCX\_SetVarOp 0x14**

Set function

**6.197.2.62 #define RCX\_SetWatchOp 0x22**

Set the watch source/value

**6.197.2.63 #define RCX\_SgnVarOp 0x64**

Sign function

---

|                   |   |                         |
|-------------------|---|-------------------------|
| <b>6.197.2.64</b> | <b>#define RCX_SoundOp 0x57</b>         | Sound opcode            |
| <b>6.197.2.65</b> | <b>#define RCX_StartTaskOp 0x71</b>     | Start a task            |
| <b>6.197.2.66</b> | <b>#define RCX_StopAllTasksOp 0x50</b>  | Stop all tasks          |
| <b>6.197.2.67</b> | <b>#define RCX_StopTaskOp 0x81</b>      | Stop a task             |
| <b>6.197.2.68</b> | <b>#define RCX_SubVarOp 0x34</b>        | Subtract function       |
| <b>6.197.2.69</b> | <b>#define RCX_SumVarOp 0x24</b>        | Sum function            |
| <b>6.197.2.70</b> | <b>#define RCX_UnlockFirmOp 0xa5</b>    | Unlock the firmware     |
| <b>6.197.2.71</b> | <b>#define RCX_UnlockOp 0x15</b>        | Unlock the brick        |
| <b>6.197.2.72</b> | <b>#define RCX_UnmuteSoundOp 0xe0</b>   | Unmute sound            |
| <b>6.197.2.73</b> | <b>#define RCX_UploadDatalogOp 0xa4</b> | Upload datalog contents |

**6.197.2.74 #define RCX\_ViewSourceValOp 0xe5**

[View a source/value](#)

**6.197.2.75 #define RCX\_VLLOp 0xe2**

[Send visual light link \(VLL\) data](#)

**6.198 HiTechnic/mindsensors Power Function/IR Train constants**

Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.

**Modules**

- [Power Function command constants](#)  
*Constants that are for sending Power Function commands.*
- [Power Function channel constants](#)  
*Constants that are for specifying Power Function channels.*
- [Power Function mode constants](#)  
*Constants that are for choosing Power Function modes.*
- [PF/IR Train function constants](#)  
*Constants that are for sending PF/IR Train functions.*
- [IR Train channel constants](#)  
*Constants that are for specifying IR Train channels.*
- [Power Function output constants](#)  
*Constants that are for choosing a Power Function output.*
- [Power Function pin constants](#)  
*Constants that are for choosing a Power Function pin.*
- [Power Function single pin function constants](#)  
*Constants that are for sending Power Function single pin functions.*
- [Power Function CST options constants](#)  
*Constants that are for specifying Power Function CST options.*

- [Power Function PWM option constants](#)

*Constants that are for specifying Power Function PWM options.*

### 6.198.1 Detailed Description

Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.

## 6.199 Power Function command constants

Constants that are for sending Power Function commands.

### Defines

- #define [PF\\_CMD\\_STOP](#) 0
- #define [PF\\_CMD\\_FLOAT](#) 0
- #define [PF\\_CMD\\_FWD](#) 1
- #define [PF\\_CMD\\_REV](#) 2
- #define [PF\\_CMD\\_BRAKE](#) 3

### 6.199.1 Detailed Description

Constants that are for sending Power Function commands.

### 6.199.2 Define Documentation

#### 6.199.2.1 #define [PF\\_CMD\\_BRAKE](#) 3

Power function command brake

#### 6.199.2.2 #define [PF\\_CMD\\_FLOAT](#) 0

Power function command float (same as stop)

#### 6.199.2.3 #define [PF\\_CMD\\_FWD](#) 1

Power function command forward

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_PFMate.nxc](#).

**6.199.2.4 #define PF\_CMD\_REV 2**

Power function command reverse

**Examples:**

[ex\\_PFMate.nxc](#).

**6.199.2.5 #define PF\_CMD\_STOP 0**

Power function command stop

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), and [ex\\_MSPFComboDirect.nxc](#).

**6.200 Power Function channel constants**

Constants that are for specifying Power Function channels.

**Defines**

- #define [PF\\_CHANNEL\\_1](#) 0
- #define [PF\\_CHANNEL\\_2](#) 1
- #define [PF\\_CHANNEL\\_3](#) 2
- #define [PF\\_CHANNEL\\_4](#) 3

**6.200.1 Detailed Description**

Constants that are for specifying Power Function channels.

**6.200.2 Define Documentation****6.200.2.1 #define PF\_CHANNEL\_1 0**

Power function channel 1

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), [ex\\_HTPFComboPWM.nxc](#), [ex\\_HTPFSingleOutputCST.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#), [ex\\_HTPFSinglePin.nxc](#), [ex\\_HTPFTrain.nxc](#), [ex\\_MSPFComboDirect.nxc](#), [ex\\_MSPFComboPWM.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFTrain.nxc](#).

**6.200.2.2 #define PF\_CHANNEL\_2 1**

Power function channel 2

**6.200.2.3 #define PF\_CHANNEL\_3 2**

Power function channel 3

**6.200.2.4 #define PF\_CHANNEL\_4 3**

Power function channel 4

**6.201 Power Function mode constants**

Constants that are for choosing Power Function modes.

**Defines**

- [#define PF\\_MODE\\_TRAIN 0](#)
- [#define PF\\_MODE\\_COMBO\\_DIRECT 1](#)
- [#define PF\\_MODE\\_SINGLE\\_PIN\\_CONT 2](#)
- [#define PF\\_MODE\\_SINGLE\\_PIN\\_TIME 3](#)
- [#define PF\\_MODE\\_COMBO\\_PWM 4](#)
- [#define PF\\_MODE\\_SINGLE\\_OUTPUT\\_PWM 4](#)
- [#define PF\\_MODE\\_SINGLE\\_OUTPUT\\_CST 6](#)

**6.201.1 Detailed Description**

Constants that are for choosing Power Function modes.

### 6.201.2 Define Documentation

#### 6.201.2.1 #define PF\_MODE\_COMBO\_DIRECT 1

Power function mode combo direct

#### 6.201.2.2 #define PF\_MODE\_COMBO\_PWM 4

Power function mode combo pulse width modulation (PWM)

#### 6.201.2.3 #define PF\_MODE\_SINGLE\_OUTPUT\_CST 6

Power function mode single output clear, set, toggle (CST)

#### 6.201.2.4 #define PF\_MODE\_SINGLE\_OUTPUT\_PWM 4

Power function mode single output pulse width modulation (PWM)

#### 6.201.2.5 #define PF\_MODE\_SINGLE\_PIN\_CONT 2

Power function mode single pin continuous

#### 6.201.2.6 #define PF\_MODE\_SINGLE\_PIN\_TIME 3

Power function mode single pin timed

#### 6.201.2.7 #define PF\_MODE\_TRAIN 0

Power function mode IR Train

## 6.202 PF/IR Train function constants

Constants that are for sending PF/IR Train functions.

### Defines

- #define [TRAIN\\_FUNC\\_STOP](#) 0
- #define [TRAIN\\_FUNC\\_INCR\\_SPEED](#) 1
- #define [TRAIN\\_FUNC\\_DECR\\_SPEED](#) 2
- #define [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#) 4

### 6.202.1 Detailed Description

Constants that are for sending PF/IR Train functions.

### 6.202.2 Define Documentation

#### 6.202.2.1 #define TRAIN\_FUNC\_DECR\_SPEED 2

PF/IR Train function decrement speed

#### 6.202.2.2 #define TRAIN\_FUNC\_INCR\_SPEED 1

PF/IR Train function increment speed

#### Examples:

[ex\\_HTIRTrain.nxc](#), [ex\\_HTPFTrain.nxc](#), [ex\\_MSIRTrain.nxc](#), and [ex\\_MSPFTrain.nxc](#).

#### 6.202.2.3 #define TRAIN\_FUNC\_STOP 0

PF/IR Train function stop

#### 6.202.2.4 #define TRAIN\_FUNC\_TOGGLE\_LIGHT 4

PF/IR Train function toggle light

## 6.203 IR Train channel constants

Constants that are for specifying IR Train channels.

### Defines

- #define [TRAIN\\_CHANNEL\\_1](#) 0
- #define [TRAIN\\_CHANNEL\\_2](#) 1
- #define [TRAIN\\_CHANNEL\\_3](#) 2
- #define [TRAIN\\_CHANNEL\\_ALL](#) 3

### 6.203.1 Detailed Description

Constants that are for specifying IR Train channels.

### 6.203.2 Define Documentation

#### 6.203.2.1 #define TRAIN\_CHANNEL\_1 0

IR Train channel 1

#### Examples:

[ex\\_HTIRTrain.nxc](#), and [ex\\_MSIRTrain.nxc](#).

#### 6.203.2.2 #define TRAIN\_CHANNEL\_2 1

IR Train channel 2

#### 6.203.2.3 #define TRAIN\_CHANNEL\_3 2

IR Train channel 3

#### 6.203.2.4 #define TRAIN\_CHANNEL\_ALL 3

IR Train channel all

## 6.204 Power Function output constants

Constants that are for choosing a Power Function output.

### Defines

- #define [PF\\_OUT\\_A](#) 0
- #define [PF\\_OUT\\_B](#) 1

### 6.204.1 Detailed Description

Constants that are for choosing a Power Function output.

### 6.204.2 Define Documentation

#### 6.204.2.1 #define PF\_OUT\_A 0

Power function output A

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#),  
[ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_-  
MSPFSingleOutputPWM.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).

**6.204.2.2 #define PF\_OUT\_B 1**

Power function output B

**6.205 Power Function pin constants**

Constants that are for choosing a Power Function pin.

**Defines**

- `#define PF_PIN_C1 0`
- `#define PF_PIN_C2 1`

**6.205.1 Detailed Description**

Constants that are for choosing a Power Function pin.

**6.205.2 Define Documentation****6.205.2.1 #define PF\_PIN\_C1 0**

Power function pin C1

**Examples:**

[ex\\_HTPFSinglePin.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).

**6.205.2.2 #define PF\_PIN\_C2 1**

Power function pin C2

**6.206 Power Function single pin function constants**

Constants that are for sending Power Function single pin functions.

## Defines

- #define [PF\\_FUNC\\_NOCHANGE](#) 0
- #define [PF\\_FUNC\\_CLEAR](#) 1
- #define [PF\\_FUNC\\_SET](#) 2
- #define [PF\\_FUNC\\_TOGGLE](#) 3

### 6.206.1 Detailed Description

Constants that are for sending Power Function single pin functions.

### 6.206.2 Define Documentation

#### 6.206.2.1 #define [PF\\_FUNC\\_CLEAR](#) 1

Power function single pin - clear

#### 6.206.2.2 #define [PF\\_FUNC\\_NOCHANGE](#) 0

Power function single pin - no change

#### 6.206.2.3 #define [PF\\_FUNC\\_SET](#) 2

Power function single pin - set

### Examples:

[ex\\_HTPFSinglePin.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).

#### 6.206.2.4 #define [PF\\_FUNC\\_TOGGLE](#) 3

Power function single pin - toggle

## 6.207 Power Function CST options constants

Constants that are for specifying Power Function CST options.

## Defines

- #define [PF\\_CST\\_CLEAR1\\_CLEAR2](#) 0
- #define [PF\\_CST\\_SET1\\_CLEAR2](#) 1
- #define [PF\\_CST\\_CLEAR1\\_SET2](#) 2
- #define [PF\\_CST\\_SET1\\_SET2](#) 3
- #define [PF\\_CST\\_INCREMENT\\_PWM](#) 4
- #define [PF\\_CST\\_DECREMENT\\_PWM](#) 5
- #define [PF\\_CST\\_FULL\\_FWD](#) 6
- #define [PF\\_CST\\_FULL\\_REV](#) 7
- #define [PF\\_CST\\_TOGGLE\\_DIR](#) 8

### 6.207.1 Detailed Description

Constants that are for specifying Power Function CST options.

### 6.207.2 Define Documentation

#### 6.207.2.1 #define [PF\\_CST\\_CLEAR1\\_CLEAR2](#) 0

Power function CST clear 1 and clear 2

#### 6.207.2.2 #define [PF\\_CST\\_CLEAR1\\_SET2](#) 2

Power function CST clear 1 and set 2

#### 6.207.2.3 #define [PF\\_CST\\_DECREMENT\\_PWM](#) 5

Power function CST decrement PWM

#### 6.207.2.4 #define [PF\\_CST\\_FULL\\_FWD](#) 6

Power function CST full forward

#### 6.207.2.5 #define [PF\\_CST\\_FULL\\_REV](#) 7

Power function CST full reverse

#### 6.207.2.6 #define [PF\\_CST\\_INCREMENT\\_PWM](#) 4

Power function CST increment PWM

**6.207.2.7 #define PF\_CST\_SET1\_CLEAR2 1**

Power function CST set 1 and clear 2

**6.207.2.8 #define PF\_CST\_SET1\_SET2 3**

Power function CST set 1 and set 2

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), and [ex\\_MSPFSingleOutputCST.nxc](#).

**6.207.2.9 #define PF\_CST\_TOGGLE\_DIR 8**

Power function CST toggle direction

**6.208 Power Function PWM option constants**

Constants that are for specifying Power Function PWM options.

**Defines**

- #define [PF\\_PWM\\_FLOAT](#) 0
- #define [PF\\_PWM\\_FWD1](#) 1
- #define [PF\\_PWM\\_FWD2](#) 2
- #define [PF\\_PWM\\_FWD3](#) 3
- #define [PF\\_PWM\\_FWD4](#) 4
- #define [PF\\_PWM\\_FWD5](#) 5
- #define [PF\\_PWM\\_FWD6](#) 6
- #define [PF\\_PWM\\_FWD7](#) 7
- #define [PF\\_PWM\\_BRAKE](#) 8
- #define [PF\\_PWM\\_REV7](#) 9
- #define [PF\\_PWM\\_REV6](#) 10
- #define [PF\\_PWM\\_REV5](#) 11
- #define [PF\\_PWM\\_REV4](#) 12
- #define [PF\\_PWM\\_REV3](#) 13
- #define [PF\\_PWM\\_REV2](#) 14
- #define [PF\\_PWM\\_REV1](#) 15

**6.208.1 Detailed Description**

Constants that are for specifying Power Function PWM options.

**6.208.2 Define Documentation****6.208.2.1 #define PF\_PWM\_BRAKE 8**

Power function PWM brake

**6.208.2.2 #define PF\_PWM\_FLOAT 0**

Power function PWM float

**6.208.2.3 #define PF\_PWM\_FWD1 1**

Power function PWM forward level 1

**6.208.2.4 #define PF\_PWM\_FWD2 2**

Power function PWM forward level 2

**6.208.2.5 #define PF\_PWM\_FWD3 3**

Power function PWM forward level 3

**6.208.2.6 #define PF\_PWM\_FWD4 4**

Power function PWM forward level 4

**6.208.2.7 #define PF\_PWM\_FWD5 5**

Power function PWM forward level 5

**Examples:**

[ex\\_HTPFComboPWM.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#), and [ex\\_MSPFSingleOutputPWM.nxc](#).

**6.208.2.8 #define PF\_PWM\_FWD6 6**

Power function PWM forward level 6

**6.208.2.9 #define PF\_PWM\_FWD7 7**

Power function PWM forward level 7

**6.208.2.10 #define PF\_PWM\_REV1 15**

Power function PWM reverse level 1

**6.208.2.11 #define PF\_PWM\_REV2 14**

Power function PWM reverse level 2

**6.208.2.12 #define PF\_PWM\_REV3 13**

Power function PWM reverse level 3

**6.208.2.13 #define PF\_PWM\_REV4 12**

Power function PWM reverse level 4

**Examples:**[ex\\_HTTPComboPWM.nxc](#), and [ex\\_MSPFComboPWM.nxc](#).**6.208.2.14 #define PF\_PWM\_REV5 11**

Power function PWM reverse level 5

**6.208.2.15 #define PF\_PWM\_REV6 10**

Power function PWM reverse level 6

**6.208.2.16 #define PF\_PWM\_REV7 9**

Power function PWM reverse level 7

**6.209 HiTechnic device constants**

Constants that are for use with HiTechnic devices.

## Modules

- [HiTechnic IRSeeker2 constants](#)  
*Constants that are for use with the HiTechnic IRSeeker2 device.*
- [HiTechnic IRReceiver constants](#)  
*Constants that are for use with the HiTechnic IRReceiver device.*
- [HiTechnic Color2 constants](#)  
*Constants that are for use with the HiTechnic Color2 device.*
- [HiTechnic Angle sensor constants](#)  
*Constants that are for use with the HiTechnic Angle sensor device.*
- [HiTechnic Barometric sensor constants](#)  
*Constants that are for use with the HiTechnic Barometric sensor device.*
- [HiTechnic Prototype board constants](#)  
*Constants that are for use with the HiTechnic Prototype board.*
- [HiTechnic SuperPro constants](#)  
*Constants that are for use with the HiTechnic SuperPro board.*

## Defines

- #define [HT\\_ADDR\\_IRSEEKER](#) 0x02
- #define [HT\\_ADDR\\_IRSEEKER2](#) 0x10
- #define [HT\\_ADDR\\_IRRECEIVER](#) 0x02
- #define [HT\\_ADDR\\_COMPASS](#) 0x02
- #define [HT\\_ADDR\\_ACCEL](#) 0x02
- #define [HT\\_ADDR\\_COLOR](#) 0x02
- #define [HT\\_ADDR\\_COLOR2](#) 0x02
- #define [HT\\_ADDR\\_IRLINK](#) 0x02
- #define [HT\\_ADDR\\_ANGLE](#) 0x02
- #define [HT\\_ADDR\\_BAROMETRIC](#) 0x02
- #define [HT\\_ADDR\\_PROTOBOARD](#) 0x02
- #define [HT\\_ADDR\\_SUPERPRO](#) 0x10

### 6.209.1 Detailed Description

Constants that are for use with HiTechnic devices.

**6.209.2 Define Documentation****6.209.2.1 #define HT\_ADDR\_ACCEL 0x02**

HiTechnic Accel I2C address

**6.209.2.2 #define HT\_ADDR\_ANGLE 0x02**

HiTechnic Angle I2C address

**6.209.2.3 #define HT\_ADDR\_BAROMETRIC 0x02**

HiTechnic Barometric I2C address

**6.209.2.4 #define HT\_ADDR\_COLOR 0x02**

HiTechnic Color I2C address

**6.209.2.5 #define HT\_ADDR\_COLOR2 0x02**

HiTechnic Color2 I2C address

**6.209.2.6 #define HT\_ADDR\_COMPASS 0x02**

HiTechnic Compass I2C address

**6.209.2.7 #define HT\_ADDR\_IRLINK 0x02**

HiTechnic IRLink I2C address

**6.209.2.8 #define HT\_ADDR\_IRRECEIVER 0x02**

HiTechnic IRReceiver I2C address

**6.209.2.9 #define HT\_ADDR\_IRSEEKER 0x02**

HiTechnic IRSeeker I2C address

**6.209.2.10 #define HT\_ADDR\_IRSEEKER2 0x10**

HiTechnic IRSeeker2 I2C address

**6.209.2.11 #define HT\_ADDR\_PROTOBOARD 0x02**

HiTechnic Prototype board I2C address

**6.209.2.12 #define HT\_ADDR\_SUPERPRO 0x10**

HiTechnic SuperPro board I2C address

**6.210 HiTechnic IRSeeker2 constants**

Constants that are for use with the HiTechnic IRSeeker2 device.

**Defines**

- #define HTIR2\_MODE\_1200 0
- #define HTIR2\_MODE\_600 1
- #define HTIR2\_REG\_MODE 0x41
- #define HTIR2\_REG\_DCDIR 0x42
- #define HTIR2\_REG\_DC01 0x43
- #define HTIR2\_REG\_DC02 0x44
- #define HTIR2\_REG\_DC03 0x45
- #define HTIR2\_REG\_DC04 0x46
- #define HTIR2\_REG\_DC05 0x47
- #define HTIR2\_REG\_DCAVG 0x48
- #define HTIR2\_REG\_ACDIR 0x49
- #define HTIR2\_REG\_AC01 0x4A
- #define HTIR2\_REG\_AC02 0x4B
- #define HTIR2\_REG\_AC03 0x4C
- #define HTIR2\_REG\_AC04 0x4D
- #define HTIR2\_REG\_AC05 0x4E

**6.210.1 Detailed Description**

Constants that are for use with the HiTechnic IRSeeker2 device.

## 6.210.2 Define Documentation

### 6.210.2.1 #define HTIR2\_MODE\_1200 0

Set IRSeeker2 to 1200 mode

#### Examples:

[ex\\_sethtirseeker2mode.nxc](#), and [ex\\_setsensorboolean.nxc](#).

### 6.210.2.2 #define HTIR2\_MODE\_600 1

Set IRSeeker2 to 600 mode

### 6.210.2.3 #define HTIR2\_REG\_AC01 0x4A

IRSeeker2 AC 01 register

### 6.210.2.4 #define HTIR2\_REG\_AC02 0x4B

IRSeeker2 AC 02 register

### 6.210.2.5 #define HTIR2\_REG\_AC03 0x4C

IRSeeker2 AC 03 register

### 6.210.2.6 #define HTIR2\_REG\_AC04 0x4D

IRSeeker2 AC 04 register

### 6.210.2.7 #define HTIR2\_REG\_AC05 0x4E

IRSeeker2 AC 05 register

### 6.210.2.8 #define HTIR2\_REG\_ACDIR 0x49

IRSeeker2 AC direction register

### 6.210.2.9 #define HTIR2\_REG\_DC01 0x43

IRSeeker2 DC 01 register

**6.210.2.10** `#define HTIR2_REG_DC02 0x44`

IRSeeker2 DC 02 register

**6.210.2.11** `#define HTIR2_REG_DC03 0x45`

IRSeeker2 DC 03 register

**6.210.2.12** `#define HTIR2_REG_DC04 0x46`

IRSeeker2 DC 04 register

**6.210.2.13** `#define HTIR2_REG_DC05 0x47`

IRSeeker2 DC 05 register

**6.210.2.14** `#define HTIR2_REG_DC AVG 0x48`

IRSeeker2 DC average register

#### Examples:

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**6.210.2.15** `#define HTIR2_REG_DC DIR 0x42`

IRSeeker2 DC direction register

**6.210.2.16** `#define HTIR2_REG_MODE 0x41`

IRSeeker2 mode register

## 6.211 HiTechnic IRReceiver constants

Constants that are for use with the HiTechnic IRReceiver device.

#### Defines

- `#define HT_CH1_A 0`
- `#define HT_CH1_B 1`

- #define [HT\\_CH2\\_A](#) 2
- #define [HT\\_CH2\\_B](#) 3
- #define [HT\\_CH3\\_A](#) 4
- #define [HT\\_CH3\\_B](#) 5
- #define [HT\\_CH4\\_A](#) 6
- #define [HT\\_CH4\\_B](#) 7

### 6.211.1 Detailed Description

Constants that are for use with the HiTechnic IRReceiver device.

### 6.211.2 Define Documentation

#### 6.211.2.1 #define HT\_CH1\_A 0

Use IRReceiver channel 1 output A

#### Examples:

[ex\\_ReadSensorHTIRReceiverEx.nxc](#).

#### 6.211.2.2 #define HT\_CH1\_B 1

Use IRReceiver channel 1 output B

#### 6.211.2.3 #define HT\_CH2\_A 2

Use IRReceiver channel 2 output A

#### 6.211.2.4 #define HT\_CH2\_B 3

Use IRReceiver channel 2 output B

#### 6.211.2.5 #define HT\_CH3\_A 4

Use IRReceiver channel 3 output A

#### 6.211.2.6 #define HT\_CH3\_B 5

Use IRReceiver channel 3 output B

**6.211.2.7 #define HT\_CH4\_A 6**

Use IRReceiver channel 4 output A

**6.211.2.8 #define HT\_CH4\_B 7**

Use IRReceiver channel 4 output B

**6.212 HiTechnic Color2 constants**

Constants that are for use with the HiTechnic Color2 device.

**Defines**

- #define [HT\\_CMD\\_COLOR2\\_ACTIVE](#) 0x00
- #define [HT\\_CMD\\_COLOR2\\_PASSIVE](#) 0x01
- #define [HT\\_CMD\\_COLOR2\\_RAW](#) 0x03
- #define [HT\\_CMD\\_COLOR2\\_50HZ](#) 0x35
- #define [HT\\_CMD\\_COLOR2\\_60HZ](#) 0x36
- #define [HT\\_CMD\\_COLOR2\\_BLCAL](#) 0x42
- #define [HT\\_CMD\\_COLOR2\\_WBCAL](#) 0x43
- #define [HT\\_CMD\\_COLOR2\\_FAR](#) 0x46
- #define [HT\\_CMD\\_COLOR2\\_LED\\_HI](#) 0x48
- #define [HT\\_CMD\\_COLOR2\\_LED\\_LOW](#) 0x4C
- #define [HT\\_CMD\\_COLOR2\\_NEAR](#) 0x4E

**6.212.1 Detailed Description**

Constants that are for use with the HiTechnic Color2 device.

**6.212.2 Define Documentation****6.212.2.1 #define HT\_CMD\_COLOR2\_50HZ 0x35**

Set the Color2 sensor to 50Hz mode

**6.212.2.2 #define HT\_CMD\_COLOR2\_60HZ 0x36**

Set the Color2 sensor to 60Hz mode

**6.212.2.3 #define HT\_CMD\_COLOR2\_ACTIVE 0x00**

Set the Color2 sensor to active mode

**Examples:**

[ex\\_I2CSendCommand.nxc](#), and [ex\\_sethtcolor2mode.nxc](#).

**6.212.2.4 #define HT\_CMD\_COLOR2\_BLCAL 0x42**

Set the Color2 sensor to black level calibration mode

**6.212.2.5 #define HT\_CMD\_COLOR2\_FAR 0x46**

Set the Color2 sensor to far mode

**6.212.2.6 #define HT\_CMD\_COLOR2\_LED\_HI 0x48**

Set the Color2 sensor to LED high mode

**6.212.2.7 #define HT\_CMD\_COLOR2\_LED\_LOW 0x4C**

Set the Color2 sensor to LED low mode

**6.212.2.8 #define HT\_CMD\_COLOR2\_NEAR 0x4E**

Set the Color2 sensor to near mode

**6.212.2.9 #define HT\_CMD\_COLOR2\_PASSIVE 0x01**

Set the Color2 sensor to passive mode

**6.212.2.10 #define HT\_CMD\_COLOR2\_RAW 0x03**

Set the Color2 sensor to raw mode

**6.212.2.11 #define HT\_CMD\_COLOR2\_WBCAL 0x43**

Set the Color2 sensor to white level calibration mode

## 6.213 HiTechnic Angle sensor constants

Constants that are for use with the HiTechnic Angle sensor device.

### Defines

- #define [HTANGLE\\_MODE\\_NORMAL](#) 0x00
- #define [HTANGLE\\_MODE\\_CALIBRATE](#) 0x43
- #define [HTANGLE\\_MODE\\_RESET](#) 0x52
- #define [HTANGLE\\_REG\\_MODE](#) 0x41
- #define [HTANGLE\\_REG\\_DCDIR](#) 0x42
- #define [HTANGLE\\_REG\\_DC01](#) 0x43
- #define [HTANGLE\\_REG\\_DC02](#) 0x44
- #define [HTANGLE\\_REG\\_DC03](#) 0x45
- #define [HTANGLE\\_REG\\_DC04](#) 0x46
- #define [HTANGLE\\_REG\\_DC05](#) 0x47
- #define [HTANGLE\\_REG\\_DCAVG](#) 0x48
- #define [HTANGLE\\_REG\\_ACDIR](#) 0x49

### 6.213.1 Detailed Description

Constants that are for use with the HiTechnic Angle sensor device.

### 6.213.2 Define Documentation

#### 6.213.2.1 #define HTANGLE\_MODE\_CALIBRATE 0x43

Resets 0 degree position to current shaft angle

#### 6.213.2.2 #define HTANGLE\_MODE\_NORMAL 0x00

Normal angle measurement mode

#### 6.213.2.3 #define HTANGLE\_MODE\_RESET 0x52

Resets the accumulated angle

### Examples:

[ex\\_ResetSensorHTAngle.nxc](#).

**6.213.2.4 #define HTANGLE\_REG\_ACDIR 0x49**

Angle 16 bit revolutions per minute, low byte register

**6.213.2.5 #define HTANGLE\_REG\_DC01 0x43**

Angle current angle (1 degree adder) register

**6.213.2.6 #define HTANGLE\_REG\_DC02 0x44**

Angle 32 bit accumulated angle, high byte register

**6.213.2.7 #define HTANGLE\_REG\_DC03 0x45**

Angle 32 bit accumulated angle, mid byte register

**6.213.2.8 #define HTANGLE\_REG\_DC04 0x46**

Angle 32 bit accumulated angle, mid byte register

**6.213.2.9 #define HTANGLE\_REG\_DC05 0x47**

Angle 32 bit accumulated angle, low byte register

**6.213.2.10 #define HTANGLE\_REG\_DCAVG 0x48**

Angle 16 bit revolutions per minute, high byte register

**6.213.2.11 #define HTANGLE\_REG\_DCDIR 0x42**

Angle current angle (2 degree increments) register

**6.213.2.12 #define HTANGLE\_REG\_MODE 0x41**

Angle mode register

**6.214 HiTechnic Barometric sensor constants**

Constants that are for use with the HiTechnic Barometric sensor device.

### Defines

- #define [HTBAR\\_REG\\_COMMAND](#) 0x40
- #define [HTBAR\\_REG\\_TEMPERATURE](#) 0x42
- #define [HTBAR\\_REG\\_PRESSURE](#) 0x44
- #define [HTBAR\\_REG\\_CALIBRATION](#) 0x46

#### 6.214.1 Detailed Description

Constants that are for use with the HiTechnic Barometric sensor device.

#### 6.214.2 Define Documentation

##### 6.214.2.1 #define HTBAR\_REG\_CALIBRATION 0x46

Barometric sensor calibration register (2 bytes msb/lb)

##### 6.214.2.2 #define HTBAR\_REG\_COMMAND 0x40

Barometric sensor command register

##### 6.214.2.3 #define HTBAR\_REG\_PRESSURE 0x44

Barometric sensor pressure register (2 bytes msb/lb)

##### 6.214.2.4 #define HTBAR\_REG\_TEMPERATURE 0x42

Barometric sensor temperature register (2 bytes msb/lb)

### 6.215 HiTechnic Prototype board constants

Constants that are for use with the HiTechnic Prototype board.

### Modules

- [HiTechnic Prototype board analog input constants](#)

*Constants that are for use with reading the HiTechnic Prototype board analog input values.*

## Defines

- #define [HTPROTO\\_REG\\_A0](#) 0x42
- #define [HTPROTO\\_REG\\_A1](#) 0x44
- #define [HTPROTO\\_REG\\_A2](#) 0x46
- #define [HTPROTO\\_REG\\_A3](#) 0x48
- #define [HTPROTO\\_REG\\_A4](#) 0x4A
- #define [HTPROTO\\_REG\\_DIN](#) 0x4C
- #define [HTPROTO\\_REG\\_DOUT](#) 0x4D
- #define [HTPROTO\\_REG\\_DCTRL](#) 0x4E
- #define [HTPROTO\\_REG\\_SRATE](#) 0x4F

### 6.215.1 Detailed Description

Constants that are for use with the HiTechnic Prototype board.

### 6.215.2 Define Documentation

#### 6.215.2.1 #define [HTPROTO\\_REG\\_A0](#) 0x42

Prototype board analog 0 register (2 bytes msb/lb)

#### 6.215.2.2 #define [HTPROTO\\_REG\\_A1](#) 0x44

Prototype board analog 1 register (2 bytes msb/lb)

#### 6.215.2.3 #define [HTPROTO\\_REG\\_A2](#) 0x46

Prototype board analog 2 register (2 bytes msb/lb)

#### 6.215.2.4 #define [HTPROTO\\_REG\\_A3](#) 0x48

Prototype board analog 3 register (2 bytes msb/lb)

#### 6.215.2.5 #define [HTPROTO\\_REG\\_A4](#) 0x4A

Prototype board analog 4 register (2 bytes msb/lb)

#### 6.215.2.6 #define [HTPROTO\\_REG\\_DCTRL](#) 0x4E

Prototype board digital pin control register (6 bits)

**6.215.2.7 #define HTPROTO\_REG\_DIN 0x4C**

Prototype board digital pin input register (6 bits)

**6.215.2.8 #define HTPROTO\_REG\_DOUT 0x4D**

Prototype board digital pin output register (6 bits)

**6.215.2.9 #define HTPROTO\_REG\_SRATE 0x4F**

Prototype board sample rate register

**6.216 HiTechnic Prototype board analog input constants**

Constants that are for use with reading the HiTechnic Prototype board analog input values.

**Defines**

- #define [HTPROTO\\_A0](#) 0x42
- #define [HTPROTO\\_A1](#) 0x44
- #define [HTPROTO\\_A2](#) 0x46
- #define [HTPROTO\\_A3](#) 0x48
- #define [HTPROTO\\_A4](#) 0x4A

**6.216.1 Detailed Description**

Constants that are for use with reading the HiTechnic Prototype board analog input values.

**6.216.2 Define Documentation****6.216.2.1 #define HTPROTO\_A0 0x42**

Read Prototype board analog input 0

**Examples:**

[ex\\_proto.nxc](#).

**6.216.2.2 #define HTPROTO\_A1 0x44**

Read Prototype board analog input 1

**6.216.2.3 #define HTPROTO\_A2 0x46**

Read Prototype board analog input 2

**6.216.2.4 #define HTPROTO\_A3 0x48**

Read Prototype board analog input 3

**6.216.2.5 #define HTPROTO\_A4 0x4A**

Read Prototype board analog input 4

**6.217 HiTechnic SuperPro constants**

Constants that are for use with the HiTechnic SuperPro board.

**Modules**

- [HiTechnic SuperPro analog input index constants](#)  
*Constants that are for use with reading the HiTechnic SuperPro analog input values.*
- [HiTechnic SuperPro analog output index constants](#)  
*Constants that are for use with configuring the HiTechnic SuperPro analog outputs.*
- [SuperPro LED control constants](#)  
*Constants for controlling the 2 onboard LEDs.*
- [SuperPro analog output mode constants](#)  
*Constants for controlling the 2 analog output modes.*
- [SuperPro digital pin constants](#)  
*Constants for controlling the 8 digital pins.*
- [SuperPro Strobe control constants](#)  
*Constants for manipulating the six digital strobe outputs.*

**Defines**

- #define HTSPRO\_REG\_CTRL 0x40
- #define HTSPRO\_REG\_A0 0x42
- #define HTSPRO\_REG\_A1 0x44
- #define HTSPRO\_REG\_A2 0x46
- #define HTSPRO\_REG\_A3 0x48
- #define HTSPRO\_REG\_DIN 0x4C
- #define HTSPRO\_REG\_DOUT 0x4D
- #define HTSPRO\_REG\_DCTRL 0x4E
- #define HTSPRO\_REG\_STROBE 0x50
- #define HTSPRO\_REG\_LED 0x51
- #define HTSPRO\_REG\_DAC0\_MODE 0x52
- #define HTSPRO\_REG\_DAC0\_FREQ 0x53
- #define HTSPRO\_REG\_DAC0\_VOLTAGE 0x55
- #define HTSPRO\_REG\_DAC1\_MODE 0x57
- #define HTSPRO\_REG\_DAC1\_FREQ 0x58
- #define HTSPRO\_REG\_DAC1\_VOLTAGE 0x5A
- #define HTSPRO\_REG\_DLADDRESS 0x60
- #define HTSPRO\_REG\_DLDATA 0x62
- #define HTSPRO\_REG\_DLCHKSUM 0x6A
- #define HTSPRO\_REG\_DLCONTROL 0x6B
- #define HTSPRO\_REG\_MEMORY\_20 0x80
- #define HTSPRO\_REG\_MEMORY\_21 0x84
- #define HTSPRO\_REG\_MEMORY\_22 0x88
- #define HTSPRO\_REG\_MEMORY\_23 0x8C
- #define HTSPRO\_REG\_MEMORY\_24 0x90
- #define HTSPRO\_REG\_MEMORY\_25 0x94
- #define HTSPRO\_REG\_MEMORY\_26 0x98
- #define HTSPRO\_REG\_MEMORY\_27 0x9C
- #define HTSPRO\_REG\_MEMORY\_28 0xA0
- #define HTSPRO\_REG\_MEMORY\_29 0xA4
- #define HTSPRO\_REG\_MEMORY\_2A 0xA8
- #define HTSPRO\_REG\_MEMORY\_2B 0xAC
- #define HTSPRO\_REG\_MEMORY\_2C 0xB0
- #define HTSPRO\_REG\_MEMORY\_2D 0xB4
- #define HTSPRO\_REG\_MEMORY\_2E 0xB8
- #define HTSPRO\_REG\_MEMORY\_2F 0xBC
- #define HTSPRO\_REG\_MEMORY\_30 0xC0
- #define HTSPRO\_REG\_MEMORY\_31 0xC4
- #define HTSPRO\_REG\_MEMORY\_32 0xC8
- #define HTSPRO\_REG\_MEMORY\_33 0xCC
- #define HTSPRO\_REG\_MEMORY\_34 0xD0

- #define HTSPRO\_REG\_MEMORY\_35 0xD4
- #define HTSPRO\_REG\_MEMORY\_36 0xD8
- #define HTSPRO\_REG\_MEMORY\_37 0xDC
- #define HTSPRO\_REG\_MEMORY\_38 0xE0
- #define HTSPRO\_REG\_MEMORY\_39 0xE4
- #define HTSPRO\_REG\_MEMORY\_3A 0xE8
- #define HTSPRO\_REG\_MEMORY\_3B 0xEC
- #define HTSPRO\_REG\_MEMORY\_3C 0xF0
- #define HTSPRO\_REG\_MEMORY\_3D 0xF4
- #define HTSPRO\_REG\_MEMORY\_3E 0xF8
- #define HTSPRO\_REG\_MEMORY\_3F 0xFC

### 6.217.1 Detailed Description

Constants that are for use with the HiTechnic SuperPro board.

### 6.217.2 Define Documentation

#### 6.217.2.1 #define HTSPRO\_REG\_A0 0x42

SuperPro analog 0 register (10 bits)

#### 6.217.2.2 #define HTSPRO\_REG\_A1 0x44

SuperPro analog 1 register (10 bits)

#### 6.217.2.3 #define HTSPRO\_REG\_A2 0x46

SuperPro analog 2 register (10 bits)

#### 6.217.2.4 #define HTSPRO\_REG\_A3 0x48

SuperPro analog 3 register (10 bits)

#### 6.217.2.5 #define HTSPRO\_REG\_CTRL 0x40

SuperPro program control register

#### 6.217.2.6 #define HTSPRO\_REG\_DAC0\_FREQ 0x53

SuperPro analog output 0 frequency register (2 bytes msb/lb)

**6.217.2.7 #define HTSPRO\_REG\_DAC0\_MODE 0x52**

SuperPro analog output 0 mode register

**6.217.2.8 #define HTSPRO\_REG\_DAC0\_VOLTAGE 0x55**

SuperPro analog output 0 voltage register (10 bits)

**6.217.2.9 #define HTSPRO\_REG\_DAC1\_FREQ 0x58**

SuperPro analog output 1 frequency register (2 bytes msb/lb)

**6.217.2.10 #define HTSPRO\_REG\_DAC1\_MODE 0x57**

SuperPro analog output 1 mode register

**6.217.2.11 #define HTSPRO\_REG\_DAC1\_VOLTAGE 0x5A**

SuperPro analog output 1 voltage register (10 bits)

**6.217.2.12 #define HTSPRO\_REG\_DCTRL 0x4E**

SuperPro digital pin control register (8 bits)

**6.217.2.13 #define HTSPRO\_REG\_DIN 0x4C**

SuperPro digital pin input register (8 bits)

**6.217.2.14 #define HTSPRO\_REG\_DLADDRESS 0x60**

SuperPro download address register (2 bytes msb/lb)

**6.217.2.15 #define HTSPRO\_REG\_DLCHKSUM 0x6A**

SuperPro download checksum register

**6.217.2.16 #define HTSPRO\_REG\_DLCONTROL 0x6B**

SuperPro download control register

**6.217.2.17 #define HTSPRO\_REG\_DLDATA 0x62**

SuperPro download data register (8 bytes)

**6.217.2.18 #define HTSPRO\_REG\_DOUT 0x4D**

SuperPro digital pin output register (8 bits)

**6.217.2.19 #define HTSPRO\_REG\_LED 0x51**

SuperPro LED control register

**6.217.2.20 #define HTSPRO\_REG\_MEMORY\_20 0x80**

SuperPro memory address 0x20 register (4 bytes msb/lb)

**6.217.2.21 #define HTSPRO\_REG\_MEMORY\_21 0x84**

SuperPro memory address 0x21 register (4 bytes msb/lb)

**6.217.2.22 #define HTSPRO\_REG\_MEMORY\_22 0x88**

SuperPro memory address 0x22 register (4 bytes msb/lb)

**6.217.2.23 #define HTSPRO\_REG\_MEMORY\_23 0x8C**

SuperPro memory address 0x23 register (4 bytes msb/lb)

**6.217.2.24 #define HTSPRO\_REG\_MEMORY\_24 0x90**

SuperPro memory address 0x24 register (4 bytes msb/lb)

**6.217.2.25 #define HTSPRO\_REG\_MEMORY\_25 0x94**

SuperPro memory address 0x25 register (4 bytes msb/lb)

**6.217.2.26 #define HTSPRO\_REG\_MEMORY\_26 0x98**

SuperPro memory address 0x26 register (4 bytes msb/lb)

**6.217.2.27 #define HTSPRO\_REG\_MEMORY\_27 0x9C**

SuperPro memory address 0x27 register (4 bytes msb/lb)

**6.217.2.28 #define HTSPRO\_REG\_MEMORY\_28 0xA0**

SuperPro memory address 0x28 register (4 bytes msb/lb)

**6.217.2.29 #define HTSPRO\_REG\_MEMORY\_29 0xA4**

SuperPro memory address 0x29 register (4 bytes msb/lb)

**6.217.2.30 #define HTSPRO\_REG\_MEMORY\_2A 0xA8**

SuperPro memory address 0x2A register (4 bytes msb/lb)

**6.217.2.31 #define HTSPRO\_REG\_MEMORY\_2B 0xAC**

SuperPro memory address 0x2B register (4 bytes msb/lb)

**6.217.2.32 #define HTSPRO\_REG\_MEMORY\_2C 0xB0**

SuperPro memory address 0x2C register (4 bytes msb/lb)

**6.217.2.33 #define HTSPRO\_REG\_MEMORY\_2D 0xB4**

SuperPro memory address 0x2D register (4 bytes msb/lb)

**6.217.2.34 #define HTSPRO\_REG\_MEMORY\_2E 0xB8**

SuperPro memory address 0x2E register (4 bytes msb/lb)

**6.217.2.35 #define HTSPRO\_REG\_MEMORY\_2F 0xBC**

SuperPro memory address 0x2F register (4 bytes msb/lb)

**6.217.2.36 #define HTSPRO\_REG\_MEMORY\_30 0xC0**

SuperPro memory address 0x30 register (4 bytes msb/lb)

**6.217.2.37 #define HTSPRO\_REG\_MEMORY\_31 0xC4**

SuperPro memory address 0x31 register (4 bytes msb/lb)

**6.217.2.38 #define HTSPRO\_REG\_MEMORY\_32 0xC8**

SuperPro memory address 0x32 register (4 bytes msb/lb)

**6.217.2.39 #define HTSPRO\_REG\_MEMORY\_33 0xCC**

SuperPro memory address 0x33 register (4 bytes msb/lb)

**6.217.2.40 #define HTSPRO\_REG\_MEMORY\_34 0xD0**

SuperPro memory address 0x34 register (4 bytes msb/lb)

**6.217.2.41 #define HTSPRO\_REG\_MEMORY\_35 0xD4**

SuperPro memory address 0x35 register (4 bytes msb/lb)

**6.217.2.42 #define HTSPRO\_REG\_MEMORY\_36 0xD8**

SuperPro memory address 0x36 register (4 bytes msb/lb)

**6.217.2.43 #define HTSPRO\_REG\_MEMORY\_37 0xDC**

SuperPro memory address 0x37 register (4 bytes msb/lb)

**6.217.2.44 #define HTSPRO\_REG\_MEMORY\_38 0xE0**

SuperPro memory address 0x38 register (4 bytes msb/lb)

**6.217.2.45 #define HTSPRO\_REG\_MEMORY\_39 0xE4**

SuperPro memory address 0x39 register (4 bytes msb/lb)

**6.217.2.46 #define HTSPRO\_REG\_MEMORY\_3A 0xE8**

SuperPro memory address 0x3A register (4 bytes msb/lb)

**6.217.2.47 #define HTSPRO\_REG\_MEMORY\_3B 0xEC**

SuperPro memory address 0x3B register (4 bytes msb/lb)

**6.217.2.48 #define HTSPRO\_REG\_MEMORY\_3C 0xF0**

SuperPro memory address 0x3C register (4 bytes msb/lb)

**6.217.2.49 #define HTSPRO\_REG\_MEMORY\_3D 0xF4**

SuperPro memory address 0x3D register (4 bytes msb/lb)

**6.217.2.50 #define HTSPRO\_REG\_MEMORY\_3E 0xF8**

SuperPro memory address 0x3E register (4 bytes msb/lb)

**6.217.2.51 #define HTSPRO\_REG\_MEMORY\_3F 0xFC**

SuperPro memory address 0x3F register (4 bytes msb/lb)

**6.217.2.52 #define HTSPRO\_REG\_STROBE 0x50**

SuperPro strobe control register

**6.218 HiTechnic SuperPro analog input index constants**

Constants that are for use with reading the HiTechnic SuperPro analog input values.

**Defines**

- #define [HTSPRO\\_A0](#) 0x42
- #define [HTSPRO\\_A1](#) 0x44
- #define [HTSPRO\\_A2](#) 0x46
- #define [HTSPRO\\_A3](#) 0x48

**6.218.1 Detailed Description**

Constants that are for use with reading the HiTechnic SuperPro analog input values.

### 6.218.2 Define Documentation

#### 6.218.2.1 #define HTSPRO\_A0 0x42

Read SuperPro analog input 0

#### Examples:

[ex\\_superpro.nxc](#).

#### 6.218.2.2 #define HTSPRO\_A1 0x44

Read SuperPro analog input 1

#### 6.218.2.3 #define HTSPRO\_A2 0x46

Read SuperPro analog input 2

#### 6.218.2.4 #define HTSPRO\_A3 0x48

Read SuperPro analog input 3

## 6.219 HiTechnic SuperPro analog output index constants

Constants that are for use with configuraing the HiTechnic SuperPro analog outputs.

### Defines

- #define [HTSPRO\\_DAC0](#) 0x52
- #define [HTSPRO\\_DAC1](#) 0x57

### 6.219.1 Detailed Description

Constants that are for use with configuraing the HiTechnic SuperPro analog outputs.

### 6.219.2 Define Documentation

#### 6.219.2.1 #define HTSPRO\_DAC0 0x52

Set SuperPro analog output 0 configuration

**Examples:**

[ex\\_superpro.nxc](#).

**6.219.2.2 #define HTSPRO\_DAC1 0x57**

Set SuperPro analog output 1 configuration

**Examples:**

[ex\\_superpro.nxc](#).

**6.220 MindSensors device constants**

Constants that are for use with MindSensors devices.

**Modules**

- [MindSensors DIST-Nx constants](#)  
*Constants that are for use with the MindSensors DIST-Nx device.*
- [MindSensors PSP-Nx constants](#)  
*Constants that are for use with the MindSensors PSP-Nx device.*
- [MindSensors nRLink constants](#)  
*Constants that are for use with the MindSensors nRLink device.*
- [MindSensors ACCL-Nx constants](#)  
*Constants that are for use with the MindSensors ACCL-Nx device.*
- [MindSensors PFMate constants](#)  
*Constants that are for use with the MindSensors PFMate device.*
- [MindSensors NXTServo constants](#)  
*Constants that are for use with the MindSensors NXTServo device.*
- [MindSensors NXTHID constants](#)  
*Constants that are for use with the MindSensors NXTHID device.*
- [MindSensors NXTPowerMeter constants](#)  
*Constants that are for use with the MindSensors NXTPowerMeter device.*

- [MindSensors NXTSumoEyes constants](#)

*Constants that are for use with the MindSensors NXTSumoEyes device.*

- [MindSensors NXTLineLeader constants](#)

*Constants that are for use with the MindSensors NXTLineLeader device.*

## Defines

- #define [MS\\_CMD\\_ENERGIZED](#) 0x45
- #define [MS\\_CMD\\_DEENERGIZED](#) 0x44
- #define [MS\\_CMD\\_ADPA\\_ON](#) 0x4E
- #define [MS\\_CMD\\_ADPA\\_OFF](#) 0x4F
- #define [MS\\_ADDR\\_RTCLOCK](#) 0xD0
- #define [MS\\_ADDR\\_DISTNX](#) 0x02
- #define [MS\\_ADDR\\_NRLINK](#) 0x02
- #define [MS\\_ADDR\\_ACCLNX](#) 0x02
- #define [MS\\_ADDR\\_CMPSNX](#) 0x02
- #define [MS\\_ADDR\\_PSPNX](#) 0x02
- #define [MS\\_ADDR\\_LINELDR](#) 0x02
- #define [MS\\_ADDR\\_NXTCAM](#) 0x02
- #define [MS\\_ADDR\\_NXTHID](#) 0x04
- #define [MS\\_ADDR\\_NXTSERVO](#) 0xB0
- #define [MS\\_ADDR\\_NXTSERVO\\_EM](#) 0x40
- #define [MS\\_ADDR\\_PFMATE](#) 0x48
- #define [MS\\_ADDR\\_MTRMUX](#) 0xB4
- #define [MS\\_ADDR\\_NXTMMX](#) 0x06
- #define [MS\\_ADDR\\_IVSENS](#) 0x12
- #define [MS\\_ADDR\\_RXMUX](#) 0x7E

### 6.220.1 Detailed Description

Constants that are for use with MindSensors devices.

### 6.220.2 Define Documentation

#### 6.220.2.1 #define MS\_ADDR\_ACCLNX 0x02

MindSensors ACCL-Nx I2C address

**Examples:**

[ex\\_ACCLNxCalibrateX.nxc](#),      [ex\\_ACCLNxCalibrateXEnd.nxc](#),      [ex\\_-](#)  
[ACCLNxCalibrateY.nxc](#),      [ex\\_ACCLNxCalibrateYEnd.nxc](#),      [ex\\_-](#)  
[ACCLNxCalibrateZ.nxc](#),      [ex\\_ACCLNxCalibrateZEnd.nxc](#),      [ex\\_-](#)  
[ACCLNxResetCalibration.nxc](#),      [ex\\_ACCLNxSensitivity.nxc](#),      [ex\\_-](#)  
[ACCLNxXOffset.nxc](#),      [ex\\_ACCLNxXRange.nxc](#),      [ex\\_ACCLNxYOffset.nxc](#),  
[ex\\_ACCLNxYRange.nxc](#),      [ex\\_ACCLNxZOffset.nxc](#),      [ex\\_ACCLNxZRange.nxc](#),  
[ex\\_ReadSensorMSAccel.nxc](#),      [ex\\_ReadSensorMSTilt.nxc](#),      and      [ex\\_-](#)  
[SetACCLNxSensitivity.nxc](#).

**6.220.2.2 #define MS\_ADDR\_CMPSNX 0x02**

MindSensors CMPS-Nx I2C address

**Examples:**

[ex\\_SensorMSCompass.nxc](#).

**6.220.2.3 #define MS\_ADDR\_DISTNX 0x02**

MindSensors DIST-Nx I2C address

**Examples:**

[ex\\_DISTNxDistance.nxc](#),      [ex\\_DISTNxGP2D12.nxc](#),      [ex\\_DISTNxGP2D120.nxc](#),  
[ex\\_DISTNxGP2YA02.nxc](#),      [ex\\_DISTNxGP2YA21.nxc](#),      [ex\\_-](#)  
[DISTNxMaxDistance.nxc](#),      [ex\\_DISTNxMinDistance.nxc](#),      [ex\\_-](#)  
[DISTNxModuleType.nxc](#),      [ex\\_DISTNxNumPoints.nxc](#),      [ex\\_DISTNxVoltage.nxc](#),  
[ex\\_MSADPAOff.nxc](#), and [ex\\_MSADPAOn.nxc](#).

**6.220.2.4 #define MS\_ADDR\_IVSENS 0x12**

MindSensors IVSens (NXTPowerMeter) I2C address

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**6.220.2.5 #define MS\_ADDR\_LINELDR 0x02**

MindSensors LineLdr I2C address

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**6.220.2.6 #define MS\_ADDR\_MTRMUX 0xB4**

MindSensors MTRMux I2C address

**6.220.2.7 #define MS\_ADDR\_NRLINK 0x02**

MindSensors NRLink I2C address

**Examples:**

[ex\\_MSRCXSetNRLinkPort.nxc](#), [ex\\_NRLink2400.nxc](#), [ex\\_NRLink4800.nxc](#),  
[ex\\_NRLinkFlush.nxc](#), [ex\\_NRLinkIRLong.nxc](#), [ex\\_NRLinkIRShort.nxc](#),  
[ex\\_NRLinkSetPF.nxc](#), [ex\\_NRLinkSetRCX.nxc](#), [ex\\_NRLinkSetTrain.nxc](#),  
[ex\\_NRLinkStatus.nxc](#), [ex\\_NRLinkTxRaw.nxc](#), [ex\\_ReadNRLinkBytes.nxc](#),  
[ex\\_RunNRLinkMacro.nxc](#), and [ex\\_writenrlinkbytes.nxc](#).

**6.220.2.8 #define MS\_ADDR\_NXTCAM 0x02**

MindSensors NXTCam I2C address

**6.220.2.9 #define MS\_ADDR\_NXTHID 0x04**

MindSensors NXTHID I2C address

**Examples:**

[ex\\_NXTHID.nxc](#).

**6.220.2.10 #define MS\_ADDR\_NXTMMX 0x06**

MindSensors NXTMMX I2C address

**6.220.2.11 #define MS\_ADDR\_NXTSERVO 0xB0**

MindSensors NXTServo I2C address

**Examples:**

[ex\\_NXTHID.nxc](#), and [ex\\_NXTServo.nxc](#).

**6.220.2.12 #define MS\_ADDR\_NXTSERVO\_EM 0x40**

MindSensors NXTServo in edit macro mode I2C address

**6.220.2.13 #define MS\_ADDR\_PFMATE 0x48**

MindSensors PFMate I2C address

**Examples:**[ex\\_PFMate.nxc](#).**6.220.2.14 #define MS\_ADDR\_PSPNX 0x02**

MindSensors PSP-Nx I2C address

**Examples:**[ex\\_PSPNxAAnalog.nxc](#), [ex\\_PSPNxDigital.nxc](#), and [ex\\_ReadSensorMSPlayStation.nxc](#).**6.220.2.15 #define MS\_ADDR\_RTCLOCK 0xD0**

MindSensors RTClock I2C address

**6.220.2.16 #define MS\_ADDR\_RXMUX 0x7E**

MindSensors RXMux I2C address

**6.220.2.17 #define MS\_CMD\_ADPA\_OFF 0x4F**

Turn MindSensors ADPA mode off

**6.220.2.18 #define MS\_CMD\_ADPA\_ON 0x4E**

Turn MindSensors ADPA mode on

**6.220.2.19 #define MS\_CMD\_DEENERGIZED 0x44**

De-energize the MindSensors device

**6.220.2.20 #define MS\_CMD\_ENERGIZED 0x45**

Energize the MindSensors device

**6.221 MindSensors DIST-Nx constants**

Constants that are for use with the MindSensors DIST-Nx device.

**Defines**

- #define [DIST\\_CMD\\_GP2D12](#) 0x31
- #define [DIST\\_CMD\\_GP2D120](#) 0x32
- #define [DIST\\_CMD\\_GP2YA21](#) 0x33
- #define [DIST\\_CMD\\_GP2YA02](#) 0x34
- #define [DIST\\_CMD\\_CUSTOM](#) 0x35
- #define [DIST\\_REG\\_DIST](#) 0x42
- #define [DIST\\_REG\\_VOLT](#) 0x44
- #define [DIST\\_REG\\_MODULE\\_TYPE](#) 0x50
- #define [DIST\\_REG\\_NUM\\_POINTS](#) 0x51
- #define [DIST\\_REG\\_DIST\\_MIN](#) 0x52
- #define [DIST\\_REG\\_DIST\\_MAX](#) 0x54
- #define [DIST\\_REG\\_VOLT1](#) 0x56
- #define [DIST\\_REG\\_DIST1](#) 0x58

**6.221.1 Detailed Description**

Constants that are for use with the MindSensors DIST-Nx device.

**6.221.2 Define Documentation****6.221.2.1 #define DIST\_CMD\_CUSTOM 0x35**

Set the DIST-Nx to a custom mode

**6.221.2.2 #define DIST\_CMD\_GP2D12 0x31**

Set the DIST-Nx to GP2D12 mode

**6.221.2.3 #define DIST\_CMD\_GP2D120 0x32**

Set the DIST-Nx to GP2D120 mode

**6.221.2.4 #define DIST\_CMD\_GP2YA02 0x34**

Set the DIST-Nx to GP2YA02 mode

**6.221.2.5 #define DIST\_CMD\_GP2YA21 0x33**

Set the DIST-Nx to GP2YA21 mode

**6.221.2.6 #define DIST\_REG\_DIST 0x42**

The DIST-Nx distance register

**6.221.2.7 #define DIST\_REG\_DIST1 0x58**

The DIST-Nx distance 1 register

**6.221.2.8 #define DIST\_REG\_DIST\_MAX 0x54**

The DIST-Nx maximum distance register

**6.221.2.9 #define DIST\_REG\_DIST\_MIN 0x52**

The DIST-Nx minimum distance register

**6.221.2.10 #define DIST\_REG\_MODULE\_TYPE 0x50**

The DIST-Nx module type register

**6.221.2.11 #define DIST\_REG\_NUM\_POINTS 0x51**

The DIST-Nx number of data points in Custom curve register

**6.221.2.12 #define DIST\_REG\_VOLT 0x44**

The DIST-Nx voltage register

**6.221.2.13 #define DIST\_REG\_VOLT1 0x56**

The DIST-Nx voltage 1 register

## 6.222 MindSensors PSP-Nx constants

Constants that are for use with the MindSensors PSP-Nx device.

### Modules

- [MindSensors PSP-Nx button set 1 constants](#)  
*Constants that are for interpreting MindSensors PSP-Nx button set 1 values.*
- [MindSensors PSP-Nx button set 2 constants](#)  
*Constants that are for interpreting MindSensors PSP-Nx button set 2 values.*

### Defines

- #define [PSP\\_CMD\\_DIGITAL](#) 0x41
- #define [PSP\\_CMD\\_ANALOG](#) 0x73
- #define [PSP\\_REG\\_BTNSET1](#) 0x42
- #define [PSP\\_REG\\_BTNSET2](#) 0x43
- #define [PSP\\_REG\\_XLEFT](#) 0x44
- #define [PSP\\_REG\\_YLEFT](#) 0x45
- #define [PSP\\_REG\\_XRIGHT](#) 0x46
- #define [PSP\\_REG\\_YRIGHT](#) 0x47

#### 6.222.1 Detailed Description

Constants that are for use with the MindSensors PSP-Nx device.

#### 6.222.2 Define Documentation

##### 6.222.2.1 #define [PSP\\_CMD\\_ANALOG](#) 0x73

Set the PSP-Nx to analog mode

##### 6.222.2.2 #define [PSP\\_CMD\\_DIGITAL](#) 0x41

Set the PSP-Nx to digital mode

##### 6.222.2.3 #define [PSP\\_REG\\_BTNSET1](#) 0x42

The PSP-Nx button set 1 register

**6.222.2.4 #define PSP\_REG\_BTNSET2 0x43**

The PSP-Nx button set 2 register

**6.222.2.5 #define PSP\_REG\_XLEFT 0x44**

The PSP-Nx X left register

**6.222.2.6 #define PSP\_REG\_XRIGHT 0x46**

The PSP-Nx X right register

**6.222.2.7 #define PSP\_REG\_YLEFT 0x45**

The PSP-Nx Y left register

**6.222.2.8 #define PSP\_REG\_YRIGHT 0x47**

The PSP-Nx Y right register

**6.223 MindSensors PSP-Nx button set 1 constants**

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

**Defines**

- #define [PSP\\_BTNSET1\\_LEFT](#) 0x80
- #define [PSP\\_BTNSET1\\_DOWN](#) 0x40
- #define [PSP\\_BTNSET1\\_RIGHT](#) 0x20
- #define [PSP\\_BTNSET1\\_UP](#) 0x10
- #define [PSP\\_BTNSET1\\_START](#) 0x08
- #define [PSP\\_BTNSET1\\_R3](#) 0x04
- #define [PSP\\_BTNSET1\\_L3](#) 0x02
- #define [PSP\\_BTNSET1\\_SELECT](#) 0x01

**6.223.1 Detailed Description**

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

## 6.223.2 Define Documentation

### 6.223.2.1 #define PSP\_BTNSET1\_DOWN 0x40

The PSP-Nx button set 1 down arrow

#### Examples:

[ex\\_ReadSensorMSPlayStation.nxc](#).

### 6.223.2.2 #define PSP\_BTNSET1\_L3 0x02

The PSP-Nx button set 1 L3

#### Examples:

[ex\\_ReadSensorMSPlayStation.nxc](#).

### 6.223.2.3 #define PSP\_BTNSET1\_LEFT 0x80

The PSP-Nx button set 1 left arrow

#### Examples:

[ex\\_ReadSensorMSPlayStation.nxc](#).

### 6.223.2.4 #define PSP\_BTNSET1\_R3 0x04

The PSP-Nx button set 1 R3

#### Examples:

[ex\\_ReadSensorMSPlayStation.nxc](#).

### 6.223.2.5 #define PSP\_BTNSET1\_RIGHT 0x20

The PSP-Nx button set 1 right arrow

#### Examples:

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.223.2.6 #define PSP\_BTNSET1\_SELECT 0x01**

The PSP-Nx button set 1 select

**6.223.2.7 #define PSP\_BTNSET1\_START 0x08**

The PSP-Nx button set 1 start

**6.223.2.8 #define PSP\_BTNSET1\_UP 0x10**

The PSP-Nx button set 1 up arrow

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224 MindSensors PSP-Nx button set 2 constants**

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

**Defines**

- #define [PSP\\_BTNSET2\\_SQUARE](#) 0x80
- #define [PSP\\_BTNSET2\\_CROSS](#) 0x40
- #define [PSP\\_BTNSET2\\_CIRCLE](#) 0x20
- #define [PSP\\_BTNSET2\\_TRIANGLE](#) 0x10
- #define [PSP\\_BTNSET2\\_R1](#) 0x08
- #define [PSP\\_BTNSET2\\_L1](#) 0x04
- #define [PSP\\_BTNSET2\\_R2](#) 0x02
- #define [PSP\\_BTNSET2\\_L2](#) 0x01

**6.224.1 Detailed Description**

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

**6.224.2 Define Documentation****6.224.2.1 #define PSP\_BTNSET2\_CIRCLE 0x20**

The PSP-Nx button set 2 circle

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.2 #define PSP\_BTNSET2\_CROSS 0x40**

The PSP-Nx button set 2 cross

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.3 #define PSP\_BTNSET2\_L1 0x04**

The PSP-Nx button set 2 L1

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.4 #define PSP\_BTNSET2\_L2 0x01**

The PSP-Nx button set 2 L2

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.5 #define PSP\_BTNSET2\_R1 0x08**

The PSP-Nx button set 2 R1

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.6 #define PSP\_BTNSET2\_R2 0x02**

The PSP-Nx button set 2 R2

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.7 #define PSP\_BTNSET2\_SQUARE 0x80**

The PSP-Nx button set 2 square

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.224.2.8 #define PSP\_BTNSET2\_TRIANGLE 0x10**

The PSP-Nx button set 2 triangle

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**6.225 MindSensors nRLink constants**

Constants that are for use with the MindSensors nRLink device.

**Defines**

- #define [NRLINK\\_CMD\\_2400](#) 0x44
- #define [NRLINK\\_CMD\\_FLUSH](#) 0x46
- #define [NRLINK\\_CMD\\_4800](#) 0x48
- #define [NRLINK\\_CMD\\_IR\\_LONG](#) 0x4C
- #define [NRLINK\\_CMD\\_IR\\_SHORT](#) 0x53
- #define [NRLINK\\_CMD\\_RUN\\_MACRO](#) 0x52
- #define [NRLINK\\_CMD\\_TX\\_RAW](#) 0x55
- #define [NRLINK\\_CMD\\_SET\\_RCX](#) 0x58
- #define [NRLINK\\_CMD\\_SET\\_TRAIN](#) 0x54
- #define [NRLINK\\_CMD\\_SET\\_PF](#) 0x50
- #define [NRLINK\\_REG\\_BYTES](#) 0x40
- #define [NRLINK\\_REG\\_DATA](#) 0x42
- #define [NRLINK\\_REG\\_EEPROM](#) 0x50

**6.225.1 Detailed Description**

Constants that are for use with the MindSensors nRLink device.

**6.225.2 Define Documentation****6.225.2.1 #define NRLINK\_CMD\_2400 0x44**

Set nRLink to 2400 baud

**6.225.2.2 #define NRLINK\_CMD\_4800 0x48**

Set nRLink to 4800 baud

**6.225.2.3 #define NRLINK\_CMD\_FLUSH 0x46**

Flush the nRLink

**6.225.2.4 #define NRLINK\_CMD\_IR\_LONG 0x4C**

Set the nRLink to long range IR

**6.225.2.5 #define NRLINK\_CMD\_IR\_SHORT 0x53**

Set the nRLink to short range IR

**6.225.2.6 #define NRLINK\_CMD\_RUN\_MACRO 0x52**

Run an nRLink macro

**6.225.2.7 #define NRLINK\_CMD\_SET\_PF 0x50**

Set the nRLink to Power Function mode

**6.225.2.8 #define NRLINK\_CMD\_SET\_RCX 0x58**

Set the nRLink to RCX mode

**6.225.2.9 #define NRLINK\_CMD\_SET\_TRAIN 0x54**

Set the nRLink to IR Train mode

**6.225.2.10 #define NRLINK\_CMD\_TX\_RAW 0x55**

Set the NRLink to transmit raw bytes

**6.225.2.11 #define NRLINK\_REG\_BYTES 0x40**

The NRLink bytes register

**6.225.2.12 #define NRLINK\_REG\_DATA 0x42**

The NRLink data register

**6.225.2.13 #define NRLINK\_REG\_EEPROM 0x50**

The NRLink eeprom register

**6.226 MindSensors ACCL-Nx constants**

Constants that are for use with the MindSensors ACCL-Nx device.

**Modules**

- [MindSensors ACCL-Nx sensitivity level constants](#)

*Constants that are for setting the MindSensors ACCL-Nx sensitivity level.*

**Defines**

- #define [ACCL\\_CMD\\_X\\_CAL](#) 0x58
- #define [ACCL\\_CMD\\_Y\\_CAL](#) 0x59
- #define [ACCL\\_CMD\\_Z\\_CAL](#) 0x5a
- #define [ACCL\\_CMD\\_X\\_CAL\\_END](#) 0x78
- #define [ACCL\\_CMD\\_Y\\_CAL\\_END](#) 0x79
- #define [ACCL\\_CMD\\_Z\\_CAL\\_END](#) 0x7a
- #define [ACCL\\_CMD\\_RESET\\_CAL](#) 0x52
- #define [ACCL\\_REG\\_SENS\\_LVL](#) 0x19
- #define [ACCL\\_REG\\_X\\_TILT](#) 0x42
- #define [ACCL\\_REG\\_Y\\_TILT](#) 0x43
- #define [ACCL\\_REG\\_Z\\_TILT](#) 0x44
- #define [ACCL\\_REG\\_X\\_ACCEL](#) 0x45

- #define `ACCL_REG_Y_ACCEL` 0x47
- #define `ACCL_REG_Z_ACCEL` 0x49
- #define `ACCL_REG_X_OFFSET` 0x4b
- #define `ACCL_REG_X_RANGE` 0x4d
- #define `ACCL_REG_Y_OFFSET` 0x4f
- #define `ACCL_REG_Y_RANGE` 0x51
- #define `ACCL_REG_Z_OFFSET` 0x53
- #define `ACCL_REG_Z_RANGE` 0x55

### 6.226.1 Detailed Description

Constants that are for use with the MindSensors ACCL-Nx device.

### 6.226.2 Define Documentation

#### 6.226.2.1 #define `ACCL_CMD_RESET_CAL` 0x52

Reset to factory calibration

#### 6.226.2.2 #define `ACCL_CMD_X_CAL` 0x58

Acquire X-axis calibration point

#### 6.226.2.3 #define `ACCL_CMD_X_CAL_END` 0x78

Acquire X-axis calibration point and end calibration

#### 6.226.2.4 #define `ACCL_CMD_Y_CAL` 0x59

Acquire Y-axis calibration point

#### 6.226.2.5 #define `ACCL_CMD_Y_CAL_END` 0x79

Acquire Y-axis calibration point and end calibration

#### 6.226.2.6 #define `ACCL_CMD_Z_CAL` 0x5a

Acquire Z-axis calibration point

**6.226.2.7 #define ACCL\_CMD\_Z\_CAL\_END 0x7a**

Acquire Z-axis calibration point and end calibration

**6.226.2.8 #define ACCL\_REG\_SENS\_LVL 0x19**

The current sensitivity

**6.226.2.9 #define ACCL\_REG\_X\_ACCEL 0x45**

The X-axis acceleration data

**6.226.2.10 #define ACCL\_REG\_X\_OFFSET 0x4b**

The X-axis offset

**6.226.2.11 #define ACCL\_REG\_X\_RANGE 0x4d**

The X-axis range

**6.226.2.12 #define ACCL\_REG\_X\_TILT 0x42**

The X-axis tilt data

**6.226.2.13 #define ACCL\_REG\_Y\_ACCEL 0x47**

The Y-axis acceleration data

**6.226.2.14 #define ACCL\_REG\_Y\_OFFSET 0x4f**

The Y-axis offset

**6.226.2.15 #define ACCL\_REG\_Y\_RANGE 0x51**

The Y-axis range

**6.226.2.16 #define ACCL\_REG\_Y\_TILT 0x43**

The Y-axis tilt data

**6.226.2.17 #define ACCL\_REG\_Z\_ACCEL 0x49**

The Z-axis acceleration data

**6.226.2.18 #define ACCL\_REG\_Z\_OFFSET 0x53**

The Z-axis offset

**6.226.2.19 #define ACCL\_REG\_Z\_RANGE 0x55**

The Z-axis range

**6.226.2.20 #define ACCL\_REG\_Z\_TILT 0x44**

The Z-axis tilt data

**6.227 MindSensors ACCL-Nx sensitivity level constants**

Constants that are for setting the MindSensors ACCL-Nx sensitivity level.

**Defines**

- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_1](#) 0x31
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_2](#) 0x32
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_3](#) 0x33
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_4](#) 0x34

**6.227.1 Detailed Description**

Constants that are for setting the MindSensors ACCL-Nx sensitivity level.

**6.227.2 Define Documentation****6.227.2.1 #define ACCL\_SENSITIVITY\_LEVEL\_1 0x31**

The ACCL-Nx sensitivity level 1

**Examples:**[ex\\_SetACCLNxSensitivity.nxc.](#)

**6.227.2.2 #define ACCL\_SENSITIVITY\_LEVEL\_2 0x32**

The ACCL-Nx sensitivity level 2

**6.227.2.3 #define ACCL\_SENSITIVITY\_LEVEL\_3 0x33**

The ACCL-Nx sensitivity level 3

**6.227.2.4 #define ACCL\_SENSITIVITY\_LEVEL\_4 0x34**

The ACCL-Nx sensitivity level 4

**6.228 MindSensors PFMate constants**

Constants that are for use with the MindSensors PFMate device.

**Modules**

- [PFMate motor constants](#)  
*Constants that are for specifying PFMate motors.*
- [PFMate channel constants](#)  
*Constants that are for specifying PFMate channels.*

**Defines**

- #define [PFMATE\\_REG\\_CMD](#) 0x41
- #define [PFMATE\\_REG\\_CHANNEL](#) 0x42
- #define [PFMATE\\_REG\\_MOTORS](#) 0x43
- #define [PFMATE\\_REG\\_A\\_CMD](#) 0x44
- #define [PFMATE\\_REG\\_A\\_SPEED](#) 0x45
- #define [PFMATE\\_REG\\_B\\_CMD](#) 0x46
- #define [PFMATE\\_REG\\_B\\_SPEED](#) 0x47
- #define [PFMATE\\_CMD\\_GO](#) 0x47
- #define [PFMATE\\_CMD\\_RAW](#) 0x52

**6.228.1 Detailed Description**

Constants that are for use with the MindSensors PFMate device.

**6.228.2 Define Documentation****6.228.2.1 #define PFMATE\_CMD\_GO 0x47**

Send IR signal to IR receiver

**6.228.2.2 #define PFMATE\_CMD\_RAW 0x52**

Send raw IR signal to IR receiver

**6.228.2.3 #define PFMATE\_REG\_A\_CMD 0x44**

PF command for motor A? (PF\_CMD\_FLOAT, PF\_CMD\_FWD, PF\_CMD\_REV, PF\_CMD\_BRAKE)

**6.228.2.4 #define PFMATE\_REG\_A\_SPEED 0x45**

PF speed for motor A? (0-7)

**6.228.2.5 #define PFMATE\_REG\_B\_CMD 0x46**

PF command for motor B? (PF\_CMD\_FLOAT, PF\_CMD\_FWD, PF\_CMD\_REV, PF\_CMD\_BRAKE)

**6.228.2.6 #define PFMATE\_REG\_B\_SPEED 0x47**

PF speed for motor B? (0-7)

**6.228.2.7 #define PFMATE\_REG\_CHANNEL 0x42**

PF channel? 1, 2, 3, or 4

**6.228.2.8 #define PFMATE\_REG\_CMD 0x41**

PFMate command

**6.228.2.9 #define PFMATE\_REG\_MOTORS 0x43**

PF motors? (0 = both, 1 = A, 2 = B)

## 6.229 PFMate motor constants

Constants that are for specifying PFMate motors.

### Defines

- #define [PFMATE\\_MOTORS\\_BOTH](#) 0x00
- #define [PFMATE\\_MOTORS\\_A](#) 0x01
- #define [PFMATE\\_MOTORS\\_B](#) 0x02

### 6.229.1 Detailed Description

Constants that are for specifying PFMate motors.

### 6.229.2 Define Documentation

#### 6.229.2.1 #define PFMATE\_MOTORS\_A 0x01

Control only motor A

#### 6.229.2.2 #define PFMATE\_MOTORS\_B 0x02

Control only motor B

#### 6.229.2.3 #define PFMATE\_MOTORS\_BOTH 0x00

Control both motors

### Examples:

[ex\\_PFMate.nxc](#).

## 6.230 PFMate channel constants

Constants that are for specifying PFMate channels.

### Defines

- #define [PFMATE\\_CHANNEL\\_1](#) 1
- #define [PFMATE\\_CHANNEL\\_2](#) 2
- #define [PFMATE\\_CHANNEL\\_3](#) 3
- #define [PFMATE\\_CHANNEL\\_4](#) 4

### 6.230.1 Detailed Description

Constants that are for specifying PFMate channels.

### 6.230.2 Define Documentation

#### 6.230.2.1 #define PFMATE\_CHANNEL\_1 1

Power function channel 1

#### Examples:

[ex\\_PFMate.nxc](#).

#### 6.230.2.2 #define PFMATE\_CHANNEL\_2 2

Power function channel 2

#### 6.230.2.3 #define PFMATE\_CHANNEL\_3 3

Power function channel 3

#### 6.230.2.4 #define PFMATE\_CHANNEL\_4 4

Power function channel 4

## 6.231 MindSensors NXTServo constants

Constants that are for use with the MindSensors NXTServo device.

### Modules

- [MindSensors NXTServo registers](#)  
*NXTServo device register constants.*
- [MindSensors NXTServo position constants](#)  
*NXTServo device position constants.*
- [MindSensors NXTServo quick position constants](#)  
*NXTServo device quick position constants.*

- [MindSensors NXTServo servo numbers](#)

*NXTServo device servo number constants.*

- [MindSensors NXTServo commands](#)

*NXTServo device command constants.*

### 6.231.1 Detailed Description

Constants that are for use with the MindSensors NXTServo device.

## 6.232 MindSensors NXTServo registers

NXTServo device register constants.

### Defines

- #define [NXTSERVO\\_REG\\_VOLTAGE](#) 0x41
- #define [NXTSERVO\\_REG\\_CMD](#) 0x41
- #define [NXTSERVO\\_REG\\_S1\\_POS](#) 0x42
- #define [NXTSERVO\\_REG\\_S2\\_POS](#) 0x44
- #define [NXTSERVO\\_REG\\_S3\\_POS](#) 0x46
- #define [NXTSERVO\\_REG\\_S4\\_POS](#) 0x48
- #define [NXTSERVO\\_REG\\_S5\\_POS](#) 0x4A
- #define [NXTSERVO\\_REG\\_S6\\_POS](#) 0x4C
- #define [NXTSERVO\\_REG\\_S7\\_POS](#) 0x4E
- #define [NXTSERVO\\_REG\\_S8\\_POS](#) 0x50
- #define [NXTSERVO\\_REG\\_S1\\_SPEED](#) 0x52
- #define [NXTSERVO\\_REG\\_S2\\_SPEED](#) 0x53
- #define [NXTSERVO\\_REG\\_S3\\_SPEED](#) 0x54
- #define [NXTSERVO\\_REG\\_S4\\_SPEED](#) 0x55
- #define [NXTSERVO\\_REG\\_S5\\_SPEED](#) 0x56
- #define [NXTSERVO\\_REG\\_S6\\_SPEED](#) 0x57
- #define [NXTSERVO\\_REG\\_S7\\_SPEED](#) 0x58
- #define [NXTSERVO\\_REG\\_S8\\_SPEED](#) 0x59
- #define [NXTSERVO\\_REG\\_S1\\_QPOS](#) 0x5A
- #define [NXTSERVO\\_REG\\_S2\\_QPOS](#) 0x5B
- #define [NXTSERVO\\_REG\\_S3\\_QPOS](#) 0x5C
- #define [NXTSERVO\\_REG\\_S4\\_QPOS](#) 0x5D
- #define [NXTSERVO\\_REG\\_S5\\_QPOS](#) 0x5E

- #define `NXTSERVO_REG_S6_QPOS` 0x5F
- #define `NXTSERVO_REG_S7_QPOS` 0x60
- #define `NXTSERVO_REG_S8_QPOS` 0x61
- #define `NXTSERVO_EM_REG_CMD` 0x00
- #define `NXTSERVO_EM_REG_EEPROM_START` 0x21
- #define `NXTSERVO_EM_REG_EEPROM_END` 0xFF

### 6.232.1 Detailed Description

NXTServo device register constants.

### 6.232.2 Define Documentation

#### 6.232.2.1 #define `NXTSERVO_EM_REG_CMD` 0x00

NXTServo in macro edit mode command register.

#### 6.232.2.2 #define `NXTSERVO_EM_REG_EEPROM_END` 0xFF

NXTServo in macro edit mode EEPROM end register.

#### 6.232.2.3 #define `NXTSERVO_EM_REG_EEPROM_START` 0x21

NXTServo in macro edit mode EEPROM start register.

#### 6.232.2.4 #define `NXTSERVO_REG_CMD` 0x41

NXTServo command register. See [MindSensors NXTServo commands](#) group. (write only)

#### 6.232.2.5 #define `NXTSERVO_REG_S1_POS` 0x42

NXTServo servo 1 position register.

#### 6.232.2.6 #define `NXTSERVO_REG_S1_QPOS` 0x5A

NXTServo servo 1 quick position register. (write only)

**6.232.2.7 #define NXTSERVO\_REG\_S1\_SPEED 0x52**

NXTServo servo 1 speed register.

**6.232.2.8 #define NXTSERVO\_REG\_S2\_POS 0x44**

NXTServo servo 2 position register.

**6.232.2.9 #define NXTSERVO\_REG\_S2\_QPOS 0x5B**

NXTServo servo 2 quick position register. (write only)

**6.232.2.10 #define NXTSERVO\_REG\_S2\_SPEED 0x53**

NXTServo servo 2 speed register.

**6.232.2.11 #define NXTSERVO\_REG\_S3\_POS 0x46**

NXTServo servo 3 position register.

**6.232.2.12 #define NXTSERVO\_REG\_S3\_QPOS 0x5C**

NXTServo servo 3 quick position register. (write only)

**6.232.2.13 #define NXTSERVO\_REG\_S3\_SPEED 0x54**

NXTServo servo 3 speed register.

**6.232.2.14 #define NXTSERVO\_REG\_S4\_POS 0x48**

NXTServo servo 4 position register.

**6.232.2.15 #define NXTSERVO\_REG\_S4\_QPOS 0x5D**

NXTServo servo 4 quick position register. (write only)

**6.232.2.16 #define NXTSERVO\_REG\_S4\_SPEED 0x55**

NXTServo servo 4 speed register.

**6.232.2.17 #define NXTSERVO\_REG\_S5\_POS 0x4A**

NXTServo servo 5 position register.

**6.232.2.18 #define NXTSERVO\_REG\_S5\_QPOS 0x5E**

NXTServo servo 5 quick position register. (write only)

**6.232.2.19 #define NXTSERVO\_REG\_S5\_SPEED 0x56**

NXTServo servo 5 speed register.

**6.232.2.20 #define NXTSERVO\_REG\_S6\_POS 0x4C**

NXTServo servo 6 position register.

**6.232.2.21 #define NXTSERVO\_REG\_S6\_QPOS 0x5F**

NXTServo servo 6 quick position register. (write only)

**6.232.2.22 #define NXTSERVO\_REG\_S6\_SPEED 0x57**

NXTServo servo 6 speed register.

**6.232.2.23 #define NXTSERVO\_REG\_S7\_POS 0x4E**

NXTServo servo 7 position register.

**6.232.2.24 #define NXTSERVO\_REG\_S7\_QPOS 0x60**

NXTServo servo 7 quick position register. (write only)

**6.232.2.25 #define NXTSERVO\_REG\_S7\_SPEED 0x58**

NXTServo servo 7 speed register.

**6.232.2.26 #define NXTSERVO\_REG\_S8\_POS 0x50**

NXTServo servo 8 position register.

**6.232.2.27 #define NXTSERVO\_REG\_S8\_QPOS 0x61**

NXTServo servo 8 quick position register. (write only)

**6.232.2.28 #define NXTSERVO\_REG\_S8\_SPEED 0x59**

NXTServo servo 8 speed register.

**6.232.2.29 #define NXTSERVO\_REG\_VOLTAGE 0x41**

Battery voltage register. (read only)

**6.233 MindSensors NXTServo position constants**

NXTServo device position constants.

**Defines**

- #define [NXTSERVO\\_POS\\_CENTER](#) 1500
- #define [NXTSERVO\\_POS\\_MIN](#) 500
- #define [NXTSERVO\\_POS\\_MAX](#) 2500

**6.233.1 Detailed Description**

NXTServo device position constants.

**6.233.2 Define Documentation****6.233.2.1 #define NXTSERVO\_POS\_CENTER 1500**

Center position for 1500us servos.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.233.2.2 #define NXTSERVO\_POS\_MAX 2500**

Maximum position for 1500us servos.

**6.233.2.3 #define NXTSERVO\_POS\_MIN 500**

Minimum position for 1500us servos.

**6.234 MindSensors NXTServo quick position constants**

NXTServo device quick position constants.

**Defines**

- #define [NXTSERVO\\_QPOS\\_CENTER](#) 150
- #define [NXTSERVO\\_QPOS\\_MIN](#) 50
- #define [NXTSERVO\\_QPOS\\_MAX](#) 250

**6.234.1 Detailed Description**

NXTServo device quick position constants.

**6.234.2 Define Documentation****6.234.2.1 #define NXTSERVO\_QPOS\_CENTER 150**

Center quick position for 1500us servos.

**6.234.2.2 #define NXTSERVO\_QPOS\_MAX 250**

Maximum quick position for 1500us servos.

**6.234.2.3 #define NXTSERVO\_QPOS\_MIN 50**

Minimum quick position for 1500us servos.

**Examples:**

[ex\\_NXTServo.nxc](#).

**6.235 MindSensors NXTServo servo numbers**

NXTServo device servo number constants.

## Defines

- #define `NXTSERVO_SERVO_1` 0
- #define `NXTSERVO_SERVO_2` 1
- #define `NXTSERVO_SERVO_3` 2
- #define `NXTSERVO_SERVO_4` 3
- #define `NXTSERVO_SERVO_5` 4
- #define `NXTSERVO_SERVO_6` 5
- #define `NXTSERVO_SERVO_7` 6
- #define `NXTSERVO_SERVO_8` 7

### 6.235.1 Detailed Description

NXTServo device servo number constants.

### 6.235.2 Define Documentation

#### 6.235.2.1 #define `NXTSERVO_SERVO_1` 0

NXTServo server number 1.

#### Examples:

`ex_NXTServo.nxc`.

#### 6.235.2.2 #define `NXTSERVO_SERVO_2` 1

NXTServo server number 2.

#### 6.235.2.3 #define `NXTSERVO_SERVO_3` 2

NXTServo server number 3.

#### 6.235.2.4 #define `NXTSERVO_SERVO_4` 3

NXTServo server number 4.

#### 6.235.2.5 #define `NXTSERVO_SERVO_5` 4

NXTServo server number 5.

**6.235.2.6 #define NXTSERVO\_SERVO\_6 5**

NXTServo server number 6.

**6.235.2.7 #define NXTSERVO\_SERVO\_7 6**

NXTServo server number 7.

**6.235.2.8 #define NXTSERVO\_SERVO\_8 7**

NXTServo server number 8.

**6.236 MindSensors NXTServo commands**

NXTServo device command constants.

**Defines**

- #define [NXTSERVO\\_CMD\\_INIT](#) 0x49
- #define [NXTSERVO\\_CMD\\_RESET](#) 0x53
- #define [NXTSERVO\\_CMD\\_HALT](#) 0x48
- #define [NXTSERVO\\_CMD\\_RESUME](#) 0x52
- #define [NXTSERVO\\_CMD\\_GOTO](#) 0x47
- #define [NXTSERVO\\_CMD\\_PAUSE](#) 0x50
- #define [NXTSERVO\\_CMD\\_EDIT1](#) 0x45
- #define [NXTSERVO\\_CMD\\_EDIT2](#) 0x4D
- #define [NXTSERVO\\_EM\\_CMD\\_QUIT](#) 0x51

**6.236.1 Detailed Description**

NXTServo device command constants. These are written to the command register to control the device.

**6.236.2 Define Documentation****6.236.2.1 #define NXTSERVO\_CMD\_EDIT1 0x45**

Edit Macro (part 1 of 2 character command sequence)

**6.236.2.2 #define NXTSERVO\_CMD\_EDIT2 0x4D**

Edit Macro (part 2 of 2 character command sequence)

**6.236.2.3 #define NXTSERVO\_CMD\_GOTO 0x47**

Goto EEPROM position x. This command re-initializes the macro environment.

**6.236.2.4 #define NXTSERVO\_CMD\_HALT 0x48**

Halt Macro. This command re-initializes the macro environment.

**6.236.2.5 #define NXTSERVO\_CMD\_INIT 0x49**

Store the initial speed and position properties of the servo motor 'n'. Current speed and position values of the nth servo is read from the servo speed register and servo position register and written to permanent memory.

**6.236.2.6 #define NXTSERVO\_CMD\_PAUSE 0x50**

Pause Macro. This command will pause the macro, and save the environment for subsequent resumption.

**6.236.2.7 #define NXTSERVO\_CMD\_RESET 0x53**

Reset servo properties to factory default. Initial Position of servos to 1500, and speed to 0.

**6.236.2.8 #define NXTSERVO\_CMD\_RESUME 0x52**

Resume macro Execution. This command resumes macro where it was paused last, using the same environment.

**6.236.2.9 #define NXTSERVO\_EM\_CMD\_QUIT 0x51**

Exit edit macro mode

**6.237 MindSensors NXTHID constants**

Constants that are for use with the MindSensors NXTHID device.

## Modules

- [MindSensors NXTHID registers](#)  
*NXTHID device register constants.*
- [MindSensors NXTHID modifier keys](#)  
*NXTHID device modifier key constants.*
- [MindSensors NXTHID commands](#)  
*NXTHID device command constants.*

### 6.237.1 Detailed Description

Constants that are for use with the MindSensors NXTHID device.

## 6.238 MindSensors NXTHID registers

NXTHID device register constants.

### Defines

- #define [NXTHID\\_REG\\_CMD](#) 0x41
- #define [NXTHID\\_REG\\_MODIFIER](#) 0x42
- #define [NXTHID\\_REG\\_DATA](#) 0x43

### 6.238.1 Detailed Description

NXTHID device register constants.

### 6.238.2 Define Documentation

#### 6.238.2.1 #define NXTHID\_REG\_CMD 0x41

NXTHID command register. See [MindSensors NXTHID commands](#) group.

#### 6.238.2.2 #define NXTHID\_REG\_DATA 0x43

NXTHID data register.

### 6.238.2.3 #define NXTHID\_REG\_MODIFIER 0x42

NXTHID modifier register. See [MindSensors NXTHID modifier keys](#) group.

## 6.239 MindSensors NXTHID modifier keys

NXTHID device modifier key constants.

### Defines

- #define [NXTHID\\_MOD\\_NONE](#) 0x00
- #define [NXTHID\\_MOD\\_LEFT\\_CTRL](#) 0x01
- #define [NXTHID\\_MOD\\_LEFT\\_SHIFT](#) 0x02
- #define [NXTHID\\_MOD\\_LEFT\\_ALT](#) 0x04
- #define [NXTHID\\_MOD\\_LEFT\\_GUI](#) 0x08
- #define [NXTHID\\_MOD\\_RIGHT\\_CTRL](#) 0x10
- #define [NXTHID\\_MOD\\_RIGHT\\_SHIFT](#) 0x20
- #define [NXTHID\\_MOD\\_RIGHT\\_ALT](#) 0x40
- #define [NXTHID\\_MOD\\_RIGHT\\_GUI](#) 0x80

### 6.239.1 Detailed Description

NXTHID device modifier key constants.

### 6.239.2 Define Documentation

#### 6.239.2.1 #define NXTHID\_MOD\_LEFT\_ALT 0x04

NXTHID left alt modifier.

#### 6.239.2.2 #define NXTHID\_MOD\_LEFT\_CTRL 0x01

NXTHID left control modifier.

### Examples:

[ex\\_NXTHID.nxc](#).

#### 6.239.2.3 #define NXTHID\_MOD\_LEFT\_GUI 0x08

NXTHID left gui modifier.

**6.239.2.4 #define NXTHID\_MOD\_LEFT\_SHIFT 0x02**

NXTHID left shift modifier.

**6.239.2.5 #define NXTHID\_MOD\_NONE 0x00**

NXTHID no modifier.

**Examples:**

[ex\\_NXTHID.nxc](#).

**6.239.2.6 #define NXTHID\_MOD\_RIGHT\_ALT 0x40**

NXTHID right alt modifier.

**6.239.2.7 #define NXTHID\_MOD\_RIGHT\_CTRL 0x10**

NXTHID right control modifier.

**6.239.2.8 #define NXTHID\_MOD\_RIGHT\_GUI 0x80**

NXTHID right gui modifier.

**6.239.2.9 #define NXTHID\_MOD\_RIGHT\_SHIFT 0x20**

NXTHID right shift modifier.

**6.240 MindSensors NXTHID commands**

NXTHID device command constants.

**Defines**

- #define [NXTHID\\_CMD\\_ASCII](#) 0x41
- #define [NXTHID\\_CMD\\_DIRECT](#) 0x44
- #define [NXTHID\\_CMD\\_TRANSMIT](#) 0x54

### 6.240.1 Detailed Description

NXTHID device command constants. These are written to the command register to control the device.

### 6.240.2 Define Documentation

#### 6.240.2.1 #define NXTHID\_CMD\_ASCII 0x41

Use ASCII data mode. In ASCII mode no non-printable characters can be sent.

#### 6.240.2.2 #define NXTHID\_CMD\_DIRECT 0x44

Use direct data mode In direct mode any character can be sent.

#### 6.240.2.3 #define NXTHID\_CMD\_TRANSMIT 0x54

Transmit data to the host computer.

## 6.241 MindSensors NXTPowerMeter constants

Constants that are for use with the MindSensors NXTPowerMeter device.

### Modules

- [MindSensors NXTPowerMeter registers](#)  
*NXTPowerMeter device register constants.*
- [MindSensors NXTPowerMeter commands](#)  
*NXTPowerMeter device command constants.*

### 6.241.1 Detailed Description

Constants that are for use with the MindSensors NXTPowerMeter device.

## 6.242 MindSensors NXTPowerMeter registers

NXTPowerMeter device register constants.

## Defines

- #define [NXTPM\\_REG\\_CMD](#) 0x41
- #define [NXTPM\\_REG\\_CURRENT](#) 0x42
- #define [NXTPM\\_REG\\_VOLTAGE](#) 0x44
- #define [NXTPM\\_REG\\_CAPACITY](#) 0x46
- #define [NXTPM\\_REG\\_POWER](#) 0x48
- #define [NXTPM\\_REG\\_TOTALPOWER](#) 0x4A
- #define [NXTPM\\_REG\\_MAXCURRENT](#) 0x4E
- #define [NXTPM\\_REG\\_MINCURRENT](#) 0x50
- #define [NXTPM\\_REG\\_MAXVOLTAGE](#) 0x52
- #define [NXTPM\\_REG\\_MINVOLTAGE](#) 0x54
- #define [NXTPM\\_REG\\_TIME](#) 0x56
- #define [NXTPM\\_REG\\_USERGAIN](#) 0x5A
- #define [NXTPM\\_REG\\_GAIN](#) 0x5E
- #define [NXTPM\\_REG\\_ERRORCOUNT](#) 0x5F

### 6.242.1 Detailed Description

NXTPowerMeter device register constants.

### 6.242.2 Define Documentation

#### 6.242.2.1 #define [NXTPM\\_REG\\_CAPACITY](#) 0x46

NXTPowerMeter capacity used since last reset register. (2 bytes)

#### 6.242.2.2 #define [NXTPM\\_REG\\_CMD](#) 0x41

NXTPowerMeter command register. See the [MindSensors NXTPowerMeter commands](#) group.

#### 6.242.2.3 #define [NXTPM\\_REG\\_CURRENT](#) 0x42

NXTPowerMeter present current in mA register. (2 bytes)

#### 6.242.2.4 #define [NXTPM\\_REG\\_ERRORCOUNT](#) 0x5F

NXTPowerMeter error count register. (2 bytes)

**6.242.2.5 #define NXTPM\_REG\_GAIN 0x5E**

NXTPowerMeter gain register. (1 byte)

**6.242.2.6 #define NXTPM\_REG\_MAXCURRENT 0x4E**

NXTPowerMeter max current register. (2 bytes)

**6.242.2.7 #define NXTPM\_REG\_MAXVOLTAGE 0x52**

NXTPowerMeter max voltage register. (2 bytes)

**6.242.2.8 #define NXTPM\_REG\_MINCURRENT 0x50**

NXTPowerMeter min current register. (2 bytes)

**6.242.2.9 #define NXTPM\_REG\_MINVOLTAGE 0x54**

NXTPowerMeter min voltage register. (2 bytes)

**6.242.2.10 #define NXTPM\_REG\_POWER 0x48**

NXTPowerMeter present power register. (2 bytes)

**6.242.2.11 #define NXTPM\_REG\_TIME 0x56**

NXTPowerMeter time register. (4 bytes)

**6.242.2.12 #define NXTPM\_REG\_TOTALPOWER 0x4A**

NXTPowerMeter total power consumed since last reset register. (4 bytes)

**6.242.2.13 #define NXTPM\_REG\_USERGAIN 0x5A**

NXTPowerMeter user gain register. Not yet implemented. (4 bytes)

**6.242.2.14 #define NXTPM\_REG\_VOLTAGE 0x44**

NXTPowerMeter present voltage in mV register. (2 bytes)

## 6.243 MindSensors NXTPowerMeter commands

NXTPowerMeter device command constants.

### Defines

- `#define NXTPM_CMD_RESET 0x52`

#### 6.243.1 Detailed Description

NXTPowerMeter device command constants. These are written to the command register to control the device.

#### 6.243.2 Define Documentation

##### 6.243.2.1 `#define NXTPM_CMD_RESET 0x52`

Reset counters.

## 6.244 MindSensors NXTSumoEyes constants

Constants that are for use with the MindSensors NXTSumoEyes device.

### Defines

- `#define NXTSE_ZONE_NONE 0`
- `#define NXTSE_ZONE_FRONT 1`
- `#define NXTSE_ZONE_LEFT 2`
- `#define NXTSE_ZONE_RIGHT 3`

#### 6.244.1 Detailed Description

Constants that are for use with the MindSensors NXTSumoEyes device.

#### 6.244.2 Define Documentation

##### 6.244.2.1 `#define NXTSE_ZONE_FRONT 1`

Obstacle zone front.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.244.2.2 #define NXTSE\_ZONE\_LEFT 2**

Obstacle zone left.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.244.2.3 #define NXTSE\_ZONE\_NONE 0**

Obstacle zone none.

**6.244.2.4 #define NXTSE\_ZONE\_RIGHT 3**

Obstacle zone right.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**6.245 MindSensors NXTLineLeader constants**

Constants that are for use with the MindSensors NXTLineLeader device.

**Modules**

- [MindSensors NXTLineLeader registers](#)  
*NXTLineLeader device register constants.*
- [MindSensors NXTLineLeader commands](#)  
*NXTLineLeader device command constants.*

**6.245.1 Detailed Description**

Constants that are for use with the MindSensors NXTLineLeader device.

## 6.246 MindSensors NXTLineLeader registers

NXTLineLeader device register constants.

### Defines

- #define `NXTLL_REG_CMD` 0x41
- #define `NXTLL_REG_STEERING` 0x42
- #define `NXTLL_REG_AVERAGE` 0x43
- #define `NXTLL_REG_RESULT` 0x44
- #define `NXTLL_REG_SETPOINT` 0x45
- #define `NXTLL_REG_KP_VALUE` 0x46
- #define `NXTLL_REG_KI_VALUE` 0x47
- #define `NXTLL_REG_KD_VALUE` 0x48
- #define `NXTLL_REG_CALIBRATED` 0x49
- #define `NXTLL_REG_WHITELIMITS` 0x51
- #define `NXTLL_REG_BLACKLIMITS` 0x59
- #define `NXTLL_REG_KP_FACTOR` 0x61
- #define `NXTLL_REG_KI_FACTOR` 0x62
- #define `NXTLL_REG_KD_FACTOR` 0x63
- #define `NXTLL_REG_WHITEDATA` 0x64
- #define `NXTLL_REG_BLACKDATA` 0x6C
- #define `NXTLL_REG_RAWVOLTAGE` 0x74

### 6.246.1 Detailed Description

NXTLineLeader device register constants.

### 6.246.2 Define Documentation

#### 6.246.2.1 #define `NXTLL_REG_AVERAGE` 0x43

NXTLineLeader average result register.

#### 6.246.2.2 #define `NXTLL_REG_BLACKDATA` 0x6C

NXTLineLeader black calibration data registers. 8 bytes.

#### 6.246.2.3 #define `NXTLL_REG_BLACKLIMITS` 0x59

NXTLineLeader black limit registers. 8 bytes.

**6.246.2.4 #define NXTLL\_REG\_CALIBRATED 0x49**

NXTLineLeader calibrated sensor reading registers. 8 bytes.

**6.246.2.5 #define NXTLL\_REG\_CMD 0x41**

NXTLineLeader command register. See the [MindSensors NXTLineLeader commands](#) group.

**6.246.2.6 #define NXTLL\_REG\_KD\_FACTOR 0x63**

NXTLineLeader Kd factor register. Default = 32.

**6.246.2.7 #define NXTLL\_REG\_KD\_VALUE 0x48**

NXTLineLeader Kd value register. Default = 8.

**6.246.2.8 #define NXTLL\_REG\_KI\_FACTOR 0x62**

NXTLineLeader Ki factor register. Default = 32.

**6.246.2.9 #define NXTLL\_REG\_KI\_VALUE 0x47**

NXTLineLeader Ki value register. Default = 0.

**6.246.2.10 #define NXTLL\_REG\_KP\_FACTOR 0x61**

NXTLineLeader Kp factor register. Default = 32.

**6.246.2.11 #define NXTLL\_REG\_KP\_VALUE 0x46**

NXTLineLeader Kp value register. Default = 25.

**6.246.2.12 #define NXTLL\_REG\_RAWVOLTAGE 0x74**

NXTLineLeader uncalibrated sensor voltage registers. 16 bytes.

**6.246.2.13 #define NXTLL\_REG\_RESULT 0x44**

NXTLineLeader result register (sensor bit values).

**6.246.2.14 #define NXTLL\_REG\_SETPOINT 0x45**

NXTLineLeader user settable average (setpoint) register. Default = 45.

**6.246.2.15 #define NXTLL\_REG\_STEERING 0x42**

NXTLineLeader steering register.

**6.246.2.16 #define NXTLL\_REG\_WHITEDATA 0x64**

NXTLineLeader white calibration data registers. 8 bytes.

**6.246.2.17 #define NXTLL\_REG\_WHITELIMITS 0x51**

NXTLineLeader white limit registers. 8 bytes.

**6.247 MindSensors NXTLineLeader commands**

NXTLineLeader device command constants.

**Defines**

- #define [NXTLL\\_CMD\\_USA](#) 0x41
- #define [NXTLL\\_CMD\\_BLACK](#) 0x42
- #define [NXTLL\\_CMD\\_POWERDOWN](#) 0x44
- #define [NXTLL\\_CMD\\_EUROPEAN](#) 0x45
- #define [NXTLL\\_CMD\\_INVERT](#) 0x49
- #define [NXTLL\\_CMD\\_POWERUP](#) 0x50
- #define [NXTLL\\_CMD\\_RESET](#) 0x52
- #define [NXTLL\\_CMD\\_SNAPSHOT](#) 0x53
- #define [NXTLL\\_CMD\\_UNIVERSAL](#) 0x55
- #define [NXTLL\\_CMD\\_WHITE](#) 0x57

**6.247.1 Detailed Description**

NXTLineLeader device command constants. These are written to the command register to control the device.

**6.247.2 Define Documentation****6.247.2.1 #define NXTLL\_CMD\_BLACK 0x42**

Black calibration.

**6.247.2.2 #define NXTLL\_CMD\_EUROPEAN 0x45**

European power frequency. (50hz)

**6.247.2.3 #define NXTLL\_CMD\_INVERT 0x49**

Invert color.

**6.247.2.4 #define NXTLL\_CMD\_POWERDOWN 0x44**

Power down the device.

**6.247.2.5 #define NXTLL\_CMD\_POWERUP 0x50**

Power up the device.

**6.247.2.6 #define NXTLL\_CMD\_RESET 0x52**

Reset inversion.

**6.247.2.7 #define NXTLL\_CMD\_SNAPSHOT 0x53**

Setpoint based on snapshot (automatically sets invert if needed).

**6.247.2.8 #define NXTLL\_CMD\_UNIVERSAL 0x55**

Universal power frequency. The sensor auto adjusts for any frequency. This is the default mode.

**6.247.2.9 #define NXTLL\_CMD\_USA 0x41**

USA power frequency. (60hz)

**6.247.2.10 #define NXTLL\_CMD\_WHITE 0x57**

White balance calibration.

**6.248 Codatex device constants**

Constants that are for use with Codatex devices.

**Modules**

- [Codatex RFID sensor constants](#)

*Constants that are for use with the Codatex RFID sensor device.*

**6.248.1 Detailed Description**

Constants that are for use with Codatex devices.

**6.249 Codatex RFID sensor constants**

Constants that are for use with the Codatex RFID sensor device.

**Modules**

- [Codatex RFID sensor modes](#)

*Constants that are for configuring the Codatex RFID sensor mode.*

**Defines**

- #define [CT\\_ADDR\\_RFID](#) 0x04
- #define [CT\\_REG\\_STATUS](#) 0x32
- #define [CT\\_REG\\_MODE](#) 0x41
- #define [CT\\_REG\\_DATA](#) 0x42

**6.249.1 Detailed Description**

Constants that are for use with the Codatex RFID sensor device.

### 6.249.2 Define Documentation

#### 6.249.2.1 #define CT\_ADDR\_RFID 0x04

RFID I2C address

#### 6.249.2.2 #define CT\_REG\_DATA 0x42

RFID data register

#### 6.249.2.3 #define CT\_REG\_MODE 0x41

RFID mode register

#### 6.249.2.4 #define CT\_REG\_STATUS 0x32

RFID status register

## 6.250 Codatex RFID sensor modes

Constants that are for configuring the Codatex RFID sensor mode.

### Defines

- #define [RFID\\_MODE\\_STOP](#) 0
- #define [RFID\\_MODE\\_SINGLE](#) 1
- #define [RFID\\_MODE\\_CONTINUOUS](#) 2

### 6.250.1 Detailed Description

Constants that are for configuring the Codatex RFID sensor mode.

### 6.250.2 Define Documentation

#### 6.250.2.1 #define RFID\_MODE\_CONTINUOUS 2

Configure the RFID device for continuous reading

### Examples:

[ex\\_RFIDMode.nxc](#).

**6.250.2.2 #define RFID\_MODE\_SINGLE 1**

Configure the RFID device for a single reading

**6.250.2.3 #define RFID\_MODE\_STOP 0**

Stop the RFID device

**6.251 Dexter Industries device constants**

Constants that are for use with Dexter Industries devices.

**Modules**

- [Dexter Industries GPS sensor constants](#)  
*Constants that are for use with the Dexter Industries GPS sensor.*
- [Dexter Industries IMU sensor constants](#)  
*Constants that are for use with the Dexter Industries IMU sensor.*

**6.251.1 Detailed Description**

Constants that are for use with Dexter Industries devices.

**6.252 Dexter Industries GPS sensor constants**

Constants that are for use with the Dexter Industries GPS sensor.

**Defines**

- #define [DI\\_ADDR\\_DGPS](#) 0x06
- #define [DGPS\\_REG\\_TIME](#) 0x00
- #define [DGPS\\_REG\\_STATUS](#) 0x01
- #define [DGPS\\_REG\\_LATITUDE](#) 0x02
- #define [DGPS\\_REG\\_LONGITUDE](#) 0x04
- #define [DGPS\\_REG\\_VELOCITY](#) 0x06
- #define [DGPS\\_REG\\_HEADING](#) 0x07
- #define [DGPS\\_REG\\_DISTANCE](#) 0x08
- #define [DGPS\\_REG\\_WAYANGLE](#) 0x09

- #define DGPS\_REG\_LASTANGLE 0x0A
- #define DGPS\_REG\_SETLATITUDE 0x0B
- #define DGPS\_REG\_SETLONGITUDE 0x0C

### 6.252.1 Detailed Description

Constants that are for use with the Dexter Industries GPS sensor.

### 6.252.2 Define Documentation

#### 6.252.2.1 #define DGPS\_REG\_DISTANCE 0x08

Read distance to current waypoint in meters.

#### 6.252.2.2 #define DGPS\_REG\_HEADING 0x07

Read heading in degrees.

#### 6.252.2.3 #define DGPS\_REG\_LASTANGLE 0x0A

Read angle travelled since last request, resets the request coordinates on the GPS sensor, sends the angle of travel since last reset.

#### 6.252.2.4 #define DGPS\_REG\_LATITUDE 0x02

Read integer latitude.(ddddddd; Positive = North; Negative = South).

#### 6.252.2.5 #define DGPS\_REG\_LONGITUDE 0x04

Read integer longitude (ddddddd; Positive = East; Negative = West).

#### 6.252.2.6 #define DGPS\_REG\_SETLATITUDE 0x0B

Set waypoint latitude as a 4 byte integer.

#### 6.252.2.7 #define DGPS\_REG\_SETLONGITUDE 0x0C

Set waypoint longitude as a 4 byte integer.

**6.252.2.8 #define DGPS\_REG\_STATUS 0x01**

Read status of the GPS (0 - invalid signal, 1 - valid signal).

**6.252.2.9 #define DGPS\_REG\_TIME 0x00**

Read time in UTC (hhmmss).

**6.252.2.10 #define DGPS\_REG\_VELOCITY 0x06**

Read velocity in cm/s.

**6.252.2.11 #define DGPS\_REG\_WAYANGLE 0x09**

Read angle to current waypoint in degrees.

**6.252.2.12 #define DI\_ADDR\_DGPS 0x06**

Dexter Industries DGPS I2C address

**6.253 Dexter Industries IMU sensor constants**

Constants that are for use with the Dexter Industries IMU sensor.

**Modules**

- [Dexter Industries IMU Gyro register constants](#)  
*Constants that define the Dexter Industries IMU Gyro registers.*
- [Dexter Industries IMU Gyro control register 1 constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 1.*
- [Dexter Industries IMU Gyro control register 2 constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 2.*
- [Dexter Industries IMU Gyro control register 3 constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 3.*

- [Dexter Industries IMU Gyro control register 4 constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 4.*
- [Dexter Industries IMU Gyro control register 5 constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 5.*
- [Dexter Industries IMU Gyro FIFO control register onstants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's FIFO control register.*
- [Dexter Industries IMU Gyro status register constants](#)  
*Constants that are for use with the Dexter Industries IMU Gyro sensor's status register.*
- [Dexter Industries IMU Accelerometer register constants](#)  
*Constants that define the Dexter Industries IMU Accelerometer registers.*
- [Dexter Industries IMU Accelerometer status register constants](#)  
*Constants that are for use with the Dexter Industries IMU Accelerometer sensor's status register.*
- [Dexter Industries IMU Accelerometer mode control register constants](#)  
*Constants that are for use with the Dexter Industries IMU Accelerometer sensor's mode control register.*
- [Dexter Industries IMU Accelerometer interrupt latch reset register constants](#)  
*Constants that are for use with the Dexter Industries IMU Accelerometer sensor's interrupt latch reset register.*
- [Dexter Industries IMU Accelerometer control register 1 constants](#)  
*Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 1.*
- [Dexter Industries IMU Accelerometer control register 2 constants](#)  
*Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 2.*

### Defines

- `#define DI_ADDR_GYRO 0xD2`
- `#define DI_ADDR_ACCL 0x3A`

### 6.253.1 Detailed Description

Constants that are for use with the Dexter Industries IMU sensor.

### 6.253.2 Define Documentation

#### 6.253.2.1 #define DI\_ADDR\_ACCL 0x3A

Dexter Industries DIMU Accelerometer I2C address

#### 6.253.2.2 #define DI\_ADDR\_GYRO 0xD2

Dexter Industries DIMU Gyro I2C address

## 6.254 Dexter Industries IMU Gyro register constants

Constants that define the Dexter Industries IMU Gyro registers.

### Defines

- #define DIGYRO\_REG\_WHOAMI 0x0F
- #define DIGYRO\_REG\_CTRL1 0x20
- #define DIGYRO\_REG\_CTRL2 0x21
- #define DIGYRO\_REG\_CTRL3 0x22
- #define DIGYRO\_REG\_CTRL4 0x23
- #define DIGYRO\_REG\_CTRL5 0x24
- #define DIGYRO\_REG\_REFERENCE 0x25
- #define DIGYRO\_REG\_OUTTEMP 0x26
- #define DIGYRO\_REG\_STATUS 0x27
- #define DIGYRO\_REG\_XLOW 0x28
- #define DIGYRO\_REG\_XHIGH 0x29
- #define DIGYRO\_REG\_YLOW 0x2A
- #define DIGYRO\_REG\_YHIGH 0x2B
- #define DIGYRO\_REG\_ZLOW 0x2C
- #define DIGYRO\_REG\_ZHIGH 0x2D
- #define DIGYRO\_REG\_FIFOCTRL 0x2E
- #define DIGYRO\_REG\_FIFOSRC 0x2F
- #define DIGYRO\_REG\_INT1\_CFG 0x30
- #define DIGYRO\_REG\_INT1\_SRC 0x31
- #define DIGYRO\_REG\_INT1\_XHI 0x32
- #define DIGYRO\_REG\_INT1\_XLO 0x33

- #define DIGYRO\_REG\_INT1\_YHI 0x34
- #define DIGYRO\_REG\_INT1\_YLO 0x35
- #define DIGYRO\_REG\_INT1\_ZHI 0x36
- #define DIGYRO\_REG\_INT1\_ZLO 0x37
- #define DIGYRO\_REG\_INT1\_DUR 0x38
- #define DIGYRO\_REG\_CTRL1AUTO 0xA0
- #define DIGYRO\_REG\_TEMPAUTO 0xA6
- #define DIGYRO\_REG\_XLOWBURST 0xA8
- #define DIGYRO\_REG\_YLOWBURST 0xAA
- #define DIGYRO\_REG\_ZLOWBURST 0xAC

### 6.254.1 Detailed Description

Constants that define the Dexter Industries IMU Gyro registers.

### 6.254.2 Define Documentation

#### 6.254.2.1 #define DIGYRO\_REG\_CTRL1 0x20

Gyro control register 1

#### 6.254.2.2 #define DIGYRO\_REG\_CTRL1AUTO 0xA0

Gyro control register 1 - auto increment write

#### 6.254.2.3 #define DIGYRO\_REG\_CTRL2 0x21

Gyro control register 2

#### 6.254.2.4 #define DIGYRO\_REG\_CTRL3 0x22

Gyro control register 3

#### 6.254.2.5 #define DIGYRO\_REG\_CTRL4 0x23

Gyro control register 4

#### 6.254.2.6 #define DIGYRO\_REG\_CTRL5 0x24

Gyro control register 5

**6.254.2.7 #define DIGYRO\_REG\_FIFOCTRL 0x2E**

Gyro FIFO control register

**6.254.2.8 #define DIGYRO\_REG\_FIFOSRC 0x2F**

Gyro FIFO source register (read only)

**6.254.2.9 #define DIGYRO\_REG\_INT1\_CFG 0x30**

Gyro interrupt 1 config register

**6.254.2.10 #define DIGYRO\_REG\_INT1\_DUR 0x38**

Gyro interrupt 1 duration register

**6.254.2.11 #define DIGYRO\_REG\_INT1\_SRC 0x31**

Gyro interrupt 1 source register

**6.254.2.12 #define DIGYRO\_REG\_INT1\_XHI 0x32**

Gyro interrupt 1 x-axis high threshold register

**6.254.2.13 #define DIGYRO\_REG\_INT1\_XLO 0x33**

Gyro interrupt 1 x-axis low threshold register

**6.254.2.14 #define DIGYRO\_REG\_INT1\_YHI 0x34**

Gyro interrupt 1 y-axis high threshold register

**6.254.2.15 #define DIGYRO\_REG\_INT1\_YLO 0x35**

Gyro interrupt 1 y-axis low threshold register

**6.254.2.16 #define DIGYRO\_REG\_INT1\_ZHI 0x36**

Gyro interrupt 1 z-axis high threshold register

**6.254.2.17 #define DIGYRO\_REG\_INT1\_ZLO 0x37**

Gyro interrupt 1 z-axis low threshold register

**6.254.2.18 #define DIGYRO\_REG\_OUTTEMP 0x26**

Gyro temperature register (read only) - stores temperature data

**6.254.2.19 #define DIGYRO\_REG\_REFERENCE 0x25**

Gyro reference register - stores the reference value used for interrupt generation

**6.254.2.20 #define DIGYRO\_REG\_STATUS 0x27**

Gyro status register (read only)

**6.254.2.21 #define DIGYRO\_REG\_TEMPAUTO 0xA6**

Gyro temperature register - read burst mode (read only)

**6.254.2.22 #define DIGYRO\_REG\_WHOAMI 0x0F**

Gyro device identification register (read only)

**6.254.2.23 #define DIGYRO\_REG\_XHIGH 0x29**

Gyro x-axis high byte register (read only)

**6.254.2.24 #define DIGYRO\_REG\_XLOW 0x28**

Gyro x-axis low byte register (read only)

**6.254.2.25 #define DIGYRO\_REG\_XLOWBURST 0xA8**

Gyro x-axis low byte register - read burst mode (read only)

**6.254.2.26 #define DIGYRO\_REG\_YHIGH 0x2B**

Gyro y-axis high byte register (read only)

**6.254.2.27 #define DIGYRO\_REG\_YLOW 0x2A**

Gyro y-axis low byte register (read only)

**6.254.2.28 #define DIGYRO\_REG\_YLOWBURST 0xAA**

Gyro y-axis low byte register - read burst mode (read only)

**6.254.2.29 #define DIGYRO\_REG\_ZHIGH 0x2D**

Gyro z-axis high byte register (read only)

**6.254.2.30 #define DIGYRO\_REG\_ZLOW 0x2C**

Gyro z-axis low byte register (read only)

**6.254.2.31 #define DIGYRO\_REG\_ZLOWBURST 0xAC**

Gyro y-axis low byte register - read burst mode (read only)

**6.255 Dexter Industries IMU Gyro control register 1 constants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 1.

**Defines**

- #define [DIGYRO\\_CTRL1\\_XENABLE](#) 0x01
- #define [DIGYRO\\_CTRL1\\_YENABLE](#) 0x02
- #define [DIGYRO\\_CTRL1\\_ZENABLE](#) 0x04
- #define [DIGYRO\\_CTRL1\\_POWERDOWN](#) 0x00
- #define [DIGYRO\\_CTRL1\\_NORMAL](#) 0x08
- #define [DIGYRO\\_CTRL1\\_BANDWIDTH\\_1](#) 0x00
- #define [DIGYRO\\_CTRL1\\_BANDWIDTH\\_2](#) 0x10
- #define [DIGYRO\\_CTRL1\\_BANDWIDTH\\_3](#) 0x20
- #define [DIGYRO\\_CTRL1\\_BANDWIDTH\\_4](#) 0x30
- #define [DIGYRO\\_CTRL1\\_DATARATE\\_100](#) 0x00
- #define [DIGYRO\\_CTRL1\\_DATARATE\\_200](#) 0x40
- #define [DIGYRO\\_CTRL1\\_DATARATE\\_400](#) 0x80
- #define [DIGYRO\\_CTRL1\\_DATARATE\\_800](#) 0xC0

### 6.255.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 1.

### 6.255.2 Define Documentation

#### 6.255.2.1 #define DIGYRO\_CTRL1\_BANDWIDTH\_1 0x00

Gyro LPF2 cut-off frequency bandwidth level 1 (12.5hz, 12.5hz, 20hz, 30hz)

#### 6.255.2.2 #define DIGYRO\_CTRL1\_BANDWIDTH\_2 0x10

Gyro LPF2 cut-off frequency bandwidth level 2 (12.5hz, 25hz, 50hz, 70hz)

#### 6.255.2.3 #define DIGYRO\_CTRL1\_BANDWIDTH\_3 0x20

Gyro LPF2 cut-off frequency bandwidth level 3 (20hz, 25hz, 50hz, 110hz)

#### 6.255.2.4 #define DIGYRO\_CTRL1\_BANDWIDTH\_4 0x30

Gyro LPF2 cut-off frequency bandwidth level 4 (30hz, 35hz, 50hz, 110hz)

### Examples:

[ex\\_digyro.nxc](#).

#### 6.255.2.5 #define DIGYRO\_CTRL1\_DATARATE\_100 0x00

Gyro output data rate 100 hz

#### 6.255.2.6 #define DIGYRO\_CTRL1\_DATARATE\_200 0x40

Gyro output data rate 200 hz

#### 6.255.2.7 #define DIGYRO\_CTRL1\_DATARATE\_400 0x80

Gyro output data rate 400 hz

**6.255.2.8 #define DIGYRO\_CTRL1\_DATARATE\_800 0xC0**

Gyro output data rate 800 hz

**Examples:**[ex\\_digyro.nxc](#).**6.255.2.9 #define DIGYRO\_CTRL1\_NORMAL 0x08**

Gyro disable power down mode

**6.255.2.10 #define DIGYRO\_CTRL1\_POWERDOWN 0x00**

Gyro enable power down mode

**6.255.2.11 #define DIGYRO\_CTRL1\_XENABLE 0x01**

Gyro enable X axis

**6.255.2.12 #define DIGYRO\_CTRL1\_YENABLE 0x02**

Gyro enable Y axis

**6.255.2.13 #define DIGYRO\_CTRL1\_ZENABLE 0x04**

Gyro enable Z axis

**6.256 Dexter Industries IMU Gyro control register 2 constants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 2.

**Defines**

- #define [DIGYRO\\_CTRL2\\_CUTOFF\\_FREQ\\_8](#) 0x00
- #define [DIGYRO\\_CTRL2\\_CUTOFF\\_FREQ\\_4](#) 0x01
- #define [DIGYRO\\_CTRL2\\_CUTOFF\\_FREQ\\_2](#) 0x02
- #define [DIGYRO\\_CTRL2\\_CUTOFF\\_FREQ\\_1](#) 0x03
- #define [DIGYRO\\_CTRL2\\_CUTOFF\\_FREQ\\_05](#) 0x04

- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_02 0x05
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_01 0x06
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_005 0x07
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_002 0x08
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_001 0x09
- #define DIGYRO\_CTRL2\_HPMODE\_RESET 0x00
- #define DIGYRO\_CTRL2\_HPMODE\_REFSIG 0x10
- #define DIGYRO\_CTRL2\_HPMODE\_NORMAL 0x20
- #define DIGYRO\_CTRL2\_HPMODE\_AUTOINT 0x30

### 6.256.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 2.

### 6.256.2 Define Documentation

#### 6.256.2.1 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_001 0x09

Gyro high pass filter cutoff frequency 0.01 hz

#### 6.256.2.2 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_002 0x08

Gyro high pass filter cutoff frequency 0.02 hz

#### 6.256.2.3 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_005 0x07

Gyro high pass filter cutoff frequency 0.05 hz

#### 6.256.2.4 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_01 0x06

Gyro high pass filter cutoff frequency 0.1 hz

#### 6.256.2.5 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_02 0x05

Gyro high pass filter cutoff frequency 0.2 hz

#### 6.256.2.6 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_05 0x04

Gyro high pass filter cutoff frequency 0.5 hz

**6.256.2.7 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_1 0x03**

Gyro high pass filter cutoff frequency 1 hz

**6.256.2.8 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_2 0x02**

Gyro high pass filter cutoff frequency 2 hz

**6.256.2.9 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_4 0x01**

Gyro high pass filter cutoff frequency 4 hz

**6.256.2.10 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_8 0x00**

Gyro high pass filter cutoff frequency 8 hz

**6.256.2.11 #define DIGYRO\_CTRL2\_HPMODE\_AUTOINT 0x30**

Gyro high pass filter autoreset on interrupt event mode

**6.256.2.12 #define DIGYRO\_CTRL2\_HPMODE\_NORMAL 0x20**

Gyro high pass filter normal mode

**6.256.2.13 #define DIGYRO\_CTRL2\_HPMODE\_REFSIG 0x10**

Gyro high pass filter reference signal mode

**6.256.2.14 #define DIGYRO\_CTRL2\_HPMODE\_RESET 0x00**

Gyro high pass filter reset mode

**6.257 Dexter Industries IMU Gyro control register 3 constants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 3.

## Defines

- #define DIGYRO\_CTRL3\_INT1\_ENABLE 0x80
- #define DIGYRO\_CTRL3\_INT1\_BOOT 0x40
- #define DIGYRO\_CTRL3\_INT1\_LOWACTIVE 0x20
- #define DIGYRO\_CTRL3\_OPENDRAIN 0x10
- #define DIGYRO\_CTRL3\_INT2\_DATAREADY 0x08
- #define DIGYRO\_CTRL3\_INT2\_WATERMARK 0x04
- #define DIGYRO\_CTRL3\_INT2\_OVERRUN 0x02
- #define DIGYRO\_CTRL3\_INT2\_EMPTY 0x01

### 6.257.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 3.

### 6.257.2 Define Documentation

#### 6.257.2.1 #define DIGYRO\_CTRL3\_INT1\_BOOT 0x40

Gyro boot status available on INT1

#### 6.257.2.2 #define DIGYRO\_CTRL3\_INT1\_ENABLE 0x80

Gyro interrupt enable on INT1 pin

#### 6.257.2.3 #define DIGYRO\_CTRL3\_INT1\_LOWACTIVE 0x20

Gyro interrupt active low on INT1

#### 6.257.2.4 #define DIGYRO\_CTRL3\_INT2\_DATAREADY 0x08

Gyro data ready on DRDY/INT2

#### 6.257.2.5 #define DIGYRO\_CTRL3\_INT2\_EMPTY 0x01

Gyro FIFO empty interrupt on DRDY/INT2

#### 6.257.2.6 #define DIGYRO\_CTRL3\_INT2\_OVERRUN 0x02

Gyro FIFO overrun interrupt on DRDY/INT2

**6.257.2.7 #define DIGYRO\_CTRL3\_INT2\_WATERMARK 0x04**

Gyro FIFO watermark interrupt on DRDY/INT2

**6.257.2.8 #define DIGYRO\_CTRL3\_OPENDRAIN 0x10**

Gyro use open drain rather than push-pull

**6.258 Dexter Industries IMU Gyro control register 4 constants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 4.

**Defines**

- #define [DIGYRO\\_CTRL4\\_BLOCKDATA](#) 0x80
- #define [DIGYRO\\_CTRL4\\_BIGENDIAN](#) 0x40
- #define [DIGYRO\\_CTRL4\\_SCALE\\_250](#) 0x00
- #define [DIGYRO\\_CTRL4\\_SCALE\\_500](#) 0x10
- #define [DIGYRO\\_CTRL4\\_SCALE\\_2000](#) 0x30

**6.258.1 Detailed Description**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 4.

**6.258.2 Define Documentation****6.258.2.1 #define DIGYRO\_CTRL4\_BIGENDIAN 0x40**

Gyro use big endian - MSB/LSB rather than LSB/MSB in output registers

**6.258.2.2 #define DIGYRO\_CTRL4\_BLOCKDATA 0x80**

Gyro block data update - output registers are not updated until MSB and LSB reading

**6.258.2.3 #define DIGYRO\_CTRL4\_SCALE\_2000 0x30**

Gyro 2000 degrees per second scale

**Examples:**

[ex\\_digyro.nxc](#).

**6.258.2.4 #define DIGYRO\_CTRL4\_SCALE\_250 0x00**

Gyro 250 degrees per second scale

**6.258.2.5 #define DIGYRO\_CTRL4\_SCALE\_500 0x10**

Gyro 500 degrees per second scale

**6.259 Dexter Industries IMU Gyro control register 5 constants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 5.

**Defines**

- #define [DIGYRO\\_CTRL5\\_REBOOTMEM](#) 0x80
- #define [DIGYRO\\_CTRL5\\_FIFOENABLE](#) 0x40
- #define [DIGYRO\\_CTRL5\\_HPENABLE](#) 0x10
- #define [DIGYRO\\_CTRL5\\_OUT\\_SEL\\_1](#) 0x00
- #define [DIGYRO\\_CTRL5\\_OUT\\_SEL\\_2](#) 0x01
- #define [DIGYRO\\_CTRL5\\_OUT\\_SEL\\_3](#) 0x02
- #define [DIGYRO\\_CTRL5\\_INT1\\_SEL\\_1](#) 0x00
- #define [DIGYRO\\_CTRL5\\_INT1\\_SEL\\_2](#) 0x04
- #define [DIGYRO\\_CTRL5\\_INT1\\_SEL\\_3](#) 0x08

**6.259.1 Detailed Description**

Constants that are for use with the Dexter Industries IMU Gyro sensor's control register 5.

**6.259.2 Define Documentation****6.259.2.1 #define DIGYRO\_CTRL5\_FIFOENABLE 0x40**

Gyro enable FIFO

**6.259.2.2 #define DIGYRO\_CTRL5\_HPENABLE 0x10**

Gyro enable high pass filter

**6.259.2.3 #define DIGYRO\_CTRL5\_INT1\_SEL\_1 0x00**

Gyro non-high-pass-filtered data are used for interrupt generation

**6.259.2.4 #define DIGYRO\_CTRL5\_INT1\_SEL\_2 0x04**

Gyro high-pass-filtered data are used for interrupt generation

**6.259.2.5 #define DIGYRO\_CTRL5\_INT1\_SEL\_3 0x08**

Gyro low-pass-filtered data are used for interrupt generation

**6.259.2.6 #define DIGYRO\_CTRL5\_OUT\_SEL\_1 0x00**

Gyro data in data registers and FIFO are not high-pass filtered

**6.259.2.7 #define DIGYRO\_CTRL5\_OUT\_SEL\_2 0x01**

Gyro data in data registers and FIFO are high-pass filtered

**6.259.2.8 #define DIGYRO\_CTRL5\_OUT\_SEL\_3 0x02**

Gyro data in data registers and FIFO are low-pass filtered by LPF2

**6.259.2.9 #define DIGYRO\_CTRL5\_REBOOTMEM 0x80**

Gyro reboot memory content

**6.260 Dexter Industries IMU Gyro FIFO control register onstants**

Constants that are for use with the Dexter Industries IMU Gyro sensor's FIFO control register.

## Defines

- #define DIGYRO\_FIFOCTRL\_BYPASS 0x00
- #define DIGYRO\_FIFOCTRL\_FIFO 0x20
- #define DIGYRO\_FIFOCTRL\_STREAM 0x40
- #define DIGYRO\_FIFOCTRL\_STREAM2FIFO 0x60
- #define DIGYRO\_FIFOCTRL\_BYPASS2STREAM 0x80
- #define DIGYRO\_FIFOCTRL\_WATERMARK\_MASK 0x1F

### 6.260.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Gyro sensor's FIFO control register.

### 6.260.2 Define Documentation

#### 6.260.2.1 #define DIGYRO\_FIFOCTRL\_BYPASS 0x00

Gyro FIFO bypass mode

#### 6.260.2.2 #define DIGYRO\_FIFOCTRL\_BYPASS2STREAM 0x80

Gyro FIFO bypass-to-stream mode

#### 6.260.2.3 #define DIGYRO\_FIFOCTRL\_FIFO 0x20

Gyro FIFO mode

#### 6.260.2.4 #define DIGYRO\_FIFOCTRL\_STREAM 0x40

Gyro FIFO stream mode

#### 6.260.2.5 #define DIGYRO\_FIFOCTRL\_STREAM2FIFO 0x60

Gyro FIFO stream-to-FIFO mode

#### 6.260.2.6 #define DIGYRO\_FIFOCTRL\_WATERMARK\_MASK 0x1F

Gyro FIFO threshold. Watermark level setting mask (values from 0x00 to 0x1F)

## 6.261 Dexter Industries IMU Gyro status register constants

Constants that are for use with the Dexter Industries IMU Gyro sensor's status register.

### Defines

- #define DIGYRO\_STATUS\_XDATA 0x01
- #define DIGYRO\_STATUS\_YDATA 0x02
- #define DIGYRO\_STATUS\_ZDATA 0x04
- #define DIGYRO\_STATUS\_XYZDATA 0x08
- #define DIGYRO\_STATUS\_XOVER 0x10
- #define DIGYRO\_STATUS\_YOVER 0x20
- #define DIGYRO\_STATUS\_ZOVER 0x40
- #define DIGYRO\_STATUS\_XYZOVER 0x80

#### 6.261.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Gyro sensor's status register.

#### 6.261.2 Define Documentation

##### 6.261.2.1 #define DIGYRO\_STATUS\_XDATA 0x01

Gyro X-axis new data available

##### 6.261.2.2 #define DIGYRO\_STATUS\_XOVER 0x10

Gyro X-axis data overrun - new data for the X-axis has overwritten the previous one

##### 6.261.2.3 #define DIGYRO\_STATUS\_XYZDATA 0x08

Gyro X, Y, or Z-axis new data available - a new set of data is available

##### 6.261.2.4 #define DIGYRO\_STATUS\_XYZOVER 0x80

Gyro X, Y, or Z-axis data overrun - new data has overwritten the previous one before it was read

##### 6.261.2.5 #define DIGYRO\_STATUS\_YDATA 0x02

Gyro Y-axis new data available

**6.261.2.6 #define DIGYRO\_STATUS\_YOVER 0x20**

Gyro Y-axis data overrun - new data for the Y-axis has overwritten the previous one

**6.261.2.7 #define DIGYRO\_STATUS\_ZDATA 0x04**

Gyro Z-axis new data available

**6.261.2.8 #define DIGYRO\_STATUS\_ZOVER 0x40**

Gyro Z-axis data overrun - new data for the Z-axis has overwritten the previous one

**6.262 Dexter Industries IMU Accelerometer register constants**

Constants that define the Dexter Industries IMU Accelerometer registers.

**Defines**

- #define [DIACCL\\_REG\\_XLOW](#) 0x00
- #define [DIACCL\\_REG\\_XHIGH](#) 0x01
- #define [DIACCL\\_REG\\_YLOW](#) 0x02
- #define [DIACCL\\_REG\\_YHIGH](#) 0x03
- #define [DIACCL\\_REG\\_ZLOW](#) 0x04
- #define [DIACCL\\_REG\\_ZHIGH](#) 0x05
- #define [DIACCL\\_REG\\_X8](#) 0x06
- #define [DIACCL\\_REG\\_Y8](#) 0x07
- #define [DIACCL\\_REG\\_Z8](#) 0x08
- #define [DIACCL\\_REG\\_STATUS](#) 0x09
- #define [DIACCL\\_REG\\_DETECTSRC](#) 0x0A
- #define [DIACCL\\_REG\\_OUTTEMP](#) 0x0B
- #define [DIACCL\\_REG\\_I2CADDR](#) 0x0D
- #define [DIACCL\\_REG\\_USERINFO](#) 0x0E
- #define [DIACCL\\_REG\\_WHOAMI](#) 0x0F
- #define [DIACCL\\_REG\\_XLOWDRIFT](#) 0x10
- #define [DIACCL\\_REG\\_XHIGHDRIFT](#) 0x11
- #define [DIACCL\\_REG\\_YLOWDRIFT](#) 0x12
- #define [DIACCL\\_REG\\_YHIGHDRIFT](#) 0x13
- #define [DIACCL\\_REG\\_ZLOWDRIFT](#) 0x14
- #define [DIACCL\\_REG\\_ZHIGHDRIFT](#) 0x15
- #define [DIACCL\\_REG\\_MODECTRL](#) 0x16
- #define [DIACCL\\_REG\\_INTLATCH](#) 0x17

- #define `DIACCL_REG_CTRL1` 0x18
- #define `DIACCL_REG_CTRL2` 0x19
- #define `DIACCL_REG_LVLDETTTHR` 0x1A
- #define `DIACCL_REG_PLSDETTHR` 0x1B
- #define `DIACCL_REG_PLSDURVAL` 0x1C
- #define `DIACCL_REG_LATENCYTM` 0x1D
- #define `DIACCL_REG_TIMEWINDOW` 0x1E

### 6.262.1 Detailed Description

Constants that define the Dexter Industries IMU Accelerometer registers.

### 6.262.2 Define Documentation

#### 6.262.2.1 #define `DIACCL_REG_CTRL1` 0x18

Accelerometer control register 1 (read/write)

#### 6.262.2.2 #define `DIACCL_REG_CTRL2` 0x19

Accelerometer control register 1 (read/write)

#### 6.262.2.3 #define `DIACCL_REG_DETECTSRC` 0x0A

Accelerometer detection source register (read only)

#### 6.262.2.4 #define `DIACCL_REG_I2CADDR` 0x0D

Accelerometer I2C address register (read only)

#### 6.262.2.5 #define `DIACCL_REG_INTLATCH` 0x17

Accelerometer interrupt latch reset register (read/write)

#### 6.262.2.6 #define `DIACCL_REG_LATENCYTM` 0x1D

Accelerometer latency time value register (read/write)

**6.262.2.7 #define DIACCL\_REG\_LVLDETTTHR 0x1A**

Accelerometer level detection threshold limit value register (read/write)

**6.262.2.8 #define DIACCL\_REG\_MODECTRL 0x16**

Accelerometer mode control register (read/write)

**6.262.2.9 #define DIACCL\_REG\_OUTTEMP 0x0B**

Accelerometer temperature output register (read only)

**6.262.2.10 #define DIACCL\_REG\_PLSDETTHR 0x1B**

Accelerometer pulse detection threshold limit value register (read/write)

**6.262.2.11 #define DIACCL\_REG\_PLSDURVAL 0x1C**

Accelerometer pulse duration value register (read/write)

**6.262.2.12 #define DIACCL\_REG\_STATUS 0x09**

Accelerometer status register (read only)

**6.262.2.13 #define DIACCL\_REG\_TIMEWINDOW 0x1E**

Accelerometer time window for 2nd pulse value register (read/write)

**6.262.2.14 #define DIACCL\_REG\_USERINFO 0x0E**

Accelerometer user information register (read only)

**6.262.2.15 #define DIACCL\_REG\_WHOAMI 0x0F**

Accelerometer device identification register (read only)

**6.262.2.16 #define DIACCL\_REG\_X8 0x06**

Accelerometer x-axis 8-bit register (read only)

**6.262.2.17 #define DIACCL\_REG\_XHIGH 0x01**

Accelerometer x-axis high byte register (read only)

**6.262.2.18 #define DIACCL\_REG\_XHIGHDRIFT 0x11**

Accelerometer x-axis offset drift high byte register (read/write)

**6.262.2.19 #define DIACCL\_REG\_XLOW 0x00**

Accelerometer x-axis low byte register (read only)

**6.262.2.20 #define DIACCL\_REG\_XLOWDRIFT 0x10**

Accelerometer x-axis offset drift low byte register (read/write)

**6.262.2.21 #define DIACCL\_REG\_Y8 0x07**

Accelerometer x-axis 8-bit register (read only)

**6.262.2.22 #define DIACCL\_REG\_YHIGH 0x03**

Accelerometer y-axis high byte register (read only)

**6.262.2.23 #define DIACCL\_REG\_YHIGHDRIFT 0x13**

Accelerometer y-axis offset drift high byte register (read/write)

**6.262.2.24 #define DIACCL\_REG\_YLOW 0x02**

Accelerometer y-axis low byte register (read only)

**6.262.2.25 #define DIACCL\_REG\_YLOWDRIFT 0x12**

Accelerometer y-axis offset drift low byte register (read/write)

**6.262.2.26 #define DIACCL\_REG\_Z8 0x08**

Accelerometer x-axis 8-bit register (read only)

**6.262.2.27 #define DIACCL\_REG\_ZHIGH 0x05**

Accelerometer z-axis high byte register (read only)

**6.262.2.28 #define DIACCL\_REG\_ZHIGHDRIFT 0x15**

Accelerometer z-axis offset drift high byte register (read/write)

**6.262.2.29 #define DIACCL\_REG\_ZLOW 0x04**

Accelerometer z-axis low byte register (read only)

**6.262.2.30 #define DIACCL\_REG\_ZLOWDRIFT 0x14**

Accelerometer z-axis offset drift low byte register (read/write)

**6.263 Dexter Industries IMU Accelerometer status register constants**

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's status register.

**Defines**

- #define [DIACCL\\_STATUS\\_DATAREADY](#) 0x01
- #define [DIACCL\\_STATUS\\_DATAOVER](#) 0x02
- #define [DIACCL\\_STATUS\\_PARITYERR](#) 0x04

**6.263.1 Detailed Description**

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's status register.

**6.263.2 Define Documentation**

**6.263.2.1 #define DIACCL\_STATUS\_DATAOVER 0x02**

Accelerometer data is overwritten

## 6.264 Dexter Industries IMU Accelerometer mode control register constants

### 6.263.2.2 #define DIACCL\_STATUS\_DATAREADY 0x01

Accelerometer data is ready

### 6.263.2.3 #define DIACCL\_STATUS\_PARITYERR 0x04

Accelerometer parity error is detected in trim data

## 6.264 Dexter Industries IMU Accelerometer mode control register constants

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's mode control register.

### Defines

- #define [DIACCL\\_MODE\\_STANDBY](#) 0x00
- #define [DIACCL\\_MODE\\_MEASURE](#) 0x01
- #define [DIACCL\\_MODE\\_LVLDETECT](#) 0x02
- #define [DIACCL\\_MODE\\_PLSDTECT](#) 0x03
- #define [DIACCL\\_MODE\\_GLVL8](#) 0x00
- #define [DIACCL\\_MODE\\_GLVL2](#) 0x04
- #define [DIACCL\\_MODE\\_GLVL4](#) 0x08

### 6.264.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's mode control register.

### 6.264.2 Define Documentation

#### 6.264.2.1 #define DIACCL\_MODE\_GLVL2 0x04

Accelerometer 2G measurement range

#### 6.264.2.2 #define DIACCL\_MODE\_GLVL4 0x08

Accelerometer 4G measurement range

## 6.265 Dexter Industries IMU Accelerometer interrupt latch reset register constants 1004

---

### 6.264.2.3 #define DIACCL\_MODE\_GLVL8 0x00

Accelerometer 8G measurement range

#### Examples:

[ex\\_diaccl.nxc](#).

### 6.264.2.4 #define DIACCL\_MODE\_LVLDETECT 0x02

Accelerometer level detect mode

### 6.264.2.5 #define DIACCL\_MODE\_MEASURE 0x01

Accelerometer measurement mode

### 6.264.2.6 #define DIACCL\_MODE\_PLSDETECT 0x03

Accelerometer pulse detect mode

### 6.264.2.7 #define DIACCL\_MODE\_STANDBY 0x00

Accelerometer standby mode

## 6.265 Dexter Industries IMU Accelerometer interrupt latch reset register constants

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's interrupt latch reset register.

#### Defines

- #define [DIACCL\\_INTERRUPT\\_LATCH\\_CLEAR1](#) 0x01
- #define [DIACCL\\_INTERRUPT\\_LATCH\\_CLEAR2](#) 0x02

### 6.265.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's interrupt latch reset register.

## **6.265.2 Define Documentation**

### **6.265.2.1 #define DIACCL\_INTERRUPT\_LATCH\_CLEAR1 0x01**

Accelerometer clear interrupt 1

### **6.265.2.2 #define DIACCL\_INTERRUPT\_LATCH\_CLEAR2 0x02**

Accelerometer clear interrupt 2

## **6.266 Dexter Industries IMU Accelerometer control register 1 constants**

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 1.

### **Defines**

- #define [DIACCL\\_CTRL1\\_INT2TOINT1](#) 0x01
- #define [DIACCL\\_CTRL1\\_LEVELPULSE](#) 0x00
- #define [DIACCL\\_CTRL1\\_PULSELEVEL](#) 0x02
- #define [DIACCL\\_CTRL1\\_PULSEPULSE](#) 0x04
- #define [DIACCL\\_CTRL1\\_NO\\_XDETECT](#) 0x08
- #define [DIACCL\\_CTRL1\\_NO\\_YDETECT](#) 0x10
- #define [DIACCL\\_CTRL1\\_NO\\_ZDETECT](#) 0x20
- #define [DIACCL\\_CTRL1\\_THRESH\\_INT](#) 0x40
- #define [DIACCL\\_CTRL1\\_FILT\\_BW125](#) 0x80

### **6.266.1 Detailed Description**

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 1.

## **6.266.2 Define Documentation**

### **6.266.2.1 #define DIACCL\_CTRL1\_FILT\_BW125 0x80**

Accelerometer digital filter band width is 125 Hz.

## **6.267 Dexter Industries IMU Accelerometer control register 2 constants 1006**

### **6.266.2.2 #define DIACCL\_CTRL1\_INT2TOINT1 0x01**

Accelerometer INT2 pin is routed to INT1 bit in Detection Source Register (\$0A) and INT1 pin is routed to INT2 bit in Detection Source Register (\$0A)

### **6.266.2.3 #define DIACCL\_CTRL1\_LEVELPULSE 0x00**

Accelerometer INT1 register is detecting Level while INT2 is detecting pulse

### **6.266.2.4 #define DIACCL\_CTRL1\_NO\_XDETECT 0x08**

Accelerometer disable x-axis detection.

### **6.266.2.5 #define DIACCL\_CTRL1\_NO\_YDETECT 0x10**

Accelerometer disable y-axis detection.

### **6.266.2.6 #define DIACCL\_CTRL1\_NO\_ZDETECT 0x20**

Accelerometer disable z-axis detection.

### **6.266.2.7 #define DIACCL\_CTRL1\_PULSELEVEL 0x02**

Accelerometer INT1 Register is detecting Pulse while INT2 is detecting Level

### **6.266.2.8 #define DIACCL\_CTRL1\_PULSEPULSE 0x04**

Accelerometer INT1 Register is detecting a Single Pulse and INT2 is detecting Single Pulse (if 2nd Time Window = 0) or if there is a latency time window and second time window > 0 then INT2 will detect the double pulse only.

### **6.266.2.9 #define DIACCL\_CTRL1\_THRESH\_INT 0x40**

Accelerometer threshold value can be an integer.

## **6.267 Dexter Industries IMU Accelerometer control register 2 constants**

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 2.

## Defines

- #define [DIACCL\\_CTRL2\\_LVLPOL\\_NEGAND](#) 0x01
- #define [DIACCL\\_CTRL2\\_DETPOL\\_NEGAND](#) 0x02
- #define [DIACCL\\_CTRL2\\_DRIVE\\_STRONG](#) 0x04

### 6.267.1 Detailed Description

Constants that are for use with the Dexter Industries IMU Accelerometer sensor's control register 2.

### 6.267.2 Define Documentation

#### 6.267.2.1 #define [DIACCL\\_CTRL2\\_DETPOL\\_NEGAND](#) 0x02

Accelerometer pulse detection polarity is negative and detecting condition is AND all 3 axes

#### 6.267.2.2 #define [DIACCL\\_CTRL2\\_DRIVE\\_STRONG](#) 0x04

Accelerometer strong drive strength on SDA/SDO pin

#### 6.267.2.3 #define [DIACCL\\_CTRL2\\_LVLPOL\\_NEGAND](#) 0x01

Accelerometer level detection polarity is negative and detecting condition is AND all 3 axes

## 6.268 Microinfinity device constants

Constants that are for use with Microinfinity devices.

### Modules

- [Microinfinity CruzCore XG1300L sensor constants](#)  
*Constants that are for use with the CruzCore XG1300L sensor.*

### 6.268.1 Detailed Description

Constants that are for use with Microinfinity devices.

## 6.269 Microinfinity CruzCore XG1300L sensor constants

Constants that are for use with the CruzCore XG1300L sensor.

### Modules

- [Microinfinity CruzCore XG1300L](#)

*sensor scale factor constants* Constants for setting the scale factor of the CruzCore XG1300L sensor.

### Defines

- #define [MI\\_ADDR\\_XG1300L](#) 0x02
- #define [XG1300L\\_REG\\_ANGLE](#) 0x42
- #define [XG1300L\\_REG\\_TURNRATE](#) 0x44
- #define [XG1300L\\_REG\\_XAXIS](#) 0x46
- #define [XG1300L\\_REG\\_YAXIS](#) 0x48
- #define [XG1300L\\_REG\\_ZAXIS](#) 0x4A
- #define [XG1300L\\_REG\\_RESET](#) 0x60
- #define [XG1300L\\_REG\\_2G](#) 0x61
- #define [XG1300L\\_REG\\_4G](#) 0x62
- #define [XG1300L\\_REG\\_8G](#) 0x63

#### 6.269.1 Detailed Description

Constants that are for use with the CruzCore XG1300L sensor.

#### 6.269.2 Define Documentation

##### 6.269.2.1 #define MI\_ADDR\_XG1300L 0x02

XG1300L I2C address

##### 6.269.2.2 #define XG1300L\_REG\_2G 0x61

Select +/- 2G accelerometer range.

##### 6.269.2.3 #define XG1300L\_REG\_4G 0x62

Select +/- 4G accelerometer range.

**6.269.2.4 #define XG1300L\_REG\_8G 0x63**

Select +/- 8G accelerometer range.

**6.269.2.5 #define XG1300L\_REG\_ANGLE 0x42**

Read accumulated angle (2 bytes little endian) in 1/100s of degrees.

**6.269.2.6 #define XG1300L\_REG\_RESET 0x60**

Reset the XG1300L device.

**6.269.2.7 #define XG1300L\_REG\_TURNRATE 0x44**

Read rate of turn (2 bytes little endian) in 1/100s of degrees/second.

**6.269.2.8 #define XG1300L\_REG\_XAXIS 0x46**

Read x-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**6.269.2.9 #define XG1300L\_REG\_YAXIS 0x48**

Read y-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**6.269.2.10 #define XG1300L\_REG\_ZAXIS 0x4A**

Read z-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**6.270 Microinfinity CruzCore XG1300L**

sensor scale factor constants Constants for setting the scale factor of the CruzCore XG1300L sensor.

**Defines**

- #define [XG1300L\\_SCALE\\_2G](#) 0x01
- #define [XG1300L\\_SCALE\\_4G](#) 0x02

- #define `XG1300L_SCALE_8G` 0x04

### 6.270.1 Detailed Description

sensor scale factor constants Constants for setting the scale factor of the CruzCore XG1300L sensor.

### 6.270.2 Define Documentation

#### 6.270.2.1 #define `XG1300L_SCALE_2G` 0x01

Select +/- 2G accelerometer range.

#### Examples:

`ex_xg1300.nxc`.

#### 6.270.2.2 #define `XG1300L_SCALE_4G` 0x02

Select +/- 4G accelerometer range.

#### Examples:

`ex_xg1300.nxc`.

#### 6.270.2.3 #define `XG1300L_SCALE_8G` 0x04

Select +/- 8G accelerometer range.

#### Examples:

`ex_xg1300.nxc`.

## 6.271 Data type limits

Constants that define various data type limits.

### Defines

- #define `CHAR_BIT` 8
- #define `SCHAR_MIN` -128

- #define `SCHAR_MAX` 127
- #define `UCHAR_MAX` 255
- #define `CHAR_MIN` -128
- #define `CHAR_MAX` 127
- #define `SHRT_MIN` -32768
- #define `SHRT_MAX` 32767
- #define `USHRT_MAX` 65535
- #define `INT_MIN` -32768
- #define `INT_MAX` 32767
- #define `UINT_MAX` 65535
- #define `LONG_MIN` -2147483648
- #define `LONG_MAX` 2147483647
- #define `ULONG_MAX` 4294967295
- #define `RAND_MAX` 2147483646

### 6.271.1 Detailed Description

Constants that define various data type limits.

### 6.271.2 Define Documentation

#### 6.271.2.1 #define `CHAR_BIT` 8

The number of bits in the char type

#### 6.271.2.2 #define `CHAR_MAX` 127

The maximum value of the char type

#### 6.271.2.3 #define `CHAR_MIN` -128

The minimum value of the char type

#### 6.271.2.4 #define `INT_MAX` 32767

The maximum value of the int type

#### 6.271.2.5 #define `INT_MIN` -32768

The minimum value of the int type

**6.271.2.6 #define LONG\_MAX 2147483647**

The maximum value of the long type

**6.271.2.7 #define LONG\_MIN -2147483648**

The minimum value of the long type

**6.271.2.8 #define RAND\_MAX 2147483646**

The maximum long random number returned by rand

**6.271.2.9 #define SCHAR\_MAX 127**

The maximum value of the signed char type

**6.271.2.10 #define SCHAR\_MIN -128**

The minimum value of the signed char type

**6.271.2.11 #define SHRT\_MAX 32767**

The maximum value of the short type

**6.271.2.12 #define SHRT\_MIN -32768**

The minimum value of the short type

**6.271.2.13 #define UCHAR\_MAX 255**

The maximum value of the unsigned char type

**6.271.2.14 #define UINT\_MAX 65535**

The maximum value of the unsigned int type

**6.271.2.15 #define ULONG\_MAX 4294967295**

The maximum value of the unsigned long type

**6.271.2.16 #define USHRT\_MAX 65535**

The maximum value of the unsigned short type

**6.272 Graphics library begin modes**

Constants that are used to specify the polygon surface begin mode.

**Defines**

- #define [GL\\_POLYGON](#) 1
- #define [GL\\_LINE](#) 2
- #define [GL\\_POINT](#) 3
- #define [GL\\_CIRCLE](#) 4

**6.272.1 Detailed Description**

Constants that are used to specify the polygon surface begin mode.

**6.272.2 Define Documentation****6.272.2.1 #define GL\_CIRCLE 4**

Use circle mode.

**Examples:**

[glCircleDemo.nxc](#).

**6.272.2.2 #define GL\_LINE 2**

Use line mode.

**6.272.2.3 #define GL\_POINT 3**

Use point mode.

#### 6.272.2.4 #define GL\_POLYGON 1

Use polygon mode.

#### Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

## 6.273 Graphics library actions

Constants that are used to specify a graphics library action.

### Defines

- #define [GL\\_TRANSLATE\\_X](#) 1
- #define [GL\\_TRANSLATE\\_Y](#) 2
- #define [GL\\_TRANSLATE\\_Z](#) 3
- #define [GL\\_ROTATE\\_X](#) 4
- #define [GL\\_ROTATE\\_Y](#) 5
- #define [GL\\_ROTATE\\_Z](#) 6
- #define [GL\\_SCALE\\_X](#) 7
- #define [GL\\_SCALE\\_Y](#) 8
- #define [GL\\_SCALE\\_Z](#) 9

### 6.273.1 Detailed Description

Constants that are used to specify a graphics library action.

### 6.273.2 Define Documentation

#### 6.273.2.1 #define GL\_ROTATE\_X 4

Rotate around the X axis.

#### Examples:

[glRotateDemo.nxc](#).

**6.273.2.2 #define GL\_ROTATE\_Y 5**

Rotate around the Y axis.

**Examples:**

[glRotateDemo.nxc](#).

**6.273.2.3 #define GL\_ROTATE\_Z 6**

Rotate around the Z axis.

**6.273.2.4 #define GL\_SCALE\_X 7**

Scale along the X axis.

**Examples:**

[glScaleDemo.nxc](#).

**6.273.2.5 #define GL\_SCALE\_Y 8**

Scale along the Y axis.

**6.273.2.6 #define GL\_SCALE\_Z 9**

Scale along the Z axis.

**6.273.2.7 #define GL\_TRANSLATE\_X 1**

Translate along the X axis.

**Examples:**

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.273.2.8 #define GL\_TRANSLATE\_Y 2**

Translate along the Y axis.

**Examples:**

[glTranslateDemo.nxc](#).

**6.273.2.9 #define GL\_TRANSLATE\_Z 3**

Translate along the Z axis.

**Examples:**

[glTranslateDemo.nxc](#).

**6.274 Graphics library settings**

Constants that are used to configure the graphics library settings.

**Defines**

- #define [GL\\_CIRCLE\\_SIZE](#) 1
- #define [GL\\_CULL\\_MODE](#) 2
- #define [GL\\_CAMERA\\_DEPTH](#) 3
- #define [GL\\_ZOOM\\_FACTOR](#) 4

**6.274.1 Detailed Description**

Constants that are used to configure the graphics library settings.

**6.274.2 Define Documentation****6.274.2.1 #define GL\_CAMERA\_DEPTH 3**

Set the camera depth.

**6.274.2.2 #define GL\_CIRCLE\_SIZE 1**

Set the circle size.

**6.274.2.3 #define GL\_CULL\_MODE 2**

Set the cull mode.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**6.274.2.4 #define GL\_ZOOM\_FACTOR 4**

Set the zoom factor.

**6.275 Graphics library cull mode**

Constants to use when setting the graphics library cull mode.

**Defines**

- #define [GL\\_CULL\\_BACK](#) 2
- #define [GL\\_CULL\\_FRONT](#) 3
- #define [GL\\_CULL\\_NONE](#) 4

**6.275.1 Detailed Description**

Constants to use when setting the graphics library cull mode.

**6.275.2 Define Documentation****6.275.2.1 #define GL\_CULL\_BACK 2**

Cull lines in back.

**6.275.2.2 #define GL\_CULL\_FRONT 3**

Cull lines in front.

**6.275.2.3 #define GL\_CULL\_NONE 4**

Do not cull any lines.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

## 7 Data Structure Documentation

### 7.1 ColorSensorReadType Struct Reference

Parameters for the ColorSensorRead system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- byte [Port](#)
- int [ColorValue](#)
- unsigned int [RawArray](#) []
- unsigned int [NormalizedArray](#) []
- int [ScaledArray](#) []
- bool [Invalid](#)

#### 7.1.1 Detailed Description

Parameters for the ColorSensorRead system call. This structure is used when calling the [SysColorSensorRead](#) system call function. Choose the sensor port ([Input port constants](#)) and after calling the function read the sensor values from the ColorValue field or the raw, normalized, or scaled value arrays.

See also:

[SysColorSensorRead\(\)](#)

Examples:

[ex\\_SysColorSensorRead.nxc](#).

#### 7.1.2 Field Documentation

##### 7.1.2.1 int ColorSensorReadType::ColorValue

The color value returned by the sensor. See the [Color values](#) group.

Examples:

[ex\\_SysColorSensorRead.nxc](#).

### 7.1.2.2 bool ColorSensorReadType::Invalid

Are the sensor values valid?

### 7.1.2.3 unsigned int ColorSensorReadType::NormalizedArray[]

Normalized color values returned by the sensor. See the [Color sensor array indices](#) group.

### 7.1.2.4 byte ColorSensorReadType::Port

The sensor port. See the constants in the [Input port constants](#) group.

#### Examples:

[ex\\_SysColorSensorRead.nxc](#).

### 7.1.2.5 unsigned int ColorSensorReadType::RawArray[]

Raw color values returned by the sensor. See the [Color sensor array indices](#) group.

### 7.1.2.6 char ColorSensorReadType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### Examples:

[ex\\_SysColorSensorRead.nxc](#).

### 7.1.2.7 int ColorSensorReadType::ScaledArray[]

Scaled color values returned by the sensor. See the [Color sensor array indices](#) group.  
The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.2 CommBTCheckStatusType Struct Reference

Parameters for the CommBTCheckStatus system call.

```
#include <NXCDefs.h>
```

## Data Fields

- char [Result](#)
- byte [Connection](#)

### 7.2.1 Detailed Description

Parameters for the CommBTCheckStatus system call. This structure is used when calling the [SysCommBTCheckStatus](#) system call function.

#### See also:

[SysCommBTCheckStatus\(\)](#)

#### Examples:

[ex\\_syscommbtcheckstatus.nxc.](#)

### 7.2.2 Field Documentation

#### 7.2.2.1 byte CommBTCheckStatusType::Connection

The connection to check.

#### Examples:

[ex\\_syscommbtcheckstatus.nxc.](#)

#### 7.2.2.2 char CommBTCheckStatusType::Result

The function call result. Possible values include [ERR\\_INVALID\\_PORT](#), [STAT\\_COMM\\_PENDING](#), [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#), and [LDR\\_SUCCESS](#).

#### Examples:

[ex\\_syscommbtcheckstatus.nxc.](#)

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.3 CommBTConnectionType Struct Reference

Parameters for the CommBTConnection system call.

```
#include <NXCDefs.h>
```

## Data Fields

- unsigned int [Result](#)
- byte [Action](#)
- string [Name](#)
- byte [ConnectionSlot](#)

### 7.3.1 Detailed Description

Parameters for the CommBTConnection system call. This structure is used when calling the [SysCommBTConnection](#) system call function.

See also:

[SysCommBTConnection\(\)](#)

Examples:

[ex\\_syscommbtconnection.nxc.](#)

### 7.3.2 Field Documentation

#### 7.3.2.1 byte CommBTConnectionType::Action

The connection action (connect or disconnect).

Examples:

[ex\\_syscommbtconnection.nxc.](#)

#### 7.3.2.2 byte CommBTConnectionType::ConnectionSlot

The connection slot to connect or disconnect.

Examples:

[ex\\_syscommbtconnection.nxc.](#)

#### 7.3.2.3 string CommBTConnectionType::Name

The name of the device to connect or disconnect.

Examples:

[ex\\_syscommbtconnection.nxc.](#)

### 7.3.2.4 unsigned int CommBTConnectionType::Result

The function call result.

#### Examples:

[ex\\_syscommbtconnection.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.4 CommBTONOffType Struct Reference

Parameters for the CommBTONOff system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- unsigned int [Result](#)
- bool [PowerState](#)

### 7.4.1 Detailed Description

Parameters for the CommBTONOff system call. This structure is used when calling the [SysCommBTONOff](#) system call function.

#### See also:

[SysCommBTONOff\(\)](#)

#### Examples:

[ex\\_SysCommBTONOff.nxc](#).

### 7.4.2 Field Documentation

#### 7.4.2.1 bool CommBTONOffType::PowerState

If true then turn on bluetooth, otherwise, turn it off.

#### Examples:

[ex\\_SysCommBTONOff.nxc](#).

### 7.4.2.2 unsigned int CommBTOffType::Result

The function call result.

#### Examples:

[ex\\_SysCommBTOff.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.5 CommBTWriteType Struct Reference

Parameters for the CommBTWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Connection](#)
- byte [Buffer](#) []

### 7.5.1 Detailed Description

Parameters for the CommBTWrite system call. This structure is used when calling the [SysCommBTWrite](#) system call function.

#### See also:

[SysCommBTWrite\(\)](#)

#### Examples:

[ex\\_syscommbtwrite.nxc](#).

### 7.5.2 Field Documentation

#### 7.5.2.1 byte CommBTWriteType::Buffer[]

The data to write to the connection.

#### Examples:

[ex\\_syscommbtwrite.nxc](#).

### 7.5.2.2 byte CommBTWriteType::Connection

The connection to use.

#### Examples:

[ex\\_syscommbtwrite.nxc](#).

### 7.5.2.3 char CommBTWriteType::Result

The function call result. Possible values include [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#) and [STAT\\_COMM\\_PENDING](#) (write accepted).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.6 CommExecuteFunctionType Struct Reference

Parameters for the CommExecuteFunction system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [Cmd](#)
- byte [Param1](#)
- byte [Param2](#)
- byte [Param3](#)
- string [Name](#)
- unsigned int [RetVal](#)

### 7.6.1 Detailed Description

Parameters for the CommExecuteFunction system call. This structure is used when calling the [SysCommExecuteFunction](#) system call function.

The fields usage depends on the requested command and are documented in the table below. If a field member is shown as 'x' it is ignored by the specified command.

| Cmd                 | Meaning                                 | (Param1,Param2,Param3,Name)             |
|---------------------|---|---|
| INTF_SENDFILE       | Send a file over a Bluetooth connection | (Connection,x,x,Filename)               |
| INTF_SEARCH         | Search for Bluetooth devices            | (x,x,x,x)                               |
| INTF_STOPSEARCH     | Stop searching for Bluetooth devices    | (x,x,x,x)                               |
| INTF_CONNECT        | Connect to a Bluetooth device           | (DeviceIndex,Connection,x,x)            |
| INTF_DISCONNECT     | Disconnect a Bluetooth device           | (Connection,x,x,x)                      |
| INTF_-DISCONNECTALL | Disconnect all Bluetooth devices        | (x,x,x,x)                               |
| INTF_-REMOVEDEVICE  | Remove device from My Contacts          | (DeviceIndex,x,x,x)                     |
| INTF_VISIBILITY     | Set Bluetooth visibility                | (true/false,x,x,x)                      |
| INTF_SETCMDMODE     | Set command mode                        | (x,x,x,x)                               |
| INTF_OPENSTREAM     | Open a stream                           | (x,Connection,x,x)                      |
| INTF_SENDDATA       | Send data                               | (Length, Connection, WaitForIt, Buffer) |
| INTF_-FACTORYRESET  | Bluetooth factory reset                 | (x,x,x,x)                               |
| INTF_BTON           | Turn Bluetooth on                       | (x,x,x,x)                               |
| INTF_BTOFF          | Turn Bluetooth off                      | (x,x,x,x)                               |
| INTF_SETBTNAME      | Set Bluetooth name                      | (x,x,x,x)                               |
| INTF_EXTREAD        | Handle external? read                   | (x,x,x,x)                               |
| INTF_PINREQ         | Handle Bluetooth PIN request            | (x,x,x,x)                               |
| INTF_CONNECTREQ     | Handle Bluetooth connect request        | (x,x,x,x)                               |

See also:

[SysCommExecuteFunction\(\)](#)

Examples:

[ex\\_syscommexecutefunction.nxc](#).

## 7.6.2 Field Documentation

### 7.6.2.1 byte CommExecuteFunctionType::Cmd

The command to execute.

**Examples:**

[ex\\_syscommexecutefunction.nxc](#).

**7.6.2.2 string CommExecuteFunctionType::Name**

The name parameter, see table.

**7.6.2.3 byte CommExecuteFunctionType::Param1**

The first parameter, see table.

**7.6.2.4 byte CommExecuteFunctionType::Param2**

The second parameter, see table.

**7.6.2.5 byte CommExecuteFunctionType::Param3**

The third parameter, see table.

**7.6.2.6 unsigned int CommExecuteFunctionType::Result**

The function call result. Possible values include [Loader module error codes](#).

**7.6.2.7 unsigned int CommExecuteFunctionType::RetVal**

The function call return value. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.7 CommHSCheckStatusType Struct Reference**

Parameters for the CommHSCheckStatus system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- byte [SendingData](#)
- byte [DataAvailable](#)

### 7.7.1 Detailed Description

Parameters for the CommHSCheckStatus system call. This structure is used when calling the [SysCommHSCheckStatus](#) system call function.

**See also:**

[SysCommHSCheckStatus\(\)](#)

**Examples:**

[ex\\_SysCommHSCheckStatus.nxc](#).

### 7.7.2 Field Documentation

#### 7.7.2.1 byte CommHSCheckStatusType::DataAvailable

Number of bytes of data available for reading.

**Examples:**

[ex\\_SysCommHSCheckStatus.nxc](#).

#### 7.7.2.2 byte CommHSCheckStatusType::SendingData

Number of bytes of data currently being sent.

**Examples:**

[ex\\_SysCommHSCheckStatus.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.8 CommHSControlType Struct Reference

Parameters for the CommHSControl system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)

- byte [Command](#)
- byte [BaudRate](#)
- unsigned int [Mode](#)

### 7.8.1 Detailed Description

Parameters for the CommHSControl system call. This structure is used when calling the [SysCommHSControl](#) system call function.

See also:

[SysCommHSControl\(\)](#)

Examples:

[ex\\_SysCommHSControl.nxc](#).

### 7.8.2 Field Documentation

#### 7.8.2.1 byte CommHSControlType::BaudRate

The hi-speed port baud rate. See [Hi-speed port baud rate constants](#).

#### 7.8.2.2 byte CommHSControlType::Command

The hi-speed port configuration command. See [Hi-speed port SysCommHSControl constants](#).

Examples:

[ex\\_SysCommHSControl.nxc](#).

#### 7.8.2.3 unsigned int CommHSControlType::Mode

The hi-speed port mode. See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

#### 7.8.2.4 char CommHSControlType::Result

The function call result.

**Todo**

values?

**Examples:**

[ex\\_SysCommHSControl.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.9 CommHSReadWriteType Struct Reference

Parameters for the CommHSReadWrite system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Status](#)
- byte [Buffer](#) []
- byte [BufferLen](#)

### 7.9.1 Detailed Description

Parameters for the CommHSReadWrite system call. This structure is used when calling the [SysCommHSRead](#) and [SysCommHSWrite](#) system call functions.

**See also:**

[SysCommHSRead\(\)](#), [SysCommHSWrite\(\)](#)

**Examples:**

[ex\\_SysCommHSRead.nxc](#), and [ex\\_SysCommHSWrite.nxc](#).

### 7.9.2 Field Documentation

#### 7.9.2.1 byte CommHSReadWriteType::Buffer[]

The buffer of data to write or to contain the data read from the hi-speed port.

**Examples:**

[ex\\_SysCommHSRead.nxc](#), and [ex\\_SysCommHSWrite.nxc](#).

### 7.9.2.2 byte CommHSReadWriteType::BufferLen

The size of the output buffer on input. Determines the maximum number of bytes read from the hi-speed port. This field is not updated during the function call and it is only used for the Read operation.

### 7.9.2.3 char CommHSReadWriteType::Status

The result of the function call.

#### Examples:

[ex\\_SysCommHSRead.nxc](#), and [ex\\_SysCommHSWrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.10 CommLSCheckStatusType Struct Reference

Parameters for the CommLSCheckStatus system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [BytesReady](#)

### 7.10.1 Detailed Description

Parameters for the CommLSCheckStatus system call. This structure is used when calling the [SysCommLSCheckStatus](#) system call function.

#### See also:

[SysCommLSCheckStatus\(\)](#)

#### Examples:

[ex\\_syscommLSCheckStatus.nxc](#).

## 7.10.2 Field Documentation

### 7.10.2.1 byte CommLSCheckStatusType::BytesReady

The number of bytes ready to read from the specified port.

### 7.10.2.2 byte CommLSCheckStatusType::Port

The port to which the I2C device is connected.

#### Examples:

[ex\\_syscommLScheckstatus.nxc](#).

### 7.10.2.3 char CommLSCheckStatusType::Result

The function call result.

Possible values include [ERR\\_COMM\\_BUS\\_ERR](#), [ERR\\_COMM\\_CHAN\\_INVALID](#), [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#), [STAT\\_COMM\\_PENDING](#), and [NO\\_ERR](#).

#### Examples:

[ex\\_syscommLScheckstatus.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.11 CommLSReadType Struct Reference

Parameters for the CommLSRead system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [BufferLen](#)

### 7.11.1 Detailed Description

Parameters for the CommLSRead system call. This structure is used when calling the [SysCommLSRead](#) system call function.

**See also:**

[SysCommLSRead\(\)](#)

**Examples:**

[ex\\_syscommlsread.nxc](#).

### 7.11.2 Field Documentation

#### 7.11.2.1 byte CommLSReadType::Buffer[ ]

The buffer used to store the bytes read from the I2C device.

**Examples:**

[ex\\_syscommlsread.nxc](#).

#### 7.11.2.2 byte CommLSReadType::BufferLen

The size of the output buffer on input. This field is not updated during the function call.

**Examples:**

[ex\\_syscommlsread.nxc](#).

#### 7.11.2.3 byte CommLSReadType::Port

The port to which the I2C device is connected.

**Examples:**

[ex\\_syscommlsread.nxc](#).

#### 7.11.2.4 char CommLSReadType::Result

The function call result. Possible values include [ERR\\_COMM\\_-BUS\\_ERR](#), [ERR\\_COMM\\_CHAN\\_INVALID](#), [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#), [ERR\\_INVALID\\_SIZE](#), [STAT\\_COMM\\_PENDING](#), and [NO\\_ERR](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.12 CommLSWriteExType Struct Reference

Parameters for the CommLSWriteEx system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)
- bool [NoRestartOnRead](#)

### 7.12.1 Detailed Description

Parameters for the CommLSWriteEx system call. This structure is used when calling the [SysCommLSWriteEx](#) system call function.

**See also:**

[SysCommLSWriteEx\(\)](#)

**Examples:**

[ex\\_syscommmlswriteex.nxc.](#)

### 7.12.2 Field Documentation

#### 7.12.2.1 byte CommLSWriteExType::Buffer[]

The buffer written to the I2C device.

**Examples:**

[ex\\_syscommmlswriteex.nxc.](#)

#### 7.12.2.2 bool CommLSWriteExType::NoRestartOnRead

Should a restart occur before reading from the device?

##### Examples:

[ex\\_syscommlswriteex.nxc](#).

#### 7.12.2.3 byte CommLSWriteExType::Port

The port to which the I2C device is connected.

##### Examples:

[ex\\_syscommlswriteex.nxc](#).

#### 7.12.2.4 char CommLSWriteExType::Result

The function call result. Possible values include [ERR\\_COMM\\_CHAN\\_INVALID](#), [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#), [ERR\\_INVALID\\_SIZE](#), and [NO\\_ERR](#).

##### Examples:

[ex\\_syscommlswriteex.nxc](#).

#### 7.12.2.5 byte CommLSWriteExType::ReturnLen

The number of bytes that you want to read from the I2C device.

##### Examples:

[ex\\_syscommlswriteex.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.13 CommLSWriteType Struct Reference

Parameters for the CommLSWrite system call.

```
#include <NXCDefs.h>
```

## Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)

### 7.13.1 Detailed Description

Parameters for the CommLSWrite system call. This structure is used when calling the [SysCommLSWrite](#) system call function.

#### See also:

[SysCommLSWrite\(\)](#)

#### Examples:

[ex\\_syscommlswrite.nxc](#).

### 7.13.2 Field Documentation

#### 7.13.2.1 byte CommLSWriteType::Buffer[]

The buffer containing data to be written to the I2C device.

#### Examples:

[ex\\_syscommlswrite.nxc](#).

#### 7.13.2.2 byte CommLSWriteType::Port

The port to which the I2C device is connected.

#### Examples:

[ex\\_syscommlswrite.nxc](#).

#### 7.13.2.3 char CommLSWriteType::Result

The function call result. Possible values include [ERR\\_COMM\\_CHAN\\_INVALID](#), [ERR\\_COMM\\_CHAN\\_NOT\\_READY](#), [ERR\\_INVALID\\_SIZE](#), and [NO\\_ERR](#).

#### 7.13.2.4 byte CommLSWriteType::ReturnLen

The number of bytes that you want to read from the I2C device after writing the data. If no read is planned set this to zero.

#### Examples:

[ex\\_syscommmlswrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.14 ComputeCalibValueType Struct Reference

Parameters for the ComputeCalibValue system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- string [Name](#)
- unsigned int [RawVal](#)

#### 7.14.1 Detailed Description

Parameters for the ComputeCalibValue system call. This structure is used when calling the [SysComputeCalibValue](#) system call function.

#### See also:

[SysComputeCalibValue\(\)](#)

#### Examples:

[ex\\_SysComputeCalibValue.nxc](#).

#### 7.14.2 Field Documentation

##### 7.14.2.1 string ComputeCalibValueType::Name

The name of the sensor calibration cache.

**Todo**

?

**Examples:**

[ex\\_SysComputeCalibValue.nxc](#).

**7.14.2.2 unsigned int ComputeCalibValueType::RawVal**

The raw value.

**Todo**

?

**Examples:**

[ex\\_SysComputeCalibValue.nxc](#).

**7.14.2.3 byte ComputeCalibValueType::Result**

The function call result.

**Todo**

?

**Examples:**

[ex\\_SysComputeCalibValue.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.15 DatalogGetTimesType Struct Reference**

Parameters for the DatalogGetTimes system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- unsigned long [SyncTime](#)
- unsigned long [SyncTick](#)

### 7.15.1 Detailed Description

Parameters for the DatalogGetTimes system call. This structure is used when calling the [SysDatalogGetTimes](#) system call function.

**See also:**

[SysDatalogGetTimes\(\)](#)

**Examples:**

[ex\\_sysdataloggettimes.nxc](#).

### 7.15.2 Field Documentation

#### 7.15.2.1 unsigned long DatalogGetTimesType::SyncTick

The datalog synchronized tick.

**Examples:**

[ex\\_sysdataloggettimes.nxc](#).

#### 7.15.2.2 unsigned long DatalogGetTimesType::SyncTime

The datalog synchronized time.

**Examples:**

[ex\\_sysdataloggettimes.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.16 DatalogWriteType Struct Reference

Parameters for the DatalogWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Message](#) []

### 7.16.1 Detailed Description

Parameters for the DatalogWrite system call. This structure is used when calling the [SysDatalogWrite](#) system call function.

See also:

[SysDatalogWrite\(\)](#)

Examples:

[ex\\_SysDatalogWrite.nxc](#).

### 7.16.2 Field Documentation

#### 7.16.2.1 byte DatalogWriteType::Message[ ]

A buffer containing data to write to the datalog.

Examples:

[ex\\_SysDatalogWrite.nxc](#).

#### 7.16.2.2 char DatalogWriteType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

Examples:

[ex\\_SysDatalogWrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.17 DisplayExecuteFunctionType Struct Reference

Parameters for the DisplayExecuteFunction system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- byte [Status](#)

- byte [Cmd](#)
- bool [On](#)
- byte [X1](#)
- byte [Y1](#)
- byte [X2](#)
- byte [Y2](#)

### 7.17.1 Detailed Description

Parameters for the DisplayExecuteFunction system call. This structure is used when calling the [SysDisplayExecuteFunction](#) system call function.

The fields usage depends on the requested command and are documented in the table below. If a field member is shown as 'x' it is ignored by the specified display command.

| Cmd                          | Meaning                 | Expected parameters           |
|------------------------------|-------------------------|-------------------------------|
| DISPLAY_ERASE_ALL            | erase entire screen     | ()                            |
| DISPLAY_PIXEL                | set pixel (on/off)      | (true/false,X1,Y1,x,x)        |
| DISPLAY_-<br>HORIZONTAL_LINE | draw horizontal line    | (true/false,X1,Y1,X2,x)       |
| DISPLAY_-<br>VERTICAL_LINE   | draw vertical line      | (true/false,X1,Y1,x,Y2)       |
| DISPLAY_CHAR                 | draw char (actual font) | (true/false,X1,Y1,Char,x)     |
| DISPLAY_ERASE_-<br>LINE      | erase a single line     | (x,LINE,x,x,x)                |
| DISPLAY_FILL_-<br>REGION     | fill screen region      | (true/-<br>false,X1,Y1,X2,Y2) |
| DISPLAY_FILLED_-<br>FRAME    | draw a frame (on / off) | (true/-<br>false,X1,Y1,X2,Y2) |

See also:

[SysDisplayExecuteFunction\(\)](#)

Examples:

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

### 7.17.2 Field Documentation

#### 7.17.2.1 byte DisplayExecuteFunctionType::Cmd

The command to execute.

Examples:

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

### 7.17.2.2 bool DisplayExecuteFunctionType::On

The On parameter, see table.

#### Examples:

[ex\\_dispfunc.nxc](#).

### 7.17.2.3 byte DisplayExecuteFunctionType::Status

The function call result, always [NO\\_ERR](#).

### 7.17.2.4 byte DisplayExecuteFunctionType::X1

The X1 parameter, see table.

#### Examples:

[ex\\_dispfunc.nxc](#).

### 7.17.2.5 byte DisplayExecuteFunctionType::X2

The X2 parameter, see table.

#### Examples:

[ex\\_dispfunc.nxc](#).

### 7.17.2.6 byte DisplayExecuteFunctionType::Y1

The Y1 parameter, see table.

#### Examples:

[ex\\_dispfunc.nxc](#).

### 7.17.2.7 byte DisplayExecuteFunctionType::Y2

The Y2 parameter, see table.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.18 `div_t` Struct Reference

Output type of the `div` function.

```
#include <NXCDefs.h>
```

### Data Fields

- `int quot`
- `int rem`

### 7.18.1 Detailed Description

Output type of the `div` function. `div_t` structure. Structure used to represent the value of an integral division performed by `div`. It has two members of the same type, defined in either order as: `int quot`; `int rem`;

#### See also:

[div\(\)](#)

#### Examples:

[ex\\_div.nxc](#).

### 7.18.2 Field Documentation

#### 7.18.2.1 `int div_t::quot`

Represents the quotient of the integral division operation performed by `div`, which is the integer of lesser magnitude that is nearest to the algebraic quotient.

#### Examples:

[ex\\_div.nxc](#).

#### 7.18.2.2 `int div_t::rem`

Represents the remainder of the integral division operation performed by `div`, which is the integer resulting from subtracting `quot` to the numerator of the operation.

#### Examples:

[ex\\_div.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.19 DrawCircleType Struct Reference

Parameters for the DrawCircle system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType Center](#)
- byte [Size](#)
- unsigned long [Options](#)

### 7.19.1 Detailed Description

Parameters for the DrawCircle system call. This structure is used when calling the [SysDrawCircle](#) system call function. It lets you specify the center of the circle to draw using the [LocationType](#) structure member, the radius, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawCircle\(\)](#)

Examples:

[ex\\_sysdrawcircle.nxc](#).

### 7.19.2 Field Documentation

#### 7.19.2.1 LocationType DrawCircleType::Center

The location of the circle center.

Examples:

[ex\\_sysdrawcircle.nxc](#).

### 7.19.2.2 unsigned long DrawCircleType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

#### Examples:

[ex\\_sysdrawcircle.nxc](#).

### 7.19.2.3 char DrawCircleType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

### 7.19.2.4 byte DrawCircleType::Size

The circle radius.

#### Examples:

[ex\\_sysdrawcircle.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.20 DrawEllipseType Struct Reference

Parameters for the DrawEllipse system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType Center](#)
- byte [SizeX](#)
- byte [SizeY](#)
- unsigned long [Options](#)

### 7.20.1 Detailed Description

Parameters for the DrawEllipse system call. This structure is used when calling the [SysDrawEllipse](#) system call function. It lets you specify the center of the ellipse using the [LocationType](#) structure member, the x and y axis radii, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawEllipse\(\)](#)

Examples:

[ex\\_SysDrawEllipse.nxc](#).

### 7.20.2 Field Documentation

#### 7.20.2.1 LocationType DrawEllipseType::Center

The location of the ellipse center.

Examples:

[ex\\_SysDrawEllipse.nxc](#).

#### 7.20.2.2 unsigned long DrawEllipseType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex\\_SysDrawEllipse.nxc](#).

#### 7.20.2.3 char DrawEllipseType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 7.20.2.4 byte DrawEllipseType::SizeX

The horizontal ellipse radius.

Examples:

[ex\\_SysDrawEllipse.nxc](#).

#### 7.20.2.5 byte DrawEllipseType::SizeY

The vertical ellipse radius.

**Examples:**

[ex\\_SysDrawEllipse.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.21 DrawFontType Struct Reference

Parameters for the DrawFont system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Result](#)
- [LocationType](#) Location
- string [Filename](#)
- string [Text](#)
- unsigned long [Options](#)

### 7.21.1 Detailed Description

Parameters for the DrawFont system call. This structure is used when calling the [SysDrawFont](#) system call function. It lets you specify the text to draw, the LCD line and horizontal position using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

**See also:**

[SysDrawFont\(\)](#)

**Examples:**

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

### 7.21.2 Field Documentation

#### 7.21.2.1 string DrawFontType::Filename

The filename of the RIC-based font file.

**Examples:**

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

### 7.21.2.2 LocationType DrawFontType::Location

The location in X, LCD line number coordinates.

#### Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

### 7.21.2.3 unsigned long DrawFontType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

#### Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

### 7.21.2.4 char DrawFontType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

### 7.21.2.5 string DrawFontType::Text

The text to draw on the LCD.

#### Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.22 DrawGraphicArrayType Struct Reference

Parameters for the DrawGraphicArray system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- byte [Data](#) []
- long [Variables](#) []
- unsigned long [Options](#)

### 7.22.1 Detailed Description

Parameters for the DrawGraphicArray system call. This structure is used when calling the [SysDrawGraphicArray](#) system call function. It lets you specify the screen location at which to draw the image using the [LocationType](#) structure member, the graphic image data array, the image parameters (if needed), as well as drawing options defined in the [Drawing option constants](#) group.

**See also:**

[SysDrawGraphicArray\(\)](#)

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

### 7.22.2 Field Documentation

#### 7.22.2.1 byte DrawGraphicArrayType::Data[ ]

A byte array containing the RIC opcodes. [RIC Macro Wrappers](#)

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

#### 7.22.2.2 LocationType DrawGraphicArrayType::Location

The location on screen.

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

#### 7.22.2.3 unsigned long DrawGraphicArrayType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

**Examples:**

[ex\\_dispgout.nxc](#).

#### 7.22.2.4 char DrawGraphicArrayType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

### 7.22.2.5 long DrawGraphicArrayType::Variables[ ]

The variables passed as RIC arguments.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.23 DrawGraphicType Struct Reference

Parameters for the DrawGraphic system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- string [Filename](#)
- long [Variables](#) [ ]
- unsigned long [Options](#)

### 7.23.1 Detailed Description

Parameters for the DrawGraphic system call. This structure is used when calling the [SysDrawGraphic](#) system call function. It lets you specify the screen location at which to draw the image using the [LocationType](#) structure member, the filename of the graphic image, the image parameters (if needed), as well as drawing options defined in the [Drawing option constants](#) group.

#### See also:

[SysDrawGraphic\(\)](#)

#### Examples:

[ex\\_sysdrawgraphic.nxc](#).

### 7.23.2 Field Documentation

#### 7.23.2.1 string DrawGraphicType::Filename

The RIC file name.

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

**7.23.2.2 LocationType DrawGraphicType::Location**

The location on screen.

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

**7.23.2.3 unsigned long DrawGraphicType::Options**

The options to use when writing to the LCD. [Drawing option constants](#)

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

**7.23.2.4 char DrawGraphicType::Result**

The function call result. Possible values include [Loader module error codes](#), [ERR\\_FILE](#), and [NO\\_ERR](#).

**7.23.2.5 long DrawGraphicType::Variables[ ]**

The variables passed as RIC arguments.

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.24 DrawLineType Struct Reference**

Parameters for the DrawLine system call.

```
#include <NXCDefs.h>
```

## Data Fields

- char [Result](#)
- [LocationType](#) StartLoc
- [LocationType](#) EndLoc
- unsigned long [Options](#)

### 7.24.1 Detailed Description

Parameters for the DrawLine system call. This structure is used when calling the [SysDrawLine](#) system call function. It lets you specify the end points of the line to draw using two [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

#### See also:

[SysDrawLine\(\)](#)

#### Examples:

[ex\\_sysdrawline.nxc](#).

### 7.24.2 Field Documentation

#### 7.24.2.1 [LocationType](#) DrawLineType::EndLoc

The location of the ending point.

#### Examples:

[ex\\_sysdrawline.nxc](#).

#### 7.24.2.2 unsigned long DrawLineType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

#### Examples:

[ex\\_sysdrawline.nxc](#).

#### 7.24.2.3 char DrawLineType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 7.24.2.4 LocationType DrawLineType::StartLoc

The location of the starting point.

#### Examples:

[ex\\_sysdrawline.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.25 DrawPointType Struct Reference

Parameters for the DrawPoint system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- unsigned long [Options](#)

#### 7.25.1 Detailed Description

Parameters for the DrawPoint system call. This structure is used when calling the [SysDrawPoint](#) system call function. It lets you specify the pixel to draw using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

#### See also:

[SysDrawPoint\(\)](#)

#### Examples:

[ex\\_sysdrawpoint.nxc](#).

#### 7.25.2 Field Documentation

##### 7.25.2.1 LocationType DrawPointType::Location

The point location on screen.

**Examples:**

[ex\\_sysdrawpoint.nxc](#).

**7.25.2.2 unsigned long DrawPointType::Options**

The options to use when writing to the LCD. [Drawing option constants](#)

**Examples:**

[ex\\_sysdrawpoint.nxc](#).

**7.25.2.3 char DrawPointType::Result**

The function call result. [NO\\_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.26 DrawPolygonType Struct Reference**

Parameters for the DrawPolygon system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Result](#)
- [LocationType](#) Points []
- unsigned long [Options](#)

**7.26.1 Detailed Description**

Parameters for the DrawPolygon system call. This structure is used when calling the [SysDrawPolygon](#) system call function. It lets you specify the points of the polygon to draw using the [LocationType](#) array structure member, as well as drawing options defined in the [Drawing option constants](#) group.

**See also:**

[SysDrawPolygon\(\)](#)

**Examples:**

[ex\\_sysdrawpolygon.nxc](#).

**7.26.2 Field Documentation****7.26.2.1 unsigned long DrawPolygonType::Options**

The options to use when writing to the LCD. [Drawing option constants](#)

**Examples:**

[ex\\_sysdrawpolygon.nxc](#).

**7.26.2.2 LocationType DrawPolygonType::Points[ ]**

An array of [LocationType](#) structures which define the polygon's shape.

**Examples:**

[ex\\_sysdrawpolygon.nxc](#).

**7.26.2.3 char DrawPolygonType::Result**

The function call result. [NO\\_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.27 DrawRectType Struct Reference**

Parameters for the DrawRect system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Result](#)
- [LocationType](#) [Location](#)
- [SizeType](#) [Size](#)
- unsigned long [Options](#)

### 7.27.1 Detailed Description

Parameters for the DrawRect system call. This structure is used when calling the [SysDrawRect](#) system call function. It lets you specify the corner of the rectangle using the [LocationType](#) structure member, the width and height of the rectangle using the [SizeType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawRect\(\)](#)

Examples:

[ex\\_sysdrawrect.nxc](#).

### 7.27.2 Field Documentation

#### 7.27.2.1 LocationType DrawRectType::Location

The top left corner location.

Examples:

[ex\\_sysdrawrect.nxc](#).

#### 7.27.2.2 unsigned long DrawRectType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex\\_sysdrawrect.nxc](#).

#### 7.27.2.3 char DrawRectType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 7.27.2.4 SizeType DrawRectType::Size

The width and height of the rectangle.

Examples:

[ex\\_sysdrawrect.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.28 DrawTextType Struct Reference

Parameters for the DrawText system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Location
- string [Text](#)
- unsigned long [Options](#)

### 7.28.1 Detailed Description

Parameters for the DrawText system call. This structure is used when calling the [SysDrawText](#) system call function. It lets you specify the text to draw, the LCD line and horizontal position using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawText\(\)](#)

Examples:

[ex\\_syscall.nxc](#), and [ex\\_sysdrawtext.nxc](#).

### 7.28.2 Field Documentation

#### 7.28.2.1 LocationType DrawTextType::Location

The location in X, LCD line number coordinates.

Examples:

[ex\\_syscall.nxc](#), and [ex\\_sysdrawtext.nxc](#).

### 7.28.2.2 unsigned long DrawTextType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

#### Examples:

[ex\\_sysdrawtext.nxc](#).

### 7.28.2.3 char DrawTextType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

### 7.28.2.4 string DrawTextType::Text

The text to draw on the LCD.

#### Examples:

[ex\\_syscall.nxc](#), and [ex\\_sysdrawtext.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.29 FileCloseType Struct Reference

Parameters for the FileClose system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)

### 7.29.1 Detailed Description

Parameters for the FileClose system call. This structure is used when calling the [SysFileClose](#) system call function.

#### See also:

[SysFileClose\(\)](#)

**Examples:**

[ex\\_sysfileclose.nxc.](#)

**7.29.2 Field Documentation****7.29.2.1 byte FileCloseType::FileHandle**

The file handle to close.

**Examples:**

[ex\\_sysfileclose.nxc.](#)

**7.29.2.2 unsigned int FileCloseType::Result**

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.30 FileDeleteType Struct Reference**

Parameters for the FileDelete system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- unsigned int [Result](#)
- string [Filename](#)

**7.30.1 Detailed Description**

Parameters for the FileDelete system call. This structure is used when calling the [SysFileDelete](#) system call function.

**See also:**

[SysFileDelete\(\)](#)

**Examples:**

[ex\\_sysfiledelete.nxc.](#)

### 7.30.2 Field Documentation

#### 7.30.2.1 string FileDeleteType::Filename

The name of the file to delete.

#### Examples:

[ex\\_sysfiledelete.nxc](#).

#### 7.30.2.2 unsigned int FileDeleteType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.31 FileFindType Struct Reference

Parameters for the FileFind system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Filename](#)
- unsigned long [Length](#)

#### 7.31.1 Detailed Description

Parameters for the FileFind system call. This structure is used when calling the [SysFileFindFirst](#) and [SysFileFindNext](#) system call functions.

#### See also:

[SysFileFindFirst\(\)](#) and [SysFileFindNext\(\)](#)

#### Examples:

[ex\\_sysfilefindfirst.nxc](#), and [ex\\_sysfilefindnext.nxc](#).

### 7.31.2 Field Documentation

#### 7.31.2.1 byte FileFindType::FileHandle

The returned file handle to be used to continue iterations. Close it after usage.

#### Examples:

[ex\\_sysfilefindnext.nxc](#).

#### 7.31.2.2 string FileFindType::Filename

The pattern to match file name, then the returned found file name.

#### Examples:

[ex\\_sysfilefindfirst.nxc](#), and [ex\\_sysfilefindnext.nxc](#).

#### 7.31.2.3 unsigned long FileFindType::Length

The found file length.

#### 7.31.2.4 unsigned int FileFindType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.32 FileOpenType Struct Reference

Parameters for the FileOpen system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Filename](#)
- unsigned long [Length](#)

### 7.32.1 Detailed Description

Parameters for the FileOpen system call. This structure is used when calling the [SysFileOpenAppend](#), [SysFileOpenRead](#), [SysFileOpenWrite](#), [SysFileOpenReadLinear](#), [SysFileOpenWriteLinear](#) and [SysFileOpenWriteNonLinear](#) system call functions.

**See also:**

[SysFileOpenAppend\(\)](#), [SysFileOpenRead\(\)](#), [SysFileOpenWrite\(\)](#), [SysFileOpenReadLinear\(\)](#), [SysFileOpenWriteLinear\(\)](#)

**Examples:**

[ex\\_sysfileopenappend.nxc](#), [ex\\_sysfileopenread.nxc](#), [ex\\_sysfileopenreadlinear.nxc](#),  
[ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), and [ex\\_sysfileopenwritenonlinear.nxc](#).

### 7.32.2 Field Documentation

#### 7.32.2.1 byte FileOpenType::FileHandle

The returned file handle to use for subsequent file operations.

#### 7.32.2.2 string FileOpenType::Filename

The name of the file to open or create.

**Examples:**

[ex\\_sysfileopenappend.nxc](#), [ex\\_sysfileopenread.nxc](#), [ex\\_sysfileopenreadlinear.nxc](#),  
[ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), and [ex\\_sysfileopenwritenonlinear.nxc](#).

#### 7.32.2.3 unsigned long FileOpenType::Length

For [SysFileOpenWrite\(\)](#), [SysFileOpenWriteLinear\(\)](#) and [SysFileOpenWriteNonLinear\(\)](#): the desired maximum file capacity.

For [SysFileOpenAppend\(\)](#), [SysFileOpenRead\(\)](#) and [SysFileOpenReadLinear\(\)](#): the returned available length in the file.

**Examples:**

[ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), and [ex\\_sysfileopenwritenonlinear.nxc](#).

#### 7.32.2.4 unsigned int FileOpenType::Result

The function call result. Possible values include [Loader module error codes](#).

#### Examples:

[ex\\_sysfileopenappend.nxc](#), [ex\\_sysfileopenread.nxc](#), [ex\\_sysfileopenreadlinear.nxc](#),  
[ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), and [ex\\_-  
sysfileopenwritenonlinear.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

### 7.33 FileReadWriteType Struct Reference

Parameters for the FileReadWrite system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Buffer](#)
- unsigned long [Length](#)

#### 7.33.1 Detailed Description

Parameters for the FileReadWrite system call. This structure is used when calling the [SysFileRead](#) and [SysFileWrite](#) system call functions.

#### See also:

[SysFileRead\(\)](#) and [SysFileWrite\(\)](#)

#### Examples:

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

#### 7.33.2 Field Documentation

##### 7.33.2.1 string FileReadWriteType::Buffer

The buffer to store read bytes or containing bytes to write.

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

**7.33.2.2 byte FileReadWriteType::FileHandle**

The file handle to access.

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

**7.33.2.3 unsigned long FileReadWriteType::Length**

The number of bytes to read or the returned number of bytes written.

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

**7.33.2.4 unsigned int FileReadWriteType::Result**

The function call result. Possible values include [Loader module error codes](#).

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.34 FileRenameType Struct Reference**

Parameters for the FileRename system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- unsigned int [Result](#)
- string [OldFilename](#)
- string [NewFilename](#)

### 7.34.1 Detailed Description

Parameters for the FileRename system call. This structure is used when calling the [SysFileRename](#) system call function.

See also:

[SysFileRename\(\)](#)

Examples:

[ex\\_sysfilerename.nxc](#).

### 7.34.2 Field Documentation

#### 7.34.2.1 string FileRenameType::NewFilename

The new name to give to the file.

Examples:

[ex\\_sysfilerename.nxc](#).

#### 7.34.2.2 string FileRenameType::OldFilename

The name of the file to be renamed.

Examples:

[ex\\_sysfilerename.nxc](#).

#### 7.34.2.3 unsigned int FileRenameType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex\\_sysfilerename.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.35 FileResizeType Struct Reference

Parameters for the FileResize system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- unsigned int [NewSize](#)

### 7.35.1 Detailed Description

Parameters for the FileResize system call. This structure is used when calling the [SysFileResize](#) system call function.

**See also:**

[SysFileResize\(\)](#)

**Examples:**

[ex\\_sysfileresize.nxc](#).

### 7.35.2 Field Documentation

#### 7.35.2.1 byte FileResizeType::FileHandle

The handle of the file to resize.

**Examples:**

[ex\\_sysfileresize.nxc](#).

#### 7.35.2.2 unsigned int FileResizeType::NewSize

The new file size.

**Examples:**

[ex\\_sysfileresize.nxc](#).

### 7.35.2.3 unsigned int FileResizeType::Result

The function call result. Possible values include [Loader module error codes](#).

#### Examples:

[ex\\_sysfileresize.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.36 FileResolveHandleType Struct Reference

Parameters for the FileResolveHandle system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- bool [WriteHandle](#)
- string [Filename](#)

### 7.36.1 Detailed Description

Parameters for the FileResolveHandle system call. This structure is used when calling the [SysFileResolveHandle](#) system call function.

#### See also:

[SysFileResolveHandle\(\)](#)

#### Examples:

[ex\\_sysfileresolvehandle.nxc](#).

### 7.36.2 Field Documentation

#### 7.36.2.1 byte FileResolveHandleType::FileHandle

The returned resolved file handle.

### 7.36.2.2 string FileResolveHandleType::Filename

The name of the file for which to resolve a handle.

#### Examples:

[ex\\_sysfileresolvehandle.nxc](#).

### 7.36.2.3 unsigned int FileResolveHandleType::Result

The function call result. Possible values include [LDR\\_HANDLEALREADYCLOSED](#) and [LDR\\_SUCCESS](#).

#### Examples:

[ex\\_sysfileresolvehandle.nxc](#).

### 7.36.2.4 bool FileResolveHandleType::WriteHandle

True if the returned handle is a write handle.

#### Examples:

[ex\\_sysfileresolvehandle.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.37 FileSeekType Struct Reference

Parameters for the FileSeek system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- byte [Origin](#)
- long [Length](#)

### 7.37.1 Detailed Description

Parameters for the FileSeek system call. This structure is used when calling the [SysFileSeek](#) system call function.

**See also:**

[SysFileSeek\(\)](#)

**Examples:**

[ex\\_sysfileseek.nxc](#).

### 7.37.2 Field Documentation

#### 7.37.2.1 byte FileSeekType::FileHandle

The handle of the file to seek in.

**Examples:**

[ex\\_sysfileseek.nxc](#).

#### 7.37.2.2 long FileSeekType::Length

The offset from the origin to seek to.

**Examples:**

[ex\\_sysfileseek.nxc](#).

#### 7.37.2.3 byte FileSeekType::Origin

The origin of the file seek operation. See [fseek origin constants](#).

**Examples:**

[ex\\_sysfileseek.nxc](#).

#### 7.37.2.4 unsigned int FileSeekType::Result

The function call result. Possible values include [Loader module error codes](#).

**Examples:**

[ex\\_sysfileseek.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.38 FileTellType Struct Reference

Parameters for the FileTell system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- unsigned int [Result](#)
- byte [FileHandle](#)
- unsigned long [Position](#)

### 7.38.1 Detailed Description

Parameters for the FileTell system call. This structure is used when calling the [Sys-FileTell](#) system call function.

**See also:**

[SysFileTell\(\)](#)

### 7.38.2 Field Documentation

#### 7.38.2.1 byte FileTellType::FileHandle

The handle of the open file.

#### 7.38.2.2 unsigned long FileTellType::Position

The current file position in the open file.

#### 7.38.2.3 unsigned int FileTellType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.39 GetStartTickType Struct Reference

Parameters for the GetStartTick system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned long [Result](#)

### 7.39.1 Detailed Description

Parameters for the GetStartTick system call. This structure is used when calling the [SysGetStartTick](#) system call function.

**See also:**

[SysGetStartTick\(\)](#)

**Examples:**

[ex\\_sysgetstarttick.nxc.](#)

### 7.39.2 Field Documentation

#### 7.39.2.1 unsigned long GetStartTickType::Result

The returned tick value.

**Examples:**

[ex\\_sysgetstarttick.nxc.](#)

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.40 InputPinFunctionType Struct Reference

Parameters for the InputPinFunction system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [Cmd](#)
- byte [Port](#)
- byte [Pin](#)
- byte [Data](#)

#### 7.40.1 Detailed Description

Parameters for the InputPinFunction system call. This structure is used when calling the [SysInputPinFunction](#) system call function.

#### See also:

[SysInputPinFunction\(\)](#)

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

#### 7.40.2 Field Documentation

##### 7.40.2.1 byte InputPinFunctionType::Cmd

The command to execute. See [Constants to use with the Input module's Pin function](#). You can add a microsecond wait after the command by ORing [INPUT\\_PINCMD\\_WAIT\(usec\)](#) with the command value. Wait times can range from 1 to 63 microseconds.

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

##### 7.40.2.2 byte InputPinFunctionType::Data

The pin value(s). This field is only used by the [INPUT\\_PINCMD\\_READ](#) command.

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

### 7.40.2.3 byte InputPinFunctionType::Pin

The digital pin(s). See [Input port digital pin constants](#). When setting pin direction you must OR the desired direction constant into this field. See INPUT\_PINDIR\_INPUT and INPUT\_PINDIR\_OUTPUT from the [Constants to use with the Input module's Pin function](#) group. You can OR together the digital pin constants to operate on both in a single call.

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

### 7.40.2.4 byte InputPinFunctionType::Port

The input port. See [Input port constants](#).

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

### 7.40.2.5 unsigned int InputPinFunctionType::Result

The function call result. Possible return values are ERR\_INVALID\_PORT or NO\_ERR.

#### Examples:

[ex\\_sysinputpinfunction.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.41 InputValueType Struct Reference

Parameters for the [RemoteGetInputValues](#) function.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Port](#)
- bool [Valid](#)

- bool [Calibrated](#)
- byte [SensorType](#)
- byte [SensorMode](#)
- unsigned int [RawValue](#)
- unsigned int [NormalizedValue](#)
- int [ScaledValue](#)
- int [CalibratedValue](#)

### 7.41.1 Detailed Description

Parameters for the [RemoteGetInputValues](#) function. This structure is used when calling the [RemoteGetInputValues](#) function. Choose the sensor port ([Input port constants](#)) and after calling the function read the sensor values from the various structure fields.

#### Examples:

[ex\\_RemoteGetInputValues.nxc](#).

### 7.41.2 Field Documentation

#### 7.41.2.1 bool InputValueType::Calibrated

Is the sensor calibrated?

#### 7.41.2.2 int InputValueType::CalibratedValue

The calibrated value.

#### 7.41.2.3 unsigned int InputValueType::NormalizedValue

The normalized value.

#### 7.41.2.4 byte InputValueType::Port

The sensor port. See the [Input port constants](#) group.

#### 7.41.2.5 unsigned int InputValueType::RawValue

The raw value.

### 7.41.2.6 int InputValueType::ScaledValue

The scaled value.

### 7.41.2.7 byte InputValueType::SensorMode

The sensor mode. See the [Sensor mode constants](#) group.

### 7.41.2.8 byte InputValueType::SensorType

The sensor type. See the [Sensor type constants](#) group.

### 7.41.2.9 bool InputValueType::Valid

Is the sensor value valid?

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.42 IOMapReadByIDType Struct Reference

Parameters for the IOMapReadByID system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [ModuleID](#)
- unsigned int [Offset](#)
- unsigned int [Count](#)
- byte [Buffer](#) []

### 7.42.1 Detailed Description

Parameters for the IOMapReadByID system call. This structure is used when calling the [SysIOMapReadByID](#) system call function.

#### See also:

[SysIOMapReadByID\(\)](#)

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**7.42.2 Field Documentation****7.42.2.1 byte IOMapReadByIDType::Buffer[ ]**

The buffer used to store read bytes.

**Examples:**

[ex\\_reladdressof.nxc](#).

**7.42.2.2 unsigned int IOMapReadByIDType::Count**

The number of bytes to read.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**7.42.2.3 unsigned long IOMapReadByIDType::ModuleID**

The identifier of the module to read from. See the [NXT firmware module IDs](#) group.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**7.42.2.4 unsigned int IOMapReadByIDType::Offset**

The offset in the module IOMap where to start reading.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

### 7.42.2.5 char IOMapReadByIDType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### Examples:

[ex\\_sysiomapreadbyid.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.43 IOMapReadType Struct Reference

Parameters for the IOMapRead system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- string [ModuleName](#)
- unsigned int [Offset](#)
- unsigned int [Count](#)
- byte [Buffer](#) [ ]

### 7.43.1 Detailed Description

Parameters for the IOMapRead system call. This structure is used when calling the [SysIOMapRead](#) system call function.

#### See also:

[SysIOMapRead\(\)](#)

#### Examples:

[ex\\_sysiomapread.nxc](#).

### 7.43.2 Field Documentation

#### 7.43.2.1 byte IOMapReadType::Buffer [ ]

The buffer used to store read bytes.

### 7.43.2.2 unsigned int IOMapReadType::Count

The number of bytes to read.

#### Examples:

[ex\\_sysiomapread.nxc](#).

### 7.43.2.3 string IOMapReadType::ModuleName

The name of the module to read from. See the [NXT firmware module names](#) group.

#### Examples:

[ex\\_sysiomapread.nxc](#).

### 7.43.2.4 unsigned int IOMapReadType::Offset

The offset in the module IOMap where to start reading.

#### Examples:

[ex\\_sysiomapread.nxc](#).

### 7.43.2.5 char IOMapReadType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### Examples:

[ex\\_sysiomapread.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.44 IOMapWriteByIDType Struct Reference

Parameters for the IOMapWriteByID system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [ModuleID](#)
- unsigned int [Offset](#)
- byte [Buffer](#) [ ]

#### 7.44.1 Detailed Description

Parameters for the IOMapWriteByID system call. This structure is used when calling the [SysIOMapWriteByID](#) system call function.

#### See also:

[SysIOMapWriteByID\(\)](#)

#### Examples:

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

#### 7.44.2 Field Documentation

##### 7.44.2.1 byte IOMapWriteByIDType::Buffer[ ]

The buffer containing bytes to write.

#### Examples:

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

##### 7.44.2.2 unsigned long IOMapWriteByIDType::ModuleID

The identifier of the module to write to. See the [NXT firmware module IDs](#) group.

#### Examples:

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

##### 7.44.2.3 unsigned int IOMapWriteByIDType::Offset

The offset in the module IOMap where to start writing.

#### Examples:

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

#### 7.44.2.4 char IOMapWriteByIDType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.45 IOMapWriteType Struct Reference

Parameters for the IOMapWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- string [ModuleName](#)
- unsigned int [Offset](#)
- byte [Buffer](#) []

### 7.45.1 Detailed Description

Parameters for the IOMapWrite system call. This structure is used when calling the [SysIOMapWrite](#) system call function.

**See also:**

[SysIOMapWrite\(\)](#)

**Examples:**

[ex\\_sysiomapwrite.nxc.](#)

### 7.45.2 Field Documentation

#### 7.45.2.1 byte IOMapWriteType::Buffer[]

The buffer containing bytes to write.

**Examples:**

[ex\\_sysiomapwrite.nxc.](#)

### 7.45.2.2 string IOMapWriteType::ModuleName

The name of the module to write to. See the [NXT firmware module names](#) group.

#### Examples:

[ex\\_sysiomapwrite.nxc](#).

### 7.45.2.3 unsigned int IOMapWriteType::Offset

The offset in the module IOMap where to start writing.

#### Examples:

[ex\\_sysiomapwrite.nxc](#).

### 7.45.2.4 char IOMapWriteType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.46 JoystickMessageType Struct Reference

The [JoystickMessageType](#) structure.

```
#include <NXCDefs.h>
```

#### Data Fields

- byte [JoystickDir](#)
- byte [LeftMotor](#)
- byte [RightMotor](#)
- byte [BothMotors](#)
- char [LeftSpeed](#)
- char [RightSpeed](#)
- unsigned long [Buttons](#)

### 7.46.1 Detailed Description

The `JoystickMessageType` structure. This structure is used to contain Joystick values read via the `JoystickMessageRead` API function.

#### Examples:

[ex\\_joystickmsg.nxc](#).

### 7.46.2 Field Documentation

#### 7.46.2.1 byte `JoystickMessageType::BothMotors`

The left and right motors. See [RCX output constants](#) for possible values.

#### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.2 unsigned long `JoystickMessageType::Buttons`

The joystick buttons pressed state.

#### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.3 byte `JoystickMessageType::JoystickDir`

The joystick direction or position. Ranges from 1 to 9, with the values representing numeric keypad buttons. 8 is up, 2 is down, 5 is center, etc.

#### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.4 byte `JoystickMessageType::LeftMotor`

The left motor. See [RCX output constants](#) for possible values.

#### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.5 char JoystickMessageType::LeftSpeed

The left motor speed (-100 to 100).

##### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.6 byte JoystickMessageType::RightMotor

The right motor. See [RCX output constants](#) for possible values.

##### Examples:

[ex\\_joystickmsg.nxc](#).

#### 7.46.2.7 char JoystickMessageType::RightSpeed

The right motor speed (-100 to 100).

##### Examples:

[ex\\_joystickmsg.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.47 KeepAliveType Struct Reference

Parameters for the KeepAlive system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned long [Result](#)

#### 7.47.1 Detailed Description

Parameters for the KeepAlive system call. This structure is used when calling the [SysKeepAlive](#) system call function.

**See also:**

[SysKeepAlive\(\)](#)

**Examples:**

[ex\\_syskeepalive.nxc.](#)

**7.47.2 Field Documentation****7.47.2.1 unsigned long KeepAliveType::Result**

The current sleep timeout in milliseconds.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.48 `ldiv_t` Struct Reference**

Output type of the `ldiv` function.

```
#include <NXCDefs.h>
```

**Data Fields**

- long [quot](#)
- long [rem](#)

**7.48.1 Detailed Description**

Output type of the `ldiv` function. Structure used to represent the value of an integral division performed by `ldiv`. It has two members of the same type, defined in either order as: `long quot`; `long rem`;

**See also:**

[ldiv\(\)](#)

**Examples:**

[ex\\_ldiv.nxc.](#)

### 7.48.2 Field Documentation

#### 7.48.2.1 long ldiv\_t::quot

Represents the quotient of the integral division operation performed by div, which is the integer of lesser magnitude that is nearest to the algebraic quotient.

**Examples:**

[ex\\_ldiv.nxc](#).

#### 7.48.2.2 long ldiv\_t::rem

Represents the remainder of the integral division operation performed by div, which is the integer resulting from subtracting quot to the numerator of the operation.

**Examples:**

[ex\\_ldiv.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.49 ListFilesType Struct Reference

Parameters for the ListFiles system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- string [Pattern](#)
- string [FileList](#) []

### 7.49.1 Detailed Description

Parameters for the ListFiles system call. This structure is used when calling the [SysListFiles](#) system call function.

**See also:**

[SysListFiles\(\)](#)

**Examples:**

[ex\\_syslistfiles.nxc](#).

**7.49.2 Field Documentation****7.49.2.1 string ListFilesType::FileList[ ]**

An array of strings containing the list of filenames that matched the file search pattern.

**Examples:**

[ex\\_syslistfiles.nxc](#).

**7.49.2.2 string ListFilesType::Pattern**

The file search pattern.

**Examples:**

[ex\\_syslistfiles.nxc](#).

**7.49.2.3 char ListFilesType::Result**

The function call result. Possible values include [Loader module error codes](#).

**Examples:**

[ex\\_syslistfiles.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.50 LoaderExecuteFunctionType Struct Reference**

Parameters for the LoaderExecuteFunction system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- unsigned int [Result](#)
- byte [Cmd](#)
- string [Filename](#)
- byte [Buffer](#) []
- unsigned long [Length](#)

**7.50.1 Detailed Description**

Parameters for the LoaderExecuteFunction system call. This structure is used when calling the [SysLoaderExecuteFunction](#) system call function.

The fields usage depends on the requested command and are documented in the table below.

| Cmd                      | Meaning                    | Expected Parameters        |
|--------------------------|----------------------------|----------------------------|
| LDR_CMD_-OPENREAD        | Open a file for reading    | (Filename, Length)         |
| LDR_CMD_-OPENWRITE       | Create a file              | (Filename, Length)         |
| LDR_CMD_READ             | Read from a file           | (Filename, Buffer, Length) |
| LDR_CMD_WRITE            | Write to a file            | (Filename, Buffer, Length) |
| LDR_CMD_CLOSE            | Close a file               | (Filename)                 |
| LDR_CMD_DELETE           | Delete a file              | (Filename)                 |
| LDR_CMD_-FINDFIRST       | Start iterating files      | (Filename, Buffer, Length) |
| LDR_CMD_-FINDNEXT        | Continue iterating files   | (Filename, Buffer, Length) |
| LDR_CMD_-OPENWRITELINEAR | Create a linear file       | (Filename, Length)         |
| LDR_CMD_-OPENREADLINEAR  | Read a linear file         | (Filename, Buffer, Length) |
| LDR_CMD_-OPENAPPENDDATA  | Open a file for writing    | (Filename, Length)         |
| LDR_CMD_-FINDFIRSTMODULE | Start iterating modules    | (Filename, Buffer)         |
| LDR_CMD_-FINDNEXTMODULE  | Continue iterating modules | (Buffer)                   |
| LDR_CMD_-CLOSEMODHANDLE  | Close module handle        | ()                         |
| LDR_CMD_-IOMAPREAD       | Read IOMap data            | (Filename, Buffer, Length) |
| LDR_CMD_-IOMAPWRITE      | Write IOMap data           | (Filename, Buffer, Length) |
| LDR_CMD_-DELETEUSERFLASH | Delete all files           | ()                         |
| LDR_CMD_-RENAMEFILE      | Rename file                | (Filename, Buffer, Length) |

**See also:**

[SysLoaderExecuteFunction\(\)](#)

**Examples:**

[ex\\_sysloaderexecutefunction.nxc](#).

## 7.50.2 Field Documentation

### 7.50.2.1 byte LoaderExecuteFunctionType::Buffer[ ]

The Buffer parameter, see table.

### 7.50.2.2 byte LoaderExecuteFunctionType::Cmd

The command to execute.

#### Examples:

[ex\\_sysloaderexecutefunction.nxc](#).

### 7.50.2.3 string LoaderExecuteFunctionType::Filename

The Filename parameter, see table.

### 7.50.2.4 unsigned long LoaderExecuteFunctionType::Length

The Length parameter, see table.

### 7.50.2.5 unsigned int LoaderExecuteFunctionType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.51 LocationType Struct Reference

A point on the NXT LCD screen.

```
#include <NXCDefs.h>
```

#### Data Fields

- int X
- int Y

### 7.51.1 Detailed Description

A point on the NXT LCD screen. This structure is by other system call structures to specify an X, Y LCD screen coordinate.

**See also:**

[DrawTextType](#), [DrawPointType](#), [DrawLineType](#), [DrawCircleType](#), [DrawRectType](#), [DrawGraphicType](#), [DrawGraphicArrayType](#), [DrawPolygonType](#), [DrawEllipseType](#), [DrawFontType](#)

**Examples:**

[ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

### 7.51.2 Field Documentation

#### 7.51.2.1 int LocationType::X

The X coordinate. Valid range is from 0 to 99 inclusive.

**Examples:**

[ex\\_dispfout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_syscall.nxc](#), [ex\\_sysdrawcircle.nxc](#), [ex\\_SysDrawEllipse.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_sysdrawgraphic.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysdrawline.nxc](#), [ex\\_sysdrawpoint.nxc](#), [ex\\_sysdrawrect.nxc](#), and [ex\\_sysdrawtext.nxc](#).

#### 7.51.2.2 int LocationType::Y

The Y coordinate. Valid range is from 0 to 63 inclusive. For text drawing this value must be a multiple of 8.

**Examples:**

[ex\\_dispfout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_syscall.nxc](#), [ex\\_sysdrawcircle.nxc](#), [ex\\_SysDrawEllipse.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_sysdrawgraphic.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysdrawline.nxc](#), [ex\\_sysdrawpoint.nxc](#), [ex\\_sysdrawrect.nxc](#), and [ex\\_sysdrawtext.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.52 MemoryManagerType Struct Reference

Parameters for the MemoryManager system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- bool [Compact](#)
- unsigned int [PoolSize](#)
- unsigned int [DataspaceSize](#)

### 7.52.1 Detailed Description

Parameters for the MemoryManager system call. This structure is used when calling the [SysMemoryManager](#) system call function.

**See also:**

[SysMemoryManager\(\)](#)

**Examples:**

[ex\\_sysmemorymanager.nxc.](#)

### 7.52.2 Field Documentation

#### 7.52.2.1 bool MemoryManagerType::Compact

Should the dataspace be compacted or not.

**Examples:**

[ex\\_sysmemorymanager.nxc.](#)

#### 7.52.2.2 unsigned int MemoryManagerType::DataspaceSize

The returned dataspace size.

**Examples:**

[ex\\_sysmemorymanager.nxc.](#)

### 7.52.2.3 unsigned int MemoryManagerType::PoolSize

The returned pool size.

#### Examples:

[ex\\_sysmemorymanager.nxc](#).

### 7.52.2.4 char MemoryManagerType::Result

The returned status value.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.53 MessageReadType Struct Reference

Parameters for the MessageRead system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- byte [QueueID](#)
- bool [Remove](#)
- string [Message](#)

### 7.53.1 Detailed Description

Parameters for the MessageRead system call. This structure is used when calling the [SysMessageRead](#) system call function.

#### See also:

[SysMessageRead\(\)](#)

#### Examples:

[ex\\_sysmessageread.nxc](#).

### 7.53.2 Field Documentation

#### 7.53.2.1 string MessageReadType::Message

The contents of the mailbox/queue.

**Examples:**

[ex\\_sysmessageread.nxc](#).

#### 7.53.2.2 byte MessageReadType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

**Examples:**

[ex\\_sysmessageread.nxc](#).

#### 7.53.2.3 bool MessageReadType::Remove

If true, remove the read message from the queue.

**Examples:**

[ex\\_sysmessageread.nxc](#).

#### 7.53.2.4 char MessageReadType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

**Examples:**

[ex\\_sysmessageread.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.54 MessageWriteType Struct Reference

Parameters for the MessageWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [QueueID](#)
- string [Message](#)

#### 7.54.1 Detailed Description

Parameters for the MessageWrite system call. This structure is used when calling the [SysMessageWrite](#) system call function.

#### See also:

[SysMessageWrite\(\)](#)

#### Examples:

[ex\\_sysmessagewrite.nxc.](#)

#### 7.54.2 Field Documentation

##### 7.54.2.1 string MessageWriteType::Message

The message to write.

#### Examples:

[ex\\_sysmessagewrite.nxc.](#)

##### 7.54.2.2 byte MessageWriteType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

#### Examples:

[ex\\_sysmessagewrite.nxc.](#)

##### 7.54.2.3 char MessageWriteType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.55 OutputStateType Struct Reference

Parameters for the [RemoteGetOutputState](#) function.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Port](#)
- char [Power](#)
- byte [Mode](#)
- byte [RegMode](#)
- char [TurnRatio](#)
- byte [RunState](#)
- unsigned long [TachoLimit](#)
- long [TachoCount](#)
- long [BlockTachoCount](#)
- long [RotationCount](#)

### 7.55.1 Detailed Description

Parameters for the [RemoteGetOutputState](#) function. This structure is used when calling the [RemoteGetOutputState](#) function. Choose the sensor port ([Output port constants](#)) and after calling the function read the output status values from the various structure fields.

#### Examples:

```
ex_RemoteGetOutputState.nxc.
```

### 7.55.2 Field Documentation

#### 7.55.2.1 long OutputStateType::BlockTachoCount

The current block tachometer count.

#### 7.55.2.2 byte OutputStateType::Mode

The output mode. See [Output port mode constants](#) group.

#### 7.55.2.3 byte OutputStateType::Port

The output port. See the [Output port constants](#) group.

**7.55.2.4 char OutputStateType::Power**

The output power level (-100..100).

**7.55.2.5 byte OutputStateType::RegMode**

The output regulation mode. See [Output port regulation mode constants](#) group.

**7.55.2.6 long OutputStateType::RotationCount**

The current rotation count.

**7.55.2.7 byte OutputStateType::RunState**

The output run state. See [Output port run state constants](#) group.

**7.55.2.8 long OutputStateType::TachoCount**

The current tachometer count.

**7.55.2.9 unsigned long OutputStateType::TachoLimit**

The tachometer limit.

**7.55.2.10 char OutputStateType::TurnRatio**

The output turning ratio (-100..100).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.56 RandomExType Struct Reference**

Parameters for the RandomEx system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- long [Seed](#)
- bool [ReSeed](#)

### 7.56.1 Detailed Description

Parameters for the RandomEx system call. This structure is used when calling the [SysRandomEx](#) system call function.

See also:

[SysRandomEx\(\)](#)

Examples:

[ex\\_sysrandomex.nxc](#).

### 7.56.2 Field Documentation

#### 7.56.2.1 bool RandomExType::ReSeed

A flag indicating whether or not to seed the random number generator.

#### 7.56.2.2 long RandomExType::Seed

The random number or the new seed value.

Examples:

[ex\\_sysrandomex.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.57 RandomNumberType Struct Reference

Parameters for the RandomNumber system call.

```
#include <NXCDefs.h>
```

### Data Fields

- int [Result](#)

### 7.57.1 Detailed Description

Parameters for the RandomNumber system call. This structure is used when calling the [SysRandomNumber](#) system call function.

**See also:**

[SysRandomNumber\(\)](#)

**Examples:**

[ex\\_sysrandomnumber.nxc](#).

**7.57.2 Field Documentation****7.57.2.1 int RandomNumberType::Result**

The random number.

**Examples:**

[ex\\_sysrandomnumber.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.58 ReadButtonType Struct Reference**

Parameters for the ReadButton system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Result](#)
- byte [Index](#)
- bool [Pressed](#)
- byte [Count](#)
- bool [Reset](#)

**7.58.1 Detailed Description**

Parameters for the ReadButton system call. This structure is used when calling the [SysReadButton](#) system call function.

**See also:**

[SysReadButton\(\)](#)

**Examples:**

[ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

**7.58.2 Field Documentation****7.58.2.1 byte ReadButtonType::Count**

The returned button pressed count.

**7.58.2.2 byte ReadButtonType::Index**

The requested button index. See the [Button name constants](#) group.

**Examples:**

[ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

**7.58.2.3 bool ReadButtonType::Pressed**

The returned button state.

**Examples:**

[ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

**7.58.2.4 bool ReadButtonType::Reset**

If true, the count is reset after reading.

**7.58.2.5 char ReadButtonType::Result**

The function call result, [ERR\\_INVALID\\_PORT](#) or [NO\\_ERR](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.59 ReadLastResponseType Struct Reference**

Parameters for the ReadLastResponse system call.

```
#include <NXCDefs.h>
```

## Data Fields

- char [Result](#)
- bool [Clear](#)
- byte [Length](#)
- byte [Command](#)
- byte [Buffer](#) []

### 7.59.1 Detailed Description

Parameters for the ReadLastResponse system call. This structure is used when calling the [SysReadLastResponse](#) system call function.

#### See also:

[SysReadLastResponse\(\)](#)

#### Examples:

[ex\\_SysReadLastResponse.nxc](#).

### 7.59.2 Field Documentation

#### 7.59.2.1 byte ReadLastResponseType::Buffer[]

The response packet buffer.

#### 7.59.2.2 bool ReadLastResponseType::Clear

Clear the response after reading it or not.

#### Examples:

[ex\\_SysReadLastResponse.nxc](#).

#### 7.59.2.3 byte ReadLastResponseType::Command

The response packet command byte.

#### Examples:

[ex\\_SysReadLastResponse.nxc](#).

#### 7.59.2.4 byte ReadLastResponseType::Length

The response packet length.

##### Examples:

[ex\\_SysReadLastResponse.nxc](#).

#### 7.59.2.5 char ReadLastResponseType::Result

The response packet status value.

##### Examples:

[ex\\_SysReadLastResponse.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.60 ReadSemDataType Struct Reference

Parameters for the ReadSemData system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [SemData](#)
- bool [Request](#)

#### 7.60.1 Detailed Description

Parameters for the ReadSemData system call. This structure is used when calling the [SysReadSemData](#) system call function.

##### See also:

[SysReadSemData\(\)](#)

##### Examples:

[ex\\_SysReadSemData.nxc](#).

## 7.60.2 Field Documentation

### 7.60.2.1 bool ReadSemDataType::Request

Which semaphore am I reading from, usage or request?

#### Examples:

[ex\\_SysReadSemData.nxc](#).

### 7.60.2.2 byte ReadSemDataType::SemData

The semaphore data returned by the function call.

#### Examples:

[ex\\_SysReadSemData.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.61 SetScreenModeType Struct Reference

Parameters for the SetScreenMode system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [ScreenMode](#)

### 7.61.1 Detailed Description

Parameters for the SetScreenMode system call. This structure is used when calling the [SysSetScreenMode](#) system call function.

#### See also:

[SysSetScreenMode\(\)](#)

#### Examples:

[ex\\_syssetscreenmode.nxc](#).

### 7.61.2 Field Documentation

#### 7.61.2.1 char SetScreenModeType::Result

The function call result, always [NO\\_ERR](#).

#### 7.61.2.2 unsigned long SetScreenModeType::ScreenMode

The requested screen mode.

The standard NXT firmware only supports setting the ScreenMode to [SCREEN\\_MODE\\_RESTORE](#).

If you install the NBC/NXC enhanced standard NXT firmware this system function also supports setting the ScreenMode to [SCREEN\\_MODE\\_CLEAR](#).

#### Examples:

[ex\\_syssetscreenmode.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.62 SetSleepTimeoutType Struct Reference

Parameters for the SetSleepTimeout system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [TheSleepTimeoutMS](#)

#### 7.62.1 Detailed Description

Parameters for the SetSleepTimeout system call. This structure is used when calling the [SysSetSleepTimeout](#) system call function.

#### See also:

[SysSetSleepTimeout\(\)](#)

#### Examples:

[ex\\_SysSetSleepTimeout.nxc](#).

### 7.62.2 Field Documentation

#### 7.62.2.1 char SetSleepTimeoutType::Result

The result of the system call function.

#### 7.62.2.2 unsigned long SetSleepTimeoutType::TheSleepTimeoutMS

The new sleep timeout value in milliseconds.

#### Examples:

[ex\\_SysSetSleepTimeout.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.63 SizeType Struct Reference

Width and height dimensions for the DrawRect system call.

```
#include <NXCDefs.h>
```

### Data Fields

- int [Width](#)
- int [Height](#)

### 7.63.1 Detailed Description

Width and height dimensions for the DrawRect system call. This structure is by the [DrawRectType](#) to specify a width and height for a rectangle.

#### See also:

[DrawRectType](#)

### 7.63.2 Field Documentation

#### 7.63.2.1 int SizeType::Height

The rectangle height.

**Examples:**

[ex\\_sysdrawrect.nxc](#).

**7.63.2.2 int SizeType::Width**

The rectangle width.

**Examples:**

[ex\\_sysdrawrect.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.64 SoundGetType Struct Reference**

Parameters for the SoundGetType system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- byte [State](#)
- byte [Flags](#)

**7.64.1 Detailed Description**

Parameters for the SoundGetType system call. This structure is used when calling the [SysSoundGetType](#) system call function.

**See also:**

[SysSoundGetType\(\)](#)

**Examples:**

[ex\\_sysoundgetstate.nxc](#).

**7.64.2 Field Documentation****7.64.2.1 byte SoundGetType::Flags**

The returned sound flags. See the [SoundFlags constants](#) group.

### 7.64.2.2 byte SoundGetStateType::State

The returned sound state. See the [SoundState constants](#) group.

#### Examples:

[ex\\_syssoundgetstate.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.65 SoundPlayFileType Struct Reference

Parameters for the SoundPlayFile system call.

```
#include <NXCDefs.h>
```

#### Data Fields

- char [Result](#)
- string [Filename](#)
- bool [Loop](#)
- byte [SoundLevel](#)

### 7.65.1 Detailed Description

Parameters for the SoundPlayFile system call. This structure is used when calling the [SysSoundPlayFile](#) system call function.

#### See also:

[SysSoundPlayFile\(\)](#)

#### Examples:

[ex\\_syssoundplayfile.nxc](#).

### 7.65.2 Field Documentation

#### 7.65.2.1 string SoundPlayFileType::Filename

The name of the file to play.

**Examples:**

[ex\\_syssoundplayfile.nxc](#).

**7.65.2.2 bool SoundPlayFileType::Loop**

If true, loops at end of file.

**Examples:**

[ex\\_syssoundplayfile.nxc](#).

**7.65.2.3 char SoundPlayFileType::Result**

The function call result, always [NO\\_ERR](#).

**7.65.2.4 byte SoundPlayFileType::SoundLevel**

The sound level. Valid values range from 0 to 4.

**Examples:**

[ex\\_syssoundplayfile.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

**7.66 SoundPlayToneType Struct Reference**

Parameters for the SoundPlayTone system call.

```
#include <NXCDefs.h>
```

**Data Fields**

- char [Result](#)
- unsigned int [Frequency](#)
- unsigned int [Duration](#)
- bool [Loop](#)
- byte [SoundLevel](#)

### 7.66.1 Detailed Description

Parameters for the SoundPlayTone system call. This structure is used when calling the [SysSoundPlayTone](#) system call function.

See also:

[SysSoundPlayTone\(\)](#)

Examples:

[ex\\_syssoundplaytone.nxc](#).

### 7.66.2 Field Documentation

#### 7.66.2.1 unsigned int SoundPlayToneType::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

Examples:

[ex\\_syssoundplaytone.nxc](#).

#### 7.66.2.2 unsigned int SoundPlayToneType::Frequency

The tone frequency. See the [Tone constants](#) group.

Examples:

[ex\\_syssoundplaytone.nxc](#).

#### 7.66.2.3 bool SoundPlayToneType::Loop

If true, loops forever.

Examples:

[ex\\_syssoundplaytone.nxc](#).

#### 7.66.2.4 char SoundPlayToneType::Result

The function call result, always [NO\\_ERR](#).

### 7.66.2.5 byte SoundPlayToneType::SoundLevel

The sound level. Valid values range from 0 to 4.

#### Examples:

[ex\\_syssoundplaytone.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.67 SoundSetStateType Struct Reference

Parameters for the SoundSetState system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- byte [State](#)
- byte [Flags](#)

### 7.67.1 Detailed Description

Parameters for the SoundSetState system call. This structure is used when calling the [SysSoundSetState](#) system call function.

#### See also:

[SysSoundSetState\(\)](#)

#### Examples:

[ex\\_syssoundsetstate.nxc](#).

### 7.67.2 Field Documentation

#### 7.67.2.1 byte SoundSetStateType::Flags

The new sound flags. See the [SoundFlags constants](#) group.

### 7.67.2.2 byte SoundSetStateType::Result

The function call result, same as State.

### 7.67.2.3 byte SoundSetStateType::State

The new sound state. See the [SoundState constants](#) group.

#### Examples:

[ex\\_syssoundsetstate.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.68 Tone Struct Reference

Type used with the PlayTones API function.

```
#include <NXCDefs.h>
```

#### Data Fields

- unsigned int [Frequency](#)
- unsigned int [Duration](#)

### 7.68.1 Detailed Description

Type used with the PlayTones API function. An array of this structure is used when calling the [PlayTones](#) API function.

#### See also:

[PlayTones\(\)](#)

#### Examples:

[ex\\_playtones.nxc](#).

### 7.68.2 Field Documentation

#### 7.68.2.1 unsigned int Tone::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

### 7.68.2.2 unsigned int Tone::Frequency

The tone frequency. See the [Tone constants](#) group.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.69 UpdateCalibCacheInfoType Struct Reference

Parameters for the UpdateCalibCacheInfo system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- string [Name](#)
- unsigned int [MinVal](#)
- unsigned int [MaxVal](#)

### 7.69.1 Detailed Description

Parameters for the UpdateCalibCacheInfo system call. This structure is used when calling the [SysUpdateCalibCacheInfo](#) system call function.

See also:

[SysUpdateCalibCacheInfo\(\)](#)

Examples:

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

### 7.69.2 Field Documentation

#### 7.69.2.1 unsigned int UpdateCalibCacheInfoType::MaxVal

The maximum calibrated value.

Examples:

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

### 7.69.2.2 unsigned int UpdateCalibCacheInfoType::MinVal

The minimum calibrated value.

#### Examples:

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

### 7.69.2.3 string UpdateCalibCacheInfoType::Name

The name of the sensor calibration cache.

#### Todo

?

#### Examples:

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

### 7.69.2.4 byte UpdateCalibCacheInfoType::Result

The function call result.

#### Todo

?

#### Examples:

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.70 VectorType Struct Reference

This structure is used for storing three axis values in a single object.

```
#include <NXCDefs.h>
```

#### Data Fields

- float *X*
- float *Y*
- float *Z*

### 7.70.1 Detailed Description

This structure is used for storing three axis values in a single object.

#### Examples:

[ex\\_diaccl.nxc](#), and [ex\\_digyro.nxc](#).

### 7.70.2 Field Documentation

#### 7.70.2.1 float VectorType::X

The X axis value.

#### Examples:

[ex\\_diaccl.nxc](#), and [ex\\_digyro.nxc](#).

#### 7.70.2.2 float VectorType::Y

The Y axis value.

#### Examples:

[ex\\_diaccl.nxc](#), and [ex\\_digyro.nxc](#).

#### 7.70.2.3 float VectorType::Z

The Z axis value.

#### Examples:

[ex\\_diaccl.nxc](#), and [ex\\_digyro.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.71 WriteSemDataType Struct Reference

Parameters for the WriteSemData system call.

```
#include <NXCDefs.h>
```

## Data Fields

- byte [SemData](#)
- bool [Request](#)
- byte [NewVal](#)
- bool [ClearBits](#)

### 7.71.1 Detailed Description

Parameters for the WriteSemData system call. This structure is used when calling the [SysWriteSemData](#) system call function.

#### See also:

[SysWriteSemData\(\)](#)

#### Examples:

[ex\\_SysWriteSemData.nxc](#).

### 7.71.2 Field Documentation

#### 7.71.2.1 bool WriteSemDataType::ClearBits

Should I clear existing bits?

#### Examples:

[ex\\_SysWriteSemData.nxc](#).

#### 7.71.2.2 byte WriteSemDataType::NewVal

The new semaphore data.

#### Examples:

[ex\\_SysWriteSemData.nxc](#).

#### 7.71.2.3 bool WriteSemDataType::Request

Which semaphore am I writing to, usage or request?

#### Examples:

[ex\\_SysWriteSemData.nxc](#).

#### 7.71.2.4 byte WriteSemDataType::SemData

The modified semaphore data returned by the function call.

#### Examples:

[ex\\_SysWriteSemData.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 7.72 XGPacketType Struct Reference

Parameters for the [ReadSensorMIXG1300L](#) function.

```
#include <NXCDefs.h>
```

#### Data Fields

- int [AccAngle](#)
- int [TurnRate](#)
- int [XAxis](#)
- int [YAxis](#)
- int [ZAxis](#)

#### 7.72.1 Detailed Description

Parameters for the [ReadSensorMIXG1300L](#) function. This structure is used when calling the [ReadSensorMIXG1300L](#) function. After calling the function read the sensor values from the various structure fields. The values are all scaled by 100.

#### Examples:

[ex\\_xg1300.nxc](#).

#### 7.72.2 Field Documentation

##### 7.72.2.1 int XGPacketType::AccAngle

The accumulated angle.

#### Examples:

[ex\\_xg1300.nxc](#).

**7.72.2.2 int XGPacketType::TurnRate**

The turn rate.

**Examples:**

[ex\\_xg1300.nxc](#).

**7.72.2.3 int XGPacketType::XAxis**

The X axis acceleration.

**Examples:**

[ex\\_xg1300.nxc](#).

**7.72.2.4 int XGPacketType::YAxis**

The Y axis acceleration.

**Examples:**

[ex\\_xg1300.nxc](#).

**7.72.2.5 int XGPacketType::ZAxis**

The Z axis acceleration.

**Examples:**

[ex\\_xg1300.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 8 File Documentation

### 8.1 NBCCommon.h File Reference

Constants and macros common to both NBC and NXC.

## Defines

- #define [TRUE](#) 1
- #define [FALSE](#) 0
- #define [NA](#) 0xFFFF
- #define [RC\\_PROP\\_BTONOFF](#) 0x0
- #define [RC\\_PROP\\_SOUND\\_LEVEL](#) 0x1
- #define [RC\\_PROP\\_SLEEP\\_TIMEOUT](#) 0x2
- #define [RC\\_PROP\\_DEBUGGING](#) 0xF
- #define [OPARR\\_SUM](#) 0x00
- #define [OPARR\\_MEAN](#) 0x01
- #define [OPARR\\_SUMSQR](#) 0x02
- #define [OPARR\\_STD](#) 0x03
- #define [OPARR\\_MIN](#) 0x04
- #define [OPARR\\_MAX](#) 0x05
- #define [OPARR\\_SORT](#) 0x06
- #define [PI](#) 3.141593
- #define [RADIANS\\_PER\\_DEGREE](#) PI/180
- #define [DEGREES\\_PER\\_RADIAN](#) 180/PI
- #define [FileOpenRead](#) 0
- #define [FileOpenWrite](#) 1
- #define [FileOpenAppend](#) 2
- #define [FileRead](#) 3
- #define [FileWrite](#) 4
- #define [FileClose](#) 5
- #define [FileResolveHandle](#) 6
- #define [FileRename](#) 7
- #define [FileDelete](#) 8
- #define [SoundPlayFile](#) 9
- #define [SoundPlayTone](#) 10
- #define [SoundGetState](#) 11
- #define [SoundSetState](#) 12
- #define [DrawText](#) 13
- #define [DrawPoint](#) 14
- #define [DrawLine](#) 15
- #define [DrawCircle](#) 16
- #define [DrawRect](#) 17
- #define [DrawGraphic](#) 18
- #define [SetScreenMode](#) 19
- #define [ReadButton](#) 20
- #define [CommLSWrite](#) 21
- #define [CommLSRead](#) 22
- #define [CommLSCheckStatus](#) 23

- #define [RandomNumber](#) 24
- #define [GetStartTick](#) 25
- #define [MessageWrite](#) 26
- #define [MessageRead](#) 27
- #define [CommBTCheckStatus](#) 28
- #define [CommBTWrite](#) 29
- #define [CommBTRead](#) 30
- #define [KeepAlive](#) 31
- #define [IOMapRead](#) 32
- #define [IOMapWrite](#) 33
- #define [ColorSensorRead](#) 34
- #define [CommBTOnOff](#) 35
- #define [CommBTConnection](#) 36
- #define [CommHSWrite](#) 37
- #define [CommHSRead](#) 38
- #define [CommHSCheckStatus](#) 39
- #define [ReadSemData](#) 40
- #define [WriteSemData](#) 41
- #define [ComputeCalibValue](#) 42
- #define [UpdateCalibCacheInfo](#) 43
- #define [DatalogWrite](#) 44
- #define [DatalogGetTimes](#) 45
- #define [SetSleepTimeoutVal](#) 46
- #define [ListFiles](#) 47
- #define [InputPinFunction](#) 77
- #define [IOMapReadByID](#) 78
- #define [IOMapWriteByID](#) 79
- #define [DisplayExecuteFunction](#) 80
- #define [CommExecuteFunction](#) 81
- #define [LoaderExecuteFunction](#) 82
- #define [FileFindFirst](#) 83
- #define [FileFindNext](#) 84
- #define [FileOpenWriteLinear](#) 85
- #define [FileOpenWriteNonLinear](#) 86
- #define [FileOpenReadLinear](#) 87
- #define [CommHSControl](#) 88
- #define [CommLSWriteEx](#) 89
- #define [FileSeek](#) 90
- #define [FileResize](#) 91
- #define [DrawGraphicArray](#) 92
- #define [DrawPolygon](#) 93
- #define [DrawEllipse](#) 94
- #define [DrawFont](#) 95

- #define [MemoryManager](#) 96
- #define [ReadLastResponse](#) 97
- #define [FileTell](#) 98
- #define [RandomEx](#) 99
- #define [LCD\\_LINE8](#) 0
- #define [LCD\\_LINE7](#) 8
- #define [LCD\\_LINE6](#) 16
- #define [LCD\\_LINE5](#) 24
- #define [LCD\\_LINE4](#) 32
- #define [LCD\\_LINE3](#) 40
- #define [LCD\\_LINE2](#) 48
- #define [LCD\\_LINE1](#) 56
- #define [MS\\_1](#) 1
- #define [MS\\_2](#) 2
- #define [MS\\_3](#) 3
- #define [MS\\_4](#) 4
- #define [MS\\_5](#) 5
- #define [MS\\_6](#) 6
- #define [MS\\_7](#) 7
- #define [MS\\_8](#) 8
- #define [MS\\_9](#) 9
- #define [MS\\_10](#) 10
- #define [MS\\_20](#) 20
- #define [MS\\_30](#) 30
- #define [MS\\_40](#) 40
- #define [MS\\_50](#) 50
- #define [MS\\_60](#) 60
- #define [MS\\_70](#) 70
- #define [MS\\_80](#) 80
- #define [MS\\_90](#) 90
- #define [MS\\_100](#) 100
- #define [MS\\_150](#) 150
- #define [MS\\_200](#) 200
- #define [MS\\_250](#) 250
- #define [MS\\_300](#) 300
- #define [MS\\_350](#) 350
- #define [MS\\_400](#) 400
- #define [MS\\_450](#) 450
- #define [MS\\_500](#) 500
- #define [MS\\_600](#) 600
- #define [MS\\_700](#) 700
- #define [MS\\_800](#) 800
- #define [MS\\_900](#) 900

- #define SEC\_1 1000
- #define SEC\_2 2000
- #define SEC\_3 3000
- #define SEC\_4 4000
- #define SEC\_5 5000
- #define SEC\_6 6000
- #define SEC\_7 7000
- #define SEC\_8 8000
- #define SEC\_9 9000
- #define SEC\_10 10000
- #define SEC\_15 15000
- #define SEC\_20 20000
- #define SEC\_30 30000
- #define MIN\_1 60000
- #define MAILBOX1 0
- #define MAILBOX2 1
- #define MAILBOX3 2
- #define MAILBOX4 3
- #define MAILBOX5 4
- #define MAILBOX6 5
- #define MAILBOX7 6
- #define MAILBOX8 7
- #define MAILBOX9 8
- #define MAILBOX10 9
- #define CommandModuleName "Command.mod"
- #define IOCtrlModuleName "IOCtrl.mod"
- #define LoaderModuleName "Loader.mod"
- #define SoundModuleName "Sound.mod"
- #define ButtonModuleName "Button.mod"
- #define UIModuleName "Ui.mod"
- #define InputModuleName "Input.mod"
- #define OutputModuleName "Output.mod"
- #define LowSpeedModuleName "Low Speed.mod"
- #define DisplayModuleName "Display.mod"
- #define CommModuleName "Comm.mod"
- #define CommandModuleID 0x00010001
- #define IOCtrlModuleID 0x00060001
- #define LoaderModuleID 0x00090001
- #define SoundModuleID 0x00080001
- #define ButtonModuleID 0x00040001
- #define UIModuleID 0x000C0001
- #define InputModuleID 0x00030001
- #define OutputModuleID 0x00020001

- #define `LowSpeedModuleID` 0x000B0001
- #define `DisplayModuleID` 0x000A0001
- #define `CommModuleID` 0x00050001
- #define `STAT_MSG_EMPTY_MAILBOX` 64
- #define `STAT_COMM_PENDING` 32
- #define `POOL_MAX_SIZE` 32768
- #define `TIMES_UP` 6
- #define `ROTATE_QUEUE` 5
- #define `STOP_REQ` 4
- #define `BREAKOUT_REQ` 3
- #define `CLUMP_SUSPEND` 2
- #define `CLUMP_DONE` 1
- #define `NO_ERR` 0
- #define `ERR_ARG` -1
- #define `ERR_INSTR` -2
- #define `ERR_FILE` -3
- #define `ERR_VER` -4
- #define `ERR_MEM` -5
- #define `ERR_BAD_PTR` -6
- #define `ERR_CLUMP_COUNT` -7
- #define `ERR_NO_CODE` -8
- #define `ERR_INSANE_OFFSET` -9
- #define `ERR_BAD_POOL_SIZE` -10
- #define `ERR_LOADER_ERR` -11
- #define `ERR_SPOTCHECK_FAIL` -12
- #define `ERR_NO_ACTIVE_CLUMP` -13
- #define `ERR_DEFAULT_OFFSETS` -14
- #define `ERR_MEMMGR_FAIL` -15
- #define `ERR_NON_FATAL` -16
- #define `ERR_INVALID_PORT` -16
- #define `ERR_INVALID_FIELD` -17
- #define `ERR_INVALID_QUEUE` -18
- #define `ERR_INVALID_SIZE` -19
- #define `ERR_NO_PROG` -20
- #define `ERR_COMM_CHAN_NOT_READY` -32
- #define `ERR_COMM_CHAN_INVALID` -33
- #define `ERR_COMM_BUFFER_FULL` -34
- #define `ERR_COMM_BUS_ERR` -35
- #define `ERR_RC_ILLEGAL_VAL` -64
- #define `ERR_RC_BAD_PACKET` -65
- #define `ERR_RC_UNKNOWN_CMD` -66
- #define `ERR_RC_FAILED` -67
- #define `PROG_IDLE` 0

- #define `PROG_OK` 1
- #define `PROG_RUNNING` 2
- #define `PROG_ERROR` 3
- #define `PROG_ABORT` 4
- #define `PROG_RESET` 5
- #define `CommandOffsetFormatString` 0
- #define `CommandOffsetPRCHandler` 16
- #define `CommandOffsetTick` 20
- #define `CommandOffsetOffsetDS` 24
- #define `CommandOffsetOffsetDVA` 26
- #define `CommandOffsetProgStatus` 28
- #define `CommandOffsetAwake` 29
- #define `CommandOffsetActivateFlag` 30
- #define `CommandOffsetDeactivateFlag` 31
- #define `CommandOffsetFileName` 32
- #define `CommandOffsetMemoryPool` 52
- #define `CommandOffsetSyncTime` 32820
- #define `CommandOffsetSyncTick` 32824
- #define `IOCTRL_POWERDOWN` 0x5A00
- #define `IOCTRL_BOOT` 0xA55A
- #define `IOCtrlOffsetPowerOn` 0
- #define `LoaderOffsetPFunc` 0
- #define `LoaderOffsetFreeUserFlash` 4
- #define `EOF` -1
- #define `NULL` 0
- #define `LDR_SUCCESS` 0x0000
- #define `LDR_INPROGRESS` 0x0001
- #define `LDR_REQPIN` 0x0002
- #define `LDR_NOMOREHANDLES` 0x8100
- #define `LDR_NOSPACE` 0x8200
- #define `LDR_NOMOREFILES` 0x8300
- #define `LDR_EOFEXPECTED` 0x8400
- #define `LDR_ENDOFFILE` 0x8500
- #define `LDR_NOTLINEARFILE` 0x8600
- #define `LDR_FILENOTFOUND` 0x8700
- #define `LDR_HANDLEALREADYCLOSED` 0x8800
- #define `LDR_NOLINEARSPACE` 0x8900
- #define `LDR_UNDEFINEDERROR` 0x8A00
- #define `LDR_FILEISBUSY` 0x8B00
- #define `LDR_NOWRITEBUFFERS` 0x8C00
- #define `LDR_APPENDNOTPOSSIBLE` 0x8D00
- #define `LDR_FILEISFULL` 0x8E00
- #define `LDR_FILEEXISTS` 0x8F00

- #define `LDR_MODULENOTFOUND` 0x9000
- #define `LDR_OUTOFBOUNDARY` 0x9100
- #define `LDR_ILLEGALFILENAME` 0x9200
- #define `LDR_ILLEGALHANDLE` 0x9300
- #define `LDR_BTBUSY` 0x9400
- #define `LDR_BTCONNECTFAIL` 0x9500
- #define `LDR_BTTIMEOUT` 0x9600
- #define `LDR_FILETX_TIMEOUT` 0x9700
- #define `LDR_FILETX_DSTEXISTS` 0x9800
- #define `LDR_FILETX_SRCMISSING` 0x9900
- #define `LDR_FILETX_STREAMERROR` 0x9A00
- #define `LDR_FILETX_CLOSEERROR` 0x9B00
- #define `LDR_INVALIDSEEK` 0x9C00
- #define `LDR_CMD_OPENREAD` 0x80
- #define `LDR_CMD_OPENWRITE` 0x81
- #define `LDR_CMD_READ` 0x82
- #define `LDR_CMD_WRITE` 0x83
- #define `LDR_CMD_CLOSE` 0x84
- #define `LDR_CMD_DELETE` 0x85
- #define `LDR_CMD_FINDFIRST` 0x86
- #define `LDR_CMD_FINDNEXT` 0x87
- #define `LDR_CMD_VERSIONS` 0x88
- #define `LDR_CMD_OPENWRITELINEAR` 0x89
- #define `LDR_CMD_OPENREADLINEAR` 0x8A
- #define `LDR_CMD_OPENWRITEDATA` 0x8B
- #define `LDR_CMD_OPENAPPENDDATA` 0x8C
- #define `LDR_CMD_CROPDATAFILE` 0x8D
- #define `LDR_CMD_FINDFIRSTMODULE` 0x90
- #define `LDR_CMD_FINDNEXTMODULE` 0x91
- #define `LDR_CMD_CLOSEMODHANDLE` 0x92
- #define `LDR_CMD_IOMAPREAD` 0x94
- #define `LDR_CMD_IOMAPWRITE` 0x95
- #define `LDR_CMD_BOOTCMD` 0x97
- #define `LDR_CMD_SETBRICKNAME` 0x98
- #define `LDR_CMD_BTGETADR` 0x9A
- #define `LDR_CMD_DEVICEINFO` 0x9B
- #define `LDR_CMD_DELETEUSERFLASH` 0xA0
- #define `LDR_CMD_POLLCMDLEN` 0xA1
- #define `LDR_CMD_POLLCMD` 0xA2
- #define `LDR_CMD_RENAMEFILE` 0xA3
- #define `LDR_CMD_BTFACTORYRESET` 0xA4
- #define `LDR_CMD_RESIZEDATAFILE` 0xD0
- #define `LDR_CMD_SEEKFROMSTART` 0xD1

- #define `LDR_CMD_SEEKFROMCURRENT` 0xD2
- #define `LDR_CMD_SEEKFROMEND` 0xD3
- #define `SOUND_FLAGS_IDLE` 0x00
- #define `SOUND_FLAGS_UPDATE` 0x01
- #define `SOUND_FLAGS_RUNNING` 0x02
- #define `SOUND_STATE_IDLE` 0x00
- #define `SOUND_STATE_FILE` 0x02
- #define `SOUND_STATE_TONE` 0x03
- #define `SOUND_STATE_STOP` 0x04
- #define `SOUND_MODE_ONCE` 0x00
- #define `SOUND_MODE_LOOP` 0x01
- #define `SOUND_MODE_TONE` 0x02
- #define `SoundOffsetFreq` 0
- #define `SoundOffsetDuration` 2
- #define `SoundOffsetSampleRate` 4
- #define `SoundOffsetSoundFilename` 6
- #define `SoundOffsetFlags` 26
- #define `SoundOffsetState` 27
- #define `SoundOffsetMode` 28
- #define `SoundOffsetVolume` 29
- #define `FREQUENCY_MIN` 220
- #define `FREQUENCY_MAX` 14080
- #define `SAMPLERATE_MIN` 2000
- #define `SAMPLERATE_DEFAULT` 8000
- #define `SAMPLERATE_MAX` 16000
- #define `TONE_A3` 220
- #define `TONE_AS3` 233
- #define `TONE_B3` 247
- #define `TONE_C4` 262
- #define `TONE_CS4` 277
- #define `TONE_D4` 294
- #define `TONE_DS4` 311
- #define `TONE_E4` 330
- #define `TONE_F4` 349
- #define `TONE_FS4` 370
- #define `TONE_G4` 392
- #define `TONE_GS4` 415
- #define `TONE_A4` 440
- #define `TONE_AS4` 466
- #define `TONE_B4` 494
- #define `TONE_C5` 523
- #define `TONE_CS5` 554
- #define `TONE_D5` 587

- #define [TONE\\_DS5](#) 622
- #define [TONE\\_E5](#) 659
- #define [TONE\\_F5](#) 698
- #define [TONE\\_FS5](#) 740
- #define [TONE\\_G5](#) 784
- #define [TONE\\_GS5](#) 831
- #define [TONE\\_A5](#) 880
- #define [TONE\\_AS5](#) 932
- #define [TONE\\_B5](#) 988
- #define [TONE\\_C6](#) 1047
- #define [TONE\\_CS6](#) 1109
- #define [TONE\\_D6](#) 1175
- #define [TONE\\_DS6](#) 1245
- #define [TONE\\_E6](#) 1319
- #define [TONE\\_F6](#) 1397
- #define [TONE\\_FS6](#) 1480
- #define [TONE\\_G6](#) 1568
- #define [TONE\\_GS6](#) 1661
- #define [TONE\\_A6](#) 1760
- #define [TONE\\_AS6](#) 1865
- #define [TONE\\_B6](#) 1976
- #define [TONE\\_C7](#) 2093
- #define [TONE\\_CS7](#) 2217
- #define [TONE\\_D7](#) 2349
- #define [TONE\\_DS7](#) 2489
- #define [TONE\\_E7](#) 2637
- #define [TONE\\_F7](#) 2794
- #define [TONE\\_FS7](#) 2960
- #define [TONE\\_G7](#) 3136
- #define [TONE\\_GS7](#) 3322
- #define [TONE\\_A7](#) 3520
- #define [TONE\\_AS7](#) 3729
- #define [TONE\\_B7](#) 3951
- #define [BTN1](#) 0
- #define [BTN2](#) 1
- #define [BTN3](#) 2
- #define [BTN4](#) 3
- #define [BTNEXIT](#) BTN1
- #define [BTNRIGHT](#) BTN2
- #define [BTNLEFT](#) BTN3
- #define [BTNCENTER](#) BTN4
- #define [NO\\_OF\\_BTNS](#) 4
- #define [BTNSTATE\\_PRESSED\\_EV](#) 0x01

- #define `BTNSTATE_SHORT_RELEASED_EV` 0x02
- #define `BTNSTATE_LONG_PRESSED_EV` 0x04
- #define `BTNSTATE_LONG_RELEASED_EV` 0x08
- #define `BTNSTATE_PRESSED_STATE` 0x80
- #define `BTNSTATE_NONE` 0x10
- #define `ButtonOffsetPressedCnt(b)` (((b)\*8)+0)
- #define `ButtonOffsetLongPressCnt(b)` (((b)\*8)+1)
- #define `ButtonOffsetShortRelCnt(b)` (((b)\*8)+2)
- #define `ButtonOffsetLongRelCnt(b)` (((b)\*8)+3)
- #define `ButtonOffsetRelCnt(b)` (((b)\*8)+4)
- #define `ButtonOffsetState(b)` ((b)+32)
- #define `UI_FLAGS_UPDATE` 0x01
- #define `UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER` 0x02
- #define `UI_FLAGS_DISABLE_EXIT` 0x04
- #define `UI_FLAGS_REDRAW_STATUS` 0x08
- #define `UI_FLAGS_RESET_SLEEP_TIMER` 0x10
- #define `UI_FLAGS_EXECUTE_LMS_FILE` 0x20
- #define `UI_FLAGS_BUSY` 0x40
- #define `UI_FLAGS_ENABLE_STATUS_UPDATE` 0x80
- #define `UI_STATE_INIT_DISPLAY` 0
- #define `UI_STATE_INIT_LOW_BATTERY` 1
- #define `UI_STATE_INIT_INTRO` 2
- #define `UI_STATE_INIT_WAIT` 3
- #define `UI_STATE_INIT_MENU` 4
- #define `UI_STATE_NEXT_MENU` 5
- #define `UI_STATE_DRAW_MENU` 6
- #define `UI_STATE_TEST_BUTTONS` 7
- #define `UI_STATE_LEFT_PRESSED` 8
- #define `UI_STATE_RIGHT_PRESSED` 9
- #define `UI_STATE_ENTER_PRESSED` 10
- #define `UI_STATE_EXIT_PRESSED` 11
- #define `UI_STATE_CONNECT_REQUEST` 12
- #define `UI_STATE_EXECUTE_FILE` 13
- #define `UI_STATE_EXECUTING_FILE` 14
- #define `UI_STATE_LOW_BATTERY` 15
- #define `UI_STATE_BT_ERROR` 16
- #define `UI_BUTTON_NONE` 0
- #define `UI_BUTTON_LEFT` 1
- #define `UI_BUTTON_ENTER` 2
- #define `UI_BUTTON_RIGHT` 3
- #define `UI_BUTTON_EXIT` 4
- #define `UI_BT_STATE_VISIBLE` 0x01
- #define `UI_BT_STATE_CONNECTED` 0x02

- #define [UI\\_BT\\_STATE\\_OFF](#) 0x04
- #define [UI\\_BT\\_ERROR\\_ATTENTION](#) 0x08
- #define [UI\\_BT\\_CONNECT\\_REQUEST](#) 0x40
- #define [UI\\_BT\\_PIN\\_REQUEST](#) 0x80
- #define [UI\\_VM\\_IDLE](#) 0
- #define [UI\\_VM\\_RUN\\_FREE](#) 1
- #define [UI\\_VM\\_RUN\\_SINGLE](#) 2
- #define [UI\\_VM\\_RUN\\_PAUSE](#) 3
- #define [UI\\_VM\\_RESET1](#) 4
- #define [UI\\_VM\\_RESET2](#) 5
- #define [UIOffsetPMenu](#) 0
- #define [UIOffsetBatteryVoltage](#) 4
- #define [UIOffsetLMSfilename](#) 6
- #define [UIOffsetFlags](#) 26
- #define [UIOffsetState](#) 27
- #define [UIOffsetButton](#) 28
- #define [UIOffsetRunState](#) 29
- #define [UIOffsetBatteryState](#) 30
- #define [UIOffsetBluetoothState](#) 31
- #define [UIOffsetUsbState](#) 32
- #define [UIOffsetSleepTimeout](#) 33
- #define [UIOffsetSleepTimer](#) 34
- #define [UIOffsetRechargeable](#) 35
- #define [UIOffsetVolume](#) 36
- #define [UIOffsetError](#) 37
- #define [UIOffsetOBPPointer](#) 38
- #define [UIOffsetForceOff](#) 39
- #define [UIOffsetAbortFlag](#) 40
- #define [IN\\_1](#) 0x00
- #define [IN\\_2](#) 0x01
- #define [IN\\_3](#) 0x02
- #define [IN\\_4](#) 0x03
- #define [IN\\_TYPE\\_NO\\_SENSOR](#) 0x00
- #define [IN\\_TYPE\\_SWITCH](#) 0x01
- #define [IN\\_TYPE\\_TEMPERATURE](#) 0x02
- #define [IN\\_TYPE\\_REFLECTION](#) 0x03
- #define [IN\\_TYPE\\_ANGLE](#) 0x04
- #define [IN\\_TYPE\\_LIGHT\\_ACTIVE](#) 0x05
- #define [IN\\_TYPE\\_LIGHT\\_INACTIVE](#) 0x06
- #define [IN\\_TYPE\\_SOUND\\_DB](#) 0x07
- #define [IN\\_TYPE\\_SOUND\\_DBA](#) 0x08
- #define [IN\\_TYPE\\_CUSTOM](#) 0x09
- #define [IN\\_TYPE\\_LOWSPEED](#) 0x0A

- #define `IN_TYPE_LOWSPEED_9V` 0x0B
- #define `IN_TYPE_HISPEED` 0x0C
- #define `IN_TYPE_COLORFULL` 0x0D
- #define `IN_TYPE_COLORRED` 0x0E
- #define `IN_TYPE_COLORGREEN` 0x0F
- #define `IN_TYPE_COLORBLUE` 0x10
- #define `IN_TYPE_COLORNONE` 0x11
- #define `IN_TYPE_COLOREXIT` 0x12
- #define `IN_MODE_RAW` 0x00
- #define `IN_MODE_BOOLEAN` 0x20
- #define `IN_MODE_TRANSITIONCNT` 0x40
- #define `IN_MODE_PERIODCOUNTER` 0x60
- #define `IN_MODE_PCTFULLSCALE` 0x80
- #define `IN_MODE_CELSIUS` 0xA0
- #define `IN_MODE_FAHRENHEIT` 0xC0
- #define `IN_MODE_ANGLESTEP` 0xE0
- #define `IN_MODE_SLOPEMASK` 0x1F
- #define `IN_MODE_MODEMASK` 0xE0
- #define `TypeField` 0
- #define `InputModeField` 1
- #define `RawValueField` 2
- #define `NormalizedValueField` 3
- #define `ScaledValueField` 4
- #define `InvalidDataField` 5
- #define `INPUT_DIGI0` 0x01
- #define `INPUT_DIGI1` 0x02
- #define `INPUT_CUSTOMINACTIVE` 0x00
- #define `INPUT_CUSTOM9V` 0x01
- #define `INPUT_CUSTOMACTIVE` 0x02
- #define `INPUT_INVALID_DATA` 0x01
- #define `INPUT_RED` 0
- #define `INPUT_GREEN` 1
- #define `INPUT_BLUE` 2
- #define `INPUT_BLANK` 3
- #define `INPUT_NO_OF_COLORS` 4
- #define `INPUT_BLACKCOLOR` 1
- #define `INPUT_BLUECOLOR` 2
- #define `INPUT_GREENCOLOR` 3
- #define `INPUT_YELLOWCOLOR` 4
- #define `INPUT_REDCOLOR` 5
- #define `INPUT_WHITECOLOR` 6
- #define `INPUT_SENSORCAL` 0x01
- #define `INPUT_SENSOROFF` 0x02

- #define `INPUT_RUNNINGCAL` 0x20
- #define `INPUT_STARTCAL` 0x40
- #define `INPUT_RESETCAL` 0x80
- #define `INPUT_CAL_POINT_0` 0
- #define `INPUT_CAL_POINT_1` 1
- #define `INPUT_CAL_POINT_2` 2
- #define `INPUT_NO_OF_POINTS` 3
- #define `InputOffsetCustomZeroOffset(p)` (((p)\*20)+0)
- #define `InputOffsetADRaw(p)` (((p)\*20)+2)
- #define `InputOffsetSensorRaw(p)` (((p)\*20)+4)
- #define `InputOffsetSensorValue(p)` (((p)\*20)+6)
- #define `InputOffsetSensorType(p)` (((p)\*20)+8)
- #define `InputOffsetSensorMode(p)` (((p)\*20)+9)
- #define `InputOffsetSensorBoolean(p)` (((p)\*20)+10)
- #define `InputOffsetDigiPinsDir(p)` (((p)\*20)+11)
- #define `InputOffsetDigiPinsIn(p)` (((p)\*20)+12)
- #define `InputOffsetDigiPinsOut(p)` (((p)\*20)+13)
- #define `InputOffsetCustomPctFullScale(p)` (((p)\*20)+14)
- #define `InputOffsetCustomActiveStatus(p)` (((p)\*20)+15)
- #define `InputOffsetInvalidData(p)` (((p)\*20)+16)
- #define `InputOffsetColorCalibration(p, np, nc)` (80+((p)\*84)+0+((np)\*16)+((nc)\*4))
- #define `InputOffsetColorCalLimits(p, np)` (80+((p)\*84)+48+((np)\*2))
- #define `InputOffsetColorADRaw(p, nc)` (80+((p)\*84)+52+((nc)\*2))
- #define `InputOffsetColorSensorRaw(p, nc)` (80+((p)\*84)+60+((nc)\*2))
- #define `InputOffsetColorSensorValue(p, nc)` (80+((p)\*84)+68+((nc)\*2))
- #define `InputOffsetColorBoolean(p, nc)` (80+((p)\*84)+76+((nc)\*2))
- #define `InputOffsetColorCalibrationState(p)` (80+((p)\*84)+80)
- #define `INPUT_PINCMD_DIR` 0x00
- #define `INPUT_PINCMD_SET` 0x01
- #define `INPUT_PINCMD_CLEAR` 0x02
- #define `INPUT_PINCMD_READ` 0x03
- #define `INPUT_PINCMD_MASK` 0x03
- #define `INPUT_PINCMD_WAIT(_usec)` ((\_usec)<<2)
- #define `INPUT_PINDIR_OUTPUT` 0x00
- #define `INPUT_PINDIR_INPUT` 0x04
- #define `OUT_A` 0x00
- #define `OUT_B` 0x01
- #define `OUT_C` 0x02
- #define `OUT_AB` 0x03
- #define `OUT_AC` 0x04
- #define `OUT_BC` 0x05
- #define `OUT_ABC` 0x06
- #define `PID_0` 0

- #define [PID\\_1](#) 32
- #define [PID\\_2](#) 64
- #define [PID\\_3](#) 96
- #define [PID\\_4](#) 128
- #define [PID\\_5](#) 160
- #define [PID\\_6](#) 192
- #define [PID\\_7](#) 224
- #define [UF\\_UPDATE\\_MODE](#) 0x01
- #define [UF\\_UPDATE\\_SPEED](#) 0x02
- #define [UF\\_UPDATE\\_TACHO\\_LIMIT](#) 0x04
- #define [UF\\_UPDATE\\_RESET\\_COUNT](#) 0x08
- #define [UF\\_UPDATE\\_PID\\_VALUES](#) 0x10
- #define [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) 0x20
- #define [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) 0x40
- #define [UF\\_PENDING\\_UPDATES](#) 0x80
- #define [RESET\\_NONE](#) 0x00
- #define [RESET\\_COUNT](#) 0x08
- #define [RESET\\_BLOCK\\_COUNT](#) 0x20
- #define [RESET\\_ROTATION\\_COUNT](#) 0x40
- #define [RESET\\_BLOCKANDTACHO](#) 0x28
- #define [RESET\\_ALL](#) 0x68
- #define [OUT\\_MODE\\_COAST](#) 0x00
- #define [OUT\\_MODE\\_MOTORON](#) 0x01
- #define [OUT\\_MODE\\_BRAKE](#) 0x02
- #define [OUT\\_MODE\\_REGULATED](#) 0x04
- #define [OUT\\_MODE\\_REGMETHOD](#) 0xF0
- #define [OUT\\_OPTION\\_HOLDATLIMIT](#) 0x10
- #define [OUT\\_OPTION\\_RAMPDOWNTOLIMIT](#) 0x20
- #define [OUT\\_REGOPTION\\_NO\\_SATURATION](#) 0x01
- #define [OUT\\_RUNSTATE\\_IDLE](#) 0x00
- #define [OUT\\_RUNSTATE\\_RAMPUP](#) 0x10
- #define [OUT\\_RUNSTATE\\_RUNNING](#) 0x20
- #define [OUT\\_RUNSTATE\\_RAMPDOWN](#) 0x40
- #define [OUT\\_RUNSTATE\\_HOLD](#) 0x60
- #define [OUT\\_REGMODE\\_IDLE](#) 0
- #define [OUT\\_REGMODE\\_SPEED](#) 1
- #define [OUT\\_REGMODE\\_SYNC](#) 2
- #define [OUT\\_REGMODE\\_POS](#) 4
- #define [UpdateFlagsField](#) 0
  - *Update flags field.*
- #define [OutputModeField](#) 1

*Mode field.*

- #define [PowerField](#) 2  
*Power field.*
- #define [ActualSpeedField](#) 3  
*Actual speed field.*
- #define [TachoCountField](#) 4  
*Internal tachometer count field.*
- #define [TachoLimitField](#) 5  
*Tachometer limit field.*
- #define [RunStateField](#) 6  
*Run state field.*
- #define [TurnRatioField](#) 7  
*Turn ratio field.*
- #define [RegModeField](#) 8  
*Regulation mode field.*
- #define [OverloadField](#) 9  
*Overload field.*
- #define [RegPValueField](#) 10  
*Proportional field.*
- #define [RegIValueField](#) 11  
*Integral field.*
- #define [RegDValueField](#) 12  
*Derivative field.*
- #define [BlockTachoCountField](#) 13  
*NXT-G block tachometer count field.*
- #define [RotationCountField](#) 14  
*Rotation counter field.*
- #define [OutputOptionsField](#) 15

*Options field.*

- #define **MaxSpeedField** 16  
*MaxSpeed field.*
  
- #define **MaxAccelerationField** 17  
*MaxAcceleration field.*
  
- #define **OutputOffsetTachoCount**(p) (((p)\*32)+0)
- #define **OutputOffsetBlockTachoCount**(p) (((p)\*32)+4)
- #define **OutputOffsetRotationCount**(p) (((p)\*32)+8)
- #define **OutputOffsetTachoLimit**(p) (((p)\*32)+12)
- #define **OutputOffsetMotorRPM**(p) (((p)\*32)+16)
- #define **OutputOffsetFlags**(p) (((p)\*32)+18)
- #define **OutputOffsetMode**(p) (((p)\*32)+19)
- #define **OutputOffsetSpeed**(p) (((p)\*32)+20)
- #define **OutputOffsetActualSpeed**(p) (((p)\*32)+21)
- #define **OutputOffsetRegPParameter**(p) (((p)\*32)+22)
- #define **OutputOffsetRegIParameter**(p) (((p)\*32)+23)
- #define **OutputOffsetRegDParameter**(p) (((p)\*32)+24)
- #define **OutputOffsetRunState**(p) (((p)\*32)+25)
- #define **OutputOffsetRegMode**(p) (((p)\*32)+26)
- #define **OutputOffsetOverloaded**(p) (((p)\*32)+27)
- #define **OutputOffsetSyncTurnParameter**(p) (((p)\*32)+28)
- #define **OutputOffsetOptions**(p) (((p)\*32)+29)
- #define **OutputOffsetMaxSpeed**(p) (((p)\*32)+30)
- #define **OutputOffsetMaxAccel**(p) (((p)\*32)+31)
- #define **OutputOffsetRegulationTime** 96
- #define **OutputOffsetRegulationOptions** 97
- #define **COM\_CHANNEL\_NONE\_ACTIVE** 0x00
- #define **COM\_CHANNEL\_ONE\_ACTIVE** 0x01
- #define **COM\_CHANNEL\_TWO\_ACTIVE** 0x02
- #define **COM\_CHANNEL\_THREE\_ACTIVE** 0x04
- #define **COM\_CHANNEL\_FOUR\_ACTIVE** 0x08
- #define **LOWSPEED\_IDLE** 0
- #define **LOWSPEED\_INIT** 1
- #define **LOWSPEED\_LOAD\_BUFFER** 2
- #define **LOWSPEED\_COMMUNICATING** 3
- #define **LOWSPEED\_ERROR** 4
- #define **LOWSPEED\_DONE** 5
- #define **LOWSPEED\_TRANSMITTING** 1
- #define **LOWSPEED\_RECEIVING** 2
- #define **LOWSPEED\_DATA\_RECEIVED** 3

- #define `LOWSPEED_NO_ERROR` 0
- #define `LOWSPEED_CH_NOT_READY` 1
- #define `LOWSPEED_TX_ERROR` 2
- #define `LOWSPEED_RX_ERROR` 3
- #define `LowSpeedOffsetInBufBuf(p)` (((p)\*19)+0)
- #define `LowSpeedOffsetInBufInPtr(p)` (((p)\*19)+16)
- #define `LowSpeedOffsetInBufOutPtr(p)` (((p)\*19)+17)
- #define `LowSpeedOffsetInBufBytesToRx(p)` (((p)\*19)+18)
- #define `LowSpeedOffsetOutBufBuf(p)` (((p)\*19)+76)
- #define `LowSpeedOffsetOutBufInPtr(p)` (((p)\*19)+92)
- #define `LowSpeedOffsetOutBufOutPtr(p)` (((p)\*19)+93)
- #define `LowSpeedOffsetOutBufBytesToRx(p)` (((p)\*19)+94)
- #define `LowSpeedOffsetMode(p)` ((p)+152)
- #define `LowSpeedOffsetChannelState(p)` ((p)+156)
- #define `LowSpeedOffsetErrorType(p)` ((p)+160)
- #define `LowSpeedOffsetState` 164
- #define `LowSpeedOffsetSpeed` 165
- #define `LowSpeedOffsetNoRestartOnRead` 166
- #define `LSREAD_RESTART_ALL` 0x00
- #define `LSREAD_NO_RESTART_1` 0x01
- #define `LSREAD_NO_RESTART_2` 0x02
- #define `LSREAD_NO_RESTART_3` 0x04
- #define `LSREAD_NO_RESTART_4` 0x08
- #define `LSREAD_RESTART_NONE` 0x0F
- #define `LSREAD_NO_RESTART_MASK` 0x10
- #define `I2C_ADDR_DEFAULT` 0x02
- #define `I2C_REG_VERSION` 0x00
- #define `I2C_REG_VENDOR_ID` 0x08
- #define `I2C_REG_DEVICE_ID` 0x10
- #define `I2C_REG_CMD` 0x41
- #define `LEGO_ADDR_US` 0x02
- #define `LEGO_ADDR_TEMP` 0x98
- #define `LEGO_ADDR_EMETER` 0x04
- #define `US_CMD_OFF` 0x00
- #define `US_CMD_SINGLESHOT` 0x01
- #define `US_CMD_CONTINUOUS` 0x02
- #define `US_CMD_EVENTCAPTURE` 0x03
- #define `US_CMD_WARMRESET` 0x04
- #define `US_REG_CM_INTERVAL` 0x40
- #define `US_REG_ACTUAL_ZERO` 0x50
- #define `US_REG_SCALE_FACTOR` 0x51
- #define `US_REG_SCALE_DIVISOR` 0x52
- #define `US_REG_FACTORY_ACTUAL_ZERO` 0x11

- #define `US_REG_FACTORY_SCALE_FACTOR` 0x12
- #define `US_REG_FACTORY_SCALE_DIVISOR` 0x13
- #define `US_REG_MEASUREMENT_UNITS` 0x14
- #define `TEMP_RES_9BIT` 0x00
- #define `TEMP_RES_10BIT` 0x20
- #define `TEMP_RES_11BIT` 0x40
- #define `TEMP_RES_12BIT` 0x60
- #define `TEMP_SD_CONTINUOUS` 0x00
- #define `TEMP_SD_SHUTDOWN` 0x01
- #define `TEMP_TM_COMPARATOR` 0x00
- #define `TEMP_TM_INTERRUPT` 0x02
- #define `TEMP_OS_ONESHOT` 0x80
- #define `TEMP_FQ_1` 0x00
- #define `TEMP_FQ_2` 0x08
- #define `TEMP_FQ_4` 0x10
- #define `TEMP_FQ_6` 0x18
- #define `TEMP_POL_LOW` 0x00
- #define `TEMP_POL_HIGH` 0x04
- #define `TEMP_REG_TEMP` 0x00
- #define `TEMP_REG_CONFIG` 0x01
- #define `TEMP_REG_TLOW` 0x02
- #define `TEMP_REG_THIGH` 0x03
- #define `EMETER_REG_VIN` 0x0a
- #define `EMETER_REG_AIN` 0x0c
- #define `EMETER_REG_VOUT` 0x0e
- #define `EMETER_REG_AOUT` 0x10
- #define `EMETER_REG_JOULES` 0x12
- #define `EMETER_REG_WIN` 0x14
- #define `EMETER_REG_WOUT` 0x16
- #define `I2C_OPTION_STANDARD` 0x00
- #define `I2C_OPTION_NORESTART` 0x04
- #define `I2C_OPTION_FAST` 0x08
- #define `DISPLAY_ERASE_ALL` 0x00
- #define `DISPLAY_PIXEL` 0x01
- #define `DISPLAY_HORIZONTAL_LINE` 0x02
- #define `DISPLAY_VERTICAL_LINE` 0x03
- #define `DISPLAY_CHAR` 0x04
- #define `DISPLAY_ERASE_LINE` 0x05
- #define `DISPLAY_FILL_REGION` 0x06
- #define `DISPLAY_FRAME` 0x07
- #define `DRAW_OPT_NORMAL` (0x0000)
- #define `DRAW_OPT_CLEAR_WHOLE_SCREEN` (0x0001)
- #define `DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN` (0x0002)

- #define [DRAW\\_OPT\\_CLEAR\\_PIXELS](#) (0x0004)
- #define [DRAW\\_OPT\\_CLEAR](#) (0x0004)
- #define [DRAW\\_OPT\\_INVERT](#) (0x0004)
- #define [DRAW\\_OPT\\_LOGICAL\\_COPY](#) (0x0000)
- #define [DRAW\\_OPT\\_LOGICAL\\_AND](#) (0x0008)
- #define [DRAW\\_OPT\\_LOGICAL\\_OR](#) (0x0010)
- #define [DRAW\\_OPT\\_LOGICAL\\_XOR](#) (0x0018)
- #define [DRAW\\_OPT\\_FILL\\_SHAPE](#) (0x0020)
- #define [DRAW\\_OPT\\_CLEAR\\_SCREEN\\_MODES](#) (0x0003)
- #define [DRAW\\_OPT\\_LOGICAL\\_OPERATIONS](#) (0x0018)
- #define [DRAW\\_OPT\\_POLYGON\\_POLYLINE](#) (0x0400)
- #define [DRAW\\_OPT\\_FONT DIRECTIONS](#) (0x01C0)
- #define [DRAW\\_OPT\\_FONT\\_WRAP](#) (0x0200)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RB](#) (0x0000)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_L2RT](#) (0x0040)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_R2LB](#) (0x0080)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_R2LT](#) (0x00C0)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TL](#) (0x0100)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_B2TR](#) (0x0140)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_T2BL](#) (0x0180)
- #define [DRAW\\_OPT\\_FONT\\_DIR\\_T2BR](#) (0x01C0)
- #define [DISPLAY\\_ON](#) 0x01
- #define [DISPLAY\\_REFRESH](#) 0x02
- #define [DISPLAY\\_POPUP](#) 0x08
- #define [DISPLAY\\_REFRESH\\_DISABLED](#) 0x40
- #define [DISPLAY\\_BUSY](#) 0x80
- #define [DISPLAY\\_CONTRAST\\_DEFAULT](#) 0x5A
- #define [DISPLAY\\_CONTRAST\\_MAX](#) 0x7F
- #define [SCREEN\\_MODE\\_RESTORE](#) 0x00
- #define [SCREEN\\_MODE\\_CLEAR](#) 0x01
- #define [DISPLAY\\_HEIGHT](#) 64
- #define [DISPLAY\\_WIDTH](#) 100
- #define [DISPLAY\\_MENUICONS\\_Y](#) 40
- #define [DISPLAY\\_MENUICONS\\_X\\_OFFSETS](#) 7
- #define [DISPLAY\\_MENUICONS\\_X\\_DIFF](#) 31
- #define [TEXTLINE\\_1](#) 0
- #define [TEXTLINE\\_2](#) 1
- #define [TEXTLINE\\_3](#) 2
- #define [TEXTLINE\\_4](#) 3
- #define [TEXTLINE\\_5](#) 4
- #define [TEXTLINE\\_6](#) 5
- #define [TEXTLINE\\_7](#) 6
- #define [TEXTLINE\\_8](#) 7

- #define [TEXTLINES](#) 8
- #define [MENUICON\\_LEFT](#) 0
- #define [MENUICON\\_CENTER](#) 1
- #define [MENUICON\\_RIGHT](#) 2
- #define [MENUICONS](#) 3
- #define [FRAME\\_SELECT](#) 0
- #define [STATUSTEXT](#) 1
- #define [MENUTEXT](#) 2
- #define [STEPLINE](#) 3
- #define [TOPLINE](#) 4
- #define [SPECIALS](#) 5
- #define [STATUSICON\\_BLUETOOTH](#) 0
- #define [STATUSICON\\_USB](#) 1
- #define [STATUSICON\\_VM](#) 2
- #define [STATUSICON\\_BATTERY](#) 3
- #define [STATUSICONS](#) 4
- #define [SCREEN\\_BACKGROUND](#) 0
- #define [SCREEN\\_LARGE](#) 1
- #define [SCREEN\\_SMALL](#) 2
- #define [SCREENS](#) 3
- #define [BITMAP\\_1](#) 0
- #define [BITMAP\\_2](#) 1
- #define [BITMAP\\_3](#) 2
- #define [BITMAP\\_4](#) 3
- #define [BITMAPS](#) 4
- #define [STEPICON\\_1](#) 0
- #define [STEPICON\\_2](#) 1
- #define [STEPICON\\_3](#) 2
- #define [STEPICON\\_4](#) 3
- #define [STEPICON\\_5](#) 4
- #define [STEPICONS](#) 5
- #define [DisplayOffsetPFunc](#) 0
- #define [DisplayOffsetEraseMask](#) 4
- #define [DisplayOffsetUpdateMask](#) 8
- #define [DisplayOffsetPFont](#) 12
- #define [DisplayOffsetPTextLines\(p\)](#) (((p)\*4)+16)
- #define [DisplayOffsetPStatusText](#) 48
- #define [DisplayOffsetPStatusIcons](#) 52
- #define [DisplayOffsetPScreens\(p\)](#) (((p)\*4)+56)
- #define [DisplayOffsetPBitmaps\(p\)](#) (((p)\*4)+68)
- #define [DisplayOffsetPMenuText](#) 84
- #define [DisplayOffsetPMenuIcons\(p\)](#) (((p)\*4)+88)
- #define [DisplayOffsetPStepIcons](#) 100

- #define [DisplayOffsetDisplay](#) 104
- #define [DisplayOffsetStatusIcons](#)(p) ((p)+108)
- #define [DisplayOffsetStepIcons](#)(p) ((p)+112)
- #define [DisplayOffsetFlags](#) 117
- #define [DisplayOffsetTextLinesCenterFlags](#) 118
- #define [DisplayOffsetNormal](#)(l, w) (((l)\*100)+(w)+119)
- #define [DisplayOffsetPopup](#)(l, w) (((l)\*100)+(w)+919)
- #define [DisplayOffsetContrast](#) 1719
- #define [SIZE\\_OF\\_USBBUF](#) 64
- #define [USB\\_PROTOCOL\\_OVERHEAD](#) 2
- #define [SIZE\\_OF\\_USBDATA](#) 62
- #define [SIZE\\_OF\\_HSBUF](#) 128
- #define [SIZE\\_OF\\_BTBUF](#) 128
- #define [BT\\_CMD\\_BYTE](#) 1
- #define [SIZE\\_OF\\_BT\\_DEVICE\\_TABLE](#) 30
- #define [SIZE\\_OF\\_BT\\_CONNECT\\_TABLE](#) 4
- #define [SIZE\\_OF\\_BT\\_NAME](#) 16
- #define [SIZE\\_OF\\_BRICK\\_NAME](#) 8
- #define [SIZE\\_OF\\_CLASS\\_OF\\_DEVICE](#) 4
- #define [SIZE\\_OF\\_BT\\_PINCODE](#) 16
- #define [SIZE\\_OF\\_BDADDR](#) 7
- #define [MAX\\_BT\\_MSG\\_SIZE](#) 60000
- #define [BT\\_DEFAULT\\_INQUIRY\\_MAX](#) 0
- #define [BT\\_DEFAULT\\_INQUIRY\\_TIMEOUT\\_LO](#) 15
- #define [BT\\_ARM\\_OFF](#) 0
- #define [BT\\_ARM\\_CMD\\_MODE](#) 1
- #define [BT\\_ARM\\_DATA\\_MODE](#) 2
- #define [DATA\\_MODE\\_NXT](#) 0x00
- #define [DATA\\_MODE\\_GPS](#) 0x01
- #define [DATA\\_MODE\\_RAW](#) 0x02
- #define [DATA\\_MODE\\_MASK](#) 0x07
- #define [DATA\\_MODE\\_UPDATE](#) 0x08
- #define [BT\\_BRICK\\_VISIBILITY](#) 0x01
- #define [BT\\_BRICK\\_PORT\\_OPEN](#) 0x02
- #define [BT\\_CONNECTION\\_0\\_ENABLE](#) 0x10
- #define [BT\\_CONNECTION\\_1\\_ENABLE](#) 0x20
- #define [BT\\_CONNECTION\\_2\\_ENABLE](#) 0x40
- #define [BT\\_CONNECTION\\_3\\_ENABLE](#) 0x80
- #define [CONN\\_BT0](#) 0x0
- #define [CONN\\_BT1](#) 0x1
- #define [CONN\\_BT2](#) 0x2
- #define [CONN\\_BT3](#) 0x3
- #define [CONN\\_HS4](#) 0x4

- #define `CONN_HS_ALL` 0x4
- #define `CONN_HS_1` 0x5
- #define `CONN_HS_2` 0x6
- #define `CONN_HS_3` 0x7
- #define `CONN_HS_4` 0x8
- #define `CONN_HS_5` 0x9
- #define `CONN_HS_6` 0xa
- #define `CONN_HS_7` 0xb
- #define `CONN_HS_8` 0xc
- #define `BT_ENABLE` 0x00
- #define `BT_DISABLE` 0x01
- #define `HS_UPDATE` 1
- #define `HS_INITIALISE` 1
- #define `HS_INIT_RECEIVER` 2
- #define `HS_SEND_DATA` 3
- #define `HS_DISABLE` 4
- #define `HS_ENABLE` 5
- #define `HS_DEFAULT` 6
- #define `HS_BYTES_REMAINING` 16
- #define `HS_CTRL_INIT` 0
- #define `HS_CTRL_UART` 1
- #define `HS_CTRL_EXIT` 2
- #define `HS_BAUD_1200` 0
- #define `HS_BAUD_2400` 1
- #define `HS_BAUD_3600` 2
- #define `HS_BAUD_4800` 3
- #define `HS_BAUD_7200` 4
- #define `HS_BAUD_9600` 5
- #define `HS_BAUD_14400` 6
- #define `HS_BAUD_19200` 7
- #define `HS_BAUD_28800` 8
- #define `HS_BAUD_38400` 9
- #define `HS_BAUD_57600` 10
- #define `HS_BAUD_76800` 11
- #define `HS_BAUD_115200` 12
- #define `HS_BAUD_230400` 13
- #define `HS_BAUD_460800` 14
- #define `HS_BAUD_921600` 15
- #define `HS_BAUD_DEFAULT` 15
- #define `HS_MODE_UART_RS485` 0x0
- #define `HS_MODE_UART_RS232` 0x1
- #define `HS_MODE_MASK` 0xFFFF0
- #define `HS_UART_MASK` 0x000F

- #define [HS\\_MODE\\_DEFAULT](#) HS\_MODE\_8N1
- #define [HS\\_MODE\\_5\\_DATA](#) 0x0000
- #define [HS\\_MODE\\_6\\_DATA](#) 0x0040
- #define [HS\\_MODE\\_7\\_DATA](#) 0x0080
- #define [HS\\_MODE\\_8\\_DATA](#) 0x00C0
- #define [HS\\_MODE\\_10\\_STOP](#) 0x0000
- #define [HS\\_MODE\\_15\\_STOP](#) 0x1000
- #define [HS\\_MODE\\_20\\_STOP](#) 0x2000
- #define [HS\\_MODE\\_E\\_PARITY](#) 0x0000
- #define [HS\\_MODE\\_O\\_PARITY](#) 0x0200
- #define [HS\\_MODE\\_S\\_PARITY](#) 0x0400
- #define [HS\\_MODE\\_M\\_PARITY](#) 0x0600
- #define [HS\\_MODE\\_N\\_PARITY](#) 0x0800
- #define [HS\\_MODE\\_8N1](#) (HS\_MODE\_8\_DATA|HS\_MODE\_N\_PARITY|HS\_MODE\_10\_STOP)
- #define [HS\\_MODE\\_7E1](#) (HS\_MODE\_7\_DATA|HS\_MODE\_E\_PARITY|HS\_MODE\_10\_STOP)
- #define [HS\\_ADDRESS\\_ALL](#) 0
- #define [HS\\_ADDRESS\\_1](#) 1
- #define [HS\\_ADDRESS\\_2](#) 2
- #define [HS\\_ADDRESS\\_3](#) 3
- #define [HS\\_ADDRESS\\_4](#) 4
- #define [HS\\_ADDRESS\\_5](#) 5
- #define [HS\\_ADDRESS\\_6](#) 6
- #define [HS\\_ADDRESS\\_7](#) 7
- #define [HS\\_ADDRESS\\_8](#) 8
- #define [BT\\_DEVICE\\_EMPTY](#) 0x00
- #define [BT\\_DEVICE\\_UNKNOWN](#) 0x01
- #define [BT\\_DEVICE\\_KNOWN](#) 0x02
- #define [BT\\_DEVICE\\_NAME](#) 0x40
- #define [BT\\_DEVICE\\_AWAY](#) 0x80
- #define [INTF\\_SENDFILE](#) 0
- #define [INTF\\_SEARCH](#) 1
- #define [INTF\\_STOPSEARCH](#) 2
- #define [INTF\\_CONNECT](#) 3
- #define [INTF\\_DISCONNECT](#) 4
- #define [INTF\\_DISCONNECTALL](#) 5
- #define [INTF\\_REMOVEDEVICE](#) 6
- #define [INTF\\_VISIBILITY](#) 7
- #define [INTF\\_SETCMDMODE](#) 8
- #define [INTF\\_OPENSTREAM](#) 9
- #define [INTF\\_SENDDATA](#) 10
- #define [INTF\\_FACTORYRESET](#) 11

- #define [INTF\\_BTON](#) 12
- #define [INTF\\_BTOFF](#) 13
- #define [INTF\\_SETBTNAME](#) 14
- #define [INTF\\_EXTREAD](#) 15
- #define [INTF\\_PINREQ](#) 16
- #define [INTF\\_CONNECTREQ](#) 17
- #define [INTF\\_CONNECTBYNAME](#) 18
- #define [LR\\_SUCCESS](#) 0x50
- #define [LR\\_COULD\\_NOT\\_SAVE](#) 0x51
- #define [LR\\_STORE\\_IS\\_FULL](#) 0x52
- #define [LR\\_ENTRY\\_REMOVED](#) 0x53
- #define [LR\\_UNKNOWN\\_ADDR](#) 0x54
- #define [USB\\_CMD\\_READY](#) 0x01
- #define [BT\\_CMD\\_READY](#) 0x02
- #define [HS\\_CMD\\_READY](#) 0x04
- #define [CommOffsetPFunc](#) 0
- #define [CommOffsetPFuncTwo](#) 4
- #define [CommOffsetBtDeviceTableName\(p\)](#) (((p)\*31)+8)
- #define [CommOffsetBtDeviceTableClassOfDevice\(p\)](#) (((p)\*31)+24)
- #define [CommOffsetBtDeviceTableBdAddr\(p\)](#) (((p)\*31)+28)
- #define [CommOffsetBtDeviceTableDeviceStatus\(p\)](#) (((p)\*31)+35)
- #define [CommOffsetBtConnectTableName\(p\)](#) (((p)\*47)+938)
- #define [CommOffsetBtConnectTableClassOfDevice\(p\)](#) (((p)\*47)+954)
- #define [CommOffsetBtConnectTablePinCode\(p\)](#) (((p)\*47)+958)
- #define [CommOffsetBtConnectTableBdAddr\(p\)](#) (((p)\*47)+974)
- #define [CommOffsetBtConnectTableHandleNr\(p\)](#) (((p)\*47)+981)
- #define [CommOffsetBtConnectTableStreamStatus\(p\)](#) (((p)\*47)+982)
- #define [CommOffsetBtConnectTableLinkQuality\(p\)](#) (((p)\*47)+983)
- #define [CommOffsetBrickDataName](#) 1126
- #define [CommOffsetBrickDataBluecoreVersion](#) 1142
- #define [CommOffsetBrickDataBdAddr](#) 1144
- #define [CommOffsetBrickDataBtStateStatus](#) 1151
- #define [CommOffsetBrickDataBtHwStatus](#) 1152
- #define [CommOffsetBrickDataTimeOutValue](#) 1153
- #define [CommOffsetBtInBufBuf](#) 1157
- #define [CommOffsetBtInBufInPtr](#) 1285
- #define [CommOffsetBtInBufOutPtr](#) 1286
- #define [CommOffsetBtOutBufBuf](#) 1289
- #define [CommOffsetBtOutBufInPtr](#) 1417
- #define [CommOffsetBtOutBufOutPtr](#) 1418
- #define [CommOffsetHsInBufBuf](#) 1421
- #define [CommOffsetHsInBufInPtr](#) 1549
- #define [CommOffsetHsInBufOutPtr](#) 1550

- #define [CommOffsetHsOutBufBuf](#) 1553
- #define [CommOffsetHsOutBufInPtr](#) 1681
- #define [CommOffsetHsOutBufOutPtr](#) 1682
- #define [CommOffsetUsbInBufBuf](#) 1685
- #define [CommOffsetUsbInBufInPtr](#) 1749
- #define [CommOffsetUsbInBufOutPtr](#) 1750
- #define [CommOffsetUsbOutBufBuf](#) 1753
- #define [CommOffsetUsbOutBufInPtr](#) 1817
- #define [CommOffsetUsbOutBufOutPtr](#) 1818
- #define [CommOffsetUsbPollBufBuf](#) 1821
- #define [CommOffsetUsbPollBufInPtr](#) 1885
- #define [CommOffsetUsbPollBufOutPtr](#) 1886
- #define [CommOffsetBtDeviceCnt](#) 1889
- #define [CommOffsetBtDeviceNameCnt](#) 1890
- #define [CommOffsetHsFlags](#) 1891
- #define [CommOffsetHsSpeed](#) 1892
- #define [CommOffsetHsState](#) 1893
- #define [CommOffsetUsbState](#) 1894
- #define [CommOffsetHsAddress](#) 1895
- #define [CommOffsetHsMode](#) 1896
- #define [CommOffsetBtDataMode](#) 1898
- #define [CommOffsetHsDataMode](#) 1899
- #define [RCX\\_OUT\\_A](#) 0x01
- #define [RCX\\_OUT\\_B](#) 0x02
- #define [RCX\\_OUT\\_C](#) 0x04
- #define [RCX\\_OUT\\_AB](#) 0x03
- #define [RCX\\_OUT\\_AC](#) 0x05
- #define [RCX\\_OUT\\_BC](#) 0x06
- #define [RCX\\_OUT\\_ABC](#) 0x07
- #define [RCX\\_OUT\\_FLOAT](#) 0
- #define [RCX\\_OUT\\_OFF](#) 0x40
- #define [RCX\\_OUT\\_ON](#) 0x80
- #define [RCX\\_OUT\\_REV](#) 0
- #define [RCX\\_OUT\\_TOGGLE](#) 0x40
- #define [RCX\\_OUT\\_FWD](#) 0x80
- #define [RCX\\_OUT\\_LOW](#) 0
- #define [RCX\\_OUT\\_HALF](#) 3
- #define [RCX\\_OUT\\_FULL](#) 7
- #define [RCX\\_RemoteKeysReleased](#) 0x0000
- #define [RCX\\_RemotePBMessage1](#) 0x0100
- #define [RCX\\_RemotePBMessage2](#) 0x0200
- #define [RCX\\_RemotePBMessage3](#) 0x0400
- #define [RCX\\_RemoteOutAForward](#) 0x0800

- #define [RCX\\_RemoteOutBForward](#) 0x1000
- #define [RCX\\_RemoteOutCForward](#) 0x2000
- #define [RCX\\_RemoteOutABackward](#) 0x4000
- #define [RCX\\_RemoteOutBBackward](#) 0x8000
- #define [RCX\\_RemoteOutCBackward](#) 0x0001
- #define [RCX\\_RemoteSelProgram1](#) 0x0002
- #define [RCX\\_RemoteSelProgram2](#) 0x0004
- #define [RCX\\_RemoteSelProgram3](#) 0x0008
- #define [RCX\\_RemoteSelProgram4](#) 0x0010
- #define [RCX\\_RemoteSelProgram5](#) 0x0020
- #define [RCX\\_RemoteStopOutOff](#) 0x0040
- #define [RCX\\_RemotePlayASound](#) 0x0080
- #define [SOUND\\_CLICK](#) 0
- #define [SOUND\\_DOUBLE\\_BEEP](#) 1
- #define [SOUND\\_DOWN](#) 2
- #define [SOUND\\_UP](#) 3
- #define [SOUND\\_LOW\\_BEEP](#) 4
- #define [SOUND\\_FAST\\_UP](#) 5
- #define [SCOUT\\_LIGHT\\_ON](#) 0x80
- #define [SCOUT\\_LIGHT\\_OFF](#) 0
- #define [SCOUT\\_SOUND\\_REMOTE](#) 6
- #define [SCOUT\\_SOUND\\_ENTERSA](#) 7
- #define [SCOUT\\_SOUND\\_KEYERROR](#) 8
- #define [SCOUT\\_SOUND\\_NONE](#) 9
- #define [SCOUT\\_SOUND\\_TOUCH1\\_PRES](#) 10
- #define [SCOUT\\_SOUND\\_TOUCH1\\_REL](#) 11
- #define [SCOUT\\_SOUND\\_TOUCH2\\_PRES](#) 12
- #define [SCOUT\\_SOUND\\_TOUCH2\\_REL](#) 13
- #define [SCOUT\\_SOUND\\_ENTER\\_BRIGHT](#) 14
- #define [SCOUT\\_SOUND\\_ENTER\\_NORMAL](#) 15
- #define [SCOUT\\_SOUND\\_ENTER\\_DARK](#) 16
- #define [SCOUT\\_SOUND\\_1\\_BLINK](#) 17
- #define [SCOUT\\_SOUND\\_2\\_BLINK](#) 18
- #define [SCOUT\\_SOUND\\_COUNTER1](#) 19
- #define [SCOUT\\_SOUND\\_COUNTER2](#) 20
- #define [SCOUT\\_SOUND\\_TIMER1](#) 21
- #define [SCOUT\\_SOUND\\_TIMER2](#) 22
- #define [SCOUT\\_SOUND\\_TIMER3](#) 23
- #define [SCOUT\\_SOUND\\_MAIL\\_RECEIVED](#) 24
- #define [SCOUT\\_SOUND\\_SPECIAL1](#) 25
- #define [SCOUT\\_SOUND\\_SPECIAL2](#) 26
- #define [SCOUT\\_SOUND\\_SPECIAL3](#) 27
- #define [SCOUT\\_SNDSET\\_NONE](#) 0

- #define SCOUT\_SNDSET\_BASIC 1
- #define SCOUT\_SNDSET\_BUG 2
- #define SCOUT\_SNDSET\_ALARM 3
- #define SCOUT\_SNDSET\_RANDOM 4
- #define SCOUT\_SNDSET\_SCIENCE 5
- #define SCOUT\_MODE\_STANDALONE 0
- #define SCOUT\_MODE\_POWER 1
- #define SCOUT\_MR\_NO\_MOTION 0
- #define SCOUT\_MR\_FORWARD 1
- #define SCOUT\_MR\_ZIGZAG 2
- #define SCOUT\_MR\_CIRCLE\_RIGHT 3
- #define SCOUT\_MR\_CIRCLE\_LEFT 4
- #define SCOUT\_MR\_LOOP\_A 5
- #define SCOUT\_MR\_LOOP\_B 6
- #define SCOUT\_MR\_LOOP\_AB 7
- #define SCOUT\_TR\_IGNORE 0
- #define SCOUT\_TR\_REVERSE 1
- #define SCOUT\_TR\_AVOID 2
- #define SCOUT\_TR\_WAIT\_FOR 3
- #define SCOUT\_TR\_OFF\_WHEN 4
- #define SCOUT\_LR\_IGNORE 0
- #define SCOUT\_LR\_SEEK\_LIGHT 1
- #define SCOUT\_LR\_SEEK\_DARK 2
- #define SCOUT\_LR\_AVOID 3
- #define SCOUT\_LR\_WAIT\_FOR 4
- #define SCOUT\_LR\_OFF\_WHEN 5
- #define SCOUT\_TGS\_SHORT 0
- #define SCOUT\_TGS\_MEDIUM 1
- #define SCOUT\_TGS\_LONG 2
- #define SCOUT\_FXR\_NONE 0
- #define SCOUT\_FXR\_BUG 1
- #define SCOUT\_FXR\_ALARM 2
- #define SCOUT\_FXR\_RANDOM 3
- #define SCOUT\_FXR\_SCIENCE 4
- #define RCX\_VariableSrc 0
- #define RCX\_TimerSrc 1
- #define RCX\_ConstantSrc 2
- #define RCX\_OutputStatusSrc 3
- #define RCX\_RandomSrc 4
- #define RCX\_ProgramSlotSrc 8
- #define RCX\_InputValueSrc 9
- #define RCX\_InputTypeSrc 10
- #define RCX\_InputModeSrc 11

- #define [RCX\\_InputRawSrc](#) 12
- #define [RCX\\_InputBooleanSrc](#) 13
- #define [RCX\\_WatchSrc](#) 14
- #define [RCX\\_MessageSrc](#) 15
- #define [RCX\\_GlobalMotorStatusSrc](#) 17
- #define [RCX\\_ScoutRulesSrc](#) 18
- #define [RCX\\_ScoutLightParamsSrc](#) 19
- #define [RCX\\_ScoutTimerLimitSrc](#) 20
- #define [RCX\\_CounterSrc](#) 21
- #define [RCX\\_ScoutCounterLimitSrc](#) 22
- #define [RCX\\_TaskEventsSrc](#) 23
- #define [RCX\\_ScoutEventFBSrc](#) 24
- #define [RCX\\_EventStateSrc](#) 25
- #define [RCX\\_TenMSTimerSrc](#) 26
- #define [RCX\\_ClickCounterSrc](#) 27
- #define [RCX\\_UpperThresholdSrc](#) 28
- #define [RCX\\_LowerThresholdSrc](#) 29
- #define [RCX\\_HysteresisSrc](#) 30
- #define [RCX\\_DurationSrc](#) 31
- #define [RCX\\_UARTSetupSrc](#) 33
- #define [RCX\\_BatteryLevelSrc](#) 34
- #define [RCX\\_FirmwareVersionSrc](#) 35
- #define [RCX\\_IndirectVarSrc](#) 36
- #define [RCX\\_DatalogSrcIndirectSrc](#) 37
- #define [RCX\\_DatalogSrcDirectSrc](#) 38
- #define [RCX\\_DatalogValueIndirectSrc](#) 39
- #define [RCX\\_DatalogValueDirectSrc](#) 40
- #define [RCX\\_DatalogRawIndirectSrc](#) 41
- #define [RCX\\_DatalogRawDirectSrc](#) 42
- #define [RCX\\_PingOp](#) 0x10
- #define [RCX\\_BatteryLevelOp](#) 0x30
- #define [RCX\\_DeleteTasksOp](#) 0x40
- #define [RCX\\_StopAllTasksOp](#) 0x50
- #define [RCX\\_PBTurnOffOp](#) 0x60
- #define [RCX\\_DeleteSubsOp](#) 0x70
- #define [RCX\\_ClearSoundOp](#) 0x80
- #define [RCX\\_ClearMsgOp](#) 0x90
- #define [RCX\\_LSCalibrateOp](#) 0xc0
- #define [RCX\\_MuteSoundOp](#) 0xd0
- #define [RCX\\_UnmuteSoundOp](#) 0xe0
- #define [RCX\\_ClearAllEventsOp](#) 0x06
- #define [RCX\\_OnOffFloatOp](#) 0x21
- #define [RCX\\_IRModeOp](#) 0x31

- #define [RCX\\_PlaySoundOp](#) 0x51
- #define [RCX\\_DeleteTaskOp](#) 0x61
- #define [RCX\\_StartTaskOp](#) 0x71
- #define [RCX\\_StopTaskOp](#) 0x81
- #define [RCX\\_SelectProgramOp](#) 0x91
- #define [RCX\\_ClearTimerOp](#) 0xa1
- #define [RCX\\_AutoOffOp](#) 0xb1
- #define [RCX\\_DeleteSubOp](#) 0xc1
- #define [RCX\\_ClearSensorOp](#) 0xd1
- #define [RCX\\_OutputDirOp](#) 0xe1
- #define [RCX\\_PlayToneVarOp](#) 0x02
- #define [RCX\\_PollOp](#) 0x12
- #define [RCX\\_SetWatchOp](#) 0x22
- #define [RCX\\_InputTypeOp](#) 0x32
- #define [RCX\\_InputModeOp](#) 0x42
- #define [RCX\\_SetDatalogOp](#) 0x52
- #define [RCX\\_DatalogOp](#) 0x62
- #define [RCX\\_SendUARTDataOp](#) 0xc2
- #define [RCX\\_RemoteOp](#) 0xd2
- #define [RCX\\_VLLOp](#) 0xe2
- #define [RCX\\_DirectEventOp](#) 0x03
- #define [RCX\\_OutputPowerOp](#) 0x13
- #define [RCX\\_PlayToneOp](#) 0x23
- #define [RCX\\_DisplayOp](#) 0x33
- #define [RCX\\_PollMemoryOp](#) 0x63
- #define [RCX\\_SetFeedbackOp](#) 0x83
- #define [RCX\\_SetEventOp](#) 0x93
- #define [RCX\\_GOutputPowerOp](#) 0xa3
- #define [RCX\\_LSUpperThreshOp](#) 0xb3
- #define [RCX\\_LSLowerThreshOp](#) 0xc3
- #define [RCX\\_LSHysteresisOp](#) 0xd3
- #define [RCX\\_LSBlinkTimeOp](#) 0xe3
- #define [RCX\\_CalibrateEventOp](#) 0x04
- #define [RCX\\_SetVarOp](#) 0x14
- #define [RCX\\_SumVarOp](#) 0x24
- #define [RCX\\_SubVarOp](#) 0x34
- #define [RCX\\_DivVarOp](#) 0x44
- #define [RCX\\_MulVarOp](#) 0x54
- #define [RCX\\_SgnVarOp](#) 0x64
- #define [RCX\\_AbsVarOp](#) 0x74
- #define [RCX\\_AndVarOp](#) 0x84
- #define [RCX\\_OrVarOp](#) 0x94
- #define [RCX\\_UploadDatalogOp](#) 0xa4

- #define [RCX\\_SetTimerLimitOp](#) 0xc4
- #define [RCX\\_SetCounterOp](#) 0xd4
- #define [RCX\\_SetSourceValueOp](#) 0x05
- #define [RCX\\_UnlockOp](#) 0x15
- #define [RCX\\_BootModeOp](#) 0x65
- #define [RCX\\_UnlockFirmOp](#) 0xa5
- #define [RCX\\_ScoutRulesOp](#) 0xd5
- #define [RCX\\_ViewSourceValOp](#) 0xe5
- #define [RCX\\_ScoutOp](#) 0x47
- #define [RCX\\_SoundOp](#) 0x57
- #define [RCX\\_GOutputModeOp](#) 0x67
- #define [RCX\\_GOutputDirOp](#) 0x77
- #define [RCX\\_LightOp](#) 0x87
- #define [RCX\\_IncCounterOp](#) 0x97
- #define [RCX\\_DecCounterOp](#) 0xa7
- #define [RCX\\_ClearCounterOp](#) 0xb7
- #define [RCX\\_SetPriorityOp](#) 0xd7
- #define [RCX\\_MessageOp](#) 0xf7
- #define [PF\\_CMD\\_STOP](#) 0
- #define [PF\\_CMD\\_FLOAT](#) 0
- #define [PF\\_CMD\\_FWD](#) 1
- #define [PF\\_CMD\\_REV](#) 2
- #define [PF\\_CMD\\_BRAKE](#) 3
- #define [PF\\_CHANNEL\\_1](#) 0
- #define [PF\\_CHANNEL\\_2](#) 1
- #define [PF\\_CHANNEL\\_3](#) 2
- #define [PF\\_CHANNEL\\_4](#) 3
- #define [PF\\_MODE\\_TRAIN](#) 0
- #define [PF\\_MODE\\_COMBO\\_DIRECT](#) 1
- #define [PF\\_MODE\\_SINGLE\\_PIN\\_CONT](#) 2
- #define [PF\\_MODE\\_SINGLE\\_PIN\\_TIME](#) 3
- #define [PF\\_MODE\\_COMBO\\_PWM](#) 4
- #define [PF\\_MODE\\_SINGLE\\_OUTPUT\\_PWM](#) 4
- #define [PF\\_MODE\\_SINGLE\\_OUTPUT\\_CST](#) 6
- #define [TRAIN\\_FUNC\\_STOP](#) 0
- #define [TRAIN\\_FUNC\\_INCR\\_SPEED](#) 1
- #define [TRAIN\\_FUNC\\_DECR\\_SPEED](#) 2
- #define [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#) 4
- #define [TRAIN\\_CHANNEL\\_1](#) 0
- #define [TRAIN\\_CHANNEL\\_2](#) 1
- #define [TRAIN\\_CHANNEL\\_3](#) 2
- #define [TRAIN\\_CHANNEL\\_ALL](#) 3
- #define [PF\\_OUT\\_A](#) 0

- #define [PF\\_OUT\\_B](#) 1
- #define [PF\\_PIN\\_C1](#) 0
- #define [PF\\_PIN\\_C2](#) 1
- #define [PF\\_FUNC\\_NOCHANGE](#) 0
- #define [PF\\_FUNC\\_CLEAR](#) 1
- #define [PF\\_FUNC\\_SET](#) 2
- #define [PF\\_FUNC\\_TOGGLE](#) 3
- #define [PF\\_CST\\_CLEAR1\\_CLEAR2](#) 0
- #define [PF\\_CST\\_SET1\\_CLEAR2](#) 1
- #define [PF\\_CST\\_CLEAR1\\_SET2](#) 2
- #define [PF\\_CST\\_SET1\\_SET2](#) 3
- #define [PF\\_CST\\_INCREMENT\\_PWM](#) 4
- #define [PF\\_CST\\_DECREMENT\\_PWM](#) 5
- #define [PF\\_CST\\_FULL\\_FWD](#) 6
- #define [PF\\_CST\\_FULL\\_REV](#) 7
- #define [PF\\_CST\\_TOGGLE\\_DIR](#) 8
- #define [PF\\_PWM\\_FLOAT](#) 0
- #define [PF\\_PWM\\_FWD1](#) 1
- #define [PF\\_PWM\\_FWD2](#) 2
- #define [PF\\_PWM\\_FWD3](#) 3
- #define [PF\\_PWM\\_FWD4](#) 4
- #define [PF\\_PWM\\_FWD5](#) 5
- #define [PF\\_PWM\\_FWD6](#) 6
- #define [PF\\_PWM\\_FWD7](#) 7
- #define [PF\\_PWM\\_BRAKE](#) 8
- #define [PF\\_PWM\\_REV7](#) 9
- #define [PF\\_PWM\\_REV6](#) 10
- #define [PF\\_PWM\\_REV5](#) 11
- #define [PF\\_PWM\\_REV4](#) 12
- #define [PF\\_PWM\\_REV3](#) 13
- #define [PF\\_PWM\\_REV2](#) 14
- #define [PF\\_PWM\\_REV1](#) 15
- #define [HT\\_ADDR\\_IRSEEKER](#) 0x02
- #define [HT\\_ADDR\\_IRSEEKER2](#) 0x10
- #define [HT\\_ADDR\\_IRRECEIVER](#) 0x02
- #define [HT\\_ADDR\\_COMPASS](#) 0x02
- #define [HT\\_ADDR\\_ACCEL](#) 0x02
- #define [HT\\_ADDR\\_COLOR](#) 0x02
- #define [HT\\_ADDR\\_COLOR2](#) 0x02
- #define [HT\\_ADDR\\_IRLINK](#) 0x02
- #define [HT\\_ADDR\\_ANGLE](#) 0x02
- #define [HT\\_ADDR\\_BAROMETRIC](#) 0x02
- #define [HT\\_ADDR\\_PROTOBOARD](#) 0x02

- #define HT\_ADDR\_SUPERPRO 0x10
- #define HTIR2\_MODE\_1200 0
- #define HTIR2\_MODE\_600 1
- #define HTIR2\_REG\_MODE 0x41
- #define HTIR2\_REG\_DCDIR 0x42
- #define HTIR2\_REG\_DC01 0x43
- #define HTIR2\_REG\_DC02 0x44
- #define HTIR2\_REG\_DC03 0x45
- #define HTIR2\_REG\_DC04 0x46
- #define HTIR2\_REG\_DC05 0x47
- #define HTIR2\_REG\_DCAVG 0x48
- #define HTIR2\_REG\_ACDIR 0x49
- #define HTIR2\_REG\_AC01 0x4A
- #define HTIR2\_REG\_AC02 0x4B
- #define HTIR2\_REG\_AC03 0x4C
- #define HTIR2\_REG\_AC04 0x4D
- #define HTIR2\_REG\_AC05 0x4E
- #define HT\_CH1\_A 0
- #define HT\_CH1\_B 1
- #define HT\_CH2\_A 2
- #define HT\_CH2\_B 3
- #define HT\_CH3\_A 4
- #define HT\_CH3\_B 5
- #define HT\_CH4\_A 6
- #define HT\_CH4\_B 7
- #define HT\_CMD\_COLOR2\_ACTIVE 0x00
- #define HT\_CMD\_COLOR2\_PASSIVE 0x01
- #define HT\_CMD\_COLOR2\_RAW 0x03
- #define HT\_CMD\_COLOR2\_50HZ 0x35
- #define HT\_CMD\_COLOR2\_60HZ 0x36
- #define HT\_CMD\_COLOR2\_BLCAL 0x42
- #define HT\_CMD\_COLOR2\_WBCAL 0x43
- #define HT\_CMD\_COLOR2\_FAR 0x46
- #define HT\_CMD\_COLOR2\_LED\_HI 0x48
- #define HT\_CMD\_COLOR2\_LED\_LOW 0x4C
- #define HT\_CMD\_COLOR2\_NEAR 0x4E
- #define HTANGLE\_MODE\_NORMAL 0x00
- #define HTANGLE\_MODE\_CALIBRATE 0x43
- #define HTANGLE\_MODE\_RESET 0x52
- #define HTANGLE\_REG\_MODE 0x41
- #define HTANGLE\_REG\_DCDIR 0x42
- #define HTANGLE\_REG\_DC01 0x43
- #define HTANGLE\_REG\_DC02 0x44

- #define HTANGLE\_REG\_DC03 0x45
- #define HTANGLE\_REG\_DC04 0x46
- #define HTANGLE\_REG\_DC05 0x47
- #define HTANGLE\_REG\_DCAVG 0x48
- #define HTANGLE\_REG\_ACDIR 0x49
- #define HTBAR\_REG\_COMMAND 0x40
- #define HTBAR\_REG\_TEMPERATURE 0x42
- #define HTBAR\_REG\_PRESSURE 0x44
- #define HTBAR\_REG\_CALIBRATION 0x46
- #define HTPROTO\_REG\_A0 0x42
- #define HTPROTO\_REG\_A1 0x44
- #define HTPROTO\_REG\_A2 0x46
- #define HTPROTO\_REG\_A3 0x48
- #define HTPROTO\_REG\_A4 0x4A
- #define HTPROTO\_REG\_DIN 0x4C
- #define HTPROTO\_REG\_DOUT 0x4D
- #define HTPROTO\_REG\_DCTRL 0x4E
- #define HTPROTO\_REG\_SRATE 0x4F
- #define HTPROTO\_A0 0x42
- #define HTPROTO\_A1 0x44
- #define HTPROTO\_A2 0x46
- #define HTPROTO\_A3 0x48
- #define HTPROTO\_A4 0x4A
- #define HTSPRO\_REG\_CTRL 0x40
- #define HTSPRO\_REG\_A0 0x42
- #define HTSPRO\_REG\_A1 0x44
- #define HTSPRO\_REG\_A2 0x46
- #define HTSPRO\_REG\_A3 0x48
- #define HTSPRO\_REG\_DIN 0x4C
- #define HTSPRO\_REG\_DOUT 0x4D
- #define HTSPRO\_REG\_DCTRL 0x4E
- #define HTSPRO\_REG\_STROBE 0x50
- #define HTSPRO\_REG\_LED 0x51
- #define HTSPRO\_REG\_DAC0\_MODE 0x52
- #define HTSPRO\_REG\_DAC0\_FREQ 0x53
- #define HTSPRO\_REG\_DAC0\_VOLTAGE 0x55
- #define HTSPRO\_REG\_DAC1\_MODE 0x57
- #define HTSPRO\_REG\_DAC1\_FREQ 0x58
- #define HTSPRO\_REG\_DAC1\_VOLTAGE 0x5A
- #define HTSPRO\_REG\_DLADDRESS 0x60
- #define HTSPRO\_REG\_DLDATA 0x62
- #define HTSPRO\_REG\_DLCHKSUM 0x6A
- #define HTSPRO\_REG\_DLCONTROL 0x6B

- #define HTSPRO\_REG\_MEMORY\_20 0x80
- #define HTSPRO\_REG\_MEMORY\_21 0x84
- #define HTSPRO\_REG\_MEMORY\_22 0x88
- #define HTSPRO\_REG\_MEMORY\_23 0x8C
- #define HTSPRO\_REG\_MEMORY\_24 0x90
- #define HTSPRO\_REG\_MEMORY\_25 0x94
- #define HTSPRO\_REG\_MEMORY\_26 0x98
- #define HTSPRO\_REG\_MEMORY\_27 0x9C
- #define HTSPRO\_REG\_MEMORY\_28 0xA0
- #define HTSPRO\_REG\_MEMORY\_29 0xA4
- #define HTSPRO\_REG\_MEMORY\_2A 0xA8
- #define HTSPRO\_REG\_MEMORY\_2B 0xAC
- #define HTSPRO\_REG\_MEMORY\_2C 0xB0
- #define HTSPRO\_REG\_MEMORY\_2D 0xB4
- #define HTSPRO\_REG\_MEMORY\_2E 0xB8
- #define HTSPRO\_REG\_MEMORY\_2F 0xBC
- #define HTSPRO\_REG\_MEMORY\_30 0xC0
- #define HTSPRO\_REG\_MEMORY\_31 0xC4
- #define HTSPRO\_REG\_MEMORY\_32 0xC8
- #define HTSPRO\_REG\_MEMORY\_33 0xCC
- #define HTSPRO\_REG\_MEMORY\_34 0xD0
- #define HTSPRO\_REG\_MEMORY\_35 0xD4
- #define HTSPRO\_REG\_MEMORY\_36 0xD8
- #define HTSPRO\_REG\_MEMORY\_37 0xDC
- #define HTSPRO\_REG\_MEMORY\_38 0xE0
- #define HTSPRO\_REG\_MEMORY\_39 0xE4
- #define HTSPRO\_REG\_MEMORY\_3A 0xE8
- #define HTSPRO\_REG\_MEMORY\_3B 0xEC
- #define HTSPRO\_REG\_MEMORY\_3C 0xF0
- #define HTSPRO\_REG\_MEMORY\_3D 0xF4
- #define HTSPRO\_REG\_MEMORY\_3E 0xF8
- #define HTSPRO\_REG\_MEMORY\_3F 0xFC
- #define HTSPRO\_A0 0x42
- #define HTSPRO\_A1 0x44
- #define HTSPRO\_A2 0x46
- #define HTSPRO\_A3 0x48
- #define HTSPRO\_DAC0 0x52
- #define HTSPRO\_DAC1 0x57
- #define LED\_BLUE 0x02
- #define LED\_RED 0x01
- #define LED\_NONE 0x00
- #define DAC\_MODE\_DCOOUT 0
- #define DAC\_MODE\_SINEWAVE 1

- #define [DAC\\_MODE\\_SQUAREWAVE](#) 2
- #define [DAC\\_MODE\\_SAWPOSWAVE](#) 3
- #define [DAC\\_MODE\\_SAWNEGWAVE](#) 4
- #define [DAC\\_MODE\\_TRIANGLEWAVE](#) 5
- #define [DAC\\_MODE\\_PWMVOLTAGE](#) 6
- #define [DIGI\\_PIN0](#) 0x01
- #define [DIGI\\_PIN1](#) 0x02
- #define [DIGI\\_PIN2](#) 0x04
- #define [DIGI\\_PIN3](#) 0x08
- #define [DIGI\\_PIN4](#) 0x10
- #define [DIGI\\_PIN5](#) 0x20
- #define [DIGI\\_PIN6](#) 0x40
- #define [DIGI\\_PIN7](#) 0x80
- #define [STROBE\\_S0](#) 0x01
- #define [STROBE\\_S1](#) 0x02
- #define [STROBE\\_S2](#) 0x04
- #define [STROBE\\_S3](#) 0x08
- #define [STROBE\\_READ](#) 0x10
- #define [STROBE\\_WRITE](#) 0x20
- #define [MS\\_CMD\\_ENERGIZED](#) 0x45
- #define [MS\\_CMD\\_DEENERGIZED](#) 0x44
- #define [MS\\_CMD\\_ADPA\\_ON](#) 0x4E
- #define [MS\\_CMD\\_ADPA\\_OFF](#) 0x4F
- #define [MS\\_ADDR\\_RTCLOCK](#) 0xD0
- #define [MS\\_ADDR\\_DISTNX](#) 0x02
- #define [MS\\_ADDR\\_NRLINK](#) 0x02
- #define [MS\\_ADDR\\_ACCLNX](#) 0x02
- #define [MS\\_ADDR\\_CMPSNX](#) 0x02
- #define [MS\\_ADDR\\_PSPNX](#) 0x02
- #define [MS\\_ADDR\\_LINELDR](#) 0x02
- #define [MS\\_ADDR\\_NXTCAM](#) 0x02
- #define [MS\\_ADDR\\_NXTHID](#) 0x04
- #define [MS\\_ADDR\\_NXTSERVO](#) 0xB0
- #define [MS\\_ADDR\\_NXTSERVO\\_EM](#) 0x40
- #define [MS\\_ADDR\\_PFMATE](#) 0x48
- #define [MS\\_ADDR\\_MTRMUX](#) 0xB4
- #define [MS\\_ADDR\\_NXTMMX](#) 0x06
- #define [MS\\_ADDR\\_IVSENS](#) 0x12
- #define [MS\\_ADDR\\_RXMUX](#) 0x7E
- #define [DIST\\_CMD\\_GP2D12](#) 0x31
- #define [DIST\\_CMD\\_GP2D120](#) 0x32
- #define [DIST\\_CMD\\_GP2YA21](#) 0x33
- #define [DIST\\_CMD\\_GP2YA02](#) 0x34

- #define [DIST\\_CMD\\_CUSTOM](#) 0x35
- #define [DIST\\_REG\\_DIST](#) 0x42
- #define [DIST\\_REG\\_VOLT](#) 0x44
- #define [DIST\\_REG\\_MODULE\\_TYPE](#) 0x50
- #define [DIST\\_REG\\_NUM\\_POINTS](#) 0x51
- #define [DIST\\_REG\\_DIST\\_MIN](#) 0x52
- #define [DIST\\_REG\\_DIST\\_MAX](#) 0x54
- #define [DIST\\_REG\\_VOLT1](#) 0x56
- #define [DIST\\_REG\\_DIST1](#) 0x58
- #define [PSP\\_CMD\\_DIGITAL](#) 0x41
- #define [PSP\\_CMD\\_ANALOG](#) 0x73
- #define [PSP\\_REG\\_BTNSET1](#) 0x42
- #define [PSP\\_REG\\_BTNSET2](#) 0x43
- #define [PSP\\_REG\\_XLEFT](#) 0x44
- #define [PSP\\_REG\\_YLEFT](#) 0x45
- #define [PSP\\_REG\\_XRIGHT](#) 0x46
- #define [PSP\\_REG\\_YRIGHT](#) 0x47
- #define [PSP\\_BTNSET1\\_LEFT](#) 0x80
- #define [PSP\\_BTNSET1\\_DOWN](#) 0x40
- #define [PSP\\_BTNSET1\\_RIGHT](#) 0x20
- #define [PSP\\_BTNSET1\\_UP](#) 0x10
- #define [PSP\\_BTNSET1\\_START](#) 0x08
- #define [PSP\\_BTNSET1\\_R3](#) 0x04
- #define [PSP\\_BTNSET1\\_L3](#) 0x02
- #define [PSP\\_BTNSET1\\_SELECT](#) 0x01
- #define [PSP\\_BTNSET2\\_SQUARE](#) 0x80
- #define [PSP\\_BTNSET2\\_CROSS](#) 0x40
- #define [PSP\\_BTNSET2\\_CIRCLE](#) 0x20
- #define [PSP\\_BTNSET2\\_TRIANGLE](#) 0x10
- #define [PSP\\_BTNSET2\\_R1](#) 0x08
- #define [PSP\\_BTNSET2\\_L1](#) 0x04
- #define [PSP\\_BTNSET2\\_R2](#) 0x02
- #define [PSP\\_BTNSET2\\_L2](#) 0x01
- #define [NRLINK\\_CMD\\_2400](#) 0x44
- #define [NRLINK\\_CMD\\_FLUSH](#) 0x46
- #define [NRLINK\\_CMD\\_4800](#) 0x48
- #define [NRLINK\\_CMD\\_IR\\_LONG](#) 0x4C
- #define [NRLINK\\_CMD\\_IR\\_SHORT](#) 0x53
- #define [NRLINK\\_CMD\\_RUN\\_MACRO](#) 0x52
- #define [NRLINK\\_CMD\\_TX\\_RAW](#) 0x55
- #define [NRLINK\\_CMD\\_SET\\_RCX](#) 0x58
- #define [NRLINK\\_CMD\\_SET\\_TRAIN](#) 0x54
- #define [NRLINK\\_CMD\\_SET\\_PF](#) 0x50

- #define [NRLINK\\_REG\\_BYTES](#) 0x40
- #define [NRLINK\\_REG\\_DATA](#) 0x42
- #define [NRLINK\\_REG\\_EEPROM](#) 0x50
- #define [ACCL\\_CMD\\_X\\_CAL](#) 0x58
- #define [ACCL\\_CMD\\_Y\\_CAL](#) 0x59
- #define [ACCL\\_CMD\\_Z\\_CAL](#) 0x5a
- #define [ACCL\\_CMD\\_X\\_CAL\\_END](#) 0x78
- #define [ACCL\\_CMD\\_Y\\_CAL\\_END](#) 0x79
- #define [ACCL\\_CMD\\_Z\\_CAL\\_END](#) 0x7a
- #define [ACCL\\_CMD\\_RESET\\_CAL](#) 0x52
- #define [ACCL\\_REG\\_SENS\\_LVL](#) 0x19
- #define [ACCL\\_REG\\_X\\_TILT](#) 0x42
- #define [ACCL\\_REG\\_Y\\_TILT](#) 0x43
- #define [ACCL\\_REG\\_Z\\_TILT](#) 0x44
- #define [ACCL\\_REG\\_X\\_ACCEL](#) 0x45
- #define [ACCL\\_REG\\_Y\\_ACCEL](#) 0x47
- #define [ACCL\\_REG\\_Z\\_ACCEL](#) 0x49
- #define [ACCL\\_REG\\_X\\_OFFSET](#) 0x4b
- #define [ACCL\\_REG\\_X\\_RANGE](#) 0x4d
- #define [ACCL\\_REG\\_Y\\_OFFSET](#) 0x4f
- #define [ACCL\\_REG\\_Y\\_RANGE](#) 0x51
- #define [ACCL\\_REG\\_Z\\_OFFSET](#) 0x53
- #define [ACCL\\_REG\\_Z\\_RANGE](#) 0x55
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_1](#) 0x31
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_2](#) 0x32
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_3](#) 0x33
- #define [ACCL\\_SENSITIVITY\\_LEVEL\\_4](#) 0x34
- #define [PFMATE\\_REG\\_CMD](#) 0x41
- #define [PFMATE\\_REG\\_CHANNEL](#) 0x42
- #define [PFMATE\\_REG\\_MOTORS](#) 0x43
- #define [PFMATE\\_REG\\_A\\_CMD](#) 0x44
- #define [PFMATE\\_REG\\_A\\_SPEED](#) 0x45
- #define [PFMATE\\_REG\\_B\\_CMD](#) 0x46
- #define [PFMATE\\_REG\\_B\\_SPEED](#) 0x47
- #define [PFMATE\\_CMD\\_GO](#) 0x47
- #define [PFMATE\\_CMD\\_RAW](#) 0x52
- #define [PFMATE\\_MOTORS\\_BOTH](#) 0x00
- #define [PFMATE\\_MOTORS\\_A](#) 0x01
- #define [PFMATE\\_MOTORS\\_B](#) 0x02
- #define [PFMATE\\_CHANNEL\\_1](#) 1
- #define [PFMATE\\_CHANNEL\\_2](#) 2
- #define [PFMATE\\_CHANNEL\\_3](#) 3
- #define [PFMATE\\_CHANNEL\\_4](#) 4

- #define `NXTSERVO_REG_VOLTAGE` 0x41
- #define `NXTSERVO_REG_CMD` 0x41
- #define `NXTSERVO_REG_S1_POS` 0x42
- #define `NXTSERVO_REG_S2_POS` 0x44
- #define `NXTSERVO_REG_S3_POS` 0x46
- #define `NXTSERVO_REG_S4_POS` 0x48
- #define `NXTSERVO_REG_S5_POS` 0x4A
- #define `NXTSERVO_REG_S6_POS` 0x4C
- #define `NXTSERVO_REG_S7_POS` 0x4E
- #define `NXTSERVO_REG_S8_POS` 0x50
- #define `NXTSERVO_REG_S1_SPEED` 0x52
- #define `NXTSERVO_REG_S2_SPEED` 0x53
- #define `NXTSERVO_REG_S3_SPEED` 0x54
- #define `NXTSERVO_REG_S4_SPEED` 0x55
- #define `NXTSERVO_REG_S5_SPEED` 0x56
- #define `NXTSERVO_REG_S6_SPEED` 0x57
- #define `NXTSERVO_REG_S7_SPEED` 0x58
- #define `NXTSERVO_REG_S8_SPEED` 0x59
- #define `NXTSERVO_REG_S1_QPOS` 0x5A
- #define `NXTSERVO_REG_S2_QPOS` 0x5B
- #define `NXTSERVO_REG_S3_QPOS` 0x5C
- #define `NXTSERVO_REG_S4_QPOS` 0x5D
- #define `NXTSERVO_REG_S5_QPOS` 0x5E
- #define `NXTSERVO_REG_S6_QPOS` 0x5F
- #define `NXTSERVO_REG_S7_QPOS` 0x60
- #define `NXTSERVO_REG_S8_QPOS` 0x61
- #define `NXTSERVO_EM_REG_CMD` 0x00
- #define `NXTSERVO_EM_REG_EEPROM_START` 0x21
- #define `NXTSERVO_EM_REG_EEPROM_END` 0xFF
- #define `NXTSERVO_POS_CENTER` 1500
- #define `NXTSERVO_POS_MIN` 500
- #define `NXTSERVO_POS_MAX` 2500
- #define `NXTSERVO_QPOS_CENTER` 150
- #define `NXTSERVO_QPOS_MIN` 50
- #define `NXTSERVO_QPOS_MAX` 250
- #define `NXTSERVO_SERVO_1` 0
- #define `NXTSERVO_SERVO_2` 1
- #define `NXTSERVO_SERVO_3` 2
- #define `NXTSERVO_SERVO_4` 3
- #define `NXTSERVO_SERVO_5` 4
- #define `NXTSERVO_SERVO_6` 5
- #define `NXTSERVO_SERVO_7` 6
- #define `NXTSERVO_SERVO_8` 7

- #define `NXTSERVO_CMD_INIT` 0x49
- #define `NXTSERVO_CMD_RESET` 0x53
- #define `NXTSERVO_CMD_HALT` 0x48
- #define `NXTSERVO_CMD_RESUME` 0x52
- #define `NXTSERVO_CMD_GOTO` 0x47
- #define `NXTSERVO_CMD_PAUSE` 0x50
- #define `NXTSERVO_CMD_EDIT1` 0x45
- #define `NXTSERVO_CMD_EDIT2` 0x4D
- #define `NXTSERVO_EM_CMD_QUIT` 0x51
- #define `NXTHID_REG_CMD` 0x41
- #define `NXTHID_REG_MODIFIER` 0x42
- #define `NXTHID_REG_DATA` 0x43
- #define `NXTHID_MOD_NONE` 0x00
- #define `NXTHID_MOD_LEFT_CTRL` 0x01
- #define `NXTHID_MOD_LEFT_SHIFT` 0x02
- #define `NXTHID_MOD_LEFT_ALT` 0x04
- #define `NXTHID_MOD_LEFT_GUI` 0x08
- #define `NXTHID_MOD_RIGHT_CTRL` 0x10
- #define `NXTHID_MOD_RIGHT_SHIFT` 0x20
- #define `NXTHID_MOD_RIGHT_ALT` 0x40
- #define `NXTHID_MOD_RIGHT_GUI` 0x80
- #define `NXTHID_CMD_ASCII` 0x41
- #define `NXTHID_CMD_DIRECT` 0x44
- #define `NXTHID_CMD_TRANSMIT` 0x54
- #define `NXTPM_REG_CMD` 0x41
- #define `NXTPM_REG_CURRENT` 0x42
- #define `NXTPM_REG_VOLTAGE` 0x44
- #define `NXTPM_REG_CAPACITY` 0x46
- #define `NXTPM_REG_POWER` 0x48
- #define `NXTPM_REG_TOTALPOWER` 0x4A
- #define `NXTPM_REG_MAXCURRENT` 0x4E
- #define `NXTPM_REG_MINCURRENT` 0x50
- #define `NXTPM_REG_MAXVOLTAGE` 0x52
- #define `NXTPM_REG_MINVOLTAGE` 0x54
- #define `NXTPM_REG_TIME` 0x56
- #define `NXTPM_REG_USERGAIN` 0x5A
- #define `NXTPM_REG_GAIN` 0x5E
- #define `NXTPM_REG_ERRORCOUNT` 0x5F
- #define `NXTPM_CMD_RESET` 0x52
- #define `NXTSE_ZONE_NONE` 0
- #define `NXTSE_ZONE_FRONT` 1
- #define `NXTSE_ZONE_LEFT` 2
- #define `NXTSE_ZONE_RIGHT` 3

- #define `NXTLL_REG_CMD` 0x41
- #define `NXTLL_REG_STEERING` 0x42
- #define `NXTLL_REG_AVERAGE` 0x43
- #define `NXTLL_REG_RESULT` 0x44
- #define `NXTLL_REG_SETPOINT` 0x45
- #define `NXTLL_REG_KP_VALUE` 0x46
- #define `NXTLL_REG_KI_VALUE` 0x47
- #define `NXTLL_REG_KD_VALUE` 0x48
- #define `NXTLL_REG_CALIBRATED` 0x49
- #define `NXTLL_REG_WHITELIMITS` 0x51
- #define `NXTLL_REG_BLACKLIMITS` 0x59
- #define `NXTLL_REG_KP_FACTOR` 0x61
- #define `NXTLL_REG_KI_FACTOR` 0x62
- #define `NXTLL_REG_KD_FACTOR` 0x63
- #define `NXTLL_REG_WHITEDATA` 0x64
- #define `NXTLL_REG_BLACKDATA` 0x6C
- #define `NXTLL_REG_RAWVOLTAGE` 0x74
- #define `NXTLL_CMD_USA` 0x41
- #define `NXTLL_CMD_BLACK` 0x42
- #define `NXTLL_CMD_POWERDOWN` 0x44
- #define `NXTLL_CMD_EUROPEAN` 0x45
- #define `NXTLL_CMD_INVERT` 0x49
- #define `NXTLL_CMD_POWERUP` 0x50
- #define `NXTLL_CMD_RESET` 0x52
- #define `NXTLL_CMD_SNAPSHOT` 0x53
- #define `NXTLL_CMD_UNIVERSAL` 0x55
- #define `NXTLL_CMD_WHITE` 0x57
- #define `RFID_MODE_STOP` 0
- #define `RFID_MODE_SINGLE` 1
- #define `RFID_MODE_CONTINUOUS` 2
- #define `CT_ADDR_RFID` 0x04
- #define `CT_REG_STATUS` 0x32
- #define `CT_REG_MODE` 0x41
- #define `CT_REG_DATA` 0x42
- #define `DI_ADDR_DGPS` 0x06
- #define `DGPS_REG_TIME` 0x00
- #define `DGPS_REG_STATUS` 0x01
- #define `DGPS_REG_LATITUDE` 0x02
- #define `DGPS_REG_LONGITUDE` 0x04
- #define `DGPS_REG_VELOCITY` 0x06
- #define `DGPS_REG_HEADING` 0x07
- #define `DGPS_REG_DISTANCE` 0x08
- #define `DGPS_REG_WAYANGLE` 0x09

- #define `DGPS_REG_LASTANGLE` 0x0A
- #define `DGPS_REG_SETLATITUDE` 0x0B
- #define `DGPS_REG_SETLONGITUDE` 0x0C
- #define `DI_ADDR_GYRO` 0xD2
- #define `DI_ADDR_ACCL` 0x3A
- #define `DIGYRO_REG_WHOAMI` 0x0F
- #define `DIGYRO_REG_CTRL1` 0x20
- #define `DIGYRO_REG_CTRL2` 0x21
- #define `DIGYRO_REG_CTRL3` 0x22
- #define `DIGYRO_REG_CTRL4` 0x23
- #define `DIGYRO_REG_CTRL5` 0x24
- #define `DIGYRO_REG_REFERENCE` 0x25
- #define `DIGYRO_REG_OUTTEMP` 0x26
- #define `DIGYRO_REG_STATUS` 0x27
- #define `DIGYRO_REG_XLOW` 0x28
- #define `DIGYRO_REG_XHIGH` 0x29
- #define `DIGYRO_REG_YLOW` 0x2A
- #define `DIGYRO_REG_YHIGH` 0x2B
- #define `DIGYRO_REG_ZLOW` 0x2C
- #define `DIGYRO_REG_ZHIGH` 0x2D
- #define `DIGYRO_REG_FIFOCTRL` 0x2E
- #define `DIGYRO_REG_FIFOSRC` 0x2F
- #define `DIGYRO_REG_INT1_CFG` 0x30
- #define `DIGYRO_REG_INT1_SRC` 0x31
- #define `DIGYRO_REG_INT1_XHI` 0x32
- #define `DIGYRO_REG_INT1_XLO` 0x33
- #define `DIGYRO_REG_INT1_YHI` 0x34
- #define `DIGYRO_REG_INT1_YLO` 0x35
- #define `DIGYRO_REG_INT1_ZHI` 0x36
- #define `DIGYRO_REG_INT1_ZLO` 0x37
- #define `DIGYRO_REG_INT1_DUR` 0x38
- #define `DIGYRO_REG_CTRL1AUTO` 0xA0
- #define `DIGYRO_REG_TEMPAUTO` 0xA6
- #define `DIGYRO_REG_XLOWBURST` 0xA8
- #define `DIGYRO_REG_YLOWBURST` 0xAA
- #define `DIGYRO_REG_ZLOWBURST` 0xAC
- #define `DIGYRO_CTRL1_XENABLE` 0x01
- #define `DIGYRO_CTRL1_YENABLE` 0x02
- #define `DIGYRO_CTRL1_ZENABLE` 0x04
- #define `DIGYRO_CTRL1_POWERDOWN` 0x00
- #define `DIGYRO_CTRL1_NORMAL` 0x08
- #define `DIGYRO_CTRL1_BANDWIDTH_1` 0x00
- #define `DIGYRO_CTRL1_BANDWIDTH_2` 0x10

- #define DIGYRO\_CTRL1\_BANDWIDTH\_3 0x20
- #define DIGYRO\_CTRL1\_BANDWIDTH\_4 0x30
- #define DIGYRO\_CTRL1\_DATARATE\_100 0x00
- #define DIGYRO\_CTRL1\_DATARATE\_200 0x40
- #define DIGYRO\_CTRL1\_DATARATE\_400 0x80
- #define DIGYRO\_CTRL1\_DATARATE\_800 0xC0
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_8 0x00
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_4 0x01
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_2 0x02
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_1 0x03
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_05 0x04
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_02 0x05
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_01 0x06
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_005 0x07
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_002 0x08
- #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_001 0x09
- #define DIGYRO\_CTRL2\_HPMODE\_RESET 0x00
- #define DIGYRO\_CTRL2\_HPMODE\_REFSIG 0x10
- #define DIGYRO\_CTRL2\_HPMODE\_NORMAL 0x20
- #define DIGYRO\_CTRL2\_HPMODE\_AUTOINT 0x30
- #define DIGYRO\_CTRL3\_INT1\_ENABLE 0x80
- #define DIGYRO\_CTRL3\_INT1\_BOOT 0x40
- #define DIGYRO\_CTRL3\_INT1\_LOWACTIVE 0x20
- #define DIGYRO\_CTRL3\_OPENDRAIN 0x10
- #define DIGYRO\_CTRL3\_INT2\_DATAREADY 0x08
- #define DIGYRO\_CTRL3\_INT2\_WATERMARK 0x04
- #define DIGYRO\_CTRL3\_INT2\_OVERRUN 0x02
- #define DIGYRO\_CTRL3\_INT2\_EMPTY 0x01
- #define DIGYRO\_CTRL4\_BLOCKDATA 0x80
- #define DIGYRO\_CTRL4\_BIGENDIAN 0x40
- #define DIGYRO\_CTRL4\_SCALE\_250 0x00
- #define DIGYRO\_CTRL4\_SCALE\_500 0x10
- #define DIGYRO\_CTRL4\_SCALE\_2000 0x30
- #define DIGYRO\_CTRL5\_REBOOTMEM 0x80
- #define DIGYRO\_CTRL5\_FIFOENABLE 0x40
- #define DIGYRO\_CTRL5\_HPENABLE 0x10
- #define DIGYRO\_CTRL5\_OUT\_SEL\_1 0x00
- #define DIGYRO\_CTRL5\_OUT\_SEL\_2 0x01
- #define DIGYRO\_CTRL5\_OUT\_SEL\_3 0x02
- #define DIGYRO\_CTRL5\_INT1\_SEL\_1 0x00
- #define DIGYRO\_CTRL5\_INT1\_SEL\_2 0x04
- #define DIGYRO\_CTRL5\_INT1\_SEL\_3 0x08
- #define DIGYRO\_FIFOCTRL\_BYPASS 0x00

- #define DIGYRO\_FIFOCTRL\_FIFO 0x20
- #define DIGYRO\_FIFOCTRL\_STREAM 0x40
- #define DIGYRO\_FIFOCTRL\_STREAM2FIFO 0x60
- #define DIGYRO\_FIFOCTRL\_BYPASS2STREAM 0x80
- #define DIGYRO\_FIFOCTRL\_WATERMARK\_MASK 0x1F
- #define DIGYRO\_STATUS\_XDATA 0x01
- #define DIGYRO\_STATUS\_YDATA 0x02
- #define DIGYRO\_STATUS\_ZDATA 0x04
- #define DIGYRO\_STATUS\_XYZDATA 0x08
- #define DIGYRO\_STATUS\_XOVER 0x10
- #define DIGYRO\_STATUS\_YOVER 0x20
- #define DIGYRO\_STATUS\_ZOVER 0x40
- #define DIGYRO\_STATUS\_XYZOVER 0x80
- #define DIACCL\_REG\_XLOW 0x00
- #define DIACCL\_REG\_XHIGH 0x01
- #define DIACCL\_REG\_YLOW 0x02
- #define DIACCL\_REG\_YHIGH 0x03
- #define DIACCL\_REG\_ZLOW 0x04
- #define DIACCL\_REG\_ZHIGH 0x05
- #define DIACCL\_REG\_X8 0x06
- #define DIACCL\_REG\_Y8 0x07
- #define DIACCL\_REG\_Z8 0x08
- #define DIACCL\_REG\_STATUS 0x09
- #define DIACCL\_REG\_DETECTSRC 0x0A
- #define DIACCL\_REG\_OUTTEMP 0x0B
- #define DIACCL\_REG\_I2CADDR 0x0D
- #define DIACCL\_REG\_USERINFO 0x0E
- #define DIACCL\_REG\_WHOAMI 0x0F
- #define DIACCL\_REG\_XLOWDRIFT 0x10
- #define DIACCL\_REG\_XHIGHDRIFT 0x11
- #define DIACCL\_REG\_YLOWDRIFT 0x12
- #define DIACCL\_REG\_YHIGHDRIFT 0x13
- #define DIACCL\_REG\_ZLOWDRIFT 0x14
- #define DIACCL\_REG\_ZHIGHDRIFT 0x15
- #define DIACCL\_REG\_MODECTRL 0x16
- #define DIACCL\_REG\_INTLATCH 0x17
- #define DIACCL\_REG\_CTRL1 0x18
- #define DIACCL\_REG\_CTRL2 0x19
- #define DIACCL\_REG\_LVLDETTTHR 0x1A
- #define DIACCL\_REG\_PLSDETTHR 0x1B
- #define DIACCL\_REG\_PLSDURVAL 0x1C
- #define DIACCL\_REG\_LATENCYTM 0x1D
- #define DIACCL\_REG\_TIMEWINDOW 0x1E

- #define `DIACCL_STATUS_DATAREADY` 0x01
- #define `DIACCL_STATUS_DATAOVER` 0x02
- #define `DIACCL_STATUS_PARITYERR` 0x04
- #define `DIACCL_MODE_STANDBY` 0x00
- #define `DIACCL_MODE_MEASURE` 0x01
- #define `DIACCL_MODE_LVLDETECT` 0x02
- #define `DIACCL_MODE_PLSDetect` 0x03
- #define `DIACCL_MODE_GLVL8` 0x00
- #define `DIACCL_MODE_GLVL2` 0x04
- #define `DIACCL_MODE_GLVL4` 0x08
- #define `DIACCL_INTERRUPT_LATCH_CLEAR1` 0x01
- #define `DIACCL_INTERRUPT_LATCH_CLEAR2` 0x02
- #define `DIACCL_CTRL1_INT2TOINT1` 0x01
- #define `DIACCL_CTRL1_LEVELPULSE` 0x00
- #define `DIACCL_CTRL1_PULSELEVEL` 0x02
- #define `DIACCL_CTRL1_PULSEPULSE` 0x04
- #define `DIACCL_CTRL1_NO_XDETECT` 0x08
- #define `DIACCL_CTRL1_NO_YDETECT` 0x10
- #define `DIACCL_CTRL1_NO_ZDETECT` 0x20
- #define `DIACCL_CTRL1_THRESH_INT` 0x40
- #define `DIACCL_CTRL1_FILT_BW125` 0x80
- #define `DIACCL_CTRL2_LVLPOL_NEGAND` 0x01
- #define `DIACCL_CTRL2_DETPOL_NEGAND` 0x02
- #define `DIACCL_CTRL2_DRIVE_STRONG` 0x04
- #define `MI_ADDR_XG1300L` 0x02
- #define `XG1300L_REG_ANGLE` 0x42
- #define `XG1300L_REG_TURNRATE` 0x44
- #define `XG1300L_REG_XAXIS` 0x46
- #define `XG1300L_REG_YAXIS` 0x48
- #define `XG1300L_REG_ZAXIS` 0x4A
- #define `XG1300L_REG_RESET` 0x60
- #define `XG1300L_REG_2G` 0x61
- #define `XG1300L_REG_4G` 0x62
- #define `XG1300L_REG_8G` 0x63
- #define `XG1300L_SCALE_2G` 0x01
- #define `XG1300L_SCALE_4G` 0x02
- #define `XG1300L_SCALE_8G` 0x04
- #define `RICImgPoint(_X, _Y)` `(_X)&0xFF, (_X)>>8, (_Y)&0xFF, (_Y)>>8`  
*Output an RIC ImgPoint structure.*
- #define `RICImgRect(_Pt, _W, _H)` `_Pt, (_W)&0xFF, (_W)>>8, (_H)&0xFF, (_H)>>8`

*Output an RIC ImgRect structure.*

- #define `RICOpDescription`(\_Options, \_Width, \_Height) 8, 0, 0, 0, (\_Options)&0xFF, (\_Options)>>8, (\_Width)&0xFF, (\_Width)>>8, (\_Height)&0xFF, (\_Height)>>8

*Output an RIC Description opcode.*

- #define `RICOpCopyBits`(\_CopyOptions, \_DataAddr, \_SrcRect, \_DstPoint) 18, 0, 3, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, (\_DataAddr)&0xFF, (\_DataAddr)>>8, \_SrcRect, \_DstPoint

*Output an RIC CopyBits opcode.*

- #define `RICOpPixel`(\_CopyOptions, \_Point, \_Value) 10, 0, 4, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Value)&0xFF, (\_Value)>>8

*Output an RIC Pixel opcode.*

- #define `RICOpLine`(\_CopyOptions, \_Point1, \_Point2) 12, 0, 5, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point1, \_Point2

*Output an RIC Line opcode.*

- #define `RICOpRect`(\_CopyOptions, \_Point, \_Width, \_Height) 12, 0, 6, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Width)&0xFF, (\_Width)>>8, (\_Height)&0xFF, (\_Height)>>8

*Output an RIC Rect opcode.*

- #define `RICOpCircle`(\_CopyOptions, \_Point, \_Radius) 10, 0, 7, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Radius)&0xFF, (\_Radius)>>8

*Output an RIC Circle opcode.*

- #define `RICOpNumBox`(\_CopyOptions, \_Point, \_Value) 10, 0, 8, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_Value)&0xFF, (\_Value)>>8

*Output an RIC NumBox opcode.*

- #define `RICOpSprite`(\_DataAddr, \_Rows, \_BytesPerRow, \_SpriteData) ((\_Rows\*\_BytesPerRow)+((\_Rows\*\_BytesPerRow)%2)+8)&0xFF, ((\_Rows\*\_BytesPerRow)+((\_Rows\*\_BytesPerRow)%2)+8)>>8, 1, 0, (\_DataAddr)&0xFF, (\_DataAddr)>>8, (\_Rows)&0xFF, (\_Rows)>>8, (\_BytesPerRow)&0xFF, (\_BytesPerRow)>>8, \_SpriteData

*Output an RIC Sprite opcode.*

- #define `RICSpriteData`(...) \_\_VA\_ARGS\_\_

*Output RIC sprite data.*

- #define `RICOpVarMap`(\_DataAddr, \_MapCount, \_MapFunction) ((-\_MapCount\*4)+6)&0xFF, ((\_MapCount\*4)+6)>>8, 2, 0, (\_DataAddr)&0xFF, (\_DataAddr)>>8, (\_MapCount)&0xFF, (\_MapCount)>>8, \_MapFunction

*Output an RIC VarMap opcode.*

- #define `RICMapElement`(\_Domain, \_Range) (\_Domain)&0xFF, (\_Domain)>>8, (\_Range)&0xFF, (\_Range)>>8

*Output an RIC map element.*

- #define `RICMapFunction`(\_MapElement,...) \_MapElement, \_\_VA\_ARGS\_\_

*Output an RIC VarMap function.*

- #define `RICArg`(\_arg) ((\_arg)|0x1000)

*Output an RIC parameterized argument.*

- #define `RICMapArg`(\_mapidx, \_arg) ((\_arg)|0x1000|(((\_mapidx)&0xF)<<8))

*Output an RIC parameterized and mapped argument.*

- #define `RICOpPolygon`(\_CopyOptions, \_Count, \_ThePoints) ((-\_Count\*4)+6)&0xFF, ((\_Count\*4)+6)>>8, 10, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, (\_Count)&0xFF, (\_Count)>>8, \_ThePoints

*Output an RIC Polygon opcode.*

- #define `RICPolygonPoints`(\_pPoint1, \_pPoint2,...) \_pPoint1, \_pPoint2, \_\_VA\_ARGS\_\_

*Output RIC polygon points.*

- #define `RICOpEllipse`(\_CopyOptions, \_Point, \_RadiusX, \_RadiusY) 12, 0, 9, 0, (\_CopyOptions)&0xFF, (\_CopyOptions)>>8, \_Point, (\_RadiusX)&0xFF, (\_RadiusX)>>8, (\_RadiusY)&0xFF, (\_RadiusY)>>8

*Output an RIC Ellipse opcode.*

- #define `CHAR_BIT` 8
- #define `SCHAR_MIN` -128
- #define `SCHAR_MAX` 127
- #define `UCHAR_MAX` 255
- #define `CHAR_MIN` -128
- #define `CHAR_MAX` 127
- #define `SHRT_MIN` -32768
- #define `SHRT_MAX` 32767

- #define USHRT\_MAX 65535
- #define INT\_MIN -32768
- #define INT\_MAX 32767
- #define UINT\_MAX 65535
- #define LONG\_MIN -2147483648
- #define LONG\_MAX 2147483647
- #define ULONG\_MAX 4294967295
- #define RAND\_MAX 2147483646
- #define GL\_POLYGON 1
- #define GL\_LINE 2
- #define GL\_POINT 3
- #define GL\_CIRCLE 4
- #define GL\_TRANSLATE\_X 1
- #define GL\_TRANSLATE\_Y 2
- #define GL\_TRANSLATE\_Z 3
- #define GL\_ROTATE\_X 4
- #define GL\_ROTATE\_Y 5
- #define GL\_ROTATE\_Z 6
- #define GL\_SCALE\_X 7
- #define GL\_SCALE\_Y 8
- #define GL\_SCALE\_Z 9
- #define GL\_CIRCLE\_SIZE 1
- #define GL\_CULL\_MODE 2
- #define GL\_CAMERA\_DEPTH 3
- #define GL\_ZOOM\_FACTOR 4
- #define GL\_CULL\_BACK 2
- #define GL\_CULL\_FRONT 3
- #define GL\_CULL\_NONE 4

### 8.1.1 Detailed Description

Constants and macros common to both NBC and NXC. [NBCCommon.h](#) contains declarations for the NBC and NXC NEXT API functions.

License:

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of this code is John Hansen. Portions created by John Hansen are Copyright (C) 2009-2011 John Hansen. All Rights Reserved.

-----  
**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2011-10-10

**Version:**

69

**8.1.2 Define Documentation**

**8.1.2.1 #define ACCL\_CMD\_RESET\_CAL 0x52**

Reset to factory calibration

**8.1.2.2 #define ACCL\_CMD\_X\_CAL 0x58**

Acquire X-axis calibration point

**8.1.2.3 #define ACCL\_CMD\_X\_CAL\_END 0x78**

Acquire X-axis calibration point and end calibration

**8.1.2.4 #define ACCL\_CMD\_Y\_CAL 0x59**

Acquire Y-axis calibration point

**8.1.2.5 #define ACCL\_CMD\_Y\_CAL\_END 0x79**

Acquire Y-axis calibration point and end calibration

**8.1.2.6 #define ACCL\_CMD\_Z\_CAL 0x5a**

Acquire Z-axis calibration point

**8.1.2.7 #define ACCL\_CMD\_Z\_CAL\_END 0x7a**

Acquire Z-axis calibration point and end calibration

**8.1.2.8 #define ACCL\_REG\_SENS\_LVL 0x19**

The current sensitivity

**8.1.2.9 #define ACCL\_REG\_X\_ACCEL 0x45**

The X-axis acceleration data

**8.1.2.10 #define ACCL\_REG\_X\_OFFSET 0x4b**

The X-axis offset

**8.1.2.11 #define ACCL\_REG\_X\_RANGE 0x4d**

The X-axis range

**8.1.2.12 #define ACCL\_REG\_X\_TILT 0x42**

The X-axis tilt data

**8.1.2.13 #define ACCL\_REG\_Y\_ACCEL 0x47**

The Y-axis acceleration data

**8.1.2.14 #define ACCL\_REG\_Y\_OFFSET 0x4f**

The Y-axis offset

**8.1.2.15 #define ACCL\_REG\_Y\_RANGE 0x51**

The Y-axis range

**8.1.2.16 #define ACCL\_REG\_Y\_TILT 0x43**

The Y-axis tilt data

**8.1.2.17 #define ACCL\_REG\_Z\_ACCEL 0x49**

The Z-axis acceleration data

**8.1.2.18 #define ACCL\_REG\_Z\_OFFSET 0x53**

The Z-axis offset

**8.1.2.19 #define ACCL\_REG\_Z\_RANGE 0x55**

The Z-axis range

**8.1.2.20 #define ACCL\_REG\_Z\_TILT 0x44**

The Z-axis tilt data

**8.1.2.21 #define ACCL\_SENSITIVITY\_LEVEL\_1 0x31**

The ACCL-Nx sensitivity level 1

**Examples:**

[ex\\_SetACCLNxSensitivity.nxc](#).

**8.1.2.22 #define ACCL\_SENSITIVITY\_LEVEL\_2 0x32**

The ACCL-Nx sensitivity level 2

**8.1.2.23 #define ACCL\_SENSITIVITY\_LEVEL\_3 0x33**

The ACCL-Nx sensitivity level 3

**8.1.2.24 #define ACCL\_SENSITIVITY\_LEVEL\_4 0x34**

The ACCL-Nx sensitivity level 4

**8.1.2.25 #define ActualSpeedField 3**

Actual speed field. Contains the actual power level (-100 to 100). Read only. Return the percent of full power the firmware is applying to the output. This may vary from the PowerField value when auto-regulation code in the firmware responds to a load on the output.

**8.1.2.26 #define BITMAP\_1 0**

Bitmap 1

**8.1.2.27 #define BITMAP\_2 1**

Bitmap 2

**8.1.2.28 #define BITMAP\_3 2**

Bitmap 3

**8.1.2.29 #define BITMAP\_4 3**

Bitmap 4

**8.1.2.30 #define BITMAPS 4**

The number of bitmap bits

**8.1.2.31 #define BlockTachoCountField 13**

NXT-G block tachometer count field. Contains the current NXT-G block tachometer count. Read only. Return the block-relative position counter value for the specified port. Refer to the [UpdateFlagsField](#) description for information about how to use block-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) flag in [UpdateFlagsField](#) to request that the firmware reset the BlockTachoCountField. The sign of BlockTachoCountField indicates the direction of rotation. Positive values indicate forward rotation and negative values indicate reverse rotation. Forward and reverse depend on the orientation of the motor.

**8.1.2.32 #define BREAKOUT\_REQ 3**

VM should break out of current thread

**8.1.2.33 #define BT\_ARM\_CMD\_MODE 1**

BtState constant bluetooth command mode

**8.1.2.34 #define BT\_ARM\_DATA\_MODE 2**

BtState constant bluetooth data mode

**8.1.2.35 #define BT\_ARM\_OFF 0**

BtState constant bluetooth off

**8.1.2.36 #define BT\_BRICK\_PORT\_OPEN 0x02**

BtStateStatus port open bit

**8.1.2.37 #define BT\_BRICK\_VISIBILITY 0x01**

BtStateStatus brick visibility bit

**8.1.2.38 #define BT\_CMD\_BYTE 1**

Size of Bluetooth command

**8.1.2.39 #define BT\_CMD\_READY 0x02**

A constant representing bluetooth direct command

**8.1.2.40 #define BT\_CONNECTION\_0\_ENABLE 0x10**

BtStateStatus connection 0 enable/disable bit

**8.1.2.41 #define BT\_CONNECTION\_1\_ENABLE 0x20**

BtStateStatus connection 1 enable/disable bit

**8.1.2.42 #define BT\_CONNECTION\_2\_ENABLE 0x40**

BtStateStatus connection 2 enable/disable bit

**8.1.2.43 #define BT\_CONNECTION\_3\_ENABLE 0x80**

BtStateStatus connection 3 enable/disable bit

**8.1.2.44 #define BT\_DEFAULT\_INQUIRY\_MAX 0**

Bluetooth default inquiry Max (0 == unlimited)

**8.1.2.45 #define BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO 15**

Bluetooth inquiry timeout (15\*1.28 sec = 19.2 sec)

**8.1.2.46 #define BT\_DEVICE\_AWAY 0x80**

Bluetooth device away

**8.1.2.47 #define BT\_DEVICE\_EMPTY 0x00**

Bluetooth device table empty

**8.1.2.48 #define BT\_DEVICE\_KNOWN 0x02**

Bluetooth device known

**8.1.2.49 #define BT\_DEVICE\_NAME 0x40**

Bluetooth device name

**8.1.2.50 #define BT\_DEVICE\_UNKNOWN 0x01**

Bluetooth device unknown

**8.1.2.51 #define BT\_DISABLE 0x01**

BtHwStatus bluetooth disable

**8.1.2.52 #define BT\_ENABLE 0x00**

BtHwStatus bluetooth enable

**8.1.2.53 #define BTN1 0**

The exit button.

**Examples:**

[ex\\_ButtonCount.nxc](#), [ex\\_ButtonLongPressCount.nxc](#), [ex\\_ButtonLongReleaseCount.nxc](#), [ex\\_ButtonPressCount.nxc](#), [ex\\_ButtonReleaseCount.nxc](#), [ex\\_ButtonShortReleaseCount.nxc](#), [ex\\_ButtonState.nxc](#), [ex\\_ReadButtonEx.nxc](#), [ex\\_SetButtonLongPressCount.nxc](#), [ex\\_SetButtonLongReleaseCount.nxc](#), [ex\\_SetButtonPressCount.nxc](#), [ex\\_SetButtonReleaseCount.nxc](#), [ex\\_SetButtonShortReleaseCount.nxc](#), and [ex\\_SetButtonState.nxc](#).

**8.1.2.54 #define BTN2 1**

The right button.

**8.1.2.55 #define BTN3 2**

The left button.

**8.1.2.56 #define BTN4 3**

The enter button.

**8.1.2.57 #define BTNCENTER BTN4**

The enter button.

**Examples:**

[ex\\_buttonpressed.nxc](#), and [ex\\_HTGyroTest.nxc](#).

**8.1.2.58 #define BTNEXIT BTN1**

The exit button.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.1.2.59 #define BTNLEFT BTN3**

The left button.

**Examples:**

[ex\\_buttonpressed.nxc](#), and [ex\\_xg1300.nxc](#).

**8.1.2.60 #define BTNRIGHT BTN2**

The right button.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

**8.1.2.61 #define BTNSTATE\_LONG\_PRESSED\_EV 0x04**

Button is in the long pressed state.

**Examples:**

[ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.1.2.62 #define BTNSTATE\_LONG\_RELEASED\_EV 0x08**

Button is in the long released state.

**8.1.2.63 #define BTNSTATE\_NONE 0x10**

The default button state.

**8.1.2.64 #define BTNSTATE\_PRESSED\_EV 0x01**

Button is in the pressed state.

**Examples:**

[ex\\_SetButtonState.nxc](#).

**8.1.2.65 #define BTNSTATE\_PRESSED\_STATE 0x80**

A bitmask for the button pressed state

**8.1.2.66 #define BTNSTATE\_SHORT\_RELEASED\_EV 0x02**

Button is in the short released state.

**8.1.2.67 #define ButtonModuleID 0x00040001**

The button module ID

**8.1.2.68 #define ButtonModuleName "Button.mod"**

The button module name

**8.1.2.69 #define ButtonOffsetLongPressCnt(b) (((b)\*8)+1)**

Offset to the LongPressCnt field. This field stores the long press count.

**8.1.2.70 #define ButtonOffsetLongRelCnt(b) (((b)\*8)+3)**

Offset to the LongRelCnt field. This field stores the long release count.

**8.1.2.71 #define ButtonOffsetPressedCnt(b) (((b)\*8)+0)**

Offset to the PressedCnt field. This field stores the press count.

**8.1.2.72 #define ButtonOffsetRelCnt(b) (((b)\*8)+4)**

Offset to the RelCnt field. This field stores the release count.

**8.1.2.73 #define ButtonOffsetShortRelCnt(b) (((b)\*8)+2)**

Offset to the ShortRelCnt field. This field stores the short release count.

**8.1.2.74 #define ButtonOffsetState(b) ((b)+32)**

Offset to the State field. This field stores the current button state.

**8.1.2.75 #define CHAR\_BIT 8**

The number of bits in the char type

**8.1.2.76 #define CHAR\_MAX 127**

The maximum value of the char type

**8.1.2.77 #define CHAR\_MIN -128**

The minimum value of the char type

**8.1.2.78 #define CLUMP\_DONE 1**

VM has finished executing thread

**8.1.2.79 #define CLUMP\_SUSPEND 2**

VM should suspend thread

**8.1.2.80 #define ColorSensorRead 34**

Read data from the NXT 2.0 color sensor

**8.1.2.81 #define COM\_CHANNEL\_FOUR\_ACTIVE 0x08**

Low speed channel 4 is active

**8.1.2.82 #define COM\_CHANNEL\_NONE\_ACTIVE 0x00**

None of the low speed channels are active

**8.1.2.83 #define COM\_CHANNEL\_ONE\_ACTIVE 0x01**

Low speed channel 1 is active

**8.1.2.84 #define COM\_CHANNEL\_THREE\_ACTIVE 0x04**

Low speed channel 3 is active

**8.1.2.85 #define COM\_CHANNEL\_TWO\_ACTIVE 0x02**

Low speed channel 2 is active

**8.1.2.86 #define CommandModuleID 0x00010001**

The command module ID

**Examples:**

[ex\\_reladdressof.nxc](#), [ex\\_RemoteIOMapRead.nxc](#), [ex\\_RemoteIOMapWriteBytes.nxc](#), [ex\\_RemoteIOMapWriteValue.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**8.1.2.87 #define CommandModuleName "Command.mod"**

The command module name

**Examples:**

[ex\\_sysiomapread.nxc](#).

**8.1.2.88 #define CommandOffsetActivateFlag 30**

Offset to the activate flag

**8.1.2.89 #define CommandOffsetAwake 29**

Offset to the VM's awake state

**8.1.2.90 #define CommandOffsetDeactivateFlag 31**

Offset to the deactivate flag

**8.1.2.91 #define CommandOffsetFileName 32**

Offset to the running program's filename

**8.1.2.92 #define CommandOffsetFormatString 0**

Offset to the format string

**8.1.2.93 #define CommandOffsetMemoryPool 52**

Offset to the VM's memory pool

**Examples:**[ex\\_reladdressof.nxc](#).**8.1.2.94 #define CommandOffsetOffsetDS 24**

Offset to the running program's data space (DS)

**8.1.2.95 #define CommandOffsetOffsetDVA 26**

Offset to the running program's DOPE vector address (DVA)

**8.1.2.96 #define CommandOffsetPRCHandler 16**

Offset to the RC Handler function pointer

**8.1.2.97 #define CommandOffsetProgStatus 28**

Offset to the running program's status

**Examples:**[ex\\_RemoteIOMapRead.nxc](#), [ex\\_RemoteIOMapWriteBytes.nxc](#), and [ex\\_RemoteIOMapWriteValue.nxc](#).**8.1.2.98 #define CommandOffsetSyncTick 32824**

Offset to the VM sync tick

**8.1.2.99 #define CommandOffsetSyncTime 32820**

Offset to the VM sync time

**8.1.2.100 #define CommandOffsetTick 20**

Offset to the VM's current tick

**Examples:**

[ex\\_sysiomapread.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**8.1.2.101 #define CommBTCheckStatus 28**

Check the bluetooth status

**8.1.2.102 #define CommBTConnection 36**

Connect or disconnect to a known bluetooth device

**8.1.2.103 #define CommBTONOff 35**

Turn the bluetooth radio on or off

**8.1.2.104 #define CommBTRead 30**

Read from a bluetooth connection

**8.1.2.105 #define CommBTWrite 29**

Write to a bluetooth connections

**8.1.2.106 #define CommExecuteFunction 81**

Execute one of the Comm module's internal functions

**8.1.2.107 #define CommHSCheckStatus 39**

Check the status of the hi-speed port

**8.1.2.108 #define CommHSControl 88**

Control the hi-speed port

**8.1.2.109 #define CommHSRead 38**

Read data from the hi-speed port

**8.1.2.110 #define CommHSWrite 37**

Write data to the hi-speed port

**8.1.2.111 #define CommLSCheckStatus 23**

Check the status of a lowspeed (aka I2C) device

**8.1.2.112 #define CommLSRead 22**

Read from a lowspeed (aka I2C) device

**8.1.2.113 #define CommLSWrite 21**

Write to a lowspeed (aka I2C) device

**8.1.2.114 #define CommLSWriteEx 89**

Write to a lowspeed (aka I2C) device with optional restart on read

**8.1.2.115 #define CommModuleID 0x00050001**

The Comm module ID

**8.1.2.116 #define CommModuleName "Comm.mod"**

The Comm module name

**8.1.2.117 #define CommOffsetBrickDataBdAddr 1144**

Offset to Bluetooth address (7 bytes)

**8.1.2.118 #define CommOffsetBrickDataBluecoreVersion 1142**

Offset to Bluecore version (2 bytes)

**8.1.2.119 #define CommOffsetBrickDataBtHwStatus 1152**

Offset to BtHwStatus (1 byte)

**8.1.2.120 #define CommOffsetBrickDataBtStateStatus 1151**

Offset to BtStateStatus (1 byte)

**8.1.2.121 #define CommOffsetBrickDataName 1126**

Offset to brick name (16 bytes)

**8.1.2.122 #define CommOffsetBrickDataTimeOutValue 1153**

Offset to data timeout value (1 byte)

**8.1.2.123 #define CommOffsetBtConnectTableBdAddr(p) (((p)\*47)+974)**

Offset to Bluetooth connect table address (7 bytes)

**8.1.2.124 #define CommOffsetBtConnectTableClassOfDevice(p) (((p)\*47)+954)**

Offset to Bluetooth connect table device class (4 bytes)

**8.1.2.125 #define CommOffsetBtConnectTableHandleNr(p) (((p)\*47)+981)**

Offset to Bluetooth connect table handle (1 byte)

**8.1.2.126 #define CommOffsetBtConnectTableLinkQuality(p) (((p)\*47)+983)**

Offset to Bluetooth connect table link quality (1 byte)

**8.1.2.127 #define CommOffsetBtConnectTableName(p) (((p)\*47)+938)**

Offset to Bluetooth connect table name (16 bytes)

**8.1.2.128 #define CommOffsetBtConnectTablePinCode(p) (((p)\*47)+958)**

Offset to Bluetooth connect table pin code (16 bytes)

**8.1.2.129 #define CommOffsetBtConnectTableStreamStatus(p) (((p)\*47)+982)**

Offset to Bluetooth connect table stream status (1 byte)

**8.1.2.130 #define CommOffsetBtDataMode 1898**

Offset to Bluetooth data mode (1 byte)

**8.1.2.131 #define CommOffsetBtDeviceCnt 1889**

Offset to Bluetooth device count (1 byte)

**8.1.2.132 #define CommOffsetBtDeviceNameCnt 1890**

Offset to Bluetooth device name count (1 byte)

**8.1.2.133 #define CommOffsetBtDeviceTableBdAddr(p) (((p)\*31)+28)**

Offset to Bluetooth device table address (7 bytes)

**8.1.2.134 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)\*31)+24)**

Offset to Bluetooth device table device class (4 bytes)

**8.1.2.135 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)\*31)+35)**

Offset to Bluetooth device table status (1 byte)

**8.1.2.136 #define CommOffsetBtDeviceTableName(p) (((p)\*31)+8)**

Offset to BT device table name (16 bytes)

**8.1.2.137 #define CommOffsetBtInBufBuf 1157**

Offset to Bluetooth input buffer data (128 bytes)

**8.1.2.138 #define CommOffsetBtInBufInPtr 1285**

Offset to Bluetooth input buffer front pointer (1 byte)

**8.1.2.139 #define CommOffsetBtInBufOutPtr 1286**

Offset to Bluetooth output buffer back pointer (1 byte)

**8.1.2.140 #define CommOffsetBtOutBufBuf 1289**

Offset to Bluetooth output buffer offset data (128 bytes)

**8.1.2.141 #define CommOffsetBtOutBufInPtr 1417**

Offset to Bluetooth output buffer front pointer (1 byte)

**8.1.2.142 #define CommOffsetBtOutBufOutPtr 1418**

Offset to Bluetooth output buffer back pointer (1 byte)

**8.1.2.143 #define CommOffsetHsAddress 1895**

Offset to High Speed address (1 byte)

**8.1.2.144 #define CommOffsetHsDataMode 1899**

Offset to High Speed data mode (1 byte)

**8.1.2.145 #define CommOffsetHsFlags 1891**

Offset to High Speed flags (1 byte)

**8.1.2.146 #define CommOffsetHsInBufBuf 1421**

Offset to High Speed input buffer data (128 bytes)

**8.1.2.147 #define CommOffsetHsInBufInPtr 1549**

Offset to High Speed input buffer front pointer (1 byte)

**8.1.2.148 #define CommOffsetHsInBufOutPtr 1550**

Offset to High Speed input buffer back pointer (1 byte)

**8.1.2.149 #define CommOffsetHsMode 1896**

Offset to High Speed mode (2 bytes)

**8.1.2.150 #define CommOffsetHsOutBufBuf 1553**

Offset to High Speed output buffer data (128 bytes)

**8.1.2.151 #define CommOffsetHsOutBufInPtr 1681**

Offset to High Speed output buffer front pointer (1 byte)

**8.1.2.152 #define CommOffsetHsOutBufOutPtr 1682**

Offset to High Speed output buffer back pointer (1 byte)

**8.1.2.153 #define CommOffsetHsSpeed 1892**

Offset to High Speed speed (1 byte)

**8.1.2.154 #define CommOffsetHsState 1893**

Offset to High Speed state (1 byte)

**8.1.2.155 #define CommOffsetPFunc 0**

Offset to the Comm module first function pointer (4 bytes)

**8.1.2.156 #define CommOffsetPFuncTwo 4**

Offset to the Comm module second function pointer (4 bytes)

**8.1.2.157 #define CommOffsetUsbInBufBuf 1685**

Offset to Usb input buffer data (64 bytes)

**8.1.2.158 #define CommOffsetUsbInBufInPtr 1749**

Offset to Usb input buffer front pointer (1 byte)

**8.1.2.159 #define CommOffsetUsbInBufOutPtr 1750**

Offset to Usb input buffer back pointer (1 byte)

**8.1.2.160 #define CommOffsetUsbOutBufBuf 1753**

Offset to Usb output buffer data (64 bytes)

**8.1.2.161 #define CommOffsetUsbOutBufInPtr 1817**

Offset to Usb output buffer front pointer (1 byte)

**8.1.2.162 #define CommOffsetUsbOutBufOutPtr 1818**

Offset to Usb output buffer back pointer (1 byte)

**8.1.2.163 #define CommOffsetUsbPollBufBuf 1821**

Offset to Usb Poll buffer data (64 bytes)

**8.1.2.164 #define CommOffsetUsbPollBufInPtr 1885**

Offset to Usb Poll buffer front pointer (1 byte)

**8.1.2.165 #define CommOffsetUsbPollBufOutPtr 1886**

Offset to Usb Poll buffer back pointer (1 byte)

**8.1.2.166 #define CommOffsetUsbState 1894**

Offset to Usb State (1 byte)

**8.1.2.167 #define ComputeCalibValue 42**

Compute a calibration value

**8.1.2.168 #define CONN\_BT0 0x0**

Bluetooth connection 0

**8.1.2.169 #define CONN\_BT1 0x1**

Bluetooth connection 1

**Examples:**

ex\_RemoteCloseFile.nxc,           ex\_RemoteConnectionIdle.nxc,           ex\_  
 RemoteConnectionWrite.nxc,        ex\_RemoteDatalogRead.nxc,           ex\_  
 RemoteDatalogSetTimes.nxc,        ex\_RemoteDeleteFile.nxc,           ex\_  
 RemoteDeleteUserFlash.nxc,        ex\_RemoteFindFirstFile.nxc,        ex\_  
 RemoteFindNextFile.nxc,           ex\_RemoteGetBatteryLevel.nxc,        ex\_  
 RemoteGetBluetoothAddress.nxc,     ex\_RemoteGetConnectionCount.nxc,    ex\_  
 ex\_RemoteGetConnectionName.nxc,   ex\_RemoteGetContactCount.nxc,       ex\_  
 RemoteGetContactName.nxc,         ex\_RemoteGetCurrentProgramName.nxc, ex\_  
 ex\_RemoteGetDeviceInfo.nxc,        ex\_RemoteGetFirmwareVersion.nxc,    ex\_  
 ex\_RemoteGetInputValues.nxc,       ex\_RemoteGetOutputState.nxc,        ex\_  
 ex\_RemoteGetProperty.nxc,          ex\_RemoteIOMapRead.nxc,            ex\_  
 RemoteIOMapWriteBytes.nxc,         ex\_RemoteIOMapWriteValue.nxc,       ex\_  
 ex\_RemoteLowSpeedGetStatus.nxc,    ex\_RemoteLowSpeedRead.nxc,         ex\_  
 ex\_RemoteLowSpeedWrite.nxc,        ex\_RemoteOpenAppendData.nxc,        ex\_  
 ex\_RemoteOpenRead.nxc,            ex\_RemoteOpenWrite.nxc,            ex\_  
 RemoteOpenWriteData.nxc,           ex\_RemoteOpenWriteLinear.nxc,       ex\_  
 RemotePollCommand.nxc,            ex\_RemotePollCommandLength.nxc,    ex\_  
 RemoteRead.nxc, ex\_RemoteRenameFile.nxc, ex\_RemoteResetTachoCount.nxc,  
 ex\_RemoteSetProperty.nxc, and ex\_RemoteWrite.nxc.

**8.1.2.170 #define CONN\_BT2 0x2**

Bluetooth connection 2

**8.1.2.171 #define CONN\_BT3 0x3**

Bluetooth connection 3

**8.1.2.172 #define CONN\_HS4 0x4**

RS485 (hi-speed) connection (port 4, all devices)

**8.1.2.173 #define CONN\_HS\_1 0x5**

RS485 (hi-speed) connection (port 4, device address 1)

**8.1.2.174 #define CONN\_HS\_2 0x6**

RS485 (hi-speed) connection (port 4, device address 2)

**8.1.2.175 #define CONN\_HS\_3 0x7**

RS485 (hi-speed) connection (port 4, device address 3)

**8.1.2.176 #define CONN\_HS\_4 0x8**

RS485 (hi-speed) connection (port 4, device address 4)

**8.1.2.177 #define CONN\_HS\_5 0x9**

RS485 (hi-speed) connection (port 4, device address 5)

**8.1.2.178 #define CONN\_HS\_6 0xa**

RS485 (hi-speed) connection (port 4, device address 6)

**8.1.2.179 #define CONN\_HS\_7 0xb**

RS485 (hi-speed) connection (port 4, device address 7)

**8.1.2.180 #define CONN\_HS\_8 0xc**

RS485 (hi-speed) connection (port 4, device address 8)

**8.1.2.181 #define CONN\_HS\_ALL 0x4**

RS485 (hi-speed) connection (port 4, all devices)

**8.1.2.182 #define CT\_ADDR\_RFID 0x04**

RFID I2C address

**8.1.2.183 #define CT\_REG\_DATA 0x42**

RFID data register

**8.1.2.184 #define CT\_REG\_MODE 0x41**

RFID mode register

**8.1.2.185 #define CT\_REG\_STATUS 0x32**

RFID status register

**8.1.2.186 #define DAC\_MODE\_DCOUT 0**

Steady (DC) voltage output.

**8.1.2.187 #define DAC\_MODE\_PWMVOLTAGE 6**

PWM square wave output.

**8.1.2.188 #define DAC\_MODE\_SAWNEGWAVE 4**

Negative going sawtooth output.

**8.1.2.189 #define DAC\_MODE\_SAWPOSWAVE 3**

Positive going sawtooth output.

**8.1.2.190 #define DAC\_MODE\_SINEWAVE 1**

Sine wave output.

**Examples:**

[ex\\_superpro.nxc](#).

**8.1.2.191 #define DAC\_MODE\_SQUAREWAVE 2**

Square wave output.

**8.1.2.192 #define DAC\_MODE\_TRIANGLEWAVE 5**

Triangle wave output.

**8.1.2.193 #define DATA\_MODE\_GPS 0x01**

Use GPS data mode

**Examples:**

[ex\\_DataMode.nxc](#).

**8.1.2.194 #define DATA\_MODE\_MASK 0x07**

A mask for the data mode bits.

**8.1.2.195 #define DATA\_MODE\_NXT 0x00**

Use NXT data mode

**Examples:**

[ex\\_DataMode.nxc](#).

**8.1.2.196 #define DATA\_MODE\_RAW 0x02**

Use RAW data mode

**8.1.2.197 #define DATA\_MODE\_UPDATE 0x08**

Indicates that the data mode has been changed.

**8.1.2.198 #define DatalogGetTimes 45**

Get datalog timing information

**8.1.2.199 #define DatalogWrite 44**

Write to the datalog

**8.1.2.200 #define DEGREES\_PER\_RADIAN 180/PI**

Used for converting from radians to degrees

**8.1.2.201 #define DGPS\_REG\_DISTANCE 0x08**

Read distance to current waypoint in meters.

**8.1.2.202 #define DGPS\_REG\_HEADING 0x07**

Read heading in degrees.

**8.1.2.203 #define DGPS\_REG\_LASTANGLE 0x0A**

Read angle travelled since last request, resets the request coordinates on the GPS sensor, sends the angle of travel since last reset.

**8.1.2.204 #define DGPS\_REG\_LATITUDE 0x02**

Read integer latitude.(ddddddd; Positive = North; Negative = South).

**8.1.2.205 #define DGPS\_REG\_LONGITUDE 0x04**

Read integer longitude (ddddddd; Positive = East; Negative = West).

**8.1.2.206 #define DGPS\_REG\_SETLATITUDE 0x0B**

Set waypoint latitude as a 4 byte integer.

**8.1.2.207 #define DGPS\_REG\_SETLONGITUDE 0x0C**

Set waypoint longitude as a 4 byte integer.

**8.1.2.208 #define DGPS\_REG\_STATUS 0x01**

Read status of the GPS (0 - invalid signal, 1 - valid signal).

**8.1.2.209 #define DGPS\_REG\_TIME 0x00**

Read time in UTC (hhmmss).

**8.1.2.210 #define DGPS\_REG\_VELOCITY 0x06**

Read velocity in cm/s.

**8.1.2.211 #define DGPS\_REG\_WAYANGLE 0x09**

Read angle to current waypoint in degrees.

**8.1.2.212 #define DI\_ADDR\_ACCL 0x3A**

Dexter Industries DIMU Accelerometer I2C address

**8.1.2.213 #define DI\_ADDR\_DGPS 0x06**

Dexter Industries DGPS I2C address

**8.1.2.214 #define DI\_ADDR\_GYRO 0xD2**

Dexter Industries DIMU Gyro I2C address

**8.1.2.215 #define DIACCL\_CTRL1\_FILT\_BW125 0x80**

Accelerometer digital filter band width is 125 Hz.

**8.1.2.216 #define DIACCL\_CTRL1\_INT2TOINT1 0x01**

Accelerometer INT2 pin is routed to INT1 bit in Detection Source Register (\$0A) and INT1 pin is routed to INT2 bit in Detection Source Register (\$0A)

**8.1.2.217 #define DIACCL\_CTRL1\_LEVELPULSE 0x00**

Accelerometer INT1 register is detecting Level while INT2 is detecting pulse

**8.1.2.218 #define DIACCL\_CTRL1\_NO\_XDETECT 0x08**

Accelerometer disable x-axis detection.

**8.1.2.219 #define DIACCL\_CTRL1\_NO\_YDETECT 0x10**

Accelerometer disable y-axis detection.

**8.1.2.220 #define DIACCL\_CTRL1\_NO\_ZDETECT 0x20**

Accelerometer disable z-axis detection.

**8.1.2.221 #define DIACCL\_CTRL1\_PULSELEVEL 0x02**

Accelerometer INT1 Register is detecting Pulse while INT2 is detecting Level

**8.1.2.222 #define DIACCL\_CTRL1\_PULSEPULSE 0x04**

Accelerometer INT1 Register is detecting a Single Pulse and INT2 is detecting Single Pulse (if 2nd Time Window = 0) or if there is a latency time window and second time window > 0 then INT2 will detect the double pulse only.

**8.1.2.223 #define DIACCL\_CTRL1\_THRESH\_INT 0x40**

Accelerometer threshold value can be an integer.

**8.1.2.224 #define DIACCL\_CTRL2\_DETPOL\_NEGAND 0x02**

Accelerometer pulse detection polarity is negative and detecting condition is AND all 3 axes

**8.1.2.225 #define DIACCL\_CTRL2\_DRIVE\_STRONG 0x04**

Accelerometer strong drive strength on SDA/SDO pin

**8.1.2.226 #define DIACCL\_CTRL2\_LVLPOL\_NEGAND 0x01**

Accelerometer level detection polarity is negative and detecting condition is AND all 3 axes

**8.1.2.227 #define DIACCL\_INTERRUPT\_LATCH\_CLEAR1 0x01**

Accelerometer clear interrupt 1

**8.1.2.228 #define DIACCL\_INTERRUPT\_LATCH\_CLEAR2 0x02**

Accelerometer clear interrupt 2

**8.1.2.229 #define DIACCL\_MODE\_GLVL2 0x04**

Accelerometer 2G measurement range

**8.1.2.230 #define DIACCL\_MODE\_GLVL4 0x08**

Accelerometer 4G measurement range

**8.1.2.231 #define DIACCL\_MODE\_GLVL8 0x00**

Accelerometer 8G measurement range

**Examples:**[ex\\_diaccl.nxc](#).**8.1.2.232 #define DIACCL\_MODE\_LVLDETECT 0x02**

Accelerometer level detect mode

**8.1.2.233 #define DIACCL\_MODE\_MEASURE 0x01**

Accelerometer measurement mode

**8.1.2.234 #define DIACCL\_MODE\_PLSDETECT 0x03**

Accelerometer pulse detect mode

**8.1.2.235 #define DIACCL\_MODE\_STANDBY 0x00**

Accelerometer standby mode

**8.1.2.236 #define DIACCL\_REG\_CTRL1 0x18**

Accelerometer control register 1 (read/write)

**8.1.2.237 #define DIACCL\_REG\_CTRL2 0x19**

Accelerometer control register 1 (read/write)

**8.1.2.238 #define DIACCL\_REG\_DETECTSRC 0x0A**

Accelerometer detection source register (read only)

**8.1.2.239 #define DIACCL\_REG\_I2CADDR 0x0D**

Accelerometer I2C address register (read only)

**8.1.2.240 #define DIACCL\_REG\_INTLATCH 0x17**

Accelerometer interrupt latch reset register (read/write)

**8.1.2.241 #define DIACCL\_REG\_LATENCYTM 0x1D**

Accelerometer latency time value register (read/write)

**8.1.2.242 #define DIACCL\_REG\_LVLDETTHR 0x1A**

Accelerometer level detection threshold limit value register (read/write)

**8.1.2.243 #define DIACCL\_REG\_MODECTRL 0x16**

Accelerometer mode control register (read/write)

**8.1.2.244 #define DIACCL\_REG\_OUTTEMP 0x0B**

Accelerometer temperature output register (read only)

**8.1.2.245 #define DIACCL\_REG\_PLSDETTHR 0x1B**

Accelerometer pulse detection threshold limit value register (read/write)

**8.1.2.246 #define DIACCL\_REG\_PLSDURVAL 0x1C**

Accelerometer pulse duration value register (read/write)

**8.1.2.247 #define DIACCL\_REG\_STATUS 0x09**

Accelerometer status register (read only)

**8.1.2.248 #define DIACCL\_REG\_TIMEWINDOW 0x1E**

Accelerometer time window for 2nd pulse value register (read/write)

**8.1.2.249 #define DIACCL\_REG\_USERINFO 0x0E**

Accelerometer user information register (read only)

**8.1.2.250 #define DIACCL\_REG\_WHOAMI 0x0F**

Accelerometer device identification register (read only)

**8.1.2.251 #define DIACCL\_REG\_X8 0x06**

Accelerometer x-axis 8-bit register (read only)

**8.1.2.252 #define DIACCL\_REG\_XHIGH 0x01**

Accelerometer x-axis high byte register (read only)

**8.1.2.253 #define DIACCL\_REG\_XHIGHDRIFT 0x11**

Accelerometer x-axis offset drift high byte register (read/write)

**8.1.2.254 #define DIACCL\_REG\_XLOW 0x00**

Accelerometer x-axis low byte register (read only)

**8.1.2.255 #define DIACCL\_REG\_XLOWDRIFT 0x10**

Accelerometer x-axis offset drift low byte register (read/write)

**8.1.2.256 #define DIACCL\_REG\_Y8 0x07**

Accelerometer x-axis 8-bit register (read only)

**8.1.2.257 #define DIACCL\_REG\_YHIGH 0x03**

Accelerometer y-axis high byte register (read only)

**8.1.2.258 #define DIACCL\_REG\_YHIGHDRIFT 0x13**

Accelerometer y-axis offset drift high byte register (read/write)

**8.1.2.259 #define DIACCL\_REG\_YLOW 0x02**

Accelerometer y-axis low byte register (read only)

**8.1.2.260 #define DIACCL\_REG\_YLOWDRIFT 0x12**

Accelerometer y-axis offset drift low byte register (read/write)

**8.1.2.261 #define DIACCL\_REG\_Z8 0x08**

Accelerometer x-axis 8-bit register (read only)

**8.1.2.262 #define DIACCL\_REG\_ZHIGH 0x05**

Accelerometer z-axis high byte register (read only)

**8.1.2.263 #define DIACCL\_REG\_ZHIGHDRIFT 0x15**

Accelerometer z-axis offset drift high byte register (read/write)

**8.1.2.264 #define DIACCL\_REG\_ZLOW 0x04**

Accelerometer z-axis low byte register (read only)

**8.1.2.265 #define DIACCL\_REG\_ZLOWDRIFT 0x14**

Accelerometer z-axis offset drift low byte register (read/write)

**8.1.2.266 #define DIACCL\_STATUS\_DATAOVER 0x02**

Accelerometer data is overwritten

**8.1.2.267 #define DIACCL\_STATUS\_DATAREADY 0x01**

Accelerometer data is ready

**8.1.2.268 #define DIACCL\_STATUS\_PARITYERR 0x04**

Accelerometer parity error is detected in trim data

**8.1.2.269 #define DIGI\_PIN0 0x01**

Access digital pin 0 (B0)

**Examples:**

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

**8.1.2.270 #define DIGI\_PIN1 0x02**

Access digital pin 1 (B1)

**Examples:**

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

**8.1.2.271 #define DIGI\_PIN2 0x04**

Access digital pin 2 (B2)

**Examples:**

[ex\\_proto.nxc](#), and [ex\\_superpro.nxc](#).

**8.1.2.272 #define DIGI\_PIN3 0x08**

Access digital pin 3 (B3)

**8.1.2.273 #define DIGI\_PIN4 0x10**

Access digital pin 4 (B4)

**8.1.2.274 #define DIGI\_PIN5 0x20**

Access digital pin 5 (B5)

**8.1.2.275 #define DIGI\_PIN6 0x40**

Access digital pin 6 (B6)

**8.1.2.276 #define DIGI\_PIN7 0x80**

Access digital pin 7 (B7)

**8.1.2.277 #define DIGYRO\_CTRL1\_BANDWIDTH\_1 0x00**

Gyro LPF2 cut-off frequency bandwidth level 1 (12.5hz, 12.5hz, 20hz, 30hz)

**8.1.2.278 #define DIGYRO\_CTRL1\_BANDWIDTH\_2 0x10**

Gyro LPF2 cut-off frequency bandwidth level 2 (12.5hz, 25hz, 50hz, 70hz)

**8.1.2.279 #define DIGYRO\_CTRL1\_BANDWIDTH\_3 0x20**

Gyro LPF2 cut-off frequency bandwidth level 3 (20hz, 25hz, 50hz, 110hz)

**8.1.2.280 #define DIGYRO\_CTRL1\_BANDWIDTH\_4 0x30**

Gyro LPF2 cut-off frequency bandwidth level 4 (30hz, 35hz, 50hz, 110hz)

**Examples:**

[ex\\_digyro.nxc.](#)

**8.1.2.281 #define DIGYRO\_CTRL1\_DATARATE\_100 0x00**

Gyro output data rate 100 hz

**8.1.2.282 #define DIGYRO\_CTRL1\_DATARATE\_200 0x40**

Gyro output data rate 200 hz

**8.1.2.283 #define DIGYRO\_CTRL1\_DATARATE\_400 0x80**

Gyro output data rate 400 hz

**8.1.2.284 #define DIGYRO\_CTRL1\_DATARATE\_800 0xC0**

Gyro output data rate 800 hz

**Examples:**

[ex\\_digyro.nxc.](#)

- 8.1.2.285 #define DIGYRO\_CTRL1\_NORMAL 0x08**  
Gyro disable power down mode
- 8.1.2.286 #define DIGYRO\_CTRL1\_POWERDOWN 0x00**  
Gyro enable power down mode
- 8.1.2.287 #define DIGYRO\_CTRL1\_XENABLE 0x01**  
Gyro enable X axis
- 8.1.2.288 #define DIGYRO\_CTRL1\_YENABLE 0x02**  
Gyro enable Y axis
- 8.1.2.289 #define DIGYRO\_CTRL1\_ZENABLE 0x04**  
Gyro enable Z axis
- 8.1.2.290 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_001 0x09**  
Gyro high pass filter cutoff frequency 0.01 hz
- 8.1.2.291 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_002 0x08**  
Gyro high pass filter cutoff frequency 0.02 hz
- 8.1.2.292 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_005 0x07**  
Gyro high pass filter cutoff frequency 0.05 hz
- 8.1.2.293 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_01 0x06**  
Gyro high pass filter cutoff frequency 0.1 hz
- 8.1.2.294 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_02 0x05**  
Gyro high pass filter cutoff frequency 0.2 hz

**8.1.2.295 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_05 0x04**

Gyro high pass filter cutoff frequency 0.5 hz

**8.1.2.296 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_1 0x03**

Gyro high pass filter cutoff frequency 1 hz

**8.1.2.297 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_2 0x02**

Gyro high pass filter cutoff frequency 2 hz

**8.1.2.298 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_4 0x01**

Gyro high pass filter cutoff frequency 4 hz

**8.1.2.299 #define DIGYRO\_CTRL2\_CUTOFF\_FREQ\_8 0x00**

Gyro high pass filter cutoff frequency 8 hz

**8.1.2.300 #define DIGYRO\_CTRL2\_HPMODE\_AUTOINT 0x30**

Gyro high pass filter autoreset on interrupt event mode

**8.1.2.301 #define DIGYRO\_CTRL2\_HPMODE\_NORMAL 0x20**

Gyro high pass filter normal mode

**8.1.2.302 #define DIGYRO\_CTRL2\_HPMODE\_REFSIG 0x10**

Gyro high pass filter reference signal mode

**8.1.2.303 #define DIGYRO\_CTRL2\_HPMODE\_RESET 0x00**

Gyro high pass filter reset mode

**8.1.2.304 #define DIGYRO\_CTRL3\_INT1\_BOOT 0x40**

Gyro boot status available on INT1

**8.1.2.305 #define DIGYRO\_CTRL3\_INT1\_ENABLE 0x80**

Gyro interrupt enable on INT1 pin

**8.1.2.306 #define DIGYRO\_CTRL3\_INT1\_LOWACTIVE 0x20**

Gyro interrupt active low on INT1

**8.1.2.307 #define DIGYRO\_CTRL3\_INT2\_DATAREADY 0x08**

Gyro data ready on DRDY/INT2

**8.1.2.308 #define DIGYRO\_CTRL3\_INT2\_EMPTY 0x01**

Gyro FIFO empty interrupt on DRDY/INT2

**8.1.2.309 #define DIGYRO\_CTRL3\_INT2\_OVERRUN 0x02**

Gyro FIFO overrun interrupt on DRDY/INT2

**8.1.2.310 #define DIGYRO\_CTRL3\_INT2\_WATERMARK 0x04**

Gyro FIFO watermark interrupt on DRDY/INT2

**8.1.2.311 #define DIGYRO\_CTRL3\_OPENDRAIN 0x10**

Gyro use open drain rather than push-pull

**8.1.2.312 #define DIGYRO\_CTRL4\_BIGENDIAN 0x40**

Gyro use big endian - MSB/LSB rather than LSB/MSB in output registers

**8.1.2.313 #define DIGYRO\_CTRL4\_BLOCKDATA 0x80**

Gyro block data update - output registers are not updated until MSB and LSB reading

**8.1.2.314 #define DIGYRO\_CTRL4\_SCALE\_2000 0x30**

Gyro 2000 degrees per second scale

**Examples:**[ex\\_digyro.nxc](#).**8.1.2.315 #define DIGYRO\_CTRL4\_SCALE\_250 0x00**

Gyro 250 degrees per second scale

**8.1.2.316 #define DIGYRO\_CTRL4\_SCALE\_500 0x10**

Gyro 500 degrees per second scale

**8.1.2.317 #define DIGYRO\_CTRL5\_FIFOENABLE 0x40**

Gyro enable FIFO

**8.1.2.318 #define DIGYRO\_CTRL5\_HPENABLE 0x10**

Gyro enable high pass filter

**8.1.2.319 #define DIGYRO\_CTRL5\_INT1\_SEL\_1 0x00**

Gyro non-high-pass-filtered data are used for interrupt generation

**8.1.2.320 #define DIGYRO\_CTRL5\_INT1\_SEL\_2 0x04**

Gyro high-pass-filtered data are used for interrupt generation

**8.1.2.321 #define DIGYRO\_CTRL5\_INT1\_SEL\_3 0x08**

Gyro low-pass-filtered data are used for interrupt generation

**8.1.2.322 #define DIGYRO\_CTRL5\_OUT\_SEL\_1 0x00**

Gyro data in data registers and FIFO are not high-pass filtered

**8.1.2.323 #define DIGYRO\_CTRL5\_OUT\_SEL\_2 0x01**

Gyro data in data registers and FIFO are high-pass filtered

**8.1.2.324 #define DIGYRO\_CTRL5\_OUT\_SEL\_3 0x02**

Gyro data in data registers and FIFO are low-pass filtered by LPF2

**8.1.2.325 #define DIGYRO\_CTRL5\_REBOOTMEM 0x80**

Gyro reboot memory content

**8.1.2.326 #define DIGYRO\_FIFOCTRL\_BYPASS 0x00**

Gyro FIFO bypass mode

**8.1.2.327 #define DIGYRO\_FIFOCTRL\_BYPASS2STREAM 0x80**

Gyro FIFO bypass-to-stream mode

**8.1.2.328 #define DIGYRO\_FIFOCTRL\_FIFO 0x20**

Gyro FIFO mode

**8.1.2.329 #define DIGYRO\_FIFOCTRL\_STREAM 0x40**

Gyro FIFO stream mode

**8.1.2.330 #define DIGYRO\_FIFOCTRL\_STREAM2FIFO 0x60**

Gyro FIFO stream-to-FIFO mode

**8.1.2.331 #define DIGYRO\_FIFOCTRL\_WATERMARK\_MASK 0x1F**

Gyro FIFO threshold. Watermark level setting mask (values from 0x00 to 0x1F)

**8.1.2.332 #define DIGYRO\_REG\_CTRL1 0x20**

Gyro control register 1

**8.1.2.333 #define DIGYRO\_REG\_CTRL1AUTO 0xA0**

Gyro control register 1 - auto increment write

**8.1.2.334 #define DIGYRO\_REG\_CTRL2 0x21**

Gyro control register 2

**8.1.2.335 #define DIGYRO\_REG\_CTRL3 0x22**

Gyro control register 3

**8.1.2.336 #define DIGYRO\_REG\_CTRL4 0x23**

Gyro control register 4

**8.1.2.337 #define DIGYRO\_REG\_CTRL5 0x24**

Gyro control register 5

**8.1.2.338 #define DIGYRO\_REG\_FIFOCTRL 0x2E**

Gyro FIFO control register

**8.1.2.339 #define DIGYRO\_REG\_FIFOSRC 0x2F**

Gyro FIFO source register (read only)

**8.1.2.340 #define DIGYRO\_REG\_INT1\_CFG 0x30**

Gyro interrupt 1 config register

**8.1.2.341 #define DIGYRO\_REG\_INT1\_DUR 0x38**

Gyro interrupt 1 duration register

**8.1.2.342 #define DIGYRO\_REG\_INT1\_SRC 0x31**

Gyro interrupt 1 source register

**8.1.2.343 #define DIGYRO\_REG\_INT1\_XHI 0x32**

Gyro interrupt 1 x-axis high threshold register

**8.1.2.344 #define DIGYRO\_REG\_INT1\_XLO 0x33**

Gyro interrupt 1 x-axis low threshold register

**8.1.2.345 #define DIGYRO\_REG\_INT1\_YHI 0x34**

Gyro interrupt 1 y-axis high threshold register

**8.1.2.346 #define DIGYRO\_REG\_INT1\_YLO 0x35**

Gyro interrupt 1 y-axis low threshold register

**8.1.2.347 #define DIGYRO\_REG\_INT1\_ZHI 0x36**

Gyro interrupt 1 z-axis high threshold register

**8.1.2.348 #define DIGYRO\_REG\_INT1\_ZLO 0x37**

Gyro interrupt 1 z-axis low threshold register

**8.1.2.349 #define DIGYRO\_REG\_OUTTEMP 0x26**

Gyro temperature register (read only) - stores temperature data

**8.1.2.350 #define DIGYRO\_REG\_REFERENCE 0x25**

Gyro reference register - stores the reference value used for interrupt generation

**8.1.2.351 #define DIGYRO\_REG\_STATUS 0x27**

Gyro status register (read only)

**8.1.2.352 #define DIGYRO\_REG\_TEMPAUTO 0xA6**

Gyro temperature register - read burst mode (read only)

**8.1.2.353 #define DIGYRO\_REG\_WHOAMI 0x0F**

Gyro device identification register (read only)

**8.1.2.354 #define DIGYRO\_REG\_XHIGH 0x29**

Gyro x-axis high byte register (read only)

**8.1.2.355 #define DIGYRO\_REG\_XLOW 0x28**

Gyro x-axis low byte register (read only)

**8.1.2.356 #define DIGYRO\_REG\_XLOWBURST 0xA8**

Gyro x-axis low byte register - read burst mode (read only)

**8.1.2.357 #define DIGYRO\_REG\_YHIGH 0x2B**

Gyro y-axis high byte register (read only)

**8.1.2.358 #define DIGYRO\_REG\_YLOW 0x2A**

Gyro y-axis low byte register (read only)

**8.1.2.359 #define DIGYRO\_REG\_YLOWBURST 0xAA**

Gyro y-axis low byte register - read burst mode (read only)

**8.1.2.360 #define DIGYRO\_REG\_ZHIGH 0x2D**

Gyro z-axis high byte register (read only)

**8.1.2.361 #define DIGYRO\_REG\_ZLOW 0x2C**

Gyro z-axis low byte register (read only)

**8.1.2.362 #define DIGYRO\_REG\_ZLOWBURST 0xAC**

Gyro y-axis low byte register - read burst mode (read only)

**8.1.2.363 #define DIGYRO\_STATUS\_XDATA 0x01**

Gyro X-axis new data available

**8.1.2.364 #define DIGYRO\_STATUS\_XOVER 0x10**

Gyro X-axis data overrun - new data for the X-axis has overwritten the previous one

**8.1.2.365 #define DIGYRO\_STATUS\_XYZDATA 0x08**

Gyro X, Y, or Z-axis new data available - a new set of data is available

**8.1.2.366 #define DIGYRO\_STATUS\_XYZOVER 0x80**

Gyro X, Y, or Z-axis data overrun - new data has overwritten the previous one before it was read

**8.1.2.367 #define DIGYRO\_STATUS\_YDATA 0x02**

Gyro Y-axis new data available

**8.1.2.368 #define DIGYRO\_STATUS\_YOVER 0x20**

Gyro Y-axis data overrun - new data for the Y-axis has overwritten the previous one

**8.1.2.369 #define DIGYRO\_STATUS\_ZDATA 0x04**

Gyro Z-axis new data available

**8.1.2.370 #define DIGYRO\_STATUS\_ZOVER 0x40**

Gyro Z-axis data overrun - new data for the Z-axis has overwritten the previous one

**8.1.2.371 #define DISPLAY\_BUSY 0x80**

R - Refresh in progress

**8.1.2.372 #define DISPLAY\_CHAR 0x04**

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

**8.1.2.373 #define DISPLAY\_CONTRAST\_DEFAULT 0x5A**

Default display contrast value

**Examples:**[ex\\_contrast.nxc](#), and [ex\\_setdisplaycontrast.nxc](#).**8.1.2.374 #define DISPLAY\_CONTRAST\_MAX 0x7F**

Maximum display contrast value

**Examples:**[ex\\_contrast.nxc](#).**8.1.2.375 #define DISPLAY\_ERASE\_ALL 0x00**

W - erase entire screen (CMD,x,x,x,x,x)

**Examples:**[ex\\_sysdisplayexecutefunction.nxc](#).**8.1.2.376 #define DISPLAY\_ERASE\_LINE 0x05**

W - erase a single line (CMD,x,LINE,x,x,x)

**8.1.2.377 #define DISPLAY\_FILL\_REGION 0x06**

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**8.1.2.378 #define DISPLAY\_FRAME 0x07**

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**8.1.2.379 #define DISPLAY\_HEIGHT 64**

The height of the LCD screen in pixels

**Examples:**[ex\\_LineOut.nxc](#).

**8.1.2.380 #define DISPLAY\_HORIZONTAL\_LINE 0x02**

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

**Examples:**

[ex\\_dispfunc.nxc](#).

**8.1.2.381 #define DISPLAY\_MENUICONS\_X\_DIFF 31****8.1.2.382 #define DISPLAY\_MENUICONS\_X\_OFFS 7****8.1.2.383 #define DISPLAY\_MENUICONS\_Y 40****8.1.2.384 #define DISPLAY\_ON 0x01**

W - Display on

**8.1.2.385 #define DISPLAY\_PIXEL 0x01**

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

**8.1.2.386 #define DISPLAY\_POPUP 0x08**

W - Use popup display memory

**Examples:**

[ex\\_dispmisc.nxc](#).

**8.1.2.387 #define DISPLAY\_REFRESH 0x02**

W - Enable refresh

**8.1.2.388 #define DISPLAY\_REFRESH\_DISABLED 0x40**

R - Refresh disabled

**8.1.2.389 #define DISPLAY\_VERTICAL\_LINE 0x03**

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

**8.1.2.390 #define DISPLAY\_WIDTH 100**

The width of the LCD screen in pixels

**Examples:**

[ex\\_LineOut.nxc](#).

**8.1.2.391 #define DisplayExecuteFunction 80**

Execute one of the Display module's internal functions

**8.1.2.392 #define DisplayModuleID 0x000A0001**

The display module ID

**8.1.2.393 #define DisplayModuleName "Display.mod"**

The display module name

**8.1.2.394 #define DisplayOffsetContrast 1719**

Adjust the display contrast with this field

**8.1.2.395 #define DisplayOffsetDisplay 104**

Display content copied to physical display every 17 mS

**8.1.2.396 #define DisplayOffsetEraseMask 4**

Section erase mask (executed first)

**8.1.2.397 #define DisplayOffsetFlags 117**

Update flags enumerated above

**8.1.2.398 #define DisplayOffsetNormal(l, w) (((l)\*100)+(w)+119)**

Raw display memory for normal screen

**8.1.2.399 #define DisplayOffsetPBitmaps(p) (((p)\*4)+68)**

Pointer to free bitmap files

**8.1.2.400 #define DisplayOffsetPFont 12**

Pointer to font file

**8.1.2.401 #define DisplayOffsetPFunc 0**

Simple draw entry

**8.1.2.402 #define DisplayOffsetPMenuIcons(p) (((p)\*4)+88)**

Pointer to menu icon images (NULL == none)

**8.1.2.403 #define DisplayOffsetPMenuText 84**

Pointer to menu icon text (NULL == none)

**8.1.2.404 #define DisplayOffsetPopup(l, w) (((l)\*100)+(w)+919)**

Raw display memory for popup screen

**8.1.2.405 #define DisplayOffsetPScreens(p) (((p)\*4)+56)**

Pointer to screen bitmap file

**8.1.2.406 #define DisplayOffsetPStatusIcons 52**

Pointer to status icon collection file

**8.1.2.407 #define DisplayOffsetPStatusText 48**

Pointer to status text string

**8.1.2.408 #define DisplayOffsetPStepIcons 100**

Pointer to step icon collection file

**8.1.2.409 #define DisplayOffsetPTextLines(p) (((p)\*4)+16)**

Pointer to text strings

**8.1.2.410 #define DisplayOffsetStatusIcons(p) ((p)+108)**

Index in status icon collection file (index = 0 -&gt; none)

**8.1.2.411 #define DisplayOffsetStepIcons(p) ((p)+112)**

Index in step icon collection file (index = 0 -&gt; none)

**8.1.2.412 #define DisplayOffsetTextLinesCenterFlags 118**

Mask to center TextLines

**8.1.2.413 #define DisplayOffsetUpdateMask 8**

Section update mask (executed next)

**8.1.2.414 #define DIST\_CMD\_CUSTOM 0x35**

Set the DIST-Nx to a custom mode

**8.1.2.415 #define DIST\_CMD\_GP2D12 0x31**

Set the DIST-Nx to GP2D12 mode

**8.1.2.416 #define DIST\_CMD\_GP2D120 0x32**

Set the DIST-Nx to GP2D120 mode

**8.1.2.417 #define DIST\_CMD\_GP2YA02 0x34**

Set the DIST-Nx to GP2YA02 mode

**8.1.2.418 #define DIST\_CMD\_GP2YA21 0x33**

Set the DIST-Nx to GP2YA21 mode

**8.1.2.419 #define DIST\_REG\_DIST 0x42**

The DIST-Nx distance register

**8.1.2.420 #define DIST\_REG\_DIST1 0x58**

The DIST-Nx distance 1 register

**8.1.2.421 #define DIST\_REG\_DIST\_MAX 0x54**

The DIST-Nx maximum distance register

**8.1.2.422 #define DIST\_REG\_DIST\_MIN 0x52**

The DIST-Nx minimum distance register

**8.1.2.423 #define DIST\_REG\_MODULE\_TYPE 0x50**

The DIST-Nx module type register

**8.1.2.424 #define DIST\_REG\_NUM\_POINTS 0x51**

The DIST-Nx number of data points in Custom curve register

**8.1.2.425 #define DIST\_REG\_VOLT 0x44**

The DIST-Nx voltage register

**8.1.2.426 #define DIST\_REG\_VOLT1 0x56**

The DIST-Nx voltage 1 register

**8.1.2.427 #define DRAW\_OPT\_CLEAR (0x0004)**

Clear pixels while drawing (aka draw in white)

**8.1.2.428 #define DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_-  
SCREEN (0x0002)**

Clear the screen except for the status line before drawing

**8.1.2.429 #define DRAW\_OPT\_CLEAR\_PIXELS (0x0004)**

Clear pixels while drawing (aka draw in white)

**8.1.2.430 #define DRAW\_OPT\_CLEAR\_SCREEN\_MODES (0x0003)**

Bit mask for the clear screen modes

**8.1.2.431 #define DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN (0x0001)**

Clear the entire screen before drawing

**Examples:**

[ex\\_dispgoutex.nxc](#).

**8.1.2.432 #define DRAW\_OPT\_FILL\_SHAPE (0x0020)**

Fill the shape while drawing (rectangle, circle, ellipses, and polygon)

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysDrawEllipse.nxc](#),  
and [ex\\_sysdrawpolygon.nxc](#).

**8.1.2.433 #define DRAW\_OPT\_FONT\_DIR\_B2TL (0x0100)**

Font bottom to top left align

**8.1.2.434 #define DRAW\_OPT\_FONT\_DIR\_B2TR (0x0140)**

Font bottom to top right align

**8.1.2.435 #define DRAW\_OPT\_FONT\_DIR\_L2RB (0x0000)**

Font left to right bottom align

**Examples:**[ex\\_dispftout.nxc](#).**8.1.2.436 #define DRAW\_OPT\_FONT\_DIR\_L2RT (0x0040)**

Font left to right top align

**Examples:**[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).**8.1.2.437 #define DRAW\_OPT\_FONT\_DIR\_R2LB (0x0080)**

Font right to left bottom align

**8.1.2.438 #define DRAW\_OPT\_FONT\_DIR\_R2LT (0x00C0)**

Font right to left top align

**8.1.2.439 #define DRAW\_OPT\_FONT\_DIR\_T2BL (0x0180)**

Font top to bottom left align

**Examples:**[ex\\_dispftout.nxc](#).**8.1.2.440 #define DRAW\_OPT\_FONT\_DIR\_T2BR (0x01C0)**

Font top to bottom right align

**8.1.2.441 #define DRAW\_OPT\_FONT DIRECTIONS (0x01C0)**

Bit mask for the font direction bits

**8.1.2.442 #define DRAW\_OPT\_FONT\_WRAP (0x0200)**

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

**Examples:**

[ex\\_dispftout.nxc](#).

**8.1.2.443 #define DRAW\_OPT\_INVERT (0x0004)**

Invert text or graphics

**Examples:**

[ex\\_dispftout.nxc](#).

**8.1.2.444 #define DRAW\_OPT\_LOGICAL\_AND (0x0008)**

Draw pixels using a logical AND operation

**Examples:**

[ex\\_dispftout.nxc](#).

**8.1.2.445 #define DRAW\_OPT\_LOGICAL\_COPY (0x0000)**

Draw pixels using a logical copy operation

**8.1.2.446 #define DRAW\_OPT\_LOGICAL\_OPERATIONS (0x0018)**

Bit mask for the logical drawing operations

**8.1.2.447 #define DRAW\_OPT\_LOGICAL\_OR (0x0010)**

Draw pixels using a logical OR operation

**Examples:**

[ex\\_dispftout.nxc](#).

**8.1.2.448 #define DRAW\_OPT\_LOGICAL\_XOR (0x0018)**

Draw pixels using a logical XOR operation

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysDrawEllipse.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**8.1.2.449 #define DRAW\_OPT\_NORMAL (0x0000)**

Normal drawing

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispfunc.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**8.1.2.450 #define DRAW\_OPT\_POLYGON\_POLYLINE (0x0400)**

When drawing polygons, do not close (i.e., draw a polyline instead)

**8.1.2.451 #define DrawCircle 16**

Draw a circle on the LCD screen

**8.1.2.452 #define DrawEllipse 94**

Draw an ellipse on the LCD screen

**8.1.2.453 #define DrawFont 95**

Draw text using a custom RIC-based font to the LCD screen

**8.1.2.454 #define DrawGraphic 18**

Draw a graphic image on the LCD screen

**8.1.2.455 #define DrawGraphicArray 92**

Draw a graphic image from a byte array to the LCD screen

**Examples:**

[ex\\_dispgout.nxc](#).

**8.1.2.456 #define DrawLine 15**

Draw a line on the LCD screen

**8.1.2.457 #define DrawPoint 14**

Draw a single pixel on the LCD screen

**8.1.2.458 #define DrawPolygon 93**

Draw a polygon on the LCD screen

**8.1.2.459 #define DrawRect 17**

Draw a rectangle on the LCD screen

**8.1.2.460 #define DrawText 13**

Draw text to one of 8 LCD lines

**Examples:**

[ex\\_syscall.nxc](#).

**8.1.2.461 #define EMETER\_REG\_AIN 0x0c**

The register address for amps in

**8.1.2.462 #define EMETER\_REG\_AOUT 0x10**

The register address for amps out

**8.1.2.463 #define EMETER\_REG\_JOULES 0x12**

The register address for joules

**8.1.2.464 #define EMETER\_REG\_VIN 0x0a**

The register address for voltage in

**8.1.2.465 #define EMETER\_REG\_VOUT 0x0e**

The register address for voltage out

**8.1.2.466 #define EMETER\_REG\_WIN 0x14**

The register address for watts in

**8.1.2.467 #define EMETER\_REG\_WOUT 0x16**

The register address for watts out

**8.1.2.468 #define EOF -1**

A constant representing end of file

**8.1.2.469 #define ERR\_ARG -1**

0xFF Bad arguments

**8.1.2.470 #define ERR\_BAD\_POOL\_SIZE -10**

0xF6 VarsCmd.PoolSize > POOL\_MAX\_SIZE

**8.1.2.471 #define ERR\_BAD\_PTR -6**

0xFA Someone passed us a bad pointer!

**8.1.2.472 #define ERR\_CLUMP\_COUNT -7**

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT\_A\_CLUMP)

**8.1.2.473 #define ERR\_COMM\_BUFFER\_FULL -34**

0xDE No room in comm buffer

**8.1.2.474 #define ERR\_COMM\_BUS\_ERR -35**

0xDD Something went wrong on the communications bus

**8.1.2.475 #define ERR\_COMM\_CHAN\_INVALID -33**

0xDF Specified channel/connection is not valid

**8.1.2.476 #define ERR\_COMM\_CHAN\_NOT\_READY -32**

0xE0 Specified channel/connection not configured or busy

**8.1.2.477 #define ERR\_DEFAULT\_OFFSETS -14**

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

**8.1.2.478 #define ERR\_FILE -3**

0xFD Malformed file contents

**8.1.2.479 #define ERR\_INSANE\_OFFSET -9**

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount \* 2)

**8.1.2.480 #define ERR\_INSTR -2**

0xFE Illegal bytecode instruction

**8.1.2.481 #define ERR\_INVALID\_FIELD -17**

0xEF Attempted to access invalid field of a structure

**8.1.2.482 #define ERR\_INVALID\_PORT -16**

0xF0 Bad input or output port specified

**8.1.2.483 #define ERR\_INVALID\_QUEUE -18**

0xEE Illegal queue ID specified

**8.1.2.484 #define ERR\_INVALID\_SIZE -19**

0xED Illegal size specified

**8.1.2.485 #define ERR\_LOADER\_ERR -11**

0xF5 LOADER\_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

**8.1.2.486 #define ERR\_MEM -5**

0xFB Insufficient memory available

**8.1.2.487 #define ERR\_MEMMGR\_FAIL -15**

0xF1 (UBYTE \*)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV\_ARRAY[0].Offset

**8.1.2.488 #define ERR\_NO\_ACTIVE\_CLUMP -13**

0xF3 VarsCmd.RunQ.Head == NOT\_A\_CLUMP

**8.1.2.489 #define ERR\_NO\_CODE -8**

0xF8 VarsCmd.CodespaceCount == 0

**8.1.2.490 #define ERR\_NO\_PROG -20**

0xEC No active program

**8.1.2.491 #define ERR\_NON\_FATAL -16**

Fatal errors are greater than this value

**8.1.2.492 #define ERR\_RC\_BAD\_PACKET -65**

0xBF Clearly insane packet

**8.1.2.493 #define ERR\_RC\_FAILED -67**

0xBD Request failed (i.e. specified file not found)

**8.1.2.494 #define ERR\_RC\_ILLEGAL\_VAL -64**

0xC0 Data contains out-of-range values

**8.1.2.495 #define ERR\_RC\_UNKNOWN\_CMD -66**

0xBE Unknown command opcode

**8.1.2.496 #define ERR\_SPOTCHECK\_FAIL -12**

0xF4 ((UBYTE\*)(VarsCmd.pCodespace) < pData) (c\_cmd.c 1893)

**8.1.2.497 #define ERR\_VER -4**

0xFC Version mismatch between firmware and compiler

**8.1.2.498 #define FALSE 0**

A false value

**8.1.2.499 #define FileClose 5**

Close the specified file

**8.1.2.500 #define FileDelete 8**

Delete a file

**8.1.2.501 #define FileFindFirst 83**

Start a search for a file using a filename pattern

**8.1.2.502 #define FileFindNext 84**

Continue searching for a file

**8.1.2.503 #define FileOpenAppend 2**

Open a file for appending to the end of the file

**8.1.2.504 #define FileOpenRead 0**

Open a file for reading

**8.1.2.505 #define FileOpenReadLinear 87**

Open a linear file for reading

**8.1.2.506 #define FileOpenWrite 1**

Open a file for writing (creates a new file)

**8.1.2.507 #define FileOpenWriteLinear 85**

Open a linear file for writing

**8.1.2.508 #define FileOpenWriteNonLinear 86**

Open a non-linear file for writing

**8.1.2.509 #define FileRead 3**

Read from the specified file

**8.1.2.510 #define FileRename 7**

Rename a file

**8.1.2.511 #define FileResize 91**

Resize a file (not yet implemented)

**8.1.2.512 #define FileResolveHandle 6**

Get a file handle for the specified filename if it is already open

**8.1.2.513 #define FileSeek 90**

Seek to a specific position in an open file

**8.1.2.514 #define FileTell 98**

Return the current file position in an open file

**8.1.2.515 #define FileWrite 4**

Write to the specified file

**8.1.2.516 #define FRAME\_SELECT 0**

Center icon select frame

**8.1.2.517 #define FREQUENCY\_MAX 14080**

Maximum frequency [Hz]

**8.1.2.518 #define FREQUENCY\_MIN 220**

Minimum frequency [Hz]

**8.1.2.519 #define GetStartTick 25**

Get the current system tick count

**8.1.2.520 #define GL\_CAMERA\_DEPTH 3**

Set the camera depth.

**8.1.2.521 #define GL\_CIRCLE 4**

Use circle mode.

**Examples:**

[glCircleDemo.nxc](#).

**8.1.2.522 #define GL\_CIRCLE\_SIZE 1**

Set the circle size.

**8.1.2.523 #define GL\_CULL\_BACK 2**

Cull lines in back.

**8.1.2.524 #define GL\_CULL\_FRONT 3**

Cull lines in front.

**8.1.2.525 #define GL\_CULL\_MODE 2**

Set the cull mode.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.1.2.526 #define GL\_CULL\_NONE 4**

Do not cull any lines.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.1.2.527 #define GL\_LINE 2**

Use line mode.

**8.1.2.528 #define GL\_POINT 3**

Use point mode.

**8.1.2.529 #define GL\_POLYGON 1**

Use polygon mode.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.1.2.530 #define GL\_ROTATE\_X 4**

Rotate around the X axis.

**Examples:**

[glRotateDemo.nxc](#).

**8.1.2.531 #define GL\_ROTATE\_Y 5**

Rotate around the Y axis.

**Examples:**

[glRotateDemo.nxc](#).

**8.1.2.532 #define GL\_ROTATE\_Z 6**

Rotate around the Z axis.

**8.1.2.533 #define GL\_SCALE\_X 7**

Scale along the X axis.

**Examples:**

[glScaleDemo.nxc](#).

**8.1.2.534 #define GL\_SCALE\_Y 8**

Scale along the Y axis.

**8.1.2.535 #define GL\_SCALE\_Z 9**

Scale along the Z axis.

**8.1.2.536 #define GL\_TRANSLATE\_X 1**

Translate along the X axis.

**Examples:**

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.1.2.537 #define GL\_TRANSLATE\_Y 2**

Translate along the Y axis.

**Examples:**

[glTranslateDemo.nxc](#).

**8.1.2.538 #define GL\_TRANSLATE\_Z 3**

Translate along the Z axis.

**Examples:**

[glTranslateDemo.nxc](#).

**8.1.2.539 #define GL\_ZOOM\_FACTOR 4**

Set the zoom factor.

**8.1.2.540 #define HS\_ADDRESS\_1 1**

HsAddress device address 1

**8.1.2.541 #define HS\_ADDRESS\_2 2**

HsAddress device address 2

**8.1.2.542 #define HS\_ADDRESS\_3 3**

HsAddress device address 3

**8.1.2.543 #define HS\_ADDRESS\_4 4**

HsAddress device address 4

**8.1.2.544 #define HS\_ADDRESS\_5 5**

HsAddress device address 5

---

|           |                                  |                            |
|-----------|----------------------------------|----------------------------|
| 8.1.2.545 | <b>#define HS_ADDRESS_6 6</b>    | HsAddress device address 6 |
| 8.1.2.546 | <b>#define HS_ADDRESS_7 7</b>    | HsAddress device address 7 |
| 8.1.2.547 | <b>#define HS_ADDRESS_8 8</b>    | HsAddress device address 8 |
| 8.1.2.548 | <b>#define HS_ADDRESS_ALL 0</b>  | HsAddress all devices      |
| 8.1.2.549 | <b>#define HS_BAUD_115200 12</b> | HsSpeed 115200 Baud        |
| 8.1.2.550 | <b>#define HS_BAUD_1200 0</b>    | HsSpeed 1200 Baud          |
| 8.1.2.551 | <b>#define HS_BAUD_14400 6</b>   | HsSpeed 14400 Baud         |
| 8.1.2.552 | <b>#define HS_BAUD_19200 7</b>   | HsSpeed 19200 Baud         |
| 8.1.2.553 | <b>#define HS_BAUD_230400 13</b> | HsSpeed 230400 Baud        |
| 8.1.2.554 | <b>#define HS_BAUD_2400 1</b>    | HsSpeed 2400 Baud          |

**8.1.2.555 #define HS\_BAUD\_28800 8**

HsSpeed 28800 Baud

**8.1.2.556 #define HS\_BAUD\_3600 2**

HsSpeed 3600 Baud

**8.1.2.557 #define HS\_BAUD\_38400 9**

HsSpeed 38400 Baud

**8.1.2.558 #define HS\_BAUD\_460800 14**

HsSpeed 460800 Baud

**8.1.2.559 #define HS\_BAUD\_4800 3**

HsSpeed 4800 Baud

**8.1.2.560 #define HS\_BAUD\_57600 10**

HsSpeed 57600 Baud

**8.1.2.561 #define HS\_BAUD\_7200 4**

HsSpeed 7200 Baud

**8.1.2.562 #define HS\_BAUD\_76800 11**

HsSpeed 76800 Baud

**8.1.2.563 #define HS\_BAUD\_921600 15**

HsSpeed 921600 Baud

**8.1.2.564 #define HS\_BAUD\_9600 5**

HsSpeed 9600 Baud

**8.1.2.565 #define HS\_BAUD\_DEFAULT 15**

HsSpeed default Baud (921600)

**Examples:**[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).**8.1.2.566 #define HS\_BYTES\_REMAINING 16**

HsState bytes remaining to be sent

**8.1.2.567 #define HS\_CMD\_READY 0x04**

A constant representing high speed direct command

**8.1.2.568 #define HS\_CTRL\_EXIT 2**

Ddisable the high speed port

**8.1.2.569 #define HS\_CTRL\_INIT 0**

Enable the high speed port

**Examples:**[ex\\_SysCommHSControl.nxc](#).**8.1.2.570 #define HS\_CTRL\_UART 1**

Setup the high speed port UART configuration

**8.1.2.571 #define HS\_DEFAULT 6**

HsState default

**8.1.2.572 #define HS\_DISABLE 4**

HsState disable

**8.1.2.573 #define HS\_ENABLE 5**

HsState enable

**8.1.2.574 #define HS\_INIT\_RECEIVER 2**

HsState initialize receiver

**8.1.2.575 #define HS\_INITIALISE 1**

HsState initialize

**8.1.2.576 #define HS\_MODE\_10\_STOP 0x0000**

HsMode 1 stop bit

**8.1.2.577 #define HS\_MODE\_15\_STOP 0x1000**

HsMode 1.5 stop bits

**8.1.2.578 #define HS\_MODE\_20\_STOP 0x2000**

HsMode 2 stop bits

**8.1.2.579 #define HS\_MODE\_5\_DATA 0x0000**

HsMode 5 data bits

**8.1.2.580 #define HS\_MODE\_6\_DATA 0x0040**

HsMode 6 data bits

**8.1.2.581 #define HS\_MODE\_7\_DATA 0x0080**

HsMode 7 data bits

**8.1.2.582 #define HS\_MODE\_7E1 (HS\_MODE\_7\_DATA|HS\_MODE\_E -  
PARITY|HS\_MODE\_10\_STOP)**

HsMode 7 data bits, even parity, 1 stop bit

**8.1.2.583** #define HS\_MODE\_8\_DATA 0x00C0

HsMode 8 data bits

**8.1.2.584** #define HS\_MODE\_8N1 (HS\_MODE\_8\_DATA|HS\_MODE\_N\_-  
PARITY|HS\_MODE\_10\_STOP)

HsMode 8 data bits, no parity, 1 stop bit

**Examples:**

[ex\\_sethsmode.nxc](#).

**8.1.2.585** #define HS\_MODE\_DEFAULT HS\_MODE\_8N1

HsMode default mode (8 data bits, no parity, 1 stop bit)

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.1.2.586** #define HS\_MODE\_E\_PARITY 0x0000

HsMode Even parity

**8.1.2.587** #define HS\_MODE\_M\_PARITY 0x0600

HsMode Mark parity

**8.1.2.588** #define HS\_MODE\_MASK 0xFFFF0

HsMode mode mask

**8.1.2.589** #define HS\_MODE\_N\_PARITY 0x0800

HsMode No parity

**8.1.2.590** #define HS\_MODE\_O\_PARITY 0x0200

HsMode Odd parity

**8.1.2.591 #define HS\_MODE\_S\_PARITY 0x0400**

HsMode Space parity

**8.1.2.592 #define HS\_MODE\_UART\_RS232 0x1**

HsMode UART in normal or RS232 mode

**8.1.2.593 #define HS\_MODE\_UART\_RS485 0x0**

HsMode UART in default or RS485 mode

**8.1.2.594 #define HS\_SEND\_DATA 3**

HsState send data

**8.1.2.595 #define HS\_UART\_MASK 0x000F**

HsMode UART mask

**8.1.2.596 #define HS\_UPDATE 1**

HsFlags high speed update required

**8.1.2.597 #define HT\_ADDR\_ACCEL 0x02**

HiTechnic Accel I2C address

**8.1.2.598 #define HT\_ADDR\_ANGLE 0x02**

HiTechnic Angle I2C address

**8.1.2.599 #define HT\_ADDR\_BAROMETRIC 0x02**

HiTechnic Barometric I2C address

**8.1.2.600 #define HT\_ADDR\_COLOR 0x02**

HiTechnic Color I2C address

**8.1.2.601 #define HT\_ADDR\_COLOR2 0x02**

HiTechnic Color2 I2C address

**8.1.2.602 #define HT\_ADDR\_COMPASS 0x02**

HiTechnic Compass I2C address

**8.1.2.603 #define HT\_ADDR\_IRLINK 0x02**

HiTechnic IRLink I2C address

**8.1.2.604 #define HT\_ADDR\_IRRECEIVER 0x02**

HiTechnic IRReceiver I2C address

**8.1.2.605 #define HT\_ADDR\_IRSEEKER 0x02**

HiTechnic IRSeeker I2C address

**8.1.2.606 #define HT\_ADDR\_IRSEEKER2 0x10**

HiTechnic IRSeeker2 I2C address

**8.1.2.607 #define HT\_ADDR\_PROTOBOARD 0x02**

HiTechnic Prototype board I2C address

**8.1.2.608 #define HT\_ADDR\_SUPERPRO 0x10**

HiTechnic SuperPro board I2C address

**8.1.2.609 #define HT\_CH1\_A 0**

Use IRReceiver channel 1 output A

**Examples:**

[ex\\_ReadSensorHTIRReceiverEx.nxc.](#)

**8.1.2.610 #define HT\_CH1\_B 1**

Use IRReceiver channel 1 output B

**8.1.2.611 #define HT\_CH2\_A 2**

Use IRReceiver channel 2 output A

**8.1.2.612 #define HT\_CH2\_B 3**

Use IRReceiver channel 2 output B

**8.1.2.613 #define HT\_CH3\_A 4**

Use IRReceiver channel 3 output A

**8.1.2.614 #define HT\_CH3\_B 5**

Use IRReceiver channel 3 output B

**8.1.2.615 #define HT\_CH4\_A 6**

Use IRReceiver channel 4 output A

**8.1.2.616 #define HT\_CH4\_B 7**

Use IRReceiver channel 4 output B

**8.1.2.617 #define HT\_CMD\_COLOR2\_50HZ 0x35**

Set the Color2 sensor to 50Hz mode

**8.1.2.618 #define HT\_CMD\_COLOR2\_60HZ 0x36**

Set the Color2 sensor to 60Hz mode

**8.1.2.619 #define HT\_CMD\_COLOR2\_ACTIVE 0x00**

Set the Color2 sensor to active mode

**Examples:**

[ex\\_I2CSendCommand.nxc](#), and [ex\\_sethtcolor2mode.nxc](#).

**8.1.2.620 #define HT\_CMD\_COLOR2\_BLCAL 0x42**

Set the Color2 sensor to black level calibration mode

**8.1.2.621 #define HT\_CMD\_COLOR2\_FAR 0x46**

Set the Color2 sensor to far mode

**8.1.2.622 #define HT\_CMD\_COLOR2\_LED\_HI 0x48**

Set the Color2 sensor to LED high mode

**8.1.2.623 #define HT\_CMD\_COLOR2\_LED\_LOW 0x4C**

Set the Color2 sensor to LED low mode

**8.1.2.624 #define HT\_CMD\_COLOR2\_NEAR 0x4E**

Set the Color2 sensor to near mode

**8.1.2.625 #define HT\_CMD\_COLOR2\_PASSIVE 0x01**

Set the Color2 sensor to passive mode

**8.1.2.626 #define HT\_CMD\_COLOR2\_RAW 0x03**

Set the Color2 sensor to raw mode

**8.1.2.627 #define HT\_CMD\_COLOR2\_WBCAL 0x43**

Set the Color2 sensor to white level calibration mode

**8.1.2.628 #define HTANGLE\_MODE\_CALIBRATE 0x43**

Resets 0 degree position to current shaft angle

**8.1.2.629 #define HTANGLE\_MODE\_NORMAL 0x00**

Normal angle measurement mode

**8.1.2.630 #define HTANGLE\_MODE\_RESET 0x52**

Resets the accumulated angle

**Examples:**

[ex\\_ResetSensorHTAngle.nxc.](#)

**8.1.2.631 #define HTANGLE\_REG\_ACDIR 0x49**

Angle 16 bit revolutions per minute, low byte register

**8.1.2.632 #define HTANGLE\_REG\_DC01 0x43**

Angle current angle (1 degree adder) register

**8.1.2.633 #define HTANGLE\_REG\_DC02 0x44**

Angle 32 bit accumulated angle, high byte register

**8.1.2.634 #define HTANGLE\_REG\_DC03 0x45**

Angle 32 bit accumulated angle, mid byte register

**8.1.2.635 #define HTANGLE\_REG\_DC04 0x46**

Angle 32 bit accumulated angle, mid byte register

**8.1.2.636 #define HTANGLE\_REG\_DC05 0x47**

Angle 32 bit accumulated angle, low byte register

**8.1.2.637 #define HTANGLE\_REG\_DCAVG 0x48**

Angle 16 bit revolutions per minute, high byte register

**8.1.2.638 #define HTANGLE\_REG\_DCDIR 0x42**

Angle current angle (2 degree increments) register

**8.1.2.639 #define HTANGLE\_REG\_MODE 0x41**

Angle mode register

**8.1.2.640 #define HTBAR\_REG\_CALIBRATION 0x46**

Barometric sensor calibration register (2 bytes msb/lb)

**8.1.2.641 #define HTBAR\_REG\_COMMAND 0x40**

Barometric sensor command register

**8.1.2.642 #define HTBAR\_REG\_PRESSURE 0x44**

Barometric sensor pressure register (2 bytes msb/lb)

**8.1.2.643 #define HTBAR\_REG\_TEMPERATURE 0x42**

Barometric sensor temperature register (2 bytes msb/lb)

**8.1.2.644 #define HTIR2\_MODE\_1200 0**

Set IRSeeker2 to 1200 mode

**Examples:**

[ex\\_sethirseeker2mode.nxc](#), and [ex\\_setsensorboolean.nxc](#).

**8.1.2.645 #define HTIR2\_MODE\_600 1**

Set IRSeeker2 to 600 mode

**8.1.2.646 #define HTIR2\_REG\_AC01 0x4A**

IRSeeker2 AC 01 register

---

|           |                                     |                                 |
|-----------|-------------------------------------|---------------------------------|
| 8.1.2.647 | <b>#define HTIR2_REG_AC02 0x4B</b>  | IRSeeker2 AC 02 register        |
| 8.1.2.648 | <b>#define HTIR2_REG_AC03 0x4C</b>  | IRSeeker2 AC 03 register        |
| 8.1.2.649 | <b>#define HTIR2_REG_AC04 0x4D</b>  | IRSeeker2 AC 04 register        |
| 8.1.2.650 | <b>#define HTIR2_REG_AC05 0x4E</b>  | IRSeeker2 AC 05 register        |
| 8.1.2.651 | <b>#define HTIR2_REG_ACDIR 0x49</b> | IRSeeker2 AC direction register |
| 8.1.2.652 | <b>#define HTIR2_REG_DC01 0x43</b>  | IRSeeker2 DC 01 register        |
| 8.1.2.653 | <b>#define HTIR2_REG_DC02 0x44</b>  | IRSeeker2 DC 02 register        |
| 8.1.2.654 | <b>#define HTIR2_REG_DC03 0x45</b>  | IRSeeker2 DC 03 register        |
| 8.1.2.655 | <b>#define HTIR2_REG_DC04 0x46</b>  | IRSeeker2 DC 04 register        |
| 8.1.2.656 | <b>#define HTIR2_REG_DC05 0x47</b>  | IRSeeker2 DC 05 register        |

**8.1.2.657 #define HTIR2\_REG\_DCAVG 0x48**

IRSeeker2 DC average register

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**8.1.2.658 #define HTIR2\_REG\_DCDIR 0x42**

IRSeeker2 DC direction register

**8.1.2.659 #define HTIR2\_REG\_MODE 0x41**

IRSeeker2 mode register

**8.1.2.660 #define HTPROTO\_A0 0x42**

Read Prototype board analog input 0

**Examples:**

[ex\\_proto.nxc](#).

**8.1.2.661 #define HTPROTO\_A1 0x44**

Read Prototype board analog input 1

**8.1.2.662 #define HTPROTO\_A2 0x46**

Read Prototype board analog input 2

**8.1.2.663 #define HTPROTO\_A3 0x48**

Read Prototype board analog input 3

**8.1.2.664 #define HTPROTO\_A4 0x4A**

Read Prototype board analog input 4

**8.1.2.665 #define HTPROTO\_REG\_A0 0x42**

Prototype board analog 0 register (2 bytes msb/lb)

**8.1.2.666 #define HTPROTO\_REG\_A1 0x44**

Prototype board analog 1 register (2 bytes msb/lb)

**8.1.2.667 #define HTPROTO\_REG\_A2 0x46**

Prototype board analog 2 register (2 bytes msb/lb)

**8.1.2.668 #define HTPROTO\_REG\_A3 0x48**

Prototype board analog 3 register (2 bytes msb/lb)

**8.1.2.669 #define HTPROTO\_REG\_A4 0x4A**

Prototype board analog 4 register (2 bytes msb/lb)

**8.1.2.670 #define HTPROTO\_REG\_DCTRL 0x4E**

Prototype board digital pin control register (6 bits)

**8.1.2.671 #define HTPROTO\_REG\_DIN 0x4C**

Prototype board digital pin input register (6 bits)

**8.1.2.672 #define HTPROTO\_REG\_DOUT 0x4D**

Prototype board digital pin output register (6 bits)

**8.1.2.673 #define HTPROTO\_REG\_SRATE 0x4F**

Prototype board sample rate register

**8.1.2.674 #define HTSPRO\_A0 0x42**

Read SuperPro analog input 0

**Examples:**

[ex\\_superpro.nxc](#).

**8.1.2.675 #define HTSPRO\_A1 0x44**

Read SuperPro analog input 1

**8.1.2.676 #define HTSPRO\_A2 0x46**

Read SuperPro analog input 2

**8.1.2.677 #define HTSPRO\_A3 0x48**

Read SuperPro analog input 3

**8.1.2.678 #define HTSPRO\_DAC0 0x52**

Set SuperPro analog output 0 configuration

**Examples:**

[ex\\_superpro.nxc](#).

**8.1.2.679 #define HTSPRO\_DAC1 0x57**

Set SuperPro analog output 1 configuration

**Examples:**

[ex\\_superpro.nxc](#).

**8.1.2.680 #define HTSPRO\_REG\_A0 0x42**

SuperPro analog 0 register (10 bits)

**8.1.2.681 #define HTSPRO\_REG\_A1 0x44**

SuperPro analog 1 register (10 bits)

**8.1.2.682 #define HTSPRO\_REG\_A2 0x46**

SuperPro analog 2 register (10 bits)

**8.1.2.683 #define HTSPRO\_REG\_A3 0x48**

SuperPro analog 3 register (10 bits)

**8.1.2.684 #define HTSPRO\_REG\_CTRL 0x40**

SuperPro program control register

**8.1.2.685 #define HTSPRO\_REG\_DAC0\_FREQ 0x53**

SuperPro analog output 0 frequency register (2 bytes msb/lb)

**8.1.2.686 #define HTSPRO\_REG\_DAC0\_MODE 0x52**

SuperPro analog output 0 mode register

**8.1.2.687 #define HTSPRO\_REG\_DAC0\_VOLTAGE 0x55**

SuperPro analog output 0 voltage register (10 bits)

**8.1.2.688 #define HTSPRO\_REG\_DAC1\_FREQ 0x58**

SuperPro analog output 1 frequency register (2 bytes msb/lb)

**8.1.2.689 #define HTSPRO\_REG\_DAC1\_MODE 0x57**

SuperPro analog output 1 mode register

**8.1.2.690 #define HTSPRO\_REG\_DAC1\_VOLTAGE 0x5A**

SuperPro analog output 1 voltage register (10 bits)

**8.1.2.691 #define HTSPRO\_REG\_DCTRL 0x4E**

SuperPro digital pin control register (8 bits)

**8.1.2.692 #define HTSPRO\_REG\_DIN 0x4C**

SuperPro digital pin input register (8 bits)

**8.1.2.693 #define HTSPRO\_REG\_DLADDRESS 0x60**

SuperPro download address register (2 bytes msb/lb)

**8.1.2.694 #define HTSPRO\_REG\_DLCHKSUM 0x6A**

SuperPro download checksum register

**8.1.2.695 #define HTSPRO\_REG\_DLCONTROL 0x6B**

SuperPro download control register

**8.1.2.696 #define HTSPRO\_REG\_DLDATA 0x62**

SuperPro download data register (8 bytes)

**8.1.2.697 #define HTSPRO\_REG\_DOUT 0x4D**

SuperPro digital pin output register (8 bits)

**8.1.2.698 #define HTSPRO\_REG\_LED 0x51**

SuperPro LED control register

**8.1.2.699 #define HTSPRO\_REG\_MEMORY\_20 0x80**

SuperPro memory address 0x20 register (4 bytes msb/lb)

**8.1.2.700 #define HTSPRO\_REG\_MEMORY\_21 0x84**

SuperPro memory address 0x21 register (4 bytes msb/lb)

**8.1.2.701 #define HTSPRO\_REG\_MEMORY\_22 0x88**

SuperPro memory address 0x22 register (4 bytes msb/lb)

**8.1.2.702 #define HTSPRO\_REG\_MEMORY\_23 0x8C**

SuperPro memory address 0x23 register (4 bytes msb/lb)

**8.1.2.703 #define HTSPRO\_REG\_MEMORY\_24 0x90**

SuperPro memory address 0x24 register (4 bytes msb/lb)

**8.1.2.704 #define HTSPRO\_REG\_MEMORY\_25 0x94**

SuperPro memory address 0x25 register (4 bytes msb/lb)

**8.1.2.705 #define HTSPRO\_REG\_MEMORY\_26 0x98**

SuperPro memory address 0x26 register (4 bytes msb/lb)

**8.1.2.706 #define HTSPRO\_REG\_MEMORY\_27 0x9C**

SuperPro memory address 0x27 register (4 bytes msb/lb)

**8.1.2.707 #define HTSPRO\_REG\_MEMORY\_28 0xA0**

SuperPro memory address 0x28 register (4 bytes msb/lb)

**8.1.2.708 #define HTSPRO\_REG\_MEMORY\_29 0xA4**

SuperPro memory address 0x29 register (4 bytes msb/lb)

**8.1.2.709 #define HTSPRO\_REG\_MEMORY\_2A 0xA8**

SuperPro memory address 0x2A register (4 bytes msb/lb)

**8.1.2.710 #define HTSPRO\_REG\_MEMORY\_2B 0xAC**

SuperPro memory address 0x2B register (4 bytes msb/lb)

**8.1.2.711 #define HTSPRO\_REG\_MEMORY\_2C 0xB0**

SuperPro memory address 0x2C register (4 bytes msb/lb)

**8.1.2.712 #define HTSPRO\_REG\_MEMORY\_2D 0xB4**

SuperPro memory address 0x2D register (4 bytes msb/lb)

**8.1.2.713 #define HTSPRO\_REG\_MEMORY\_2E 0xB8**

SuperPro memory address 0x2E register (4 bytes msb/lb)

**8.1.2.714 #define HTSPRO\_REG\_MEMORY\_2F 0xBC**

SuperPro memory address 0x2F register (4 bytes msb/lb)

**8.1.2.715 #define HTSPRO\_REG\_MEMORY\_30 0xC0**

SuperPro memory address 0x30 register (4 bytes msb/lb)

**8.1.2.716 #define HTSPRO\_REG\_MEMORY\_31 0xC4**

SuperPro memory address 0x31 register (4 bytes msb/lb)

**8.1.2.717 #define HTSPRO\_REG\_MEMORY\_32 0xC8**

SuperPro memory address 0x32 register (4 bytes msb/lb)

**8.1.2.718 #define HTSPRO\_REG\_MEMORY\_33 0xCC**

SuperPro memory address 0x33 register (4 bytes msb/lb)

**8.1.2.719 #define HTSPRO\_REG\_MEMORY\_34 0xD0**

SuperPro memory address 0x34 register (4 bytes msb/lb)

**8.1.2.720 #define HTSPRO\_REG\_MEMORY\_35 0xD4**

SuperPro memory address 0x35 register (4 bytes msb/lb)

**8.1.2.721 #define HTSPRO\_REG\_MEMORY\_36 0xD8**

SuperPro memory address 0x36 register (4 bytes msb/lb)

**8.1.2.722 #define HTSPRO\_REG\_MEMORY\_37 0xDC**

SuperPro memory address 0x37 register (4 bytes msb/lb)

**8.1.2.723 #define HTSPRO\_REG\_MEMORY\_38 0xE0**

SuperPro memory address 0x38 register (4 bytes msb/lb)

**8.1.2.724 #define HTSPRO\_REG\_MEMORY\_39 0xE4**

SuperPro memory address 0x39 register (4 bytes msb/lb)

**8.1.2.725 #define HTSPRO\_REG\_MEMORY\_3A 0xE8**

SuperPro memory address 0x3A register (4 bytes msb/lb)

**8.1.2.726 #define HTSPRO\_REG\_MEMORY\_3B 0xEC**

SuperPro memory address 0x3B register (4 bytes msb/lb)

**8.1.2.727 #define HTSPRO\_REG\_MEMORY\_3C 0xF0**

SuperPro memory address 0x3C register (4 bytes msb/lb)

**8.1.2.728 #define HTSPRO\_REG\_MEMORY\_3D 0xF4**

SuperPro memory address 0x3D register (4 bytes msb/lb)

**8.1.2.729 #define HTSPRO\_REG\_MEMORY\_3E 0xF8**

SuperPro memory address 0x3E register (4 bytes msb/lb)

**8.1.2.730 #define HTSPRO\_REG\_MEMORY\_3F 0xFC**

SuperPro memory address 0x3F register (4 bytes msb/lb)

**8.1.2.731 #define HTSPRO\_REG\_STROBE 0x50**

SuperPro strobe control register

**8.1.2.732 #define I2C\_ADDR\_DEFAULT 0x02**

Standard NXT I2C device address

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_I2CSendCommand.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_MSDeenergize.nxc](#), [ex\\_MSEnergize.nxc](#), [ex\\_MSIRTrain.nxc](#), [ex\\_MSPFComboDirect.nxc](#), [ex\\_MSPFComboPWM.nxc](#), [ex\\_MSPFRawOutput.nxc](#), [ex\\_MSPFRepeat.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSinglePin.nxc](#), [ex\\_MSPFTrain.nxc](#), [ex\\_MSReadValue.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

**8.1.2.733 #define I2C\_OPTION\_FAST 0x08**

Fast I2C speed

**8.1.2.734 #define I2C\_OPTION\_NORESTART 0x04**

Use no restart on I2C read

**8.1.2.735 #define I2C\_OPTION\_STANDARD 0x00**

Standard I2C speed

**8.1.2.736 #define I2C\_REG\_CMD 0x41**

Standard NXT I2C device command register

**Examples:**

[ex\\_MSReadValue.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

**8.1.2.737 #define I2C\_REG\_DEVICE\_ID 0x10**

Standard NXT I2C device ID register

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

**8.1.2.738 #define I2C\_REG\_VENDOR\_ID 0x08**

Standard NXT I2C vendor ID register

**Examples:**[ex\\_i2cdeviceinfo.nxc.](#)**8.1.2.739 #define I2C\_REG\_VERSION 0x00**

Standard NXT I2C version register

**Examples:**[ex\\_i2cdeviceinfo.nxc.](#)**8.1.2.740 #define IN\_1 0x00**

Input port 1

**8.1.2.741 #define IN\_2 0x01**

Input port 2

**8.1.2.742 #define IN\_3 0x02**

Input port 3

**8.1.2.743 #define IN\_4 0x03**

Input port 4

**8.1.2.744 #define IN\_MODE\_ANGLESTEP 0xE0**

RCX rotation sensor (16 ticks per revolution)

**8.1.2.745 #define IN\_MODE\_BOOLEAN 0x20**

Boolean value (0 or 1)

**8.1.2.746 #define IN\_MODE\_CELSIUS 0xA0**

RCX temperature sensor value in degrees celcius

**8.1.2.747 #define IN\_MODE\_FAHRENHEIT 0xC0**

RCX temperature sensor value in degrees fahrenheit

**8.1.2.748 #define IN\_MODE\_MODEMASK 0xE0**

Mask for the mode without any slope value

**8.1.2.749 #define IN\_MODE\_PCTFULLSCALE 0x80**

Scaled value from 0 to 100

**8.1.2.750 #define IN\_MODE\_PERIODCOUNTER 0x60**

Counts the number of boolean periods

**8.1.2.751 #define IN\_MODE\_RAW 0x00**

Raw value from 0 to 1023

**8.1.2.752 #define IN\_MODE\_SLOPEMASK 0x1F**

Mask for slope parameter added to mode

**8.1.2.753 #define IN\_MODE\_TRANSITIONCNT 0x40**

Counts the number of boolean transitions

**8.1.2.754 #define IN\_TYPE\_ANGLE 0x04**

RCX rotation sensor

**8.1.2.755 #define IN\_TYPE\_COLORBLUE 0x10**

NXT 2.0 color sensor with blue light

**8.1.2.756 #define IN\_TYPE\_COLOREXIT 0x12**

NXT 2.0 color sensor internal state

**8.1.2.757 #define IN\_TYPE\_COLORFULL 0x0D**

NXT 2.0 color sensor in full color mode

**8.1.2.758 #define IN\_TYPE\_COLORGREEN 0x0F**

NXT 2.0 color sensor with green light

**8.1.2.759 #define IN\_TYPE\_COLORNONE 0x11**

NXT 2.0 color sensor with no light

**8.1.2.760 #define IN\_TYPE\_COLORRED 0x0E**

NXT 2.0 color sensor with red light

**8.1.2.761 #define IN\_TYPE\_CUSTOM 0x09**

NXT custom sensor

**8.1.2.762 #define IN\_TYPE\_HISPEED 0x0C**

NXT Hi-speed port (only S4)

**8.1.2.763 #define IN\_TYPE\_LIGHT\_ACTIVE 0x05**

NXT light sensor with light

**8.1.2.764 #define IN\_TYPE\_LIGHT\_INACTIVE 0x06**

NXT light sensor without light

**8.1.2.765 #define IN\_TYPE\_LOWSPEED 0x0A**

NXT I2C digital sensor

**8.1.2.766 #define IN\_TYPE\_LOWSPEED\_9V 0x0B**

NXT I2C digital sensor with 9V power

**8.1.2.767 #define IN\_TYPE\_NO\_SENSOR 0x00**

No sensor configured

**8.1.2.768 #define IN\_TYPE\_REFLECTION 0x03**

RCX light sensor

**8.1.2.769 #define IN\_TYPE\_SOUND\_DB 0x07**

NXT sound sensor with dB scaling

**8.1.2.770 #define IN\_TYPE\_SOUND\_DBA 0x08**

NXT sound sensor with dBA scaling

**8.1.2.771 #define IN\_TYPE\_SWITCH 0x01**

NXT or RCX touch sensor

**8.1.2.772 #define IN\_TYPE\_TEMPERATURE 0x02**

RCX temperature sensor

**8.1.2.773 #define INPUT\_BLACKCOLOR 1**

The color value is black

**8.1.2.774 #define INPUT\_BLANK 3**

Access the blank value from color sensor value arrays

**8.1.2.775 #define INPUT\_BLUE 2**

Access the blue value from color sensor value arrays

**8.1.2.776 #define INPUT\_BLUECOLOR 2**

The color value is blue

**8.1.2.777 #define INPUT\_CAL\_POINT\_0 0**

Calibration point 0

**Examples:**

[ex\\_ColorCalibration.nxc](#), and [ex\\_ColorCalLimits.nxc](#).

**8.1.2.778 #define INPUT\_CAL\_POINT\_1 1**

Calibration point 1

**8.1.2.779 #define INPUT\_CAL\_POINT\_2 2**

Calibration point 2

**8.1.2.780 #define INPUT\_CUSTOM9V 0x01**

Custom sensor 9V

**8.1.2.781 #define INPUT\_CUSTOMACTIVE 0x02**

Custom sensor active

**8.1.2.782 #define INPUT\_CUSTOMINACTIVE 0x00**

Custom sensor inactive

**8.1.2.783 #define INPUT\_DIGI0 0x01**

Digital pin 0

**Examples:**

[ex\\_sysinputpinfunction.nxc](#).

**8.1.2.784 #define INPUT\_DIGI1 0x02**

Digital pin 1

**8.1.2.785 #define INPUT\_GREEN 1**

Access the green value from color sensor value arrays

**8.1.2.786 #define INPUT\_GREENCOLOR 3**

The color value is green

**8.1.2.787 #define INPUT\_INVALID\_DATA 0x01**

Invalid data flag

**8.1.2.788 #define INPUT\_NO\_OF\_COLORS 4**

The number of entries in the color sensor value arrays

**8.1.2.789 #define INPUT\_NO\_OF\_POINTS 3**

The number of calibration points

**8.1.2.790 #define INPUT\_PINCMD\_CLEAR 0x02**

Clear digital pin(s)

**Examples:**

[ex\\_sysinputpinfunction.nxc.](#)

**8.1.2.791 #define INPUT\_PINCMD\_DIR 0x00**

Set digital pin(s) direction

**Examples:**

[ex\\_sysinputpinfunction.nxc.](#)

**8.1.2.792 #define INPUT\_PINCMD\_MASK 0x03**

Mask for the two bits used by pin function commands

**8.1.2.793 #define INPUT\_PINCMD\_READ 0x03**

Read digital pin(s)

**8.1.2.794 #define INPUT\_PINCMD\_SET 0x01**

Set digital pin(s)

**Examples:**

[ex\\_sysinputpinfunction.nxc.](#)

**8.1.2.795 #define INPUT\_PINCMD\_WAIT(\_usec) ((\_usec)<<2)**

A wait value in microseconds that can be added after one of the above commands by ORing with the command

**Examples:**

[ex\\_sysinputpinfunction.nxc.](#)

**8.1.2.796 #define INPUT\_PINDIR\_INPUT 0x04**

Use with the direction command to set direction to output. OR this with the pin value.

**8.1.2.797 #define INPUT\_PINDIR\_OUTPUT 0x00**

Use with the direction command to set direction to input. OR this with the pin value.

**Examples:**

[ex\\_sysinputpinfunction.nxc.](#)

**8.1.2.798 #define INPUT\_RED 0**

Access the red value from color sensor value arrays

**Examples:**

[ex\\_ColorADRaw.nxc](#), [ex\\_ColorBoolean.nxc](#), [ex\\_ColorCalibration.nxc](#), [ex\\_ColorSensorRaw.nxc](#), and [ex\\_ColorSensorValue.nxc](#).

**8.1.2.799 #define INPUT\_REDCOLOR 5**

The color value is red

**8.1.2.800 #define INPUT\_RESETCAL 0x80**

Unused calibration state constant

**8.1.2.801 #define INPUT\_RUNNINGCAL 0x20**

Unused calibration state constant

**8.1.2.802 #define INPUT\_SENSORCAL 0x01**

The state returned while the color sensor is calibrating

**8.1.2.803 #define INPUT\_SENSOROFF 0x02**

The state returned once calibration has completed

**8.1.2.804 #define INPUT\_STARTCAL 0x40**

Unused calibration state constant

**8.1.2.805 #define INPUT\_WHITECOLOR 6**

The color value is white

**8.1.2.806 #define INPUT\_YELLOWCOLOR 4**

The color value is yellow

**8.1.2.807 #define InputModeField 1**

Input mode field. Contains one of the sensor mode constants. Read/write.

**8.1.2.808 #define InputModuleID 0x00030001**

The input module ID

**8.1.2.809 #define InputModuleName "Input.mod"**

The input module name.

**8.1.2.810 #define InputOffsetADRaw(p) (((p)\*20)+2)**

Read the AD raw sensor value (2 bytes) uword

**8.1.2.811 #define InputOffsetColorADRaw(p, nc) (80+((p)\*84)+52+((nc)\*2))**

Read AD raw color sensor values

**8.1.2.812 #define InputOffsetColorBoolean(p, nc) (80+((p)\*84)+76+((nc)\*2))**

Read color sensor boolean values

**8.1.2.813 #define InputOffsetColorCalibration(p, np, nc) (80+((p)\*84)+0+((np)\*16)+((nc)\*4))**

Read/write color calibration point values

**8.1.2.814 #define InputOffsetColorCalibrationState(p) (80+((p)\*84)+80)**

Read color sensor calibration state

**8.1.2.815 #define InputOffsetColorCalLimits(p, np) (80+((p)\*84)+48+((np)\*2))**

Read/write color calibration limits

**8.1.2.816 #define InputOffsetColorSensorRaw(p, nc) (80+((p)\*84)+60+((nc)\*2))**

Read raw color sensor values

**8.1.2.817 #define InputOffsetColorSensorValue(p, nc) (80+((p)\*84)+68+((nc)\*2))**

Read scaled color sensor values

**8.1.2.818 #define InputOffsetCustomActiveStatus(p) (((p)\*20)+15)**

Read/write the active or inactive state of the custom sensor

**8.1.2.819 #define InputOffsetCustomPctFullScale(p) (((p)\*20)+14)**

Read/write the Pct full scale of the custom sensor

**8.1.2.820 #define InputOffsetCustomZeroOffset(p) (((p)\*20)+0)**

Read/write the zero offset of a custom sensor (2 bytes) uword

**8.1.2.821 #define InputOffsetDigiPinsDir(p) (((p)\*20)+11)**

Read/write the direction of the Digital pins (1 is output, 0 is input)

**8.1.2.822 #define InputOffsetDigiPinsIn(p) (((p)\*20)+12)**

Read/write the status of the digital pins

**8.1.2.823 #define InputOffsetDigiPinsOut(p) (((p)\*20)+13)**

Read/write the output level of the digital pins

**8.1.2.824 #define InputOffsetInvalidData(p) (((p)\*20)+16)**

Indicates whether data is invalid (1) or valid (0)

**8.1.2.825 #define InputOffsetSensorBoolean(p) (((p)\*20)+10)**

Read the sensor boolean value

**8.1.2.826 #define InputOffsetSensorMode(p) (((p)\*20)+9)**

Read/write the sensor mode

**8.1.2.827 #define InputOffsetSensorRaw(p) (((p)\*20)+4)**

Read the raw sensor value (2 bytes) uword

**8.1.2.828 #define InputOffsetSensorType(p) (((p)\*20)+8)**

Read/write the sensor type

**8.1.2.829 #define InputOffsetSensorValue(p) (((p)\*20)+6)**

Read/write the scaled sensor value (2 bytes) sword

**8.1.2.830 #define InputPinFunction 77**

Execute the Input module's pin function

**8.1.2.831 #define INT\_MAX 32767**

The maximum value of the int type

**8.1.2.832 #define INT\_MIN -32768**

The minimum value of the int type

**8.1.2.833 #define INTF\_BTOFF 13**

Turn off the bluetooth radio

**Examples:**

[ex\\_syscommexecutefunction.nxc](#).

**8.1.2.834 #define INTF\_BTON 12**

Turn on the bluetooth radio

**8.1.2.835 #define INTF\_CONNECT 3**

Connect to one of the known devices

**8.1.2.836 #define INTF\_CONNECTBYNAME 18**

Connect to a bluetooth device by name

**8.1.2.837 #define INTF\_CONNECTREQ 17**

Connection request from another device

**8.1.2.838 #define INTF\_DISCONNECT 4**

Disconnect from one of the connected devices

**8.1.2.839 #define INTF\_DISCONNECTALL 5**

Disconnect all devices

**8.1.2.840 #define INTF\_EXTREAD 15**

External read request

**8.1.2.841 #define INTF\_FACTORYRESET 11**

Reset bluetooth settings to factory values

**8.1.2.842 #define INTF\_OPENSTREAM 9**

Open a bluetooth stream

**8.1.2.843 #define INTF\_PINREQ 16**

Bluetooth PIN request

**8.1.2.844 #define INTF\_REMOVEDEVICE 6**

Remove a device from the known devices table

**8.1.2.845 #define INTF\_SEARCH 1**

Search for bluetooth devices

**8.1.2.846 #define INTF\_SENDDATA 10**

Send data over a bluetooth connection

**8.1.2.847 #define INTF\_SENDFILE 0**

Send a file via bluetooth to another device

**8.1.2.848 #define INTF\_SETBTNAME 14**

Set the bluetooth name

**8.1.2.849 #define INTF\_SETCMDMODE 8**

Set bluetooth into command mode

**8.1.2.850 #define INTF\_STOPSEARCH 2**

Stop searching for bluetooth devices

**8.1.2.851 #define INTF\_VISIBILITY 7**

Set the bluetooth visibility on or off

**8.1.2.852 #define InvalidDataField 5**

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

**8.1.2.853 #define IOCTRL\_BOOT 0xA55A**

Reboot the NXT into SAMBA mode

**8.1.2.854 #define IOCTRL\_POWERDOWN 0x5A00**

Power down the NXT

**8.1.2.855 #define IOCtrlModuleID 0x00060001**

The IOCtrl module ID

**8.1.2.856 #define IOCtrlModuleName "IOCtrl.mod"**

The IOCtrl module name

**8.1.2.857 #define IOCtrlOffsetPowerOn 0**

Offset to power on field

**8.1.2.858 #define IOMapRead 32**

Read data from one of the firmware module's IOMap structures using the module's name

**8.1.2.859 #define IOMapReadByID 78**

Read data from one of the firmware module's IOMap structures using the module's ID

**8.1.2.860 #define IOMapWrite 33**

Write data to one of the firmware module's IOMap structures using the module's name

**8.1.2.861 #define IOMapWriteByID 79**

Write data to one of the firmware module's IOMap structures using the module's ID

**8.1.2.862 #define KeepAlive 31**

Reset the NXT sleep timer

**8.1.2.863 #define LCD\_LINE1 56**

The 1st line of the LCD screen

**Examples:**

[ex\\_acos.nxc](#), [ex\\_acosd.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#),

[ex\\_asin.nxc](#), [ex\\_asind.nxc](#), [ex\\_atan.nxc](#), [ex\\_atand.nxc](#), [ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_buttonpressed.nxc](#), [ex\\_clearline.nxc](#), [ex\\_contrast.nxc](#), [ex\\_copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_GetBrickDataAddress.nxc](#), [ex\\_getchar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_motoroutputoptions.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_Pos.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorHTTouchMultiplexer.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_SizeOf.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_superpro.nxc](#), [ex\\_syscall.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommBTOff.nxc](#), [ex\\_SysCommHSCheckStatus.nxc](#), [ex\\_SysCommHSControl.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_SysDatalogWrite.nxc](#), [ex\\_sysdrawtext.nxc](#), [ex\\_sysfilefindfirst.nxc](#), [ex\\_sysfilefindnext.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_systemmemorymanager.nxc](#), [ex\\_sysmessageread.nxc](#), [ex\\_SysReadLastResponse.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), [ex\\_SysWriteSemData.nxc](#), [ex\\_UnflattenVar.nxc](#), and [ex\\_xg1300.nxc](#).

#### 8.1.2.864 #define LCD\_LINE2 48

The 2nd line of the LCD screen

#### Examples:

[ex\\_acos.nxc](#), [ex\\_acosd.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_asin.nxc](#), [ex\\_asind.nxc](#), [ex\\_atan.nxc](#), [ex\\_atand.nxc](#), [ex\\_buttonpressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#),

ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_SetAbortFlag.nxc, ex\_setdisplayfont.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_string.nxc, ex\_strtod.nxc, ex\_strtol.nxc, ex\_strtoul.nxc, ex\_SubStr.nxc, ex\_superpro.nxc, ex\_syscommbtconnection.nxc, ex\_sysfileread.nxc, ex\_sysmemorymanager.nxc, ex\_SysReadLastResponse.nxc, ex\_UnflattenVar.nxc, ex\_xg1300.nxc, util\_battery\_1.nxc, util\_battery\_2.nxc, and util\_rpm.nxc.

#### 8.1.2.865 #define LCD\_LINE3 40

The 3rd line of the LCD screen

##### Examples:

ex\_acos.nxc, ex\_acosd.nxc, ex\_ArraySort.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc, ex\_atand.nxc, ex\_buttonpressed.nxc, ex\_ctype.nxc, ex\_diaccl.nxc, ex\_digps.nxc, ex\_digyro.nxc, ex\_dispmisc.nxc, ex\_findfirstfile.nxc, ex\_findnextfile.nxc, ex\_FlattenVar.nxc, ex\_i2cdeviceid.nxc, ex\_i2cdeviceinfo.nxc, ex\_i2cvendorid.nxc, ex\_i2cversion.nxc, ex\_joystickmsg.nxc, ex\_memcmp.nxc, ex\_NXTLineLeader.nxc, ex\_NXTPowerMeter.nxc, ex\_NXTServo.nxc, ex\_proto.nxc, ex\_ReadSensorHTAngle.nxc, ex\_ReadSensorHTBarometric.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_SetAbortFlag.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_StrCatOld.nxc, ex\_string.nxc, ex\_strtod.nxc, ex\_strtol.nxc, ex\_strtoul.nxc, ex\_superpro.nxc, ex\_syscommbtconnection.nxc, ex\_TextOut.nxc, ex\_UnflattenVar.nxc, and ex\_xg1300.nxc.

#### 8.1.2.866 #define LCD\_LINE4 32

The 4th line of the LCD screen

##### Examples:

ex\_acos.nxc, ex\_acosd.nxc, ex\_addressof.nxc, ex\_addressofex.nxc, ex\_ArrayBuild.nxc, ex\_ArraySort.nxc, ex\_asin.nxc, ex\_asind.nxc, ex\_atan.nxc, ex\_atand.nxc, ex\_buttonpressed.nxc, ex\_ctype.nxc, ex\_DataMode.nxc, ex\_diaccl.nxc, ex\_digps.nxc, ex\_displayfont.nxc, ex\_dispmisc.nxc, ex\_FlattenVar.nxc, ex\_joystickmsg.nxc, ex\_NXTPowerMeter.nxc, ex\_proto.nxc, ex\_ReadSensorHTBarometric.nxc, ex\_ReadSensorHTTouchMultiplexer.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_reladdressof.nxc, ex\_SetAbortFlag.nxc, ex\_setdisplayfont.nxc, ex\_SetLongAbort.nxc, ex\_SizeOf.nxc, ex\_string.nxc, ex\_StrReplace.nxc, ex\_superpro.nxc, ex\_sysdataloggettimes.nxc, and ex\_UnflattenVar.nxc.

**8.1.2.867 #define LCD\_LINE5 24**

The 5th line of the LCD screen

**Examples:**

`ex_ArrayBuild.nxc`, `ex_ArraySort.nxc`, `ex_atan.nxc`, `ex_atand.nxc`, `ex_ctype.nxc`, `ex_DataMode.nxc`, `ex_diaccl.nxc`, `ex_digps.nxc`, `ex_dispmisc.nxc`, `ex_joystickmsg.nxc`, `ex_NXTPowerMeter.nxc`, `ex_proto.nxc`, `ex_ReadSensorHTBarometric.nxc`, `ex_StrIndex.nxc`, `ex_string.nxc`, `ex_superpro.nxc`, `ex_sysdataloggettimes.nxc`, and `ex_xg1300.nxc`.

**8.1.2.868 #define LCD\_LINE6 16**

The 6th line of the LCD screen

**Examples:**

`ex_ArraySort.nxc`, `ex_ctype.nxc`, `ex_diaccl.nxc`, `ex_digps.nxc`, `ex_joystickmsg.nxc`, `ex_NXTPowerMeter.nxc`, `ex_proto.nxc`, `ex_string.nxc`, `ex_StrLenOld.nxc`, `ex_superpro.nxc`, `ex_syslistfiles.nxc`, and `ex_xg1300.nxc`.

**8.1.2.869 #define LCD\_LINE7 8**

The 7th line of the LCD screen

**Examples:**

`ex_ArraySort.nxc`, `ex_ctype.nxc`, `ex_digps.nxc`, `ex_digyro.nxc`, `ex_joystickmsg.nxc`, `ex_NXTPowerMeter.nxc`, `ex_proto.nxc`, `ex_string.nxc`, `ex_superpro.nxc`, and `ex_xg1300.nxc`.

**8.1.2.870 #define LCD\_LINE8 0**

The 8th line of the LCD screen

**Examples:**

`ex_ArraySort.nxc`, `ex_ctype.nxc`, `ex_diaccl.nxc`, `ex_digps.nxc`, `ex_digyro.nxc`, `ex_dispout.nxc`, `ex_getmemoryinfo.nxc`, `ex_joystickmsg.nxc`, `ex_proto.nxc`, `ex_SetAbortFlag.nxc`, `ex_SetLongAbort.nxc`, `ex_string.nxc`, `ex_superpro.nxc`, `ex_sysmemorymanager.nxc`, and `ex_xg1300.nxc`.

**8.1.2.871 #define LDR\_APPENDNOTPOSSIBLE 0x8D00**

Only datafiles can be appended to.

**8.1.2.872 #define LDR\_BTBUSY 0x9400**

The bluetooth system is busy.

**8.1.2.873 #define LDR\_BTCONNECTFAIL 0x9500**

Bluetooth connection attempt failed.

**8.1.2.874 #define LDR\_BTTIMEOUT 0x9600**

A timeout in the bluetooth system has occurred.

**8.1.2.875 #define LDR\_CMD\_BOOTCMD 0x97**

Reboot the NXT into SAMBA mode

**8.1.2.876 #define LDR\_CMD\_BTFACTORYRESET 0xA4**

Reset bluetooth configuration to factory defaults

**8.1.2.877 #define LDR\_CMD\_BTGETADR 0x9A**

Get the NXT's bluetooth brick address

**8.1.2.878 #define LDR\_CMD\_CLOSE 0x84**

Close a file handle

**8.1.2.879 #define LDR\_CMD\_CLOSEMODHANDLE 0x92**

Close a module handle

**8.1.2.880 #define LDR\_CMD\_CROPDATAFILE 0x8D**

Crop a data file to its used space

**8.1.2.881 #define LDR\_CMD\_DELETE 0x85**

Delete a file

**8.1.2.882 #define LDR\_CMD\_DELETEUSERFLASH 0xA0**

Delete all files from user flash memory

**8.1.2.883 #define LDR\_CMD\_DEVICEINFO 0x9B**

Read device information

**8.1.2.884 #define LDR\_CMD\_FINDFIRST 0x86**

Find the first file matching the specified pattern

**8.1.2.885 #define LDR\_CMD\_FINDFIRSTMODULE 0x90**

Find the first module matching the specified pattern

**8.1.2.886 #define LDR\_CMD\_FINDNEXT 0x87**

Find the next file matching the specified pattern

**8.1.2.887 #define LDR\_CMD\_FINDNEXTMODULE 0x91**

Find the next module matching the specified pattern

**8.1.2.888 #define LDR\_CMD\_IOMAPREAD 0x94**

Read data from a module IOMAP

**8.1.2.889 #define LDR\_CMD\_IOMAPWRITE 0x95**

Write data to a module IOMAP

**8.1.2.890 #define LDR\_CMD\_OPENAPPENDDATA 0x8C**

Open a data file for appending

**8.1.2.891 #define LDR\_CMD\_OPENREAD 0x80**

Open a file for reading

**8.1.2.892 #define LDR\_CMD\_OPENREADLINEAR 0x8A**

Open a linear file for reading

**8.1.2.893 #define LDR\_CMD\_OPENWRITE 0x81**

Open a file for writing

**8.1.2.894 #define LDR\_CMD\_OPENWRITEDATA 0x8B**

Open a data file for writing

**8.1.2.895 #define LDR\_CMD\_OPENWRITELINEAR 0x89**

Open a linear file for writing

**8.1.2.896 #define LDR\_CMD\_POLLCMD 0xA2**

Poll command

**8.1.2.897 #define LDR\_CMD\_POLLCMDLEN 0xA1**

Read poll command length

**8.1.2.898 #define LDR\_CMD\_READ 0x82**

Read from a file

**8.1.2.899 #define LDR\_CMD\_RENAMEFILE 0xA3**

Rename a file

**8.1.2.900 #define LDR\_CMD\_RESIZEDATAFILE 0xD0**

Resize a data file

**8.1.2.901 #define LDR\_CMD\_SEEKFROMCURRENT 0xD2**

Seek from the current position

**8.1.2.902 #define LDR\_CMD\_SEEKFROMEND 0xD3**

Seek from the end of the file

**8.1.2.903 #define LDR\_CMD\_SEEKFROMSTART 0xD1**

Seek from the start of the file

**8.1.2.904 #define LDR\_CMD\_SETBRICKNAME 0x98**

Set the NXT's brick name

**8.1.2.905 #define LDR\_CMD\_VERSIONS 0x88**

Read firmware version information

**8.1.2.906 #define LDR\_CMD\_WRITE 0x83**

Write to a file

**8.1.2.907 #define LDR\_ENDOFFILE 0x8500**

The end of the file has been reached.

**Examples:**

[ex\\_file\\_system.nxc](#).

**8.1.2.908 #define LDR\_EOFEXPECTED 0x8400**

EOF expected.

**Examples:**

[ex\\_file\\_system.nxc](#).

**8.1.2.909 #define LDR\_FILEEXISTS 0x8F00**

A file with the same name already exists.

**Examples:**

[ex\\_file\\_system.nxc](#).

**8.1.2.910 #define LDR\_FILEISBUSY 0x8B00**

The file is already being used.

**8.1.2.911 #define LDR\_FILEISFULL 0x8E00**

The allocated file size has been filled.

**Examples:**

[ex\\_file\\_system.nxc](#).

**8.1.2.912 #define LDR\_FILENOTFOUND 0x8700**

No files matched the search criteria.

**8.1.2.913 #define LDR\_FILETX\_CLOSEERROR 0x9B00**

Error transmitting file: attempt to close file failed.

**8.1.2.914 #define LDR\_FILETX\_DSTEXISTS 0x9800**

Error transmitting file: destination file exists.

**8.1.2.915 #define LDR\_FILETX\_SRCMISSING 0x9900**

Error transmitting file: source file is missing.

**8.1.2.916 #define LDR\_FILETX\_STREAMERROR 0x9A00**

Error transmitting file: a stream error occurred.

**8.1.2.917 #define LDR\_FILETX\_TIMEOUT 0x9700**

Error transmitting file: a timeout occurred.

**8.1.2.918 #define LDR\_HANDLEALREADYCLOSED 0x8800**

The file handle has already been closed.

**8.1.2.919 #define LDR\_ILLEGALFILENAME 0x9200**

Filename length too long or attempted open a system file (\*.rx, \*.rtm, or \*.sys) for writing as a datafile.

**8.1.2.920 #define LDR\_ILLEGALHANDLE 0x9300**

Invalid file handle.

**8.1.2.921 #define LDR\_INPROGRESS 0x0001**

The function is executing but has not yet completed.

**8.1.2.922 #define LDR\_INVALIDSEEK 0x9C00**

Invalid file seek operation.

**8.1.2.923 #define LDR\_MODULENOTFOUND 0x9000**

No modules matched the specified search criteria.

**8.1.2.924 #define LDR\_NOLINEARSPACE 0x8900**

Not enough linear flash memory is available.

**8.1.2.925 #define LDR\_NOMOREFILES 0x8300**

The maximum number of files has been reached.

**8.1.2.926 #define LDR\_NOMOREHANDLES 0x8100**

All available file handles are in use.

**8.1.2.927 #define LDR\_NOSPACE 0x8200**

Not enough free flash memory for the specified file size.

**8.1.2.928 #define LDR\_NOTLINEARFILE 0x8600**

The specified file is not linear.

**8.1.2.929 #define LDR\_NOWRITEBUFFERS 0x8C00**

No more write buffers are available.

**8.1.2.930 #define LDR\_OUTOFBOUNDARY 0x9100**

Specified IOMap offset is outside the bounds of the IOMap.

**8.1.2.931 #define LDR\_REQPIN 0x0002**

A PIN exchange request is in progress.

**8.1.2.932 #define LDR\_SUCCESS 0x0000**

The function completed successfully.

**Examples:**

[ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_syscommbtcheckstatus.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_sysfilerename.nxc](#), and [ex\\_sysfileresolvehandle.nxc](#).

**8.1.2.933 #define LDR\_UNDEFINEDERROR 0x8A00**

An undefined error has occurred.

**8.1.2.934 #define LED\_BLUE 0x02**

Turn on the blue onboard LED.

**Examples:**

[ex\\_superpro.nxc](#).

**8.1.2.935 #define LED\_NONE 0x00**

Turn off the onboard LEDs.

**8.1.2.936 #define LED\_RED 0x01**

Turn on the red onboard LED.

**8.1.2.937 #define LEGO\_ADDR\_EMETER 0x04**

The LEGO e-meter sensor's I2C address

**8.1.2.938 #define LEGO\_ADDR\_TEMP 0x98**

The LEGO temperature sensor's I2C address

**8.1.2.939 #define LEGO\_ADDR\_US 0x02**

The LEGO ultrasonic sensor's I2C address

**8.1.2.940 #define ListFiles 47**

List files that match the specified filename pattern

**8.1.2.941 #define LoaderExecuteFunction 82**

Execute one of the Loader module's internal functions

**8.1.2.942 #define LoaderModuleID 0x00090001**

The Loader module ID

**8.1.2.943 #define LoaderModuleName "Loader.mod"**

The Loader module name

**8.1.2.944 #define LoaderOffsetFreeUserFlash 4**

Offset to the amount of free user flash

**8.1.2.945 #define LoaderOffsetPFunc 0**

Offset to the Loader module function pointer

**8.1.2.946 #define LONG\_MAX 2147483647**

The maximum value of the long type

**8.1.2.947 #define LONG\_MIN -2147483648**

The minimum value of the long type

**8.1.2.948 #define LOWSPEED\_CH\_NOT\_READY 1**

Lowspeed port is not ready

**8.1.2.949 #define LOWSPEED\_COMMUNICATING 3**

Channel is actively communicating

**8.1.2.950 #define LOWSPEED\_DATA\_RECEIVED 3**

Lowspeed port is in data received mode

**8.1.2.951 #define LOWSPEED\_DONE 5**

Channel is done communicating

**8.1.2.952 #define LOWSPEED\_ERROR 4**

Channel is in an error state

**8.1.2.953 #define LOWSPEED\_IDLE 0**

Channel is idle

**Examples:**[ex\\_syscommlscheckstatus.nxc.](#)

- 8.1.2.954 #define LOWSPEED\_INIT 1**  
Channel is being initialized
- 8.1.2.955 #define LOWSPEED\_LOAD\_BUFFER 2**  
Channel buffer is loading
- 8.1.2.956 #define LOWSPEED\_NO\_ERROR 0**  
Lowspeed port has no error
- 8.1.2.957 #define LOWSPEED\_RECEIVING 2**  
Lowspeed port is in receiving mode
- 8.1.2.958 #define LOWSPEED\_RX\_ERROR 3**  
Lowspeed port encountered an error while receiving data
- 8.1.2.959 #define LOWSPEED\_TRANSMITTING 1**  
Lowspeed port is in transmitting mode
- 8.1.2.960 #define LOWSPEED\_TX\_ERROR 2**  
Lowspeed port encountered an error while transmitting data
- 8.1.2.961 #define LowSpeedModuleID 0x000B0001**  
The low speed module ID
- 8.1.2.962 #define LowSpeedModuleName "Low Speed.mod"**  
The low speed module name
- 8.1.2.963 #define LowSpeedOffsetChannelState(p) ((p)+156)**  
R - Lowspeed channel state (1 byte)

**8.1.2.964 #define LowSpeedOffsetErrorType(p) ((p)+160)**

R - Lowspeed port error type (1 byte)

**8.1.2.965 #define LowSpeedOffsetInBufBuf(p) (((p)\*19)+0)**

RW - Input buffer data buffer field offset (16 bytes)

**8.1.2.966 #define LowSpeedOffsetInBufBytesToRx(p) (((p)\*19)+18)**

RW - Input buffer bytes to receive field offset (1 byte)

**8.1.2.967 #define LowSpeedOffsetInBufInPtr(p) (((p)\*19)+16)**

RW - Input buffer in pointer field offset (1 byte)

**8.1.2.968 #define LowSpeedOffsetInBufOutPtr(p) (((p)\*19)+17)**

RW - Input buffer out pointer field offset (1 byte)

**8.1.2.969 #define LowSpeedOffsetMode(p) ((p)+152)**

R - Lowspeed port mode (1 byte)

**8.1.2.970 #define LowSpeedOffsetNoRestartOnRead 166**

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

**8.1.2.971 #define LowSpeedOffsetOutBufBuf(p) (((p)\*19)+76)**

RW - Output buffer data buffer field offset (16 bytes)

**8.1.2.972 #define LowSpeedOffsetOutBufBytesToRx(p) (((p)\*19)+94)**

RW - Output buffer bytes to receive field offset (1 byte)

**8.1.2.973 #define LowSpeedOffsetOutBufInPtr(p) (((p)\*19)+92)**

RW - Output buffer in pointer field offset (1 byte)

**8.1.2.974 #define LowSpeedOffsetOutBufOutPtr(p) (((p)\*19)+93)**

RW - Output buffer out pointer field offset (1 byte)

**8.1.2.975 #define LowSpeedOffsetSpeed 165**

R - Lowspeed speed (unused)

**8.1.2.976 #define LowSpeedOffsetState 164**

R - Lowspeed state (all channels)

**8.1.2.977 #define LR\_COULD\_NOT\_SAVE 0x51**

Bluetooth list result could not save

**8.1.2.978 #define LR\_ENTRY\_REMOVED 0x53**

Bluetooth list result entry removed

**8.1.2.979 #define LR\_STORE\_IS\_FULL 0x52**

Bluetooth list result store is full

**8.1.2.980 #define LR\_SUCCESS 0x50**

Bluetooth list result success

**8.1.2.981 #define LR\_UNKNOWN\_ADDR 0x54**

Bluetooth list result unknown address

**8.1.2.982 #define LSREAD\_NO\_RESTART\_1 0x01**

No restart on read for channel 1

**8.1.2.983 #define LSREAD\_NO\_RESTART\_2 0x02**

No restart on read for channel 2

**8.1.2.984 #define LSREAD\_NO\_RESTART\_3 0x04**

No restart on read for channel 3

**8.1.2.985 #define LSREAD\_NO\_RESTART\_4 0x08**

No restart on read for channel 4

**8.1.2.986 #define LSREAD\_NO\_RESTART\_MASK 0x10**

No restart mask

**8.1.2.987 #define LSREAD\_RESTART\_ALL 0x00**

Restart on read for all channels (default)

**8.1.2.988 #define LSREAD\_RESTART\_NONE 0x0F**

No restart on read for all channels

**8.1.2.989 #define MAILBOX1 0**

Mailbox number 1

**Examples:**

[ex\\_joystickmsg.nxc](#), [ex\\_ReceiveMessage.nxc](#), [ex\\_ReceiveRemoteBool.nxc](#),  
[ex\\_ReceiveRemoteMessageEx.nxc](#), [ex\\_ReceiveRemoteNumber.nxc](#), [ex\\_-  
SendMessage.nxc](#), [ex\\_SendRemoteBool.nxc](#), [ex\\_SendRemoteNumber.nxc](#),  
[ex\\_SendRemoteString.nxc](#), [ex\\_SendResponseBool.nxc](#), [ex\\_-  
SendResponseNumber.nxc](#), [ex\\_SendResponseString.nxc](#), [ex\\_-  
sysmessageread.nxc](#), and [ex\\_sysmessagewrite.nxc](#).

**8.1.2.990 #define MAILBOX10 9**

Mailbox number 10

**8.1.2.991 #define MAILBOX2 1**

Mailbox number 2

**8.1.2.992 #define MAILBOX3 2**

Mailbox number 3

**8.1.2.993 #define MAILBOX4 3**

Mailbox number 4

**8.1.2.994 #define MAILBOX5 4**

Mailbox number 5

**8.1.2.995 #define MAILBOX6 5**

Mailbox number 6

**8.1.2.996 #define MAILBOX7 6**

Mailbox number 7

**8.1.2.997 #define MAILBOX8 7**

Mailbox number 8

**8.1.2.998 #define MAILBOX9 8**

Mailbox number 9

**8.1.2.999 #define MAX\_BT\_MSG\_SIZE 60000**

Max Bluetooth Message Size

**8.1.2.1000 #define MaxAccelerationField 17**

MaxAcceleration field. Contains the current max acceleration value. Read/write. Set the maximum acceleration to be used during position regulation.

**8.1.2.1001 #define MaxSpeedField 16**

MaxSpeed field. Contains the current max speed value. Read/write. Set the maximum speed to be used during position regulation.

**8.1.2.1002 #define MemoryManager 96**

Read memory manager information, optionally compacting the dataspace first

**8.1.2.1003 #define MENUICON\_CENTER 1**

Center icon

**8.1.2.1004 #define MENUICON\_LEFT 0**

Left icon

**8.1.2.1005 #define MENUICON\_RIGHT 2**

Right icon

**8.1.2.1006 #define MENUICONS 3**

The number of menu icons

**8.1.2.1007 #define MENUTEXT 2**

Center icon text

**8.1.2.1008 #define MessageRead 27**

Read a message from a mailbox

**8.1.2.1009 #define MessageWrite 26**

Write a message to a mailbox

**8.1.2.1010 #define MI\_ADDR\_XG1300L 0x02**

XG1300L I2C address

**8.1.2.1011 #define MIN\_1 60000**

1 minute

**Examples:**[ex\\_SysSetSleepTimeout.nxc](#).**8.1.2.1012 #define MS\_1 1**

1 millisecond

**Examples:**[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).**8.1.2.1013 #define MS\_10 10**

10 milliseconds

**Examples:**[ex\\_diaccl.nxc](#), and [ex\\_PosReg.nxc](#).**8.1.2.1014 #define MS\_100 100**

100 milliseconds

**Examples:**[ex\\_joystickmsg.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_sysdrawpolygon.nxc](#), and [ex\\_xg1300.nxc](#).**8.1.2.1015 #define MS\_150 150**

150 milliseconds

**8.1.2.1016** `#define MS_2 2`

2 milliseconds

**8.1.2.1017** `#define MS_20 20`

20 milliseconds

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cospd.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).

**8.1.2.1018** `#define MS_200 200`

200 milliseconds

**Examples:**

[ex\\_dispgoutex.nxc](#), and [ex\\_playtones.nxc](#).

**8.1.2.1019** `#define MS_250 250`

250 milliseconds

**8.1.2.1020** `#define MS_3 3`

3 milliseconds

**8.1.2.1021** `#define MS_30 30`

30 milliseconds

**8.1.2.1022** `#define MS_300 300`

300 milliseconds

**8.1.2.1023** `#define MS_350 350`

350 milliseconds

**8.1.2.1024** `#define MS_4 4`

4 milliseconds

**8.1.2.1025** `#define MS_40 40`

40 milliseconds

**8.1.2.1026** `#define MS_400 400`

400 milliseconds

**8.1.2.1027** `#define MS_450 450`

450 milliseconds

**8.1.2.1028** `#define MS_5 5`

5 milliseconds

**Examples:**

[ex\\_getchar.nxc](#).

**8.1.2.1029** `#define MS_50 50`

50 milliseconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digyro.nxc](#), and [ex\\_playtones.nxc](#).

**8.1.2.1030** `#define MS_500 500`

500 milliseconds

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_playsound.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_xg1300.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

---

|                   |                           |                  |
|-------------------|---------------------------|------------------|
| <b>8.1.2.1031</b> | <b>#define MS_6 6</b>     | 6 milliseconds   |
| <b>8.1.2.1032</b> | <b>#define MS_60 60</b>   | 60 milliseconds  |
| <b>8.1.2.1033</b> | <b>#define MS_600 600</b> | 600 milliseconds |
| <b>8.1.2.1034</b> | <b>#define MS_7 7</b>     | 7 milliseconds   |
| <b>8.1.2.1035</b> | <b>#define MS_70 70</b>   | 70 milliseconds  |
| <b>8.1.2.1036</b> | <b>#define MS_700 700</b> | 700 milliseconds |
| <b>8.1.2.1037</b> | <b>#define MS_8 8</b>     | 8 milliseconds   |
| <b>8.1.2.1038</b> | <b>#define MS_80 80</b>   | 80 milliseconds  |
| <b>8.1.2.1039</b> | <b>#define MS_800 800</b> | 800 milliseconds |
| <b>8.1.2.1040</b> | <b>#define MS_9 9</b>     | 9 milliseconds   |

**8.1.2.1041 #define MS\_90 90**

90 milliseconds

**8.1.2.1042 #define MS\_900 900**

900 milliseconds

**8.1.2.1043 #define MS\_ADDR\_ACCLNX 0x02**

MindSensors ACCL-Nx I2C address

**Examples:**

ex\_ACCLNxCalibrateX.nxc, ex\_ACCLNxCalibrateXEnd.nxc, ex\_  
ACCLNxCalibrateY.nxc, ex\_ACCLNxCalibrateYEnd.nxc, ex\_  
ACCLNxCalibrateZ.nxc, ex\_ACCLNxCalibrateZEnd.nxc, ex\_  
ACCLNxResetCalibration.nxc, ex\_ACCLNxSensitivity.nxc, ex\_  
ACCLNxXOffset.nxc, ex\_ACCLNxXRange.nxc, ex\_ACCLNxYOffset.nxc,  
ex\_ACCLNxYRange.nxc, ex\_ACCLNxZOffset.nxc, ex\_ACCLNxZRange.nxc,  
ex\_ReadSensorMSAccel.nxc, ex\_ReadSensorMSTilt.nxc, and ex\_  
SetACCLNxSensitivity.nxc.

**8.1.2.1044 #define MS\_ADDR\_CMPSNX 0x02**

MindSensors CMPS-Nx I2C address

**Examples:**

ex\_SensorMSCompass.nxc.

**8.1.2.1045 #define MS\_ADDR\_DISTNX 0x02**

MindSensors DIST-Nx I2C address

**Examples:**

ex\_DISTNxDistance.nxc, ex\_DISTNxGP2D12.nxc, ex\_DISTNxGP2D120.nxc,  
ex\_DISTNxGP2YA02.nxc, ex\_DISTNxGP2YA21.nxc, ex\_  
DISTNxMaxDistance.nxc, ex\_DISTNxMinDistance.nxc, ex\_  
DISTNxModuleType.nxc, ex\_DISTNxNumPoints.nxc, ex\_DISTNxVoltage.nxc,  
ex\_MSADPAOff.nxc, and ex\_MSADPAOn.nxc.

**8.1.2.1046 #define MS\_ADDR\_IVSENS 0x12**

MindSensors IVSens (NXTPowerMeter) I2C address

**Examples:**[ex\\_NXTPowerMeter.nxc](#).**8.1.2.1047 #define MS\_ADDR\_LINELDR 0x02**

MindSensors LineLdr I2C address

**Examples:**[ex\\_NXTLineLeader.nxc](#).**8.1.2.1048 #define MS\_ADDR\_MTRMUX 0xB4**

MindSensors MTRMux I2C address

**8.1.2.1049 #define MS\_ADDR\_NRLINK 0x02**

MindSensors NRLink I2C address

**Examples:**[ex\\_MSRCXSetNRLinkPort.nxc](#), [ex\\_NRLink2400.nxc](#), [ex\\_NRLink4800.nxc](#),  
[ex\\_NRLinkFlush.nxc](#), [ex\\_NRLinkIRLong.nxc](#), [ex\\_NRLinkIRShort.nxc](#),  
[ex\\_NRLinkSetPF.nxc](#), [ex\\_NRLinkSetRCX.nxc](#), [ex\\_NRLinkSetTrain.nxc](#),  
[ex\\_NRLinkStatus.nxc](#), [ex\\_NRLinkTxRaw.nxc](#), [ex\\_ReadNRLinkBytes.nxc](#),  
[ex\\_RunNRLinkMacro.nxc](#), and [ex\\_writenrlinkbytes.nxc](#).**8.1.2.1050 #define MS\_ADDR\_NXTCAM 0x02**

MindSensors NXTCam I2C address

**8.1.2.1051 #define MS\_ADDR\_NXTHID 0x04**

MindSensors NXTHID I2C address

**Examples:**[ex\\_NXTHID.nxc](#).

**8.1.2.1052 #define MS\_ADDR\_NXTMMX 0x06**

MindSensors NXTMMX I2C address

**8.1.2.1053 #define MS\_ADDR\_NXTSERVO 0xB0**

MindSensors NXTServo I2C address

**Examples:**

[ex\\_NXTHID.nxc](#), and [ex\\_NXTServo.nxc](#).

**8.1.2.1054 #define MS\_ADDR\_NXTSERVO\_EM 0x40**

MindSensors NXTServo in edit macro mode I2C address

**8.1.2.1055 #define MS\_ADDR\_PFMATE 0x48**

MindSensors PFMate I2C address

**Examples:**

[ex\\_PFMate.nxc](#).

**8.1.2.1056 #define MS\_ADDR\_PSPNX 0x02**

MindSensors PSP-Nx I2C address

**Examples:**

[ex\\_PSPNxAAnalog.nxc](#), [ex\\_PSPNxDigital.nxc](#), and [ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1057 #define MS\_ADDR\_RTCLOCK 0xD0**

MindSensors RTClock I2C address

**8.1.2.1058 #define MS\_ADDR\_RXMUX 0x7E**

MindSensors RXMux I2C address

**8.1.2.1059 #define MS\_CMD\_ADPA\_OFF 0x4F**

Turn MindSensors ADPA mode off

**8.1.2.1060 #define MS\_CMD\_ADPA\_ON 0x4E**

Turn MindSensors ADPA mode on

**8.1.2.1061 #define MS\_CMD\_DEENERGIZED 0x44**

De-energize the MindSensors device

**8.1.2.1062 #define MS\_CMD\_ENERGIZED 0x45**

Energize the MindSensors device

**8.1.2.1063 #define NA 0xFFFF**

The specified argument does not apply (aka unwired)

**Examples:**

[ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), and [ex\\_ArraySumSqr.nxc](#).

**8.1.2.1064 #define NO\_ERR 0**

Successful execution of the specified command

**Examples:**

[ex\\_joystickmsg.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommBTOff.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_SysCommHSWrite.nxc](#), [ex\\_syscommmlswriteex.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_SysDatalogWrite.nxc](#), [ex\\_sysfileopenappend.nxc](#), [ex\\_sysfileopenread.nxc](#), [ex\\_sysfileopenreadlinear.nxc](#), [ex\\_sysfileopenwrite.nxc](#), [ex\\_sysfileopenwritelinear.nxc](#), [ex\\_sysfileopenwritenonlinear.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfileresize.nxc](#), [ex\\_sysfileseek.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_sysiomapread.nxc](#), [ex\\_sysiomapreadbyid.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_sysmessageread.nxc](#), and [ex\\_SysReadLastResponse.nxc](#).

**8.1.2.1065 #define NO\_OF\_BTNS 4**

The number of NXT buttons.

**8.1.2.1066 #define NormalizedValueField 3**

Normalized value field. Contains the current normalized analog sensor value. Read only.

**8.1.2.1067 #define NRLINK\_CMD\_2400 0x44**

Set NRLink to 2400 baud

**8.1.2.1068 #define NRLINK\_CMD\_4800 0x48**

Set NRLink to 4800 baud

**8.1.2.1069 #define NRLINK\_CMD\_FLUSH 0x46**

Flush the NRLink

**8.1.2.1070 #define NRLINK\_CMD\_IR\_LONG 0x4C**

Set the NRLink to long range IR

**8.1.2.1071 #define NRLINK\_CMD\_IR\_SHORT 0x53**

Set the NRLink to short range IR

**8.1.2.1072 #define NRLINK\_CMD\_RUN\_MACRO 0x52**

Run an NRLink macro

**8.1.2.1073 #define NRLINK\_CMD\_SET\_PF 0x50**

Set the NRLink to Power Function mode

**8.1.2.1074 #define NRLINK\_CMD\_SET\_RCX 0x58**

Set the NRLink to RCX mode

**8.1.2.1075 #define NRLINK\_CMD\_SET\_TRAIN 0x54**

Set the NRLink to IR Train mode

**8.1.2.1076 #define NRLINK\_CMD\_TX\_RAW 0x55**

Set the NRLink to transmit raw bytes

**8.1.2.1077 #define NRLINK\_REG\_BYTES 0x40**

The NRLink bytes register

**8.1.2.1078 #define NRLINK\_REG\_DATA 0x42**

The NRLink data register

**8.1.2.1079 #define NRLINK\_REG\_EEPROM 0x50**

The NRLink eeprom register

**8.1.2.1080 #define NULL 0**

A constant representing NULL

**8.1.2.1081 #define NXTHID\_CMD\_ASCII 0x41**

Use ASCII data mode. In ASCII mode no non-printable characters can be sent.

**8.1.2.1082 #define NXTHID\_CMD\_DIRECT 0x44**

Use direct data mode In direct mode any character can be sent.

**8.1.2.1083 #define NXTHID\_CMD\_TRANSMIT 0x54**

Transmit data to the host computer.

**8.1.2.1084 #define NXTHID\_MOD\_LEFT\_ALT 0x04**

NXTHID left alt modifier.

**8.1.2.1085** `#define NXTHID_MOD_LEFT_CTRL 0x01`

NXTHID left control modifier.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.1.2.1086** `#define NXTHID_MOD_LEFT_GUI 0x08`

NXTHID left gui modifier.

**8.1.2.1087** `#define NXTHID_MOD_LEFT_SHIFT 0x02`

NXTHID left shift modifier.

**8.1.2.1088** `#define NXTHID_MOD_NONE 0x00`

NXTHID no modifier.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.1.2.1089** `#define NXTHID_MOD_RIGHT_ALT 0x40`

NXTHID right alt modifier.

**8.1.2.1090** `#define NXTHID_MOD_RIGHT_CTRL 0x10`

NXTHID right control modifier.

**8.1.2.1091** `#define NXTHID_MOD_RIGHT_GUI 0x80`

NXTHID right gui modifier.

**8.1.2.1092** `#define NXTHID_MOD_RIGHT_SHIFT 0x20`

NXTHID right shift modifier.

**8.1.2.1093 #define NXTHID\_REG\_CMD 0x41**

NXTHID command register. See [MindSensors NXTHID commands](#) group.

**8.1.2.1094 #define NXTHID\_REG\_DATA 0x43**

NXTHID data register.

**8.1.2.1095 #define NXTHID\_REG\_MODIFIER 0x42**

NXTHID modifier register. See [MindSensors NXTHID modifier keys](#) group.

**8.1.2.1096 #define NXTLL\_CMD\_BLACK 0x42**

Black calibration.

**8.1.2.1097 #define NXTLL\_CMD\_EUROPEAN 0x45**

European power frequency. (50hz)

**8.1.2.1098 #define NXTLL\_CMD\_INVERT 0x49**

Invert color.

**8.1.2.1099 #define NXTLL\_CMD\_POWERDOWN 0x44**

Power down the device.

**8.1.2.1100 #define NXTLL\_CMD\_POWERUP 0x50**

Power up the device.

**8.1.2.1101 #define NXTLL\_CMD\_RESET 0x52**

Reset inversion.

**8.1.2.1102 #define NXTLL\_CMD\_SNAPSHOT 0x53**

Setpoint based on snapshot (automatically sets invert if needed).

**8.1.2.1103 #define NXTLL\_CMD\_UNIVERSAL 0x55**

Universal power frequency. The sensor auto adjusts for any frequency. This is the default mode.

**8.1.2.1104 #define NXTLL\_CMD\_USA 0x41**

USA power frequency. (60hz)

**8.1.2.1105 #define NXTLL\_CMD\_WHITE 0x57**

White balance calibration.

**8.1.2.1106 #define NXTLL\_REG\_AVERAGE 0x43**

NXTLineLeader average result register.

**8.1.2.1107 #define NXTLL\_REG\_BLACKDATA 0x6C**

NXTLineLeader black calibration data registers. 8 bytes.

**8.1.2.1108 #define NXTLL\_REG\_BLACKLIMITS 0x59**

NXTLineLeader black limit registers. 8 bytes.

**8.1.2.1109 #define NXTLL\_REG\_CALIBRATED 0x49**

NXTLineLeader calibrated sensor reading registers. 8 bytes.

**8.1.2.1110 #define NXTLL\_REG\_CMD 0x41**

NXTLineLeader command register. See the [MindSensors NXTLineLeader commands](#) group.

**8.1.2.1111 #define NXTLL\_REG\_KD\_FACTOR 0x63**

NXTLineLeader Kd factor register. Default = 32.

**8.1.2.1112 #define NXTLL\_REG\_KD\_VALUE 0x48**

NXTLineLeader Kd value register. Default = 8.

**8.1.2.1113 #define NXTLL\_REG\_KI\_FACTOR 0x62**

NXTLineLeader Ki factor register. Default = 32.

**8.1.2.1114 #define NXTLL\_REG\_KI\_VALUE 0x47**

NXTLineLeader Ki value register. Default = 0.

**8.1.2.1115 #define NXTLL\_REG\_KP\_FACTOR 0x61**

NXTLineLeader Kp factor register. Default = 32.

**8.1.2.1116 #define NXTLL\_REG\_KP\_VALUE 0x46**

NXTLineLeader Kp value register. Default = 25.

**8.1.2.1117 #define NXTLL\_REG\_RAWVOLTAGE 0x74**

NXTLineLeader uncalibrated sensor voltage registers. 16 bytes.

**8.1.2.1118 #define NXTLL\_REG\_RESULT 0x44**

NXTLineLeader result register (sensor bit values).

**8.1.2.1119 #define NXTLL\_REG\_SETPOINT 0x45**

NXTLineLeader user settable average (setpoint) register. Default = 45.

**8.1.2.1120 #define NXTLL\_REG\_STEERING 0x42**

NXTLineLeader steering register.

**8.1.2.1121 #define NXTLL\_REG\_WHITEDATA 0x64**

NXTLineLeader white calibration data registers. 8 bytes.

**8.1.2.1122 #define NXTLL\_REG\_WHITELIMITS 0x51**

NXTLineLeader white limit registers. 8 bytes.

**8.1.2.1123 #define NXTPM\_CMD\_RESET 0x52**

Reset counters.

**8.1.2.1124 #define NXTPM\_REG\_CAPACITY 0x46**

NXTPowerMeter capacity used since last reset register. (2 bytes)

**8.1.2.1125 #define NXTPM\_REG\_CMD 0x41**

NXTPowerMeter command register. See the [MindSensors NXTPowerMeter commands](#) group.

**8.1.2.1126 #define NXTPM\_REG\_CURRENT 0x42**

NXTPowerMeter present current in mA register. (2 bytes)

**8.1.2.1127 #define NXTPM\_REG\_ERRORCOUNT 0x5F**

NXTPowerMeter error count register. (2 bytes)

**8.1.2.1128 #define NXTPM\_REG\_GAIN 0x5E**

NXTPowerMeter gain register. (1 byte)

**8.1.2.1129 #define NXTPM\_REG\_MAXCURRENT 0x4E**

NXTPowerMeter max current register. (2 bytes)

**8.1.2.1130 #define NXTPM\_REG\_MAXVOLTAGE 0x52**

NXTPowerMeter max voltage register. (2 bytes)

**8.1.2.1131 #define NXTPM\_REG\_MINCURRENT 0x50**

NXTPowerMeter min current register. (2 bytes)

**8.1.2.1132 #define NXTPM\_REG\_MINVOLTAGE 0x54**

NXTPowerMeter min voltage register. (2 bytes)

**8.1.2.1133 #define NXTPM\_REG\_POWER 0x48**

NXTPowerMeter present power register. (2 bytes)

**8.1.2.1134 #define NXTPM\_REG\_TIME 0x56**

NXTPowerMeter time register. (4 bytes)

**8.1.2.1135 #define NXTPM\_REG\_TOTALPOWER 0x4A**

NXTPowerMeter total power consumed since last reset register. (4 bytes)

**8.1.2.1136 #define NXTPM\_REG\_USERGAIN 0x5A**

NXTPowerMeter user gain register. Not yet implemented. (4 bytes)

**8.1.2.1137 #define NXTPM\_REG\_VOLTAGE 0x44**

NXTPowerMeter present voltage in mV register. (2 bytes)

**8.1.2.1138 #define NXTSE\_ZONE\_FRONT 1**

Obstacle zone front.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.1.2.1139 #define NXTSE\_ZONE\_LEFT 2**

Obstacle zone left.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.1.2.1140 #define NXTSE\_ZONE\_NONE 0**

Obstacle zone none.

**8.1.2.1141 #define NXTSE\_ZONE\_RIGHT 3**

Obstacle zone right.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.1.2.1142 #define NXTSERVO\_CMD\_EDIT1 0x45**

Edit Macro (part 1 of 2 character command sequence)

**8.1.2.1143 #define NXTSERVO\_CMD\_EDIT2 0x4D**

Edit Macro (part 2 of 2 character command sequence)

**8.1.2.1144 #define NXTSERVO\_CMD\_GOTO 0x47**

Goto EEPROM position x. This command re-initializes the macro environment.

**8.1.2.1145 #define NXTSERVO\_CMD\_HALT 0x48**

Halt Macro. This command re-initializes the macro environment.

**8.1.2.1146 #define NXTSERVO\_CMD\_INIT 0x49**

Store the initial speed and position properties of the servo motor 'n'. Current speed and position values of the nth servo is read from the servo speed register and servo position register and written to permanent memory.

**8.1.2.1147 #define NXTSERVO\_CMD\_PAUSE 0x50**

Pause Macro. This command will pause the macro, and save the environment for subsequent resumption.

**8.1.2.1148 #define NXTSERVO\_CMD\_RESET 0x53**

Reset servo properties to factory default. Initial Position of servos to 1500, and speed to 0.

**8.1.2.1149 #define NXTSERVO\_CMD\_RESUME 0x52**

Resume macro Execution. This command resumes macro where it was paused last, using the same environment.

**8.1.2.1150 #define NXTSERVO\_EM\_CMD\_QUIT 0x51**

Exit edit macro mode

**8.1.2.1151 #define NXTSERVO\_EM\_REG\_CMD 0x00**

NXTServo in macro edit mode command register.

**8.1.2.1152 #define NXTSERVO\_EM\_REG\_EEPROM\_END 0xFF**

NXTServo in macro edit mode EEPROM end register.

**8.1.2.1153 #define NXTSERVO\_EM\_REG\_EEPROM\_START 0x21**

NXTServo in macro edit mode EEPROM start register.

**8.1.2.1154 #define NXTSERVO\_POS\_CENTER 1500**

Center position for 1500us servos.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.1.2.1155 #define NXTSERVO\_POS\_MAX 2500**

Maximum position for 1500us servos.

**8.1.2.1156 #define NXTSERVO\_POS\_MIN 500**

Minimum position for 1500us servos.

**8.1.2.1157 #define NXTSERVO\_QPOS\_CENTER 150**

Center quick position for 1500us servos.

**8.1.2.1158 #define NXTSERVO\_QPOS\_MAX 250**

Maximum quick position for 1500us servos.

**8.1.2.1159 #define NXTSERVO\_QPOS\_MIN 50**

Minimum quick position for 1500us servos.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.1.2.1160 #define NXTSERVO\_REG\_CMD 0x41**

NXTServo command register. See [MindSensors NXTServo commands](#) group. (write only)

**8.1.2.1161 #define NXTSERVO\_REG\_S1\_POS 0x42**

NXTServo servo 1 position register.

**8.1.2.1162 #define NXTSERVO\_REG\_S1\_QPOS 0x5A**

NXTServo servo 1 quick position register. (write only)

**8.1.2.1163 #define NXTSERVO\_REG\_S1\_SPEED 0x52**

NXTServo servo 1 speed register.

**8.1.2.1164 #define NXTSERVO\_REG\_S2\_POS 0x44**

NXTServo servo 2 position register.

**8.1.2.1165 #define NXTSERVO\_REG\_S2\_QPOS 0x5B**

NXTServo servo 2 quick position register. (write only)

**8.1.2.1166 #define NXTSERVO\_REG\_S2\_SPEED 0x53**

NXTServo servo 2 speed register.

**8.1.2.1167 #define NXTSERVO\_REG\_S3\_POS 0x46**

NXTServo servo 3 position register.

**8.1.2.1168 #define NXTSERVO\_REG\_S3\_QPOS 0x5C**

NXTServo servo 3 quick position register. (write only)

**8.1.2.1169 #define NXTSERVO\_REG\_S3\_SPEED 0x54**

NXTServo servo 3 speed register.

**8.1.2.1170 #define NXTSERVO\_REG\_S4\_POS 0x48**

NXTServo servo 4 position register.

**8.1.2.1171 #define NXTSERVO\_REG\_S4\_QPOS 0x5D**

NXTServo servo 4 quick position register. (write only)

**8.1.2.1172 #define NXTSERVO\_REG\_S4\_SPEED 0x55**

NXTServo servo 4 speed register.

**8.1.2.1173 #define NXTSERVO\_REG\_S5\_POS 0x4A**

NXTServo servo 5 position register.

**8.1.2.1174 #define NXTSERVO\_REG\_S5\_QPOS 0x5E**

NXTServo servo 5 quick position register. (write only)

**8.1.2.1175 #define NXTSERVO\_REG\_S5\_SPEED 0x56**

NXTServo servo 5 speed register.

**8.1.2.1176 #define NXTSERVO\_REG\_S6\_POS 0x4C**

NXTServo servo 6 position register.

**8.1.2.1177 #define NXTSERVO\_REG\_S6\_QPOS 0x5F**

NXTServo servo 6 quick position register. (write only)

**8.1.2.1178 #define NXTSERVO\_REG\_S6\_SPEED 0x57**

NXTServo servo 6 speed register.

**8.1.2.1179 #define NXTSERVO\_REG\_S7\_POS 0x4E**

NXTServo servo 7 position register.

**8.1.2.1180 #define NXTSERVO\_REG\_S7\_QPOS 0x60**

NXTServo servo 7 quick position register. (write only)

**8.1.2.1181 #define NXTSERVO\_REG\_S7\_SPEED 0x58**

NXTServo servo 7 speed register.

**8.1.2.1182 #define NXTSERVO\_REG\_S8\_POS 0x50**

NXTServo servo 8 position register.

**8.1.2.1183 #define NXTSERVO\_REG\_S8\_QPOS 0x61**

NXTServo servo 8 quick position register. (write only)

**8.1.2.1184 #define NXTSERVO\_REG\_S8\_SPEED 0x59**

NXTServo servo 8 speed register.

**8.1.2.1185 #define NXTSERVO\_REG\_VOLTAGE 0x41**

Battery voltage register. (read only)

**8.1.2.1186** `#define NXTSERVO_SERVO_1 0`

NXTServo server number 1.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.1.2.1187** `#define NXTSERVO_SERVO_2 1`

NXTServo server number 2.

**8.1.2.1188** `#define NXTSERVO_SERVO_3 2`

NXTServo server number 3.

**8.1.2.1189** `#define NXTSERVO_SERVO_4 3`

NXTServo server number 4.

**8.1.2.1190** `#define NXTSERVO_SERVO_5 4`

NXTServo server number 5.

**8.1.2.1191** `#define NXTSERVO_SERVO_6 5`

NXTServo server number 6.

**8.1.2.1192** `#define NXTSERVO_SERVO_7 6`

NXTServo server number 7.

**8.1.2.1193** `#define NXTSERVO_SERVO_8 7`

NXTServo server number 8.

**8.1.2.1194** `#define OPARR_MAX 0x05`

Calculate the maximum value of the elements in the numeric input array

**Examples:**

[ex\\_ArrayOp.nxc](#).

**8.1.2.1195 #define OPARR\_MEAN 0x01**

Calculate the mean value for the elements in the numeric input array

**8.1.2.1196 #define OPARR\_MIN 0x04**

Calculate the minimum value of the elements in the numeric input array

**8.1.2.1197 #define OPARR\_SORT 0x06**

Sort the elements in the numeric input array

**8.1.2.1198 #define OPARR\_STD 0x03**

Calculate the standard deviation of the elements in the numeric input array

**8.1.2.1199 #define OPARR\_SUM 0x00**

Calculate the sum of the elements in the numeric input array

**8.1.2.1200 #define OPARR\_SUMSQR 0x02**

Calculate the sum of the squares of the elements in the numeric input array

**8.1.2.1201 #define OUT\_A 0x00**

Output port A

**Examples:**

[ex\\_coast.nxc](#), [ex\\_coastex.nxc](#), [ex\\_float.nxc](#), [ex\\_getoutput.nxc](#), [ex\\_motoractualspeed.nxc](#), [ex\\_motorblocktachocount.nxc](#), [ex\\_motormode.nxc](#), [ex\\_motoroutputoptions.nxc](#), [ex\\_motoroverload.nxc](#), [ex\\_motorpower.nxc](#), [ex\\_motorregdvalue.nxc](#), [ex\\_motorregivalue.nxc](#), [ex\\_motorregpvalue.nxc](#), [ex\\_motorregulation.nxc](#), [ex\\_motorrotationcount.nxc](#), [ex\\_motorrunstate.nxc](#), [ex\\_motortachocount.nxc](#), [ex\\_motortacholimit.nxc](#), [ex\\_motorturnratio.nxc](#), [ex\\_off.nxc](#), [ex\\_offex.nxc](#), [ex\\_onfwd.nxc](#), [ex\\_onfwdex.nxc](#), [ex\\_onfwdreg.nxc](#),

[ex\\_onfwdregex.nxc](#), [ex\\_onfwdregexpid.nxc](#), [ex\\_onfwdregpid.nxc](#), [ex\\_onrev.nxc](#), [ex\\_onrevex.nxc](#), [ex\\_onrevreg.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), [ex\\_onrevregpid.nxc](#), [ex\\_PosReg.nxc](#), [ex\\_RemoteResetMotorPosition.nxc](#), [ex\\_RemoteResetTachoCount.nxc](#), [ex\\_RemoteSetOutputState.nxc](#), [ex\\_rotatemotor.nxc](#), [ex\\_rotatemotorpid.nxc](#), and [ex\\_yield.nxc](#).

#### 8.1.2.1202 #define OUT\_AB 0x03

Output ports A and B

##### Examples:

[ex\\_onfwdsync.nxc](#), [ex\\_onfwdsyncex.nxc](#), [ex\\_onfwdsyncexpid.nxc](#), [ex\\_onfwdsyncpid.nxc](#), [ex\\_onrevsync.nxc](#), [ex\\_onrevsyncex.nxc](#), [ex\\_onrevsyncexpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_resetalltachocounts.nxc](#), [ex\\_resetblocktachocount.nxc](#), [ex\\_resetrotationcount.nxc](#), [ex\\_resettachocount.nxc](#), [ex\\_rotatemotorex.nxc](#), [ex\\_rotatemotorexp.nxc](#), and [ex\\_setoutput.nxc](#).

#### 8.1.2.1203 #define OUT\_ABC 0x06

Output ports A, B, and C

#### 8.1.2.1204 #define OUT\_AC 0x04

Output ports A and C

#### 8.1.2.1205 #define OUT\_B 0x01

Output port B

#### 8.1.2.1206 #define OUT\_BC 0x05

Output ports B and C

#### 8.1.2.1207 #define OUT\_C 0x02

Output port C

#### 8.1.2.1208 #define OUT\_MODE\_BRAKE 0x02

Uses electronic braking to outputs

**8.1.2.1209 #define OUT\_MODE\_COAST 0x00**

No power and no braking so motors rotate freely.

**8.1.2.1210 #define OUT\_MODE\_MOTORON 0x01**

Enables PWM power to the outputs given the power setting

**Examples:**

[ex\\_RemoteSetOutputState.nxc](#).

**8.1.2.1211 #define OUT\_MODE\_REGMETHOD 0xF0**

Mask for unimplemented regulation mode

**8.1.2.1212 #define OUT\_MODE\_REGULATED 0x04**

Enables active power regulation using the regulation mode value

**8.1.2.1213 #define OUT\_OPTION\_HOLDATLIMIT 0x10**

Option to have the firmware hold the motor when it reaches the tachometer limit

**8.1.2.1214 #define OUT\_OPTION\_RAMPDOWNTOLIMIT 0x20**

Option to have the firmware rampdown the motor power as it approaches the tachometer limit

**8.1.2.1215 #define OUT\_REGMODE\_IDLE 0**

No motor regulation.

**Examples:**

[ex\\_RemoteSetOutputState.nxc](#).

**8.1.2.1216 #define OUT\_REGMODE\_POS 4**

Regulate a motor's position.

**8.1.2.1217 #define OUT\_REGMODE\_SPEED 1**

Regulate a motor's speed (aka power).

**Examples:**

[ex\\_onfwdreg.nxc](#), [ex\\_onfwdregex.nxc](#), [ex\\_onfwdregexpid.nxc](#), [ex\\_onfwdregpid.nxc](#), [ex\\_onrevreg.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), and [ex\\_onrevregpid.nxc](#).

**8.1.2.1218 #define OUT\_REGMODE\_SYNC 2**

Synchronize the rotation of two motors.

**8.1.2.1219 #define OUT\_REGOPTION\_NO\_SATURATION 0x01**

Do not limit intermediary regulation results

**Examples:**

[ex\\_PosReg.nxc](#).

**8.1.2.1220 #define OUT\_RUNSTATE\_HOLD 0x60**

Set motor run state to hold at the current position.

**8.1.2.1221 #define OUT\_RUNSTATE\_IDLE 0x00**

Disable all power to motors.

**8.1.2.1222 #define OUT\_RUNSTATE\_RAMPDOWN 0x40**

Enable ramping down from a current power to a new (lower) power over a specified [TachoLimitField](#) goal.

**8.1.2.1223 #define OUT\_RUNSTATE\_RAMPU 0x10**

Enable ramping up from a current power to a new (higher) power over a specified [TachoLimitField](#) goal.

**8.1.2.1224 #define OUT\_RUNSTATE\_RUNNING 0x20**

Enable power to motors at the specified power level.

**Examples:**

[ex\\_RemoteSetOutputState.nxc](#).

**8.1.2.1225 #define OutputModeField 1**

Mode field. Contains a combination of the output mode constants. Read/write. The [OUT\\_MODE\\_MOTORON](#) bit must be set in order for power to be applied to the motors. Add [OUT\\_MODE\\_BRAKE](#) to enable electronic braking. Braking means that the output voltage is not allowed to float between active PWM pulses. It improves the accuracy of motor output but uses more battery power. To use motor regulation include [OUT\\_MODE\\_REGULATED](#) in the [OutputModeField](#) value. Use [UF\\_UPDATE\\_MODE](#) with [UpdateFlagsField](#) to commit changes to this field.

**8.1.2.1226 #define OutputModuleID 0x00020001**

The output module ID

**8.1.2.1227 #define OutputModuleName "Output.mod"**

The output module name

**8.1.2.1228 #define OutputOffsetActualSpeed(p) (((p)\*32)+21)**

R - Holds the current motor speed (1 byte) sbyte

**8.1.2.1229 #define OutputOffsetBlockTachoCount(p) (((p)\*32)+4)**

R - Holds current number of counts for the current output block (4 bytes) slong

**8.1.2.1230 #define OutputOffsetFlags(p) (((p)\*32)+18)**

RW - Holds flags for which data should be updated (1 byte) ubyte

**8.1.2.1231 #define OutputOffsetMaxAccel(p) (((p)\*32)+31)**

RW - holds the maximum acceleration for position regulation (1 byte) sbyte (NBC/NXC)

**8.1.2.1232 #define OutputOffsetMaxSpeed(p) (((p)\*32)+30)**

RW - holds the maximum speed for position regulation (1 byte) sbyte (NBC/NXC)

**8.1.2.1233 #define OutputOffsetMode(p) (((p)\*32)+19)**

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

**8.1.2.1234 #define OutputOffsetMotorRPM(p) (((p)\*32)+16)**

Not updated, will be removed later !! (2 bytes) sword

**8.1.2.1235 #define OutputOffsetOptions(p) (((p)\*32)+29)**

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

**8.1.2.1236 #define OutputOffsetOverloaded(p) (((p)\*32)+27)**

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

**8.1.2.1237 #define OutputOffsetRegDParameter(p) (((p)\*32)+24)**

RW - Holds the D-constant used in the regulation (1 byte) ubyte

**8.1.2.1238 #define OutputOffsetRegIParameter(p) (((p)\*32)+23)**

RW - Holds the I-constant used in the regulation (1 byte) ubyte

**8.1.2.1239 #define OutputOffsetRegMode(p) (((p)\*32)+26)**

RW - Tells which regulation mode should be used (1 byte) ubyte

**8.1.2.1240 #define OutputOffsetRegPParameter(p) (((p)\*32)+22)**

RW - Holds the P-constant used in the regulation (1 byte) ubyte

**8.1.2.1241 #define OutputOffsetRegulationOptions 97**

use for position regulation options (1 byte) ubyte (NBC/NXC)

**8.1.2.1242 #define OutputOffsetRegulationTime 96**

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

**8.1.2.1243 #define OutputOffsetRotationCount(p) (((p)\*32)+8)**

R - Holds current number of counts for the rotation counter to the output (4 bytes) slong

**8.1.2.1244 #define OutputOffsetRunState(p) (((p)\*32)+25)**

RW - Holds the current motor run state in the output module (1 byte) ubyte

**8.1.2.1245 #define OutputOffsetSpeed(p) (((p)\*32)+20)**

RW - Holds the wanted speed (1 byte) sbyte

**8.1.2.1246 #define OutputOffsetSyncTurnParameter(p) (((p)\*32)+28)**

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

**8.1.2.1247 #define OutputOffsetTachoCount(p) (((p)\*32)+0)**

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes) slong

**8.1.2.1248 #define OutputOffsetTachoLimit(p) (((p)\*32)+12)**

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

**8.1.2.1249 #define OutputOptionsField 15**

Options field. Contains a combination of the output options constants. Read/write. Set options for how the output module will act when a tachometer limit is reached. Option constants can be combined with bitwise OR. Use `OUT_OPTION_HOLDATLIMIT` to have the output module hold the motor when it reaches the tachometer limit. Use `OUT_OPTION_RAMPDOWNTOLIMIT` to have the output module ramp down the motor power as it approaches the tachometer limit.

**8.1.2.1250 #define OverloadField 9**

Overload field. Contains a boolean value which is TRUE if the motor is overloaded. Read only. This field will have a value of 1 (true) if the firmware speed regulation cannot overcome a physical load on the motor. In other words, the motor is turning more slowly than expected. If the motor speed can be maintained in spite of loading then this field value is zero (false). In order to use this field the motor must have a non-idle `RunStateField`, an `OutputModeField` which includes `OUT_MODE_MOTORON` and `OUT_MODE_REGULATED`, and its `RegModeField` must be set to `OUT_REGMODE_SPEED`.

**8.1.2.1251 #define PF\_CHANNEL\_1 0**

Power function channel 1

**Examples:**

`ex_HTPFComboDirect.nxc`, `ex_HTPFComboPWM.nxc`, `ex_HTPFSingleOutputCST.nxc`, `ex_HTPFSingleOutputPWM.nxc`, `ex_HTPFSinglePin.nxc`, `ex_HTPFTrain.nxc`, `ex_MSPFComboDirect.nxc`, `ex_MSPFComboPWM.nxc`, `ex_MSPFSingleOutputCST.nxc`, `ex_MSPFSingleOutputPWM.nxc`, `ex_MSPFSinglePin.nxc`, and `ex_MSPFTrain.nxc`.

**8.1.2.1252 #define PF\_CHANNEL\_2 1**

Power function channel 2

**8.1.2.1253 #define PF\_CHANNEL\_3 2**

Power function channel 3

**8.1.2.1254** `#define PF_CHANNEL_4 3`

Power function channel 4

**8.1.2.1255** `#define PF_CMD_BRAKE 3`

Power function command brake

**8.1.2.1256** `#define PF_CMD_FLOAT 0`

Power function command float (same as stop)

**8.1.2.1257** `#define PF_CMD_FWD 1`

Power function command forward

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_PFMate.nxc](#).

**8.1.2.1258** `#define PF_CMD_REV 2`

Power function command reverse

**Examples:**

[ex\\_PFMate.nxc](#).

**8.1.2.1259** `#define PF_CMD_STOP 0`

Power function command stop

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), and [ex\\_MSPFComboDirect.nxc](#).

**8.1.2.1260** `#define PF_CST_CLEAR1_CLEAR2 0`

Power function CST clear 1 and clear 2

**8.1.2.1261** `#define PF_CST_CLEAR1_SET2 2`

Power function CST clear 1 and set 2

**8.1.2.1262** `#define PF_CST_DECREMENT_PWM 5`

Power function CST decrement PWM

**8.1.2.1263** `#define PF_CST_FULL_FWD 6`

Power function CST full forward

**8.1.2.1264** `#define PF_CST_FULL_REV 7`

Power function CST full reverse

**8.1.2.1265** `#define PF_CST_INCREMENT_PWM 4`

Power function CST increment PWM

**8.1.2.1266** `#define PF_CST_SET1_CLEAR2 1`

Power function CST set 1 and clear 2

**8.1.2.1267** `#define PF_CST_SET1_SET2 3`

Power function CST set 1 and set 2

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), and [ex\\_MSPFSingleOutputCST.nxc](#).

**8.1.2.1268** `#define PF_CST_TOGGLE_DIR 8`

Power function CST toggle direction

**8.1.2.1269** `#define PF_FUNC_CLEAR 1`

Power function single pin - clear

**8.1.2.1270 #define PF\_FUNC\_NOCHANGE 0**

Power function single pin - no change

**8.1.2.1271 #define PF\_FUNC\_SET 2**

Power function single pin - set

**Examples:**[ex\\_HTPFSinglePin.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).**8.1.2.1272 #define PF\_FUNC\_TOGGLE 3**

Power function single pin - toggle

**8.1.2.1273 #define PF\_MODE\_COMBO\_DIRECT 1**

Power function mode combo direct

**8.1.2.1274 #define PF\_MODE\_COMBO\_PWM 4**

Power function mode combo pulse width modulation (PWM)

**8.1.2.1275 #define PF\_MODE\_SINGLE\_OUTPUT\_CST 6**

Power function mode single output clear, set, toggle (CST)

**8.1.2.1276 #define PF\_MODE\_SINGLE\_OUTPUT\_PWM 4**

Power function mode single output pulse width modulation (PWM)

**8.1.2.1277 #define PF\_MODE\_SINGLE\_PIN\_CONT 2**

Power function mode single pin continuous

**8.1.2.1278 #define PF\_MODE\_SINGLE\_PIN\_TIME 3**

Power function mode single pin timed

**8.1.2.1279** `#define PF_MODE_TRAIN 0`

Power function mode IR Train

**8.1.2.1280** `#define PF_OUT_A 0`

Power function output A

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#),  
[ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_-](#)  
[MSPFSingleOutputPWM.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).

**8.1.2.1281** `#define PF_OUT_B 1`

Power function output B

**8.1.2.1282** `#define PF_PIN_C1 0`

Power function pin C1

**Examples:**

[ex\\_HTPFSinglePin.nxc](#), and [ex\\_MSPFSinglePin.nxc](#).

**8.1.2.1283** `#define PF_PIN_C2 1`

Power function pin C2

**8.1.2.1284** `#define PF_PWM_BRAKE 8`

Power function PWM brake

**8.1.2.1285** `#define PF_PWM_FLOAT 0`

Power function PWM float

**8.1.2.1286** `#define PF_PWM_FWD1 1`

Power function PWM forward level 1

**8.1.2.1287 #define PF\_PWM\_FWD2 2**

Power function PWM forward level 2

**8.1.2.1288 #define PF\_PWM\_FWD3 3**

Power function PWM forward level 3

**8.1.2.1289 #define PF\_PWM\_FWD4 4**

Power function PWM forward level 4

**8.1.2.1290 #define PF\_PWM\_FWD5 5**

Power function PWM forward level 5

**Examples:**

[ex\\_HTPFComboPWM.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#), and [ex\\_MSPFSingleOutputPWM.nxc](#).

**8.1.2.1291 #define PF\_PWM\_FWD6 6**

Power function PWM forward level 6

**8.1.2.1292 #define PF\_PWM\_FWD7 7**

Power function PWM forward level 7

**8.1.2.1293 #define PF\_PWM\_REV1 15**

Power function PWM reverse level 1

**8.1.2.1294 #define PF\_PWM\_REV2 14**

Power function PWM reverse level 2

**8.1.2.1295 #define PF\_PWM\_REV3 13**

Power function PWM reverse level 3

**8.1.2.1296 #define PF\_PWM\_REV4 12**

Power function PWM reverse level 4

**Examples:**

[ex\\_HTPFComboPWM.nxc](#), and [ex\\_MSPFComboPWM.nxc](#).

**8.1.2.1297 #define PF\_PWM\_REV5 11**

Power function PWM reverse level 5

**8.1.2.1298 #define PF\_PWM\_REV6 10**

Power function PWM reverse level 6

**8.1.2.1299 #define PF\_PWM\_REV7 9**

Power function PWM reverse level 7

**8.1.2.1300 #define PFMATE\_CHANNEL\_1 1**

Power function channel 1

**Examples:**

[ex\\_PFMate.nxc](#).

**8.1.2.1301 #define PFMATE\_CHANNEL\_2 2**

Power function channel 2

**8.1.2.1302 #define PFMATE\_CHANNEL\_3 3**

Power function channel 3

**8.1.2.1303 #define PFMATE\_CHANNEL\_4 4**

Power function channel 4

**8.1.2.1304 #define PFMATE\_CMD\_GO 0x47**

Send IR signal to IR receiver

**8.1.2.1305 #define PFMATE\_CMD\_RAW 0x52**

Send raw IR signal to IR receiver

**8.1.2.1306 #define PFMATE\_MOTORS\_A 0x01**

Control only motor A

**8.1.2.1307 #define PFMATE\_MOTORS\_B 0x02**

Control only motor B

**8.1.2.1308 #define PFMATE\_MOTORS\_BOTH 0x00**

Control both motors

**Examples:**

[ex\\_PFMate.nxc.](#)

**8.1.2.1309 #define PFMATE\_REG\_A\_CMD 0x44**

PF command for motor A? (PF\_CMD\_FLOAT, PF\_CMD\_FWD, PF\_CMD\_REV, PF\_CMD\_BRAKE)

**8.1.2.1310 #define PFMATE\_REG\_A\_SPEED 0x45**

PF speed for motor A? (0-7)

**8.1.2.1311 #define PFMATE\_REG\_B\_CMD 0x46**

PF command for motor B? (PF\_CMD\_FLOAT, PF\_CMD\_FWD, PF\_CMD\_REV, PF\_CMD\_BRAKE)

**8.1.2.1312 #define PFMATE\_REG\_B\_SPEED 0x47**

PF speed for motor B? (0-7)

**8.1.2.1313 #define PFMATE\_REG\_CHANNEL 0x42**

PF channel? 1, 2, 3, or 4

**8.1.2.1314 #define PFMATE\_REG\_CMD 0x41**

PFMate command

**8.1.2.1315 #define PFMATE\_REG\_MOTORS 0x43**

PF motors? (0 = both, 1 = A, 2 = B)

**8.1.2.1316 #define PI 3.141593**

A constant for PI

**Examples:**

[ex\\_dispfnout.nxc](#), and [ex\\_string.nxc](#).

**8.1.2.1317 #define PID\_0 0**

PID zero

**8.1.2.1318 #define PID\_1 32**

PID one

**8.1.2.1319 #define PID\_2 64**

PID two

**8.1.2.1320 #define PID\_3 96**

PID three

**8.1.2.1321 #define PID\_4 128**

PID four

**8.1.2.1322 #define PID\_5 160**

PID five

**8.1.2.1323 #define PID\_6 192**

PID six

**8.1.2.1324 #define PID\_7 224**

PID seven

**8.1.2.1325 #define POOL\_MAX\_SIZE 32768**

Maximum size of memory pool, in bytes

**8.1.2.1326 #define PowerField 2**

Power field. Contains the desired power level (-100 to 100). Read/write. Specify the power level of the output. The absolute value of PowerField is a percentage of the full power of the motor. The sign of PowerField controls the rotation direction. Positive values tell the firmware to turn the motor forward, while negative values turn the motor backward. Use [UF\\_UPDATE\\_SPEED](#) with [UpdateFlagsField](#) to commit changes to this field.

**8.1.2.1327 #define PROG\_ABORT 4**

Program has been aborted

**8.1.2.1328 #define PROG\_ERROR 3**

A program error has occurred

**8.1.2.1329 #define PROG\_IDLE 0**

Program state is idle

**8.1.2.1330 #define PROG\_OK 1**

Program state is okay

**8.1.2.1331 #define PROG\_RESET 5**

Program has been reset

**8.1.2.1332 #define PROG\_RUNNING 2**

Program is running

**8.1.2.1333 #define PSP\_BTNSET1\_DOWN 0x40**

The PSP-Nx button set 1 down arrow

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1334 #define PSP\_BTNSET1\_L3 0x02**

The PSP-Nx button set 1 L3

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1335 #define PSP\_BTNSET1\_LEFT 0x80**

The PSP-Nx button set 1 left arrow

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1336 #define PSP\_BTNSET1\_R3 0x04**

The PSP-Nx button set 1 R3

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)

**8.1.2.1337 #define PSP\_BTNSET1\_RIGHT 0x20**

The PSP-Nx button set 1 right arrow

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1338 #define PSP\_BTNSET1\_SELECT 0x01**

The PSP-Nx button set 1 select

**8.1.2.1339 #define PSP\_BTNSET1\_START 0x08**

The PSP-Nx button set 1 start

**8.1.2.1340 #define PSP\_BTNSET1\_UP 0x10**

The PSP-Nx button set 1 up arrow

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1341 #define PSP\_BTNSET2\_CIRCLE 0x20**

The PSP-Nx button set 2 circle

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1342 #define PSP\_BTNSET2\_CROSS 0x40**

The PSP-Nx button set 2 cross

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1343 #define PSP\_BTNSET2\_L1 0x04**

The PSP-Nx button set 2 L1

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1344 #define PSP\_BTNSET2\_L2 0x01**

The PSP-Nx button set 2 L2

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1345 #define PSP\_BTNSET2\_R1 0x08**

The PSP-Nx button set 2 R1

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1346 #define PSP\_BTNSET2\_R2 0x02**

The PSP-Nx button set 2 R2

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)**8.1.2.1347 #define PSP\_BTNSET2\_SQUARE 0x80**

The PSP-Nx button set 2 square

**Examples:**[ex\\_ReadSensorMSPlayStation.nxc.](#)

**8.1.2.1348** `#define PSP_BTNSET2_TRIANGLE 0x10`

The PSP-Nx button set 2 triangle

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.1.2.1349** `#define PSP_CMD_ANALOG 0x73`

Set the PSP-Nx to analog mode

**8.1.2.1350** `#define PSP_CMD_DIGITAL 0x41`

Set the PSP-Nx to digital mode

**8.1.2.1351** `#define PSP_REG_BTNSET1 0x42`

The PSP-Nx button set 1 register

**8.1.2.1352** `#define PSP_REG_BTNSET2 0x43`

The PSP-Nx button set 2 register

**8.1.2.1353** `#define PSP_REG_XLEFT 0x44`

The PSP-Nx X left register

**8.1.2.1354** `#define PSP_REG_XRIGHT 0x46`

The PSP-Nx X right register

**8.1.2.1355** `#define PSP_REG_YLEFT 0x45`

The PSP-Nx Y left register

**8.1.2.1356** `#define PSP_REG_YRIGHT 0x47`

The PSP-Nx Y right register

**8.1.2.1357 #define RADIANS\_PER\_DEGREE PI/180**

Used for converting from degrees to radians

**Examples:**

[ex\\_sin\\_cos.nxc](#).

**8.1.2.1358 #define RAND\_MAX 2147483646**

The maximum long random number returned by rand

**8.1.2.1359 #define RandomEx 99**

Generate a random number or seed the RNG.

**8.1.2.1360 #define RandomNumber 24**

Generate a random number

**8.1.2.1361 #define RawValueField 2**

Raw value field. Contains the current raw analog sensor value. Read only.

**8.1.2.1362 #define RC\_PROP\_BTONOFF 0x0**

Set/get whether bluetooth is on or off

**8.1.2.1363 #define RC\_PROP\_DEBUGGING 0xF**

Set/get enhanced firmware debugging information

**8.1.2.1364 #define RC\_PROP\_SLEEP\_TIMEOUT 0x2**

Set/get the NXT sleep timeout value (times 60000)

**8.1.2.1365 #define RC\_PROP\_SOUND\_LEVEL 0x1**

Set/get the NXT sound level

**Examples:**

[ex\\_RemoteGetProperty.nxc](#), and [ex\\_RemoteSetProperty.nxc](#).

|                   |  |                              |
|-------------------|--|------------------------------|
| <b>8.1.2.1366</b> | <b>#define RCX_AbsVarOp 0x74</b>         | Absolute value function      |
| <b>8.1.2.1367</b> | <b>#define RCX_AndVarOp 0x84</b>         | AND function                 |
| <b>8.1.2.1368</b> | <b>#define RCX_AutoOffOp 0xb1</b>        | Set auto off timer           |
| <b>8.1.2.1369</b> | <b>#define RCX_BatteryLevelOp 0x30</b>   | Read the battery level       |
| <b>8.1.2.1370</b> | <b>#define RCX_BatteryLevelSrc 34</b>    | The RCX battery level source |
| <b>8.1.2.1371</b> | <b>#define RCX_BootModeOp 0x65</b>       | Set into book mode           |
| <b>8.1.2.1372</b> | <b>#define RCX_CalibrateEventOp 0x04</b> | Calibrate event              |
| <b>8.1.2.1373</b> | <b>#define RCX_ClearAllEventsOp 0x06</b> | Clear all events             |
| <b>8.1.2.1374</b> | <b>#define RCX_ClearCounterOp 0xb7</b>   | Clear a counter              |

**8.1.2.1375 #define RCX\_ClearMsgOp 0x90**

Clear message

**8.1.2.1376 #define RCX\_ClearSensorOp 0xd1**

Clear a sensor

**8.1.2.1377 #define RCX\_ClearSoundOp 0x80**

Clear sound

**8.1.2.1378 #define RCX\_ClearTimerOp 0xa1**

Clear a timer

**8.1.2.1379 #define RCX\_ClickCounterSrc 27**

The RCX event click counter source

**8.1.2.1380 #define RCX\_ConstantSrc 2**

The RCX constant value source

**Examples:**

[ex\\_HTRCXEvent.nxc](#), [ex\\_HTRCXSetEvent.nxc](#), [ex\\_HTRCXSetMaxPower.nxc](#),  
[ex\\_HTRCXSetPower.nxc](#), [ex\\_HTSscoutSendVLL.nxc](#), [ex\\_-  
HTScoutSetEventFeedback.nxc](#), [ex\\_HTSscoutSetSensorClickTime.nxc](#),  
[ex\\_HTSscoutSetSensorHysteresis.nxc](#), [ex\\_MSRCXAndVar.nxc](#), [ex\\_-  
MSRCXDivVar.nxc](#), [ex\\_MSRCXEvent.nxc](#), [ex\\_MSRCXOrVar.nxc](#), [ex\\_-  
MSRCXSetEvent.nxc](#), [ex\\_MSRCXSetMaxPower.nxc](#), [ex\\_MSRCXSetPower.nxc](#),  
[ex\\_MSScoutSendVLL.nxc](#), [ex\\_MSScoutSetCounterLimit.nxc](#), [ex\\_-  
MSScoutSetEventFeedback.nxc](#), [ex\\_MSScoutSetSensorClickTime.nxc](#), [ex\\_-  
MSScoutSetSensorHysteresis.nxc](#), and [ex\\_MSScoutSetTimerLimit.nxc](#).

**8.1.2.1381 #define RCX\_CounterSrc 21**

The RCX counter source

**8.1.2.1382 #define RCX\_DatalogOp 0x62**

Datalog the specified source/value

**8.1.2.1383 #define RCX\_DatalogRawDirectSrc 42**

The RCX direct datalog raw source

**8.1.2.1384 #define RCX\_DatalogRawIndirectSrc 41**

The RCX indirect datalog raw source

**8.1.2.1385 #define RCX\_DatalogSrcDirectSrc 38**

The RCX direct datalog source source

**8.1.2.1386 #define RCX\_DatalogSrcIndirectSrc 37**

The RCX indirect datalog source source

**8.1.2.1387 #define RCX\_DatalogValueDirectSrc 40**

The RCX direct datalog value source

**8.1.2.1388 #define RCX\_DatalogValueIndirectSrc 39**

The RCX indirect datalog value source

**8.1.2.1389 #define RCX\_DecCounterOp 0xa7**

Decrement a counter

**8.1.2.1390 #define RCX\_DeleteSubOp 0xc1**

Delete a subroutine

**8.1.2.1391 #define RCX\_DeleteSubsOp 0x70**

Delete subroutines

- 
- 8.1.2.1392 **#define RCX\_DeleteTaskOp 0x61**  
Delete a task
- 8.1.2.1393 **#define RCX\_DeleteTasksOp 0x40**  
Delete tasks
- 8.1.2.1394 **#define RCX\_DirectEventOp 0x03**  
Fire an event
- 8.1.2.1395 **#define RCX\_DisplayOp 0x33**  
Set LCD display value
- 8.1.2.1396 **#define RCX\_DivVarOp 0x44**  
Divide function
- 8.1.2.1397 **#define RCX\_DurationSrc 31**  
The RCX event duration source
- 8.1.2.1398 **#define RCX\_EventStateSrc 25**  
The RCX event static source
- 8.1.2.1399 **#define RCX\_FirmwareVersionSrc 35**  
The RCX firmware version source
- 8.1.2.1400 **#define RCX\_GlobalMotorStatusSrc 17**  
The RCX global motor status source
- 8.1.2.1401 **#define RCX\_GOutputDirOp 0x77**  
Set global motor direction

**8.1.2.1402 #define RCX\_GOutputModeOp 0x67**

Set global motor mode

**8.1.2.1403 #define RCX\_GOutputPowerOp 0xa3**

Set global motor power levels

**8.1.2.1404 #define RCX\_HysteresisSrc 30**

The RCX event hysteresis source

**8.1.2.1405 #define RCX\_IncCounterOp 0x97**

Increment a counter

**8.1.2.1406 #define RCX\_IndirectVarSrc 36**

The RCX indirect variable source

**8.1.2.1407 #define RCX\_InputBooleanSrc 13**

The RCX input boolean source

**8.1.2.1408 #define RCX\_InputModeOp 0x42**

Set the input mode

**8.1.2.1409 #define RCX\_InputModeSrc 11**

The RCX input mode source

**8.1.2.1410 #define RCX\_InputRawSrc 12**

The RCX input raw source

**8.1.2.1411 #define RCX\_InputTypeOp 0x32**

Set the input type

**8.1.2.1412 #define RCX\_InputTypeSrc 10**

The RCX input type source

**8.1.2.1413 #define RCX\_InputValueSrc 9**

The RCX input value source

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#), [ex\\_MSRCXAddToDatalog.nxc](#), and [ex\\_MSRCXSumVar.nxc](#).

**8.1.2.1414 #define RCX\_IRModeOp 0x31**

Set the IR transmit mode

**8.1.2.1415 #define RCX\_LightOp 0x87**

Light opcode

**8.1.2.1416 #define RCX\_LowerThresholdSrc 29**

The RCX event lower threshold source

**8.1.2.1417 #define RCX\_LSblinkTimeOp 0xe3**

Set the light sensor blink time

**8.1.2.1418 #define RCX\_LSCalibrateOp 0xc0**

Calibrate the light sensor

**8.1.2.1419 #define RCX\_LSHysteresisOp 0xd3**

Set the light sensor hysteresis

**8.1.2.1420 #define RCX\_LSLowerThreshOp 0xc3**

Set the light sensor lower threshold

**8.1.2.1421 #define RCX\_LSUpperThreshOp 0xb3**

Set the light sensor upper threshold

**8.1.2.1422 #define RCX\_MessageOp 0xf7**

Set message

**8.1.2.1423 #define RCX\_MessageSrc 15**

The RCX message source

**8.1.2.1424 #define RCX\_MulVarOp 0x54**

Multiply function

**8.1.2.1425 #define RCX\_MuteSoundOp 0xd0**

Mute sound

**8.1.2.1426 #define RCX\_OnOffFloatOp 0x21**

Control motor state - on, off, float

**8.1.2.1427 #define RCX\_OrVarOp 0x94**

OR function

**8.1.2.1428 #define RCX\_OUT\_A 0x01**

RCX Output A

**Examples:**

```
ex_HTRCXDisableOutput.nxc,    ex_HTRCXEnableOutput.nxc,    ex_-  
HTRCXFloat.nxc,              ex_HTRCXFwd.nxc,            ex_HTRCXInvertOutput.nxc,  
ex_HTRCXObvertOutput.nxc,    ex_HTRCXOff.nxc,           ex_-  
HTRCXOn.nxc,                 ex_HTRCXOnFor.nxc,        ex_HTRCXOnFwd.nxc,    ex_-  
HTRCXOnRev.nxc,             ex_HTRCXRev.nxc,          ex_HTRCXSetDirection.nxc,  
ex_HTRCXSetGlobalDirection.nxc,  ex_HTRCXSetGlobalOutput.nxc,
```

[ex\\_HTRCXSetMaxPower.nxc](#), [ex\\_HTRCXSetOutput.nxc](#), [ex\\_HTRCXSetPower.nxc](#), [ex\\_HTRCXToggle.nxc](#), [ex\\_MSRCXDisableOutput.nxc](#), [ex\\_MSRCXEnableOutput.nxc](#), [ex\\_MSRCXFloat.nxc](#), [ex\\_MSRCXFwd.nxc](#), [ex\\_MSRCXInvertOutput.nxc](#), [ex\\_MSRCXObvertOutput.nxc](#), [ex\\_MSRCXOff.nxc](#), [ex\\_MSRCXOn.nxc](#), [ex\\_MSRCXOnFor.nxc](#), [ex\\_MSRCXOnFwd.nxc](#), [ex\\_MSRCXOnRev.nxc](#), [ex\\_MSRCXRev.nxc](#), [ex\\_MSRCXSetDirection.nxc](#), [ex\\_MSRCXSetGlobalDirection.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), [ex\\_MSRCXSetMaxPower.nxc](#), [ex\\_MSRCXSetOutput.nxc](#), [ex\\_MSRCXSetPower.nxc](#), and [ex\\_MSRCXToggle.nxc](#).

**8.1.2.1429 #define RCX\_OUT\_AB 0x03**

RCX Outputs A and B

**8.1.2.1430 #define RCX\_OUT\_ABC 0x07**

RCX Outputs A, B, and C

**8.1.2.1431 #define RCX\_OUT\_AC 0x05**

RCX Outputs A and C

**8.1.2.1432 #define RCX\_OUT\_B 0x02**

RCX Output B

**8.1.2.1433 #define RCX\_OUT\_BC 0x06**

RCX Outputs B and C

**8.1.2.1434 #define RCX\_OUT\_C 0x04**

RCX Output C

**8.1.2.1435 #define RCX\_OUT\_FLOAT 0**

Set RCX output to float

**8.1.2.1436 #define RCX\_OUT\_FULL 7**

Set RCX output power level to full

**Examples:**[ex\\_HTRCXSetPower.nxc](#), and [ex\\_MSRCXSetPower.nxc](#).**8.1.2.1437 #define RCX\_OUT\_FWD 0x80**

Set RCX output direction to forward

**Examples:**[ex\\_HTRCXSetDirection.nxc](#), [ex\\_HTRCXSetGlobalDirection.nxc](#), [ex\\_MSRCXSetDirection.nxc](#), and [ex\\_MSRCXSetGlobalDirection.nxc](#).**8.1.2.1438 #define RCX\_OUT\_HALF 3**

Set RCX output power level to half

**8.1.2.1439 #define RCX\_OUT\_LOW 0**

Set RCX output power level to low

**8.1.2.1440 #define RCX\_OUT\_OFF 0x40**

Set RCX output to off

**8.1.2.1441 #define RCX\_OUT\_ON 0x80**

Set RCX output to on

**Examples:**[ex\\_HTRCXSetGlobalOutput.nxc](#), [ex\\_HTRCXSetOutput.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), and [ex\\_MSRCXSetOutput.nxc](#).**8.1.2.1442 #define RCX\_OUT\_REV 0**

Set RCX output direction to reverse

**8.1.2.1443 #define RCX\_OUT\_TOGGLE 0x40**

Set RCX output direction to toggle

**8.1.2.1444 #define RCX\_OutputDirOp 0xe1**

Set the motor direction

**8.1.2.1445 #define RCX\_OutputPowerOp 0x13**

Set the motor power level

**8.1.2.1446 #define RCX\_OutputStatusSrc 3**

The RCX output status source

**8.1.2.1447 #define RCX\_PBTurnOffOp 0x60**

Turn off the brick

**8.1.2.1448 #define RCX\_PingOp 0x10**

Ping the brick

**8.1.2.1449 #define RCX\_PlaySoundOp 0x51**

Play a sound

**8.1.2.1450 #define RCX\_PlayToneOp 0x23**

Play a tone

**8.1.2.1451 #define RCX\_PlayToneVarOp 0x02**

Play a tone using a variable

**8.1.2.1452 #define RCX\_PollMemoryOp 0x63**

Poll a memory location

**8.1.2.1453 #define RCX\_PollOp 0x12**

Poll a source/value combination

**8.1.2.1454 #define RCX\_ProgramSlotSrc 8**

The RCX program slot source

**8.1.2.1455 #define RCX\_RandomSrc 4**

The RCX random number source

**Examples:**

[ex\\_MSRCXSet.nxc](#), and [ex\\_MSRCXSubVar.nxc](#).

**8.1.2.1456 #define RCX\_RemoteKeysReleased 0x0000**

All remote keys have been released

**8.1.2.1457 #define RCX\_RemoteOp 0xd2**

Execute simulated remote control buttons

**8.1.2.1458 #define RCX\_RemoteOutABackward 0x4000**

Set output A backward

**8.1.2.1459 #define RCX\_RemoteOutAForward 0x0800**

Set output A forward

**8.1.2.1460 #define RCX\_RemoteOutBBackward 0x8000**

Set output B backward

**8.1.2.1461 #define RCX\_RemoteOutBForward 0x1000**

Set output B forward

**8.1.2.1462 #define RCX\_RemoteOutCBackward 0x0001**

Set output C backward

**8.1.2.1463 #define RCX\_RemoteOutCForward 0x2000**

Set output C forward

**8.1.2.1464 #define RCX\_RemotePBMessage1 0x0100**

Send PB message 1

**8.1.2.1465 #define RCX\_RemotePBMessage2 0x0200**

Send PB message 2

**8.1.2.1466 #define RCX\_RemotePBMessage3 0x0400**

Send PB message 3

**8.1.2.1467 #define RCX\_RemotePlayASound 0x0080**

Play a sound

**Examples:**

[ex\\_HTRCXRemote.nxc](#), and [ex\\_MSRCXRemote.nxc](#).

**8.1.2.1468 #define RCX\_RemoteSelProgram1 0x0002**

Select program 1

**8.1.2.1469 #define RCX\_RemoteSelProgram2 0x0004**

Select program 2

**8.1.2.1470 #define RCX\_RemoteSelProgram3 0x0008**

Select program 3

- 
- 8.1.2.1471** `#define RCX_RemoteSelProgram4 0x0010`  
Select program 4
- 8.1.2.1472** `#define RCX_RemoteSelProgram5 0x0020`  
Select program 5
- 8.1.2.1473** `#define RCX_RemoteStopOutOff 0x0040`  
Stop and turn off outputs
- 8.1.2.1474** `#define RCX_ScoutCounterLimitSrc 22`  
The Scout counter limit source
- 8.1.2.1475** `#define RCX_ScoutEventFBSrc 24`  
The Scout event feedback source
- 8.1.2.1476** `#define RCX_ScoutLightParamsSrc 19`  
The Scout light parameters source
- 8.1.2.1477** `#define RCX_ScoutOp 0x47`  
Scout opcode
- 8.1.2.1478** `#define RCX_ScoutRulesOp 0xd5`  
Set Scout rules
- 8.1.2.1479** `#define RCX_ScoutRulesSrc 18`  
The Scout rules source
- 8.1.2.1480** `#define RCX_ScoutTimerLimitSrc 20`  
The Scout timer limit source

**8.1.2.1481 #define RCX\_SelectProgramOp 0x91**

Select a program slot

**8.1.2.1482 #define RCX\_SendUARTDataOp 0xc2**

Send data via IR using UART settings

**8.1.2.1483 #define RCX\_SetCounterOp 0xd4**

Set counter value

**8.1.2.1484 #define RCX\_SetDatalogOp 0x52**

Set the datalog size

**8.1.2.1485 #define RCX\_SetEventOp 0x93**

Set an event

**8.1.2.1486 #define RCX\_SetFeedbackOp 0x83**

Set Scout feedback

**8.1.2.1487 #define RCX\_SetPriorityOp 0xd7**

Set task priority

**8.1.2.1488 #define RCX\_SetSourceValueOp 0x05**

Set a source/value

**8.1.2.1489 #define RCX\_SetTimerLimitOp 0xc4**

Set timer limit

**8.1.2.1490 #define RCX\_SetVarOp 0x14**

Set function

- 
- 8.1.2.1491** `#define RCX_SetWatchOp 0x22`  
Set the watch source/value
- 8.1.2.1492** `#define RCX_SgnVarOp 0x64`  
Sign function
- 8.1.2.1493** `#define RCX_SoundOp 0x57`  
Sound opcode
- 8.1.2.1494** `#define RCX_StartTaskOp 0x71`  
Start a task
- 8.1.2.1495** `#define RCX_StopAllTasksOp 0x50`  
Stop all tasks
- 8.1.2.1496** `#define RCX_StopTaskOp 0x81`  
Stop a task
- 8.1.2.1497** `#define RCX_SubVarOp 0x34`  
Subtract function
- 8.1.2.1498** `#define RCX_SumVarOp 0x24`  
Sum function
- 8.1.2.1499** `#define RCX_TaskEventsSrc 23`  
The RCX task events source
- 8.1.2.1500** `#define RCX_TenMSTimerSrc 26`  
The RCX 10ms timer source

**8.1.2.1501 #define RCX\_TimerSrc 1**

The RCX timer source

**8.1.2.1502 #define RCX\_UARTSetupSrc 33**

The RCX UART setup source

**8.1.2.1503 #define RCX\_UnlockFirmOp 0xa5**

Unlock the firmware

**8.1.2.1504 #define RCX\_UnlockOp 0x15**

Unlock the brick

**8.1.2.1505 #define RCX\_UnmuteSoundOp 0xe0**

Unmute sound

**8.1.2.1506 #define RCX\_UploadDatalogOp 0xa4**

Upload datalog contents

**8.1.2.1507 #define RCX\_UpperThresholdSrc 28**

The RCX event upper threshold source

**8.1.2.1508 #define RCX\_VariableSrc 0**

The RCX variable source

**Examples:**

```
ex_HTRCXPoll.nxc,          ex_HTRCXSelectDisplay.nxc,          ex_-
HTScoutSetSensorLowerLimit.nxc,    ex-HTScoutSetSensorUpperLimit.nxc,
ex_MSRCXAbsVar.nxc,    ex_MSRCXMulVar.nxc,    ex_MSRCXPoll.nxc,
ex_MSRCXSelectDisplay.nxc,    ex_MSRCXSet.nxc,    ex_-
MSRCXSetUserDisplay.nxc,    ex_MSRCXSetVar.nxc,    ex_-
MSRCXSgnVar.nxc,    ex_MSScoutSetSensorLowerLimit.nxc,    and    ex_-
MSScoutSetSensorUpperLimit.nxc.
```

**8.1.2.1509 #define RCX\_ViewSourceValOp 0xe5**

View a source/value

**8.1.2.1510 #define RCX\_VLLOp 0xe2**

Send visual light link (VLL) data

**8.1.2.1511 #define RCX\_WatchSrc 14**

The RCX watch source

**8.1.2.1512 #define ReadButton 20**

Read the current button state

**8.1.2.1513 #define ReadLastResponse 97**

Read the last response packet received by the NXT. Optionally clear the value after reading it.

**8.1.2.1514 #define ReadSemData 40**

Read motor semaphore data

**8.1.2.1515 #define RegDValueField 12**

Derivative field. Contains the derivative constant for the PID motor controller. Read/write. This field specifies the derivative term used in the internal proportional-integral-derivative (PID) control algorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

**8.1.2.1516 #define RegIValueField 11**

Integral field. Contains the integral constant for the PID motor controller. Read/write. This field specifies the integral term used in the internal proportional-integral-derivative (PID) control algorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

**8.1.2.1517 #define RegModeField 8**

Regulation mode field. Contains one of the regulation mode constants. Read/write. This field specifies the regulation mode to use with the specified port(s). It is ignored if the `OUT_MODE_REGULATED` bit is not set in the `OutputModeField` field. Unlike `OutputModeField`, `RegModeField` is not a bitfield. Only one regulation mode value can be set at a time. Speed regulation means that the firmware tries to maintain a certain speed based on the `PowerField` setting. The firmware adjusts the PWM duty cycle if the motor is affected by a physical load. This adjustment is reflected by the value of the `ActualSpeedField` property. When using speed regulation, do not set `PowerField` to its maximum value since the firmware cannot adjust to higher power levels in that situation. Synchronization means the firmware tries to keep two motors in sync regardless of physical loads. Use this mode to maintain a straight path for a mobile robot automatically. Also use this mode with the `TurnRatioField` property to provide proportional turning. Set `OUT_REGMODE_SYNC` on at least two motor ports in order for synchronization to function. Setting `OUT_REGMODE_SYNC` on all three motor ports will result in only the first two (`OUT_A` and `OUT_B`) being synchronized.

**8.1.2.1518 #define RegPValueField 10**

Proportional field. Contains the proportional constant for the PID motor controller. Read/write. This field specifies the proportional term used in the internal proportional-integral-derivative (PID) control algorithm. Set `UF_UPDATE_PID_VALUES` to commit changes to `RegPValue`, `RegIValue`, and `RegDValue` simultaneously.

**8.1.2.1519 #define RESET\_ALL 0x68**

Reset all three tachometer counters

**8.1.2.1520 #define RESET\_BLOCK\_COUNT 0x20**

Reset the NXT-G block tachometer counter

**8.1.2.1521 #define RESET\_BLOCKANDTACHO 0x28**

Reset both the internal counter and the NXT-G block counter

**8.1.2.1522 #define RESET\_COUNT 0x08**

Reset the internal tachometer counter

**8.1.2.1523 #define RESET\_NONE 0x00**

No counters will be reset

**Examples:**

[ex\\_coastex.nxc](#), [ex\\_offex.nxc](#), [ex\\_onfwdex.nxc](#), [ex\\_onfwdregex.nxc](#), [ex\\_onfwdregexpid.nxc](#), [ex\\_onfwsyncex.nxc](#), [ex\\_onfwsyncexpid.nxc](#), [ex\\_onrevex.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), [ex\\_onrevsyncex.nxc](#), and [ex\\_onrevsyncexpid.nxc](#).

**8.1.2.1524 #define RESET\_ROTATION\_COUNT 0x40**

Reset the rotation counter

**8.1.2.1525 #define RFID\_MODE\_CONTINUOUS 2**

Configure the RFID device for continuous reading

**Examples:**

[ex\\_RFIDMode.nxc](#).

**8.1.2.1526 #define RFID\_MODE\_SINGLE 1**

Configure the RFID device for a single reading

**8.1.2.1527 #define RFID\_MODE\_STOP 0**

Stop the RFID device

**8.1.2.1528 #define RICArg(\_arg) ((\_arg)|0x1000)**

Output an RIC parameterized argument.

**Parameters:**

*\_arg* The argument that you want to parameterize.

**Examples:**

[ex\\_dispgaoutex.nxc](#).

```
8.1.2.1529 #define RICImgPoint(_X, _Y) (_X)&0xFF, (_X)>>8, (_Y)&0xFF,
           (_Y)>>8
```

Output an RIC ImgPoint structure.

**Parameters:**

*\_X* The X coordinate.

*\_Y* The Y coordinate.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```
8.1.2.1530 #define RICImgRect(_Pt, _W, _H) _Pt, (_W)&0xFF, (_W)>>8,
           (_H)&0xFF, (_H)>>8
```

Output an RIC ImgRect structure.

**Parameters:**

*\_Pt* An ImgPoint. See [RICImgPoint](#).

*\_W* The rectangle width.

*\_H* The rectangle height.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**8.1.2.1531** `#define RICMapArg(_mapidx, _arg) ((_arg)|0x1000|((( _mapidx)&0xF)<<8))`

Output an RIC parameterized and mapped argument.

**Parameters:**

*\_mapidx* The varmap data address.

*\_arg* The parameterized argument you want to pass through a varmap.

**8.1.2.1532** `#define RICMapElement(_Domain, _Range) (_Domain)&0xFF, (_Domain)>>8, (_Range)&0xFF, (_Range)>>8`

Output an RIC map element.

**Parameters:**

*\_Domain* The map element domain.

*\_Range* The map element range.

**8.1.2.1533** `#define RICMapFunction(_MapElement, ...) _MapElement, __VA_ARGS__`

Output an RIC VarMap function.

**Parameters:**

*\_MapElement* An entry in the varmap function. At least 2 elements are required. See [RICMapElement](#).

**8.1.2.1534** `#define RICOpCircle(_CopyOptions, _Point, _Radius) 10, 0, 7, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_Radius)&0xFF, (_Radius)>>8`

Output an RIC Circle opcode.

**Parameters:**

*\_CopyOptions* Circle copy options. See [Drawing option constants](#).

*\_Point* The circle's center point. See [RICImgPoint](#).

*\_Radius* The circle's radius.

```
8.1.2.1535 #define RICOpCopyBits(_CopyOptions, _DataAddr, _SrcRect,
    _DstPoint) 18, 0, 3, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8,
    (_DataAddr)&0xFF, (_DataAddr)>>8, _SrcRect, _DstPoint
```

Output an RIC CopyBits opcode.

**Parameters:**

*\_CopyOptions* CopyBits copy options. See [Drawing option constants](#).

*\_DataAddr* The address of the sprite from which to copy data.

*\_SrcRect* The rectangular portion of the sprite to copy. See [RICImgRect](#).

*\_DstPoint* The LCD coordinate to which to copy the data. See [RICImgPoint](#).

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```
8.1.2.1536 #define RICOpDescription(_Options, _Width, _Height) 8, 0, 0, 0,
    (_Options)&0xFF, (_Options)>>8, (_Width)&0xFF, (_Width)>>8,
    (_Height)&0xFF, (_Height)>>8
```

Output an RIC Description opcode.

**Parameters:**

*\_Options* RIC options.

*\_Width* The total RIC width.

*\_Height* The total RIC height.

**Examples:**

[ex\\_dispgaoutex.nxc](#).

```
8.1.2.1537 #define RICOpEllipse(_CopyOptions, _Point, _RadiusX,  
    _RadiusY) 12, 0, 9, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8,  
    _Point, (_RadiusX)&0xFF, (_RadiusX)>>8, (_RadiusY)&0xFF,  
    (_RadiusY)>>8
```

Output an RIC Ellipse opcode.

**Parameters:**

*\_CopyOptions* Ellipse copy options. See [Drawing option constants](#).

*\_Point* The center of the ellipse. See [RICImgPoint](#).

*\_RadiusX* The x-axis radius of the ellipse.

*\_RadiusY* The y-axis radius of the ellipse.

```
8.1.2.1538 #define RICOpLine(_CopyOptions, _Point1, _Point2) 12, 0, 5, 0,  
    (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point1, _Point2
```

Output an RIC Line opcode.

**Parameters:**

*\_CopyOptions* Line copy options. See [Drawing option constants](#).

*\_Point1* The starting point of the line. See [RICImgPoint](#).

*\_Point2* The ending point of the line. See [RICImgPoint](#).

```
8.1.2.1539 #define RICOpNumBox(_CopyOptions, _Point, _Value) 10,  
    0, 8, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,  
    (_Value)&0xFF, (_Value)>>8
```

Output an RIC NumBox opcode.

**Parameters:**

*\_CopyOptions* NumBox copy options. See [DrawingOption constants](#).

*\_Point* The numbox bottom left corner. See [RICImgPoint](#).

*\_Value* The number to draw.

```
8.1.2.1540 #define RICOpPixel(_CopyOptions, _Point, _Value) 10, 0,  
4, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,  
(_Value)&0xFF, (_Value)>>8
```

Output an RIC Pixel opcode.

**Parameters:**

*\_CopyOptions* Pixel copy options. See [Drawing option constants](#).

*\_Point* The pixel coordinate. See [RICImgPoint](#).

*\_Value* The pixel value (unused).

```
8.1.2.1541 #define RICOpPolygon(_CopyOptions, _Count,  
_ThePoints) ((_Count*4)+6)&0xFF, ((_Count*4)+6)>>8, 10, 0,  
(_CopyOptions)&0xFF, (_CopyOptions)>>8, (_Count)&0xFF,  
(_Count)>>8, _ThePoints
```

Output an RIC Polygon opcode.

**Parameters:**

*\_CopyOptions* Polygon copy options. See [Drawing option constants](#).

*\_Count* The number of points in the polygon.

*\_ThePoints* The list of polygon points. See [RICPolygonPoints](#).

```
8.1.2.1542 #define RICOpRect(_CopyOptions, _Point, _Width, _Height) 12,  
0, 6, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,  
(_Width)&0xFF, (_Width)>>8, (_Height)&0xFF, (_Height)>>8
```

Output an RIC Rect opcode.

**Parameters:**

*\_CopyOptions* Rect copy options. See [Drawing option constants](#).

*\_Point* The rectangle's top left corner. See [RICImgPoint](#).

*\_Width* The rectangle's width.

*\_Height* The rectangle's height.

```

8.1.2.1543 #define RICOpSprite(_DataAddr, _Rows,
           _BytesPerRow, _SpriteData) ((_Rows*_
           BytesPerRow)+((_Rows*_BytesPerRow)%2)+8)&0xFF,
           ((_Rows*_BytesPerRow)+((_Rows*_BytesPerRow)%2)+8)>>8,
           1, 0, (_DataAddr)&0xFF, (_DataAddr)>>8, (_Rows)&0xFF,
           (_Rows)>>8, (_BytesPerRow)&0xFF, (_BytesPerRow)>>8,
           _SpriteData

```

Output an RIC Sprite opcode.

**Parameters:**

- \_DataAddr* The address of the sprite.
- \_Rows* The number of rows of data.
- \_BytesPerRow* The number of bytes per row.
- \_SpriteData* The actual sprite data. See [RICSpriteData](#).

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```

8.1.2.1544 #define RICOpVarMap(_DataAddr, _MapCount,
           _MapFunction) ((_MapCount*4)+6)&0xFF,
           ((_MapCount*4)+6)>>8, 2, 0, (_DataAddr)&0xFF,
           (_DataAddr)>>8, (_MapCount)&0xFF, (_MapCount)>>8,
           _MapFunction

```

Output an RIC VarMap opcode.

**Parameters:**

- \_DataAddr* The address of the varmap.
- \_MapCount* The number of points in the function.
- \_MapFunction* The definition of the varmap function. See [RICMapFunction](#).

```

8.1.2.1545 #define RICPolygonPoints(_pPoint1, _pPoint2, ...) _pPoint1,
           _pPoint2, __VA_ARGS__

```

Output RIC polygon points.

**Parameters:**

- \_pPoint1* The first polygon point. See [RICImgPoint](#).
- \_pPoint2* The second polygon point (at least 3 points are required). See [RICImgPoint](#).

**8.1.2.1546 #define RICSpriteData(...) \_\_VA\_ARGS\_\_**

Output RIC sprite data.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**8.1.2.1547 #define ROTATE\_QUEUE 5**

VM should rotate queue

**8.1.2.1548 #define RotationCountField 14**

Rotation counter field. Contains the current rotation count. Read only. Return the program-relative position counter value for the specified port. Refer to the [UpdateFlagsField](#) description for information about how to use program-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) flag in [UpdateFlagsField](#) to request that the firmware reset the [RotationCountField](#). The sign of [RotationCountField](#) indicates the direction of rotation. Positive values indicate forward rotation and negative values indicate reverse rotation. Forward and reverse depend on the orientation of the motor.

**8.1.2.1549 #define RunStateField 6**

Run state field. Contains one of the run state constants. Read/write. Use this field to specify the running state of an output. Set the [RunStateField](#) to [OUT\\_RUNSTATE\\_RUNNING](#) to enable power to any output. Use [OUT\\_RUNSTATE\\_RAMPUP](#) to enable automatic ramping to a new [PowerField](#) level greater than the current [PowerField](#) level. Use [OUT\\_RUNSTATE\\_RAMPDOWN](#) to enable automatic ramping to a new [PowerField](#) level less than the current [PowerField](#) level. Both the rampup and rampdown bits must be used in conjunction with appropriate [TachoLimitField](#) and [PowerField](#) values. In this case the firmware smoothly increases or decreases the actual power

to the new [PowerField](#) level over the total number of degrees of rotation specified in [TachoLimitField](#).

**8.1.2.1550 #define SAMPLERATE\_DEFAULT 8000**

Default sample rate [sps]

**8.1.2.1551 #define SAMPLERATE\_MAX 16000**

Max sample rate [sps]

**8.1.2.1552 #define SAMPLERATE\_MIN 2000**

Min sample rate [sps]

**8.1.2.1553 #define ScaledValueField 4**

Scaled value field. Contains the current scaled analog sensor value. Read/write.

**8.1.2.1554 #define SCHAR\_MAX 127**

The maximum value of the signed char type

**8.1.2.1555 #define SCHAR\_MIN -128**

The minimum value of the signed char type

**8.1.2.1556 #define SCOUT\_FXR\_ALARM 2**

Alarm special effects

**8.1.2.1557 #define SCOUT\_FXR\_BUG 1**

Bug special effects

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**8.1.2.1558** `#define SCOUT_FXR_NONE 0`

No special effects

**8.1.2.1559** `#define SCOUT_FXR_RANDOM 3`

Random special effects

**8.1.2.1560** `#define SCOUT_FXR_SCIENCE 4`

Science special effects

**8.1.2.1561** `#define SCOUT_LIGHT_OFF 0`

Turn off the scout light

**8.1.2.1562** `#define SCOUT_LIGHT_ON 0x80`

Turn on the scout light

**Examples:**

[ex\\_HTScoutSetLight.nxc](#).

**8.1.2.1563** `#define SCOUT_LR_AVOID 3`

Light rule avoid

**8.1.2.1564** `#define SCOUT_LR_IGNORE 0`

Light rule ignore

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**8.1.2.1565** `#define SCOUT_LR_OFF_WHEN 5`

Light rule off when

**8.1.2.1566** `#define SCOUT_LR_SEEK_DARK 2`

Light rule seek dark

**8.1.2.1567** `#define SCOUT_LR_SEEK_LIGHT 1`

Light rule seek light

**8.1.2.1568** `#define SCOUT_LR_WAIT_FOR 4`

Light rule wait for

**8.1.2.1569** `#define SCOUT_MODE_POWER 1`

Enter power mode

**Examples:**

[ex\\_HTScoutSetScoutMode.nxc](#), and [ex\\_MSScoutSetScoutMode.nxc](#).

**8.1.2.1570** `#define SCOUT_MODE_STANDALONE 0`

Enter stand alone mode

**8.1.2.1571** `#define SCOUT_MR_CIRCLE_LEFT 4`

Motion rule circle left

**8.1.2.1572** `#define SCOUT_MR_CIRCLE_RIGHT 3`

Motion rule circle right

**8.1.2.1573** `#define SCOUT_MR_FORWARD 1`

Motion rule forward

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**8.1.2.1574** `#define SCOUT_MR_LOOP_A 5`

Motion rule loop A

**8.1.2.1575** `#define SCOUT_MR_LOOP_AB 7`

Motion rule loop A then B

**8.1.2.1576** `#define SCOUT_MR_LOOP_B 6`

Motion rule loop B

**8.1.2.1577** `#define SCOUT_MR_NO_MOTION 0`

Motion rule none

**8.1.2.1578** `#define SCOUT_MR_ZIGZAG 2`

Motion rule zigzag

**8.1.2.1579** `#define SCOUT_SNDSET_ALARM 3`

Set sound set to alarm

**8.1.2.1580** `#define SCOUT_SNDSET_BASIC 1`

Set sound set to basic

**8.1.2.1581** `#define SCOUT_SNDSET_BUG 2`

Set sound set to bug

**8.1.2.1582** `#define SCOUT_SNDSET_NONE 0`

Set sound set to none

**8.1.2.1583** `#define SCOUT_SNDSET_RANDOM 4`

Set sound set to random

**8.1.2.1584 #define SCOUT\_SNDSET\_SCIENCE 5**

Set sound set to science

**8.1.2.1585 #define SCOUT\_SOUND\_1\_BLINK 17**

Play the Scout 1 blink sound

**8.1.2.1586 #define SCOUT\_SOUND\_2\_BLINK 18**

Play the Scout 2 blink sound

**8.1.2.1587 #define SCOUT\_SOUND\_COUNTER1 19**

Play the Scout counter 1 sound

**8.1.2.1588 #define SCOUT\_SOUND\_COUNTER2 20**

Play the Scout counter 2 sound

**8.1.2.1589 #define SCOUT\_SOUND\_ENTER\_BRIGHT 14**

Play the Scout enter bright sound

**8.1.2.1590 #define SCOUT\_SOUND\_ENTER\_DARK 16**

Play the Scout enter dark sound

**8.1.2.1591 #define SCOUT\_SOUND\_ENTER\_NORMAL 15**

Play the Scout enter normal sound

**8.1.2.1592 #define SCOUT\_SOUND\_ENTERSA 7**

Play the Scout enter standalone sound

**8.1.2.1593 #define SCOUT\_SOUND\_KEYERROR 8**

Play the Scout key error sound

**8.1.2.1594 #define SCOUT\_SOUND\_MAIL\_RECEIVED 24**

Play the Scout mail received sound

**8.1.2.1595 #define SCOUT\_SOUND\_NONE 9**

Play the Scout none sound

**8.1.2.1596 #define SCOUT\_SOUND\_REMOTE 6**

Play the Scout remote sound

**8.1.2.1597 #define SCOUT\_SOUND\_SPECIAL1 25**

Play the Scout special 1 sound

**8.1.2.1598 #define SCOUT\_SOUND\_SPECIAL2 26**

Play the Scout special 2 sound

**8.1.2.1599 #define SCOUT\_SOUND\_SPECIAL3 27**

Play the Scout special 3 sound

**8.1.2.1600 #define SCOUT\_SOUND\_TIMER1 21**

Play the Scout timer 1 sound

**8.1.2.1601 #define SCOUT\_SOUND\_TIMER2 22**

Play the Scout timer 2 sound

**8.1.2.1602 #define SCOUT\_SOUND\_TIMER3 23**

Play the Scout timer 3 sound

**8.1.2.1603 #define SCOUT\_SOUND\_TOUCH1\_PRES 10**

Play the Scout touch 1 pressed sound

**8.1.2.1604 #define SCOUT\_SOUND\_TOUCH1\_REL 11**

Play the Scout touch 1 released sound

**8.1.2.1605 #define SCOUT\_SOUND\_TOUCH2\_PRES 12**

Play the Scout touch 2 pressed sound

**8.1.2.1606 #define SCOUT\_SOUND\_TOUCH2\_REL 13**

Play the Scout touch 2 released sound

**8.1.2.1607 #define SCOUT\_TGS\_LONG 2**

Transmit level long

**8.1.2.1608 #define SCOUT\_TGS\_MEDIUM 1**

Transmit level medium

**8.1.2.1609 #define SCOUT\_TGS\_SHORT 0**

Transmit level short

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**8.1.2.1610 #define SCOUT\_TR\_AVOID 2**

Touch rule avoid

**8.1.2.1611 #define SCOUT\_TR\_IGNORE 0**

Touch rule ignore

**8.1.2.1612 #define SCOUT\_TR\_OFF\_WHEN 4**

Touch rule off when

**8.1.2.1613 #define SCOUT\_TR\_REVERSE 1**

Touch rule reverse

**Examples:**[ex\\_MSScoutSetScoutRules.nxc](#).**8.1.2.1614 #define SCOUT\_TR\_WAIT\_FOR 3**

Touch rule wait for

**8.1.2.1615 #define SCREEN\_BACKGROUND 0**

Entire screen

**8.1.2.1616 #define SCREEN\_LARGE 1**

Entire screen except status line

**8.1.2.1617 #define SCREEN\_MODE\_CLEAR 0x01**

Clear the screen

**See also:**[SetScreenMode\(\)](#)**8.1.2.1618 #define SCREEN\_MODE\_RESTORE 0x00**

Restore the screen

**See also:**[SetScreenMode\(\)](#)**8.1.2.1619 #define SCREEN\_SMALL 2**

Screen between menu icons and status line

**8.1.2.1620 #define SCREENS 3**

The number of screen bits

**8.1.2.1621 #define SEC\_1 1000**

1 second

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_sysdrawpolygon.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_wait.nxc](#), and [ex\\_yield.nxc](#).

**8.1.2.1622 #define SEC\_10 10000**

10 seconds

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_string.nxc](#), [ex\\_syscommmbtconnection.nxc](#), and [ex\\_SysCommHSControl.nxc](#).

**8.1.2.1623 #define SEC\_15 15000**

15 seconds

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_memcmp.nxc](#).

**8.1.2.1624 #define SEC\_2 2000**

2 seconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**8.1.2.1625 #define SEC\_20 20000**

20 seconds

**8.1.2.1626 #define SEC\_3 3000**

3 seconds

**Examples:**

[ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_div.nxc](#), and [ex\\_ldiv.nxc](#).

**8.1.2.1627 #define SEC\_30 30000**

30 seconds

**8.1.2.1628 #define SEC\_4 4000**

4 seconds

**Examples:**

[ex\\_copy.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_midstr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_syslistfiles.nxc](#), [util\\_battery\\_1.nxc](#), and [util\\_battery\\_2.nxc](#).

**8.1.2.1629 #define SEC\_5 5000**

5 seconds

**Examples:**

[ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_clearline.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_isnan.nxc](#), [ex\\_labs.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_onfwdsyncpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_PFMate.nxc](#), [ex\\_proto.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_UnflattenVar.nxc](#), and [ex\\_wait.nxc](#).

**8.1.2.1630 #define SEC\_6 6000**

6 seconds

**Examples:**

[ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), and [ex\\_strtoul.nxc](#).

**8.1.2.1631 #define SEC\_7 7000**

7 seconds

**8.1.2.1632 #define SEC\_8 8000**

8 seconds

**Examples:**

[ex\\_file\\_system.nxc](#).

**8.1.2.1633 #define SEC\_9 9000**

9 seconds

**Examples:**

[ex\\_SensorHTGyro.nxc](#).

**8.1.2.1634 #define SetScreenMode 19**

Set the screen mode

**8.1.2.1635 #define SetSleepTimeoutVal 46**

Set the NXT sleep timeout value

**8.1.2.1636 #define SHRT\_MAX 32767**

The maximum value of the short type

**8.1.2.1637 #define SHRT\_MIN -32768**

The minimum value of the short type

**8.1.2.1638 #define SIZE\_OF\_BDADDR 7**

Size of Bluetooth Address

**8.1.2.1639 #define SIZE\_OF\_BRICK\_NAME 8**

Size of NXT Brick name

**8.1.2.1640 #define SIZE\_OF\_BT\_CONNECT\_TABLE 4**

Size of Bluetooth connection table -- Index 0 is always incoming connection

**8.1.2.1641 #define SIZE\_OF\_BT\_DEVICE\_TABLE 30**

Size of Bluetooth device table

**8.1.2.1642 #define SIZE\_OF\_BT\_NAME 16**

Size of Bluetooth name

**8.1.2.1643 #define SIZE\_OF\_BT\_PINCODE 16**

Size of Bluetooth PIN

**8.1.2.1644 #define SIZE\_OF\_BTBUF 128**

Size of Bluetooth buffer

**8.1.2.1645 #define SIZE\_OF\_CLASS\_OF\_DEVICE 4**

Size of class of device

**8.1.2.1646 #define SIZE\_OF\_HSBUF 128**

Size of High Speed Port 4 buffer

**8.1.2.1647 #define SIZE\_OF\_USBBUF 64**

Size of USB Buffer in bytes

**8.1.2.1648 #define SIZE\_OF\_USBDATA 62**

Size of USB Buffer available for data

**8.1.2.1649 #define SOUND\_CLICK 0**

Play the standard key click sound

**8.1.2.1650 #define SOUND\_DOUBLE\_BEEP 1**

Play the standard double beep sound

**8.1.2.1651 #define SOUND\_DOWN 2**

Play the standard sweep down sound

**Examples:**

[ex\\_playsound.nxc.](#)

**8.1.2.1652 #define SOUND\_FAST\_UP 5**

Play the standard fast up sound

**Examples:**

[ex\\_playsound.nxc.](#)

**8.1.2.1653 #define SOUND\_FLAGS\_IDLE 0x00**

R - Sound is idle

**8.1.2.1654 #define SOUND\_FLAGS\_RUNNING 0x02**

R - Currently processing a tone or file

**8.1.2.1655 #define SOUND\_FLAGS\_UPDATE 0x01**

W - Make changes take effect

**Examples:**[ex\\_SetSoundFlags.nxc.](#)**8.1.2.1656 #define SOUND\_LOW\_BEEP 4**

Play the standard low beep sound

**Examples:**[ex\\_playsound.nxc.](#)**8.1.2.1657 #define SOUND\_MODE\_LOOP 0x01**

W - Play file until writing SOUND\_STATE\_STOP into SoundState

**8.1.2.1658 #define SOUND\_MODE\_ONCE 0x00**

W - Only play file once

**Examples:**[ex\\_SetSoundMode.nxc.](#)**8.1.2.1659 #define SOUND\_MODE\_TONE 0x02**

W - Play tone specified in Frequency for Duration ms

**8.1.2.1660 #define SOUND\_STATE\_FILE 0x02**

R - Processing a file of sound/melody data

**8.1.2.1661 #define SOUND\_STATE\_IDLE 0x00**

R - Idle, ready for start sound (SOUND\_UPDATE)

**Examples:**[ex\\_syssoundgetstate.nxc.](#)

**8.1.2.1662 #define SOUND\_STATE\_STOP 0x04**

W - Stop sound immediately and close hardware

**Examples:**

[ex\\_SetSoundModuleState.nxc](#), and [ex\\_syssoundsetstate.nxc](#).

**8.1.2.1663 #define SOUND\_STATE\_TONE 0x03**

R - Processing a play tone request

**8.1.2.1664 #define SOUND\_UP 3**

Play the standard sweep up sound

**Examples:**

[ex\\_playsound.nxc](#).

**8.1.2.1665 #define SoundGetState 11**

Get the current sound module state

**8.1.2.1666 #define SoundModuleID 0x00080001**

The sound module ID

**Examples:**

[ex\\_sysiomapwritebyid.nxc](#).

**8.1.2.1667 #define SoundModuleName "Sound.mod"**

The sound module name

**Examples:**

[ex\\_sysiomapwrite.nxc](#).

**8.1.2.1668 #define SoundOffsetDuration 2**RW - [Tone](#) duration [mS] (2 bytes)**8.1.2.1669 #define SoundOffsetFlags 26**RW - Play flag - described above (1 byte) [SoundFlags constants](#)**8.1.2.1670 #define SoundOffsetFreq 0**RW - [Tone](#) frequency [Hz] (2 bytes)**8.1.2.1671 #define SoundOffsetMode 28**RW - Play mode - described above (1 byte) [SoundMode constants](#)**8.1.2.1672 #define SoundOffsetSampleRate 4**

RW - Sound file sample rate [2000..16000] (2 bytes)

**Examples:**[ex\\_sysiomapwrite.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).**8.1.2.1673 #define SoundOffsetSoundFilename 6**

RW - Sound/melody filename (20 bytes)

**8.1.2.1674 #define SoundOffsetState 27**RW - Play state - described above (1 byte) [SoundState constants](#)**8.1.2.1675 #define SoundOffsetVolume 29**

RW - Sound/melody volume [0..4] 0 = off (1 byte)

**8.1.2.1676 #define SoundPlayFile 9**

Play a sound or melody file

**8.1.2.1677 #define SoundPlayTone 10**

Play a simple tone with the specified frequency and duration

**8.1.2.1678 #define SoundSetState 12**

Set the sound module state

**8.1.2.1679 #define SPECIALS 5**

The number of special bit values

**8.1.2.1680 #define STAT\_COMM\_PENDING 32**

Pending setup operation in progress

**8.1.2.1681 #define STAT\_MSG\_EMPTY\_MAILBOX 64**

Specified mailbox contains no new messages

**8.1.2.1682 #define STATUSICON\_BATTERY 3**

Battery status icon collection

**8.1.2.1683 #define STATUSICON\_BLUETOOTH 0**

BlueTooth status icon collection

**8.1.2.1684 #define STATUSICON\_USB 1**

USB status icon collection

**8.1.2.1685 #define STATUSICON\_VM 2**

VM status icon collection

**8.1.2.1686 #define STATUSICONS 4**

The number of status icons

**8.1.2.1687 #define STATUSTEXT 1**

Status text (BT name)

**8.1.2.1688 #define STEPICON\_1 0**

Left most step icon

**8.1.2.1689 #define STEPICON\_2 1**

**8.1.2.1690 #define STEPICON\_3 2**

**8.1.2.1691 #define STEPICON\_4 3**

**8.1.2.1692 #define STEPICON\_5 4**

Right most step icon

**8.1.2.1693 #define STEPICONS 5**

**8.1.2.1694 #define STEPLINE 3**

Step collection lines

**8.1.2.1695 #define STOP\_REQ 4**

VM should stop executing program

**8.1.2.1696 #define STROBE\_READ 0x10**

Access read pin (RD)

**8.1.2.1697 #define STROBE\_S0 0x01**

Access strobe 0 pin (S0)

**Examples:**[ex\\_superpro.nxc](#).**8.1.2.1698 #define STROBE\_S1 0x02**

Access strobe 1 pin (S1)

**8.1.2.1699 #define STROBE\_S2 0x04**

Access strobe 2 pin (S2)

**8.1.2.1700 #define STROBE\_S3 0x08**

Access strobe 3 pin (S3)

**8.1.2.1701 #define STROBE\_WRITE 0x20**

Access write pin (WR)

**8.1.2.1702 #define TachoCountField 4**

Internal tachometer count field. Contains the current internal tachometer count. Read only. Return the internal position counter value for the specified output. The internal count is reset automatically when a new goal is set using the [TachoLimitField](#) and the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag. Set the [UF\\_UPDATE\\_RESET\\_COUNT](#) flag in [UpdateFlagsField](#) to reset [TachoCountField](#) and cancel any [TachoLimitField](#). The sign of [TachoCountField](#) indicates the motor rotation direction.

**8.1.2.1703 #define TachoLimitField 5**

Tachometer limit field. Contains the current tachometer limit. Read/write. Specify the number of degrees the motor should rotate. Use [UF\\_UPDATE\\_TACHO\\_LIMIT](#) with the [UpdateFlagsField](#) field to commit changes to the [TachoLimitField](#). The value of

this field is a relative distance from the current motor position at the moment when the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag is processed.

**8.1.2.1704 #define TEMP\_FQ\_1 0x00**

Set fault queue to 1 fault before alert

**8.1.2.1705 #define TEMP\_FQ\_2 0x08**

Set fault queue to 2 faults before alert

**8.1.2.1706 #define TEMP\_FQ\_4 0x10**

Set fault queue to 4 faults before alert

**8.1.2.1707 #define TEMP\_FQ\_6 0x18**

Set fault queue to 6 faults before alert

**8.1.2.1708 #define TEMP\_OS\_ONESHOT 0x80**

Set the sensor into oneshot mode. When the device is in shutdown mode this will start a single temperature conversion. The device returns to shutdown mode when it completes.

**8.1.2.1709 #define TEMP\_POL\_HIGH 0x04**

Set polarity of ALERT pin to be active HIGH

**8.1.2.1710 #define TEMP\_POL\_LOW 0x00**

Set polarity of ALERT pin to be active LOW

**8.1.2.1711 #define TEMP\_REG\_CONFIG 0x01**

The register for reading/writing sensor configuration values

**8.1.2.1712 #define TEMP\_REG\_TEMP 0x00**

The register where temperature values can be read

**8.1.2.1713 #define TEMP\_REG\_THIGH 0x03**

The register for reading/writing a user-defined high temperature limit

**8.1.2.1714 #define TEMP\_REG\_TLOW 0x02**

The register for reading/writing a user-defined low temperature limit

**8.1.2.1715 #define TEMP\_RES\_10BIT 0x20**

Set the temperature conversion resolution to 10 bit

**8.1.2.1716 #define TEMP\_RES\_11BIT 0x40**

Set the temperature conversion resolution to 11 bit

**8.1.2.1717 #define TEMP\_RES\_12BIT 0x60**

Set the temperature conversion resolution to 12 bit

**Examples:**

[ex\\_ConfigureTemperatureSensor.nxc](#).

**8.1.2.1718 #define TEMP\_RES\_9BIT 0x00**

Set the temperature conversion resolution to 9 bit

**8.1.2.1719 #define TEMP\_SD\_CONTINUOUS 0x00**

Set the sensor mode to continuous

**8.1.2.1720 #define TEMP\_SD\_SHUTDOWN 0x01**

Set the sensor mode to shutdown. The device will shut down after the current conversion is completed.

**8.1.2.1721 #define TEMP\_TM\_COMPARATOR 0x00**

Set the thermostat mode to comparator

**8.1.2.1722** `#define TEMP_TM_INTERRUPT 0x02`

Set the thermostat mode to interrupt

**8.1.2.1723** `#define TEXTLINE_1 0`

Text line 1

**Examples:**

[ex\\_GetDisplayNormal.nxc](#), [ex\\_GetDisplayPopup.nxc](#), [ex\\_SetDisplayNormal.nxc](#),  
and [ex\\_SetDisplayPopup.nxc](#).

**8.1.2.1724** `#define TEXTLINE_2 1`

Text line 2

**8.1.2.1725** `#define TEXTLINE_3 2`

Text line 3

**8.1.2.1726** `#define TEXTLINE_4 3`

Text line 4

**8.1.2.1727** `#define TEXTLINE_5 4`

Text line 5

**8.1.2.1728** `#define TEXTLINE_6 5`

Text line 6

**8.1.2.1729** `#define TEXTLINE_7 6`

Text line 7

**8.1.2.1730** `#define TEXTLINE_8 7`

Text line 8

**8.1.2.1731 #define TEXTLINES 8**

The number of text lines on the LCD

**8.1.2.1732 #define TIMES\_UP 6**

VM time is up

**8.1.2.1733 #define TONE\_A3 220**

Third octave A

**8.1.2.1734 #define TONE\_A4 440**

Fourth octave A

**Examples:**

[ex\\_yield.nxc.](#)

**8.1.2.1735 #define TONE\_A5 880**

Fifth octave A

**8.1.2.1736 #define TONE\_A6 1760**

Sixth octave A

**8.1.2.1737 #define TONE\_A7 3520**

Seventh octave A

**8.1.2.1738 #define TONE\_AS3 233**

Third octave A sharp

**8.1.2.1739 #define TONE\_AS4 466**

Fourth octave A sharp

**8.1.2.1740** `#define TONE_AS5 932`

Fifth octave A sharp

**8.1.2.1741** `#define TONE_AS6 1865`

Sixth octave A sharp

**8.1.2.1742** `#define TONE_AS7 3729`

Seventh octave A sharp

**8.1.2.1743** `#define TONE_B3 247`

Third octave B

**8.1.2.1744** `#define TONE_B4 494`

Fourth octave B

**8.1.2.1745** `#define TONE_B5 988`

Fifth octave B

**8.1.2.1746** `#define TONE_B6 1976`

Sixth octave B

**8.1.2.1747** `#define TONE_B7 3951`

Seventh octave B

**8.1.2.1748** `#define TONE_C4 262`

Fourth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**8.1.2.1749** `#define TONE_C5 523`

Fifth octave C

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_playtones.nxc](#).

**8.1.2.1750** `#define TONE_C6 1047`

Sixth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**8.1.2.1751** `#define TONE_C7 2093`

Seventh octave C

**8.1.2.1752** `#define TONE_CS4 277`

Fourth octave C sharp

**8.1.2.1753** `#define TONE_CS5 554`

Fifth octave C sharp

**8.1.2.1754** `#define TONE_CS6 1109`

Sixth octave C sharp

**8.1.2.1755** `#define TONE_CS7 2217`

Seventh octave C sharp

**8.1.2.1756** `#define TONE_D4 294`

Fourth octave D

**8.1.2.1757** `#define TONE_D5 587`

Fifth octave D

**8.1.2.1758** `#define TONE_D6 1175`

Sixth octave D

**8.1.2.1759** `#define TONE_D7 2349`

Seventh octave D

**8.1.2.1760** `#define TONE_DS4 311`

Fourth octave D sharp

**8.1.2.1761** `#define TONE_DS5 622`

Fifth octave D sharp

**8.1.2.1762** `#define TONE_DS6 1245`

Sixth octave D sharp

**8.1.2.1763** `#define TONE_DS7 2489`

Seventh octave D sharp

**8.1.2.1764** `#define TONE_E4 330`

Fourth octave E

**Examples:**

[ex\\_playtones.nxc](#).

**8.1.2.1765** `#define TONE_E5 659`

Fifth octave E

**Examples:**

[ex\\_playtones.nxc](#).

**8.1.2.1766 #define TONE\_E6 1319**

Sixth octave E

**8.1.2.1767 #define TONE\_E7 2637**

Seventh octave E

**8.1.2.1768 #define TONE\_F4 349**

Fourth octave F

**8.1.2.1769 #define TONE\_F5 698**

Fifth octave F

**8.1.2.1770 #define TONE\_F6 1397**

Sixth octave F

**8.1.2.1771 #define TONE\_F7 2794**

Seventh octave F

**8.1.2.1772 #define TONE\_FS4 370**

Fourth octave F sharp

**8.1.2.1773 #define TONE\_FS5 740**

Fifth octave F sharp

**8.1.2.1774 #define TONE\_FS6 1480**

Sixth octave F sharp

**8.1.2.1775** `#define TONE_FS7 2960`

Seventh octave F sharp

**8.1.2.1776** `#define TONE_G4 392`

Fourth octave G

**Examples:**

[ex\\_playtones.nxc](#).

**8.1.2.1777** `#define TONE_G5 784`

Fifth octave G

**Examples:**

[ex\\_playtones.nxc](#).

**8.1.2.1778** `#define TONE_G6 1568`

Sixth octave G

**8.1.2.1779** `#define TONE_G7 3136`

Seventh octave G

**8.1.2.1780** `#define TONE_GS4 415`

Fourth octave G sharp

**8.1.2.1781** `#define TONE_GS5 831`

Fifth octave G sharp

**8.1.2.1782** `#define TONE_GS6 1661`

Sixth octave G sharp

**8.1.2.1783** `#define TONE_GS7 3322`

Seventh octave G sharp

**8.1.2.1784** `#define TOPLINE 4`

Top status underline

**8.1.2.1785** `#define TRAIN_CHANNEL_1 0`

IR Train channel 1

**Examples:**

[ex\\_HTIRTrain.nxc](#), and [ex\\_MSIRTrain.nxc](#).

**8.1.2.1786** `#define TRAIN_CHANNEL_2 1`

IR Train channel 2

**8.1.2.1787** `#define TRAIN_CHANNEL_3 2`

IR Train channel 3

**8.1.2.1788** `#define TRAIN_CHANNEL_ALL 3`

IR Train channel all

**8.1.2.1789** `#define TRAIN_FUNC_DECR_SPEED 2`

PF/IR Train function decrement speed

**8.1.2.1790** `#define TRAIN_FUNC_INCR_SPEED 1`

PF/IR Train function increment speed

**Examples:**

[ex\\_HTIRTrain.nxc](#), [ex\\_HTPFTrain.nxc](#), [ex\\_MSIRTrain.nxc](#), and [ex\\_MSPFTrain.nxc](#).

**8.1.2.1791 #define TRAIN\_FUNC\_STOP 0**

PF/IR Train function stop

**8.1.2.1792 #define TRAIN\_FUNC\_TOGGLE\_LIGHT 4**

PF/IR Train function toggle light

**8.1.2.1793 #define TRUE 1**

A true value

**Examples:**[ex\\_syscommbtconnection.nxc.](#)**8.1.2.1794 #define TurnRatioField 7**

Turn ratio field. Contains the current turn ratio. Only applicable when synchronizing multiple motors. Read/write. Use this field to specify a proportional turning ratio. This field must be used in conjunction with other field values: `OutputModeField` must include `OUT_MODE_MOTORON` and `OUT_MODE_REGULATED`, `RegModeField` must be set to `OUT_REGMODE_SYNC`, `RunStateField` must not be `OUT_RUNSTATE_IDLE`, and `PowerField` must be non-zero. There are only three valid combinations of left and right motors for use with `TurnRatioField`: `OUT_AB`, `OUT_BC`, and `OUT_AC`. In each of these three options the first motor listed is considered to be the left motor and the second motor is the right motor, regardless of the physical configuration of the robot. Negative turn ratio values shift power toward the left motor while positive values shift power toward the right motor. An absolute value of 50 usually results in one motor stopping. An absolute value of 100 usually results in two motors turning in opposite directions at equal power.

**8.1.2.1795 #define TypeField 0**

Type field. Contains one of the sensor type constants. Read/write.

**8.1.2.1796 #define UCHAR\_MAX 255**

The maximum value of the unsigned char type

**8.1.2.1797 #define UF\_PENDING\_UPDATES 0x80**

Are there any pending motor updates?

**8.1.2.1798 #define UF\_UPDATE\_MODE 0x01**

Commits changes to the [OutputModeField](#) output property

**8.1.2.1799 #define UF\_UPDATE\_PID\_VALUES 0x10**

Commits changes to the PID motor regulation properties

**8.1.2.1800 #define UF\_UPDATE\_RESET\_BLOCK\_COUNT 0x20**

Resets the NXT-G block-relative rotation counter

**8.1.2.1801 #define UF\_UPDATE\_RESET\_COUNT 0x08**

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

**8.1.2.1802 #define UF\_UPDATE\_RESET\_ROTATION\_COUNT 0x40**

Resets the program-relative (user) rotation counter

**8.1.2.1803 #define UF\_UPDATE\_SPEED 0x02**

Commits changes to the [PowerField](#) output property

**8.1.2.1804 #define UF\_UPDATE\_TACHO\_LIMIT 0x04**

Commits changes to the [TachoLimitField](#) output property

**8.1.2.1805 #define UI\_BT\_CONNECT\_REQUEST 0x40**

RW - BT get connect accept in progress

**8.1.2.1806 #define UI\_BT\_ERROR\_ATTENTION 0x08**

W - BT error attention

**8.1.2.1807 #define UI\_BT\_PIN\_REQUEST 0x80**

RW - BT get pin code

**8.1.2.1808 #define UI\_BT\_STATE\_CONNECTED 0x02**

RW - BT connected to something

**8.1.2.1809 #define UI\_BT\_STATE\_OFF 0x04**

RW - BT power off

**Examples:**

[ex\\_SetBluetoothState.nxc.](#)

**8.1.2.1810 #define UI\_BT\_STATE\_VISIBLE 0x01**

RW - BT visible

**8.1.2.1811 #define UI\_BUTTON\_ENTER 2**

W - Insert enter button

**Examples:**

[ex\\_SetUIButton.nxc.](#)

**8.1.2.1812 #define UI\_BUTTON\_EXIT 4**

W - Insert exit button

**8.1.2.1813 #define UI\_BUTTON\_LEFT 1**

W - Insert left arrow button

**8.1.2.1814 #define UI\_BUTTON\_NONE 0**

R - Button inserted are executed

**8.1.2.1815 #define UI\_BUTTON\_RIGHT 3**

W - Insert right arrow button

**8.1.2.1816 #define UI\_FLAGS\_BUSY 0x40**

R - UI busy running or datalogging (popup disabled)

**8.1.2.1817 #define UI\_FLAGS\_DISABLE\_EXIT 0x04**

RW - Disable exit button

**8.1.2.1818 #define UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER 0x02**

RW - Disable left, right and enter button

**8.1.2.1819 #define UI\_FLAGS\_ENABLE\_STATUS\_UPDATE 0x80**

W - Enable status line to be updated

**8.1.2.1820 #define UI\_FLAGS\_EXECUTE\_LMS\_FILE 0x20**

W - Execute LMS file in "LMSfilename" (Try It)

**8.1.2.1821 #define UI\_FLAGS\_REDRAW\_STATUS 0x08**

W - Redraw entire status line

**Examples:**

[ex\\_SetCommandFlags.nxc.](#)

**8.1.2.1822 #define UI\_FLAGS\_RESET\_SLEEP\_TIMER 0x10**

W - Reset sleep timeout timer

**8.1.2.1823 #define UI\_FLAGS\_UPDATE 0x01**

W - Make changes take effect

8.1.2.1824 **#define UI\_STATE\_BT\_ERROR 16**

R - BT error

8.1.2.1825 **#define UI\_STATE\_CONNECT\_REQUEST 12**

RW - Request for connection accept

8.1.2.1826 **#define UI\_STATE\_DRAW\_MENU 6**

RW - Execute function and draw menu icons

8.1.2.1827 **#define UI\_STATE\_ENTER\_PRESSED 10**

RW - Load selected function and next menu id

8.1.2.1828 **#define UI\_STATE\_EXECUTE\_FILE 13**

RW - Execute file in "LMSfilename"

8.1.2.1829 **#define UI\_STATE\_EXECUTING\_FILE 14**

R - Executing file in "LMSfilename"

8.1.2.1830 **#define UI\_STATE\_EXIT\_PRESSED 11**

RW - Load selected function and next menu id

8.1.2.1831 **#define UI\_STATE\_INIT\_DISPLAY 0**

RW - Init display and load font, menu etc.

8.1.2.1832 **#define UI\_STATE\_INIT\_INTRO 2**

R - Display intro

8.1.2.1833 **#define UI\_STATE\_INIT\_LOW\_BATTERY 1**

R - Low battery voltage at power on

**8.1.2.1834 #define UI\_STATE\_INIT\_MENU 4**

RW - Init menu system

**8.1.2.1835 #define UI\_STATE\_INIT\_WAIT 3**

RW - Wait for initialization end

**8.1.2.1836 #define UI\_STATE\_LEFT\_PRESSED 8**

RW - Load selected function and next menu id

**8.1.2.1837 #define UI\_STATE\_LOW\_BATTERY 15**

R - Low battery at runtime

**Examples:**

[ex\\_SetUIState.nxc.](#)

**8.1.2.1838 #define UI\_STATE\_NEXT\_MENU 5**

RW - Next menu icons ready for drawing

**8.1.2.1839 #define UI\_STATE\_RIGHT\_PRESSED 9**

RW - Load selected function and next menu id

**8.1.2.1840 #define UI\_STATE\_TEST\_BUTTONS 7**

RW - Wait for buttons to be pressed

**8.1.2.1841 #define UI\_VM\_IDLE 0**

VM\_IDLE: Just sitting around. Request to run program will lead to ONE of the VM\_RUN\* states.

**8.1.2.1842 #define UI\_VM\_RESET1 4**

VM\_RESET1: Initialize state variables and some I/O devices -- executed when programs end

**8.1.2.1843 #define UI\_VM\_RESET2 5**

VM\_RESET2: Final clean up and return to IDLE

**8.1.2.1844 #define UI\_VM\_RUN\_FREE 1**

VM\_RUN\_FREE: Attempt to run as many instructions as possible within our timeslice

**8.1.2.1845 #define UI\_VM\_RUN\_PAUSE 3**

VM\_RUN\_PAUSE: Program still "active", but someone has asked us to pause

**8.1.2.1846 #define UI\_VM\_RUN\_SINGLE 2**

VM\_RUN\_SINGLE: Run exactly one instruction per timeslice

**8.1.2.1847 #define UIModuleID 0x000C0001**

The Ui module ID

**8.1.2.1848 #define UIModuleName "Ui.mod"**

The Ui module name

**8.1.2.1849 #define UINT\_MAX 65535**

The maximum value of the unsigned int type

**8.1.2.1850 #define UIOffsetAbortFlag 40**

RW - Long Abort (true == use long press to abort) (1 byte)

**8.1.2.1851 #define UIOffsetBatteryState 30**

W - Battery state (0..4 capacity) (1 byte)

**8.1.2.1852 #define UIOffsetBatteryVoltage 4**

R - Battery voltage in millivolts (2 bytes)

**8.1.2.1853 #define UIOffsetBluetoothState 31**

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)

**8.1.2.1854 #define UIOffsetButton 28**

RW - Insert button (buttons enumerated above) (1 byte)

**8.1.2.1855 #define UIOffsetError 37**

W - Error code (1 byte)

**8.1.2.1856 #define UIOffsetFlags 26**

RW - Update command flags (flags enumerated above) (1 byte)

**8.1.2.1857 #define UIOffsetForceOff 39**

W - Force off (> 0 = off) (1 byte)

**8.1.2.1858 #define UIOffsetLMSfilename 6**

W - LMS filename to execute (Try It) (20 bytes)

**8.1.2.1859 #define UIOffsetOBPPointer 38**

W - Actual OBP step (0 - 4) (1 byte)

**8.1.2.1860 #define UIOffsetPMenu 0**

W - Pointer to menu file (4 bytes)

**8.1.2.1861 #define UIOffsetRechargeable 35**

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

**8.1.2.1862 #define UIOffsetRunState 29**

W - VM Run state (0 = stopped, 1 = running) (1 byte)

**8.1.2.1863 #define UIOffsetSleepTimeout 33**

RW - Sleep timeout time (min) (1 byte)

**8.1.2.1864 #define UIOffsetSleepTimer 34**

RW - Sleep timer (min) (1 byte)

**8.1.2.1865 #define UIOffsetState 27**

RW - UI state (states enumerated above) (1 byte)

**8.1.2.1866 #define UIOffsetUsbState 32**

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

**8.1.2.1867 #define UIOffsetVolume 36**

RW - Volume used in UI (0 - 4) (1 byte)

**8.1.2.1868 #define ULONG\_MAX 4294967295**

The maximum value of the unsigned long type

**8.1.2.1869 #define UpdateCalibCacheInfo 43**

Update sensor calibration cache information

**8.1.2.1870 #define UpdateFlagsField 0**

Update flags field. Contains a combination of the update flag constants. Read/write. Use [UF\\_UPDATE\\_MODE](#), [UF\\_UPDATE\\_SPEED](#), [UF\\_UPDATE\\_TACHO\\_LIMIT](#), and [UF\\_UPDATE\\_PID\\_VALUES](#) along with other fields to commit changes to the state of outputs. Set the appropriate flags after setting one or more of the output fields in order for the changes to actually go into affect.

**8.1.2.1871 #define US\_CMD\_CONTINUOUS 0x02**

Command to put the ultrasonic sensor into continuous polling mode (default)

**8.1.2.1872 #define US\_CMD\_EVENTCAPTURE 0x03**

Command to put the ultrasonic sensor into event capture mode

**8.1.2.1873 #define US\_CMD\_OFF 0x00**

Command to turn off the ultrasonic sensor

**Examples:**

[ex\\_writei2cregister.nxc](#).

**8.1.2.1874 #define US\_CMD\_SINGLESHOT 0x01**

Command to put the ultrasonic sensor into single shot mode

**8.1.2.1875 #define US\_CMD\_WARMRESET 0x04**

Command to warm reset the ultrasonic sensor

**8.1.2.1876 #define US\_REG\_ACTUAL\_ZERO 0x50**

The register address used to store the actual zero value

**8.1.2.1877 #define US\_REG\_CM\_INTERVAL 0x40**

The register address used to store the CM interval

**8.1.2.1878 #define US\_REG\_FACTORY\_ACTUAL\_ZERO 0x11**

The register address containing the factory setting for the actual zero value

**8.1.2.1879 #define US\_REG\_FACTORY\_SCALE\_DIVISOR 0x13**

The register address containing the factory setting for the scale divisor value

**8.1.2.1880 #define US\_REG\_FACTORY\_SCALE\_FACTOR 0x12**

The register address containing the factory setting for the scale factor value

**8.1.2.1881 #define US\_REG\_MEASUREMENT\_UNITS 0x14**

The register address containing the measurement units (degrees C or F)

**8.1.2.1882 #define US\_REG\_SCALE\_DIVISOR 0x52**

The register address used to store the scale divisor value

**8.1.2.1883 #define US\_REG\_SCALE\_FACTOR 0x51**

The register address used to store the scale factor value

**8.1.2.1884 #define USB\_CMD\_READY 0x01**

A constant representing usb direct command

**8.1.2.1885 #define USB\_PROTOCOL\_OVERHEAD 2**

Size of USB Overhead in bytes -- Command type byte + Command

**8.1.2.1886 #define USHRT\_MAX 65535**

The maximum value of the unsigned short type

**8.1.2.1887 #define WriteSemData 41**

Write motor semaphore data

**8.1.2.1888 #define XG1300L\_REG\_2G 0x61**

Select +/- 2G accelerometer range.

**8.1.2.1889 #define XG1300L\_REG\_4G 0x62**

Select +/- 4G accelerometer range.

**8.1.2.1890 #define XG1300L\_REG\_8G 0x63**

Select +/- 8G accelerometer range.

**8.1.2.1891 #define XG1300L\_REG\_ANGLE 0x42**

Read accumulated angle (2 bytes little endian) in 1/100s of degrees.

**8.1.2.1892 #define XG1300L\_REG\_RESET 0x60**

Reset the XG1300L device.

**8.1.2.1893 #define XG1300L\_REG\_TURNRATE 0x44**

Read rate of turn (2 bytes little endian) in 1/100s of degrees/second.

**8.1.2.1894 #define XG1300L\_REG\_XAXIS 0x46**

Read x-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**8.1.2.1895 #define XG1300L\_REG\_YAXIS 0x48**

Read y-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**8.1.2.1896 #define XG1300L\_REG\_ZAXIS 0x4A**

Read z-axis acceleration (2 bytes little endian) in  $m/s^2$  scaled by  $100/ACC\_RANGE*2$ , where  $ACC\_RANGE$  is 2, 4, or 8.

**8.1.2.1897 #define XG1300L\_SCALE\_2G 0x01**

Select +/- 2G accelerometer range.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.1.2.1898 #define XG1300L\_SCALE\_4G 0x02**

Select +/- 4G accelerometer range.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.1.2.1899 #define XG1300L\_SCALE\_8G 0x04**

Select +/- 8G accelerometer range.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.2 NXCAPIDocs.h File Reference**

Additional documentation for the NXC API. `#include "NXCDefs.h"`

**8.2.1 Detailed Description**

Additional documentation for the NXC API. [NXCAPIDocs.h](#) contains additional documentation for the NXC API

License:

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of this code is John Hansen. Portions created by John Hansen are Copyright (C) 2009-2011 John Hansen. All Rights Reserved.

-----  
**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2011-10-16

**Version:**

22

## 8.3 NXCDefs.h File Reference

Constants, macros, and API functions for NXC. `#include "NBCCCommon.h"`

### Data Structures

- struct [ColorSensorReadType](#)  
*Parameters for the ColorSensorRead system call.*
- struct [InputValuesType](#)  
*Parameters for the RemoteGetInputValues function.*
- struct [InputPinFunctionType](#)  
*Parameters for the InputPinFunction system call.*
- struct [OutputStateType](#)  
*Parameters for the RemoteGetOutputState function.*
- struct [LocationType](#)  
*A point on the NXT LCD screen.*
- struct [SizeType](#)  
*Width and height dimensions for the DrawRect system call.*
- struct [DrawTextType](#)  
*Parameters for the DrawText system call.*
- struct [DrawPointType](#)  
*Parameters for the DrawPoint system call.*
- struct [DrawLineType](#)  
*Parameters for the DrawLine system call.*
- struct [DrawCircleType](#)  
*Parameters for the DrawCircle system call.*
- struct [DrawRectType](#)  
*Parameters for the DrawRect system call.*
- struct [DrawGraphicType](#)  
*Parameters for the DrawGraphic system call.*

- struct [SetScreenModeType](#)  
*Parameters for the SetScreenMode system call.*
- struct [DisplayExecuteFunctionType](#)  
*Parameters for the DisplayExecuteFunction system call.*
- struct [DrawGraphicArrayType](#)  
*Parameters for the DrawGraphicArray system call.*
- struct [DrawPolygonType](#)  
*Parameters for the DrawPolygon system call.*
- struct [DrawEllipseType](#)  
*Parameters for the DrawEllipse system call.*
- struct [DrawFontType](#)  
*Parameters for the DrawFont system call.*
- struct [Tone](#)  
*Type used with the PlayTones API function.*
- struct [SoundPlayFileType](#)  
*Parameters for the SoundPlayFile system call.*
- struct [SoundPlayToneType](#)  
*Parameters for the SoundPlayTone system call.*
- struct [SoundGetStateType](#)  
*Parameters for the SoundGetState system call.*
- struct [SoundSetStateType](#)  
*Parameters for the SoundSetState system call.*
- struct [CommLSWriteType](#)  
*Parameters for the CommLSWrite system call.*
- struct [CommLSReadType](#)  
*Parameters for the CommLSRead system call.*
- struct [CommLSCheckStatusType](#)  
*Parameters for the CommLSCheckStatus system call.*

- struct [CommLSWriteExType](#)  
*Parameters for the CommLSWriteEx system call.*
- struct [GetStartTickType](#)  
*Parameters for the GetStartTick system call.*
- struct [KeepAliveType](#)  
*Parameters for the KeepAlive system call.*
- struct [IOMapReadType](#)  
*Parameters for the IOMapRead system call.*
- struct [IOMapWriteType](#)  
*Parameters for the IOMapWrite system call.*
- struct [IOMapReadByIDType](#)  
*Parameters for the IOMapReadByID system call.*
- struct [IOMapWriteByIDType](#)  
*Parameters for the IOMapWriteByID system call.*
- struct [DatalogWriteType](#)  
*Parameters for the DatalogWrite system call.*
- struct [DatalogGetTimesType](#)  
*Parameters for the DatalogGetTimes system call.*
- struct [ReadSemDataType](#)  
*Parameters for the ReadSemData system call.*
- struct [WriteSemDataType](#)  
*Parameters for the WriteSemData system call.*
- struct [UpdateCalibCacheInfoType](#)  
*Parameters for the UpdateCalibCacheInfo system call.*
- struct [ComputeCalibValueType](#)  
*Parameters for the ComputeCalibValue system call.*
- struct [MemoryManagerType](#)  
*Parameters for the MemoryManager system call.*

- struct [ReadLastResponseType](#)  
*Parameters for the ReadLastResponse system call.*
- struct [MessageWriteType](#)  
*Parameters for the MessageWrite system call.*
- struct [MessageReadType](#)  
*Parameters for the MessageRead system call.*
- struct [CommBTCheckStatusType](#)  
*Parameters for the CommBTCheckStatus system call.*
- struct [CommBTWriteType](#)  
*Parameters for the CommBTWrite system call.*
- struct [JoystickMessageType](#)  
*The JoystickMessageType structure.*
- struct [CommExecuteFunctionType](#)  
*Parameters for the CommExecuteFunction system call.*
- struct [CommHSControlType](#)  
*Parameters for the CommHSControl system call.*
- struct [CommHSCheckStatusType](#)  
*Parameters for the CommHSCheckStatus system call.*
- struct [CommHSReadWriteType](#)  
*Parameters for the CommHSReadWrite system call.*
- struct [CommBTOnOffType](#)  
*Parameters for the CommBTOnOff system call.*
- struct [CommBTConnectionType](#)  
*Parameters for the CommBTConnection system call.*
- struct [ReadButtonType](#)  
*Parameters for the ReadButton system call.*
- struct [SetSleepTimeoutType](#)  
*Parameters for the SetSleepTimeout system call.*

- struct [FileOpenType](#)  
*Parameters for the FileOpen system call.*
- struct [FileReadWriteType](#)  
*Parameters for the FileReadWrite system call.*
- struct [FileCloseType](#)  
*Parameters for the FileClose system call.*
- struct [FileResolveHandleType](#)  
*Parameters for the FileResolveHandle system call.*
- struct [FileRenameType](#)  
*Parameters for the FileRename system call.*
- struct [FileDeleteType](#)  
*Parameters for the FileDelete system call.*
- struct [LoaderExecuteFunctionType](#)  
*Parameters for the LoaderExecuteFunction system call.*
- struct [FileFindType](#)  
*Parameters for the FileFind system call.*
- struct [FileSeekType](#)  
*Parameters for the FileSeek system call.*
- struct [FileResizeType](#)  
*Parameters for the FileResize system call.*
- struct [FileTellType](#)  
*Parameters for the FileTell system call.*
- struct [ListFilesType](#)  
*Parameters for the ListFiles system call.*
- struct [XGPacketType](#)  
*Parameters for the [ReadSensorMIXG1300L](#) function.*
- struct [VectorType](#)  
*This structure is used for storing three axis values in a single object.*

- struct [RandomNumberType](#)  
*Parameters for the RandomNumber system call.*
- struct [RandomExType](#)  
*Parameters for the RandomEx system call.*
- struct [div\\_t](#)  
*Output type of the div function.*
- struct [ldiv\\_t](#)  
*Output type of the ldiv function.*

### Defines

- #define [u8](#) unsigned char
- #define [s8](#) char
- #define [u16](#) unsigned int
- #define [s16](#) int
- #define [u32](#) unsigned long
- #define [s32](#) long
- #define [S1](#) 0
- #define [S2](#) 1
- #define [S3](#) 2
- #define [S4](#) 3
- #define [SENSOR\\_TYPE\\_NONE](#) IN\_TYPE\_NO\_SENSOR
- #define [SENSOR\\_TYPE\\_TOUCH](#) IN\_TYPE\_SWITCH
- #define [SENSOR\\_TYPE\\_TEMPERATURE](#) IN\_TYPE\_TEMPERATURE
- #define [SENSOR\\_TYPE\\_LIGHT](#) IN\_TYPE\_REFLECTION
- #define [SENSOR\\_TYPE\\_ROTATION](#) IN\_TYPE\_ANGLE
- #define [SENSOR\\_TYPE\\_LIGHT\\_ACTIVE](#) IN\_TYPE\_LIGHT\_ACTIVE
- #define [SENSOR\\_TYPE\\_LIGHT\\_INACTIVE](#) IN\_TYPE\_LIGHT\_INACTIVE
- #define [SENSOR\\_TYPE\\_SOUND\\_DB](#) IN\_TYPE\_SOUND\_DB
- #define [SENSOR\\_TYPE\\_SOUND\\_DBA](#) IN\_TYPE\_SOUND\_DBA
- #define [SENSOR\\_TYPE\\_CUSTOM](#) IN\_TYPE\_CUSTOM
- #define [SENSOR\\_TYPE\\_LOWSPEED](#) IN\_TYPE\_LOWSPEED
- #define [SENSOR\\_TYPE\\_LOWSPEED\\_9V](#) IN\_TYPE\_LOWSPEED\_9V
- #define [SENSOR\\_TYPE\\_HIGHSPEED](#) IN\_TYPE\_HISPEED
- #define [SENSOR\\_TYPE\\_COLORFULL](#) IN\_TYPE\_COLORFULL
- #define [SENSOR\\_TYPE\\_COLORRED](#) IN\_TYPE\_COLORRED
- #define [SENSOR\\_TYPE\\_COLORGREEN](#) IN\_TYPE\_COLORGREEN
- #define [SENSOR\\_TYPE\\_COLORBLUE](#) IN\_TYPE\_COLORBLUE

- #define `SENSOR_TYPE_COLORNONE` `IN_TYPE_COLORNONE`
- #define `SENSOR_MODE_RAW` `IN_MODE_RAW`
- #define `SENSOR_MODE_BOOL` `IN_MODE_BOOLEAN`
- #define `SENSOR_MODE_EDGE` `IN_MODE_TRANSITIONCNT`
- #define `SENSOR_MODE_PULSE` `IN_MODE_PERIODCOUNTER`
- #define `SENSOR_MODE_PERCENT` `IN_MODE_PCTFULLSCALE`
- #define `SENSOR_MODE_CELSIUS` `IN_MODE_CELSIUS`
- #define `SENSOR_MODE_FAHRENHEIT` `IN_MODE_FAHRENHEIT`
- #define `SENSOR_MODE_ROTATION` `IN_MODE_ANGLESTEP`
- #define `_SENSOR_CFG(_type, _mode)` `(((_type)<<8)+(_mode))`
- #define `SENSOR_TOUCH` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_BOOL)`
- #define `SENSOR_LIGHT` `_SENSOR_CFG(SENSOR_TYPE_LIGHT, SENSOR_MODE_PERCENT)`
- #define `SENSOR_ROTATION` `_SENSOR_CFG(SENSOR_TYPE_ROTATION, SENSOR_MODE_ROTATION)`
- #define `SENSOR_CELSIUS` `_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_CELSIUS)`
- #define `SENSOR_FAHRENHEIT` `_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_FAHRENHEIT)`
- #define `SENSOR_PULSE` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_PULSE)`
- #define `SENSOR_EDGE` `_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_EDGE)`
- #define `SENSOR_NXTLIGHT` `_SENSOR_CFG(SENSOR_TYPE_LIGHT_ACTIVE, SENSOR_MODE_PERCENT)`
- #define `SENSOR_SOUND` `_SENSOR_CFG(SENSOR_TYPE_SOUND_DB, SENSOR_MODE_PERCENT)`
- #define `SENSOR_LOWSPEED_9V` `_SENSOR_CFG(SENSOR_TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)`
- #define `SENSOR_LOWSPEED` `_SENSOR_CFG(SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW)`
- #define `SENSOR_COLORFULL` `_SENSOR_CFG(SENSOR_TYPE_COLORFULL, SENSOR_MODE_RAW)`
- #define `SENSOR_COLORRED` `_SENSOR_CFG(SENSOR_TYPE_COLORRED, SENSOR_MODE_PERCENT)`
- #define `SENSOR_COLORGREEN` `_SENSOR_CFG(SENSOR_TYPE_COLORGREEN, SENSOR_MODE_PERCENT)`
- #define `SENSOR_COLORBLUE` `_SENSOR_CFG(SENSOR_TYPE_COLORBLUE, SENSOR_MODE_PERCENT)`
- #define `SENSOR_COLORNONE` `_SENSOR_CFG(SENSOR_TYPE_COLORNONE, SENSOR_MODE_PERCENT)`
- #define `SENSOR_1` `Sensor(S1)`
- #define `SENSOR_2` `Sensor(S2)`

- #define [SENSOR\\_3](#) Sensor(S3)
- #define [SENSOR\\_4](#) Sensor(S4)
- #define [LT](#) 0x00
- #define [GT](#) 0x01
- #define [LTEQ](#) 0x02
- #define [GTEQ](#) 0x03
- #define [EQ](#) 0x04
- #define [NEQ](#) 0x05
- #define [Sqrt](#)(\_X) asm { sqrt \_\_FLTRETVAL\_\_, \_X }  
*Compute square root.*
- #define [Sin](#)(\_X) asm { sin \_\_FLTRETVAL\_\_, \_X }  
*Compute sine.*
- #define [Cos](#)(\_X) asm { cos \_\_FLTRETVAL\_\_, \_X }  
*Compute cosine.*
- #define [Asin](#)(\_X) asm { asin \_\_FLTRETVAL\_\_, \_X }  
*Compute arc sine.*
- #define [Acos](#)(\_X) asm { acos \_\_FLTRETVAL\_\_, \_X }  
*Compute arc cosine.*
- #define [Atan](#)(\_X) asm { atan \_\_FLTRETVAL\_\_, \_X }  
*Compute arc tangent.*
- #define [Ceil](#)(\_X) asm { ceil \_\_FLTRETVAL\_\_, \_X }  
*Round up value.*
- #define [Exp](#)(\_X) asm { exp \_\_FLTRETVAL\_\_, \_X }  
*Compute exponential function .*
- #define [Floor](#)(\_X) asm { floor \_\_FLTRETVAL\_\_, \_X }  
*Round down value.*
- #define [Tan](#)(\_X) asm { tan \_\_FLTRETVAL\_\_, \_X }  
*Compute tangent.*
- #define [Tanh](#)(\_X) asm { tanh \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic tangent.*
- #define [Cosh](#)(\_X) asm { cosh \_\_FLTRETVAL\_\_, \_X }

*Compute hyperbolic cosine.*

- #define **Sinh**(\_X) asm { sinh \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic sine.*
- #define **Log**(\_X) asm { log \_\_FLTRETVAL\_\_, \_X }  
*Compute natural logarithm.*
- #define **Log10**(\_X) asm { log10 \_\_FLTRETVAL\_\_, \_X }  
*Compute common logarithm.*
- #define **Atan2**(\_Y, \_X) asm { atan2 \_\_FLTRETVAL\_\_, \_Y, \_X }  
*Compute arc tangent with 2 parameters.*
- #define **Pow**(\_Base, \_Exponent) asm { pow \_\_FLTRETVAL\_\_, \_Base, \_Exponent }  
*Raise to power.*
- #define **Trunc**(\_X) asm { trunc \_\_RETVAL\_\_, \_X }  
*Compute integral part.*
- #define **Frac**(\_X) asm { frac \_\_FLTRETVAL\_\_, \_X }  
*Compute fractional part.*
- #define **MulDiv32**(\_A, \_B, \_C) asm { muldiv \_\_RETVAL\_\_, \_A, \_B, \_C }  
*Multiply and divide.*
- #define **SinD**(\_X) asm { sind \_\_FLTRETVAL\_\_, \_X }  
*Compute sine (degrees).*
- #define **CosD**(\_X) asm { cosd \_\_FLTRETVAL\_\_, \_X }  
*Compute cosine (degrees).*
- #define **AsinD**(\_X) asm { asind \_\_FLTRETVAL\_\_, \_X }  
*Compute arch sine (degrees).*
- #define **AcosD**(\_X) asm { acosd \_\_FLTRETVAL\_\_, \_X }  
*Compute arc cosine (degrees).*
- #define **AtanD**(\_X) asm { atand \_\_FLTRETVAL\_\_, \_X }  
*Compute arc tangent (degrees).*
- #define **TanD**(\_X) asm { tand \_\_FLTRETVAL\_\_, \_X }

*Compute tangent (degrees).*

- #define **TanhD**(\_X) asm { tanhd \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic tangent (degrees).*
- #define **CoshD**(\_X) asm { coshd \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic cosine (degrees).*
- #define **SinhD**(\_X) asm { sinhd \_\_FLTRETVAL\_\_, \_X }  
*Compute hyperbolic sine (degrees).*
- #define **Atan2D**(\_Y, \_X) asm { atan2d \_\_FLTRETVAL\_\_, \_Y, \_X }  
*Compute arc tangent with two parameters (degrees).*
- #define **getc**(\_handle) fgetc(\_handle)  
*Get character from file.*
- #define **putc**(\_ch, \_handle) fputc(\_ch, \_handle)  
*Write character to file.*
- #define **SEEK\_SET** 0
- #define **SEEK\_CUR** 1
- #define **SEEK\_END** 2
- #define **RICSetValue**(\_data, \_idx, \_newval) \_data[(\_idx)] = (\_newval)&0xFF;  
\_data[(\_idx)+1] = (\_newval)>>8  
*Set the value of an element in an RIC data array.*

## Functions

- void **SetSensorType** (const byte &port, byte type)  
*Set sensor type.*
- void **SetSensorMode** (const byte &port, byte mode)  
*Set sensor mode.*
- void **ClearSensor** (const byte &port)  
*Clear a sensor value.*
- void **ResetSensor** (const byte &port)  
*Reset the sensor port.*

- void [SetSensor](#) (const byte &port, const unsigned int config)  
*Set sensor configuration.*
- void [SetSensorTouch](#) (const byte &port)  
*Configure a touch sensor.*
- void [SetSensorLight](#) (const byte &port, bool bActive=true)  
*Configure a light sensor.*
- void [SetSensorSound](#) (const byte &port, bool bdBScaling=true)  
*Configure a sound sensor.*
- void [SetSensorLowspeed](#) (const byte &port, bool bIsPowered=true)  
*Configure an I2C sensor.*
- void [SetSensorUltrasonic](#) (const byte &port)  
*Configure an ultrasonic sensor.*
- void [SetSensorEMeter](#) (const byte &port)  
*Configure an EMeter sensor.*
- void [SetSensorTemperature](#) (const byte &port)  
*Configure a temperature sensor.*
- void [SetSensorColorFull](#) (const byte &port)  
*Configure an NXT 2.0 full color sensor.*
- void [SetSensorColorRed](#) (const byte &port)  
*Configure an NXT 2.0 red light sensor.*
- void [SetSensorColorGreen](#) (const byte &port)  
*Configure an NXT 2.0 green light sensor.*
- void [SetSensorColorBlue](#) (const byte &port)  
*Configure an NXT 2.0 blue light sensor.*
- void [SetSensorColorNone](#) (const byte &port)  
*Configure an NXT 2.0 no light sensor.*
- variant [GetInput](#) (const byte &port, const byte field)  
*Get an input field value.*

- void [SetInput](#) (const byte &port, const int field, variant value)  
*Set an input field value.*
- unsigned int [Sensor](#) (const byte &port)  
*Read sensor scaled value.*
- bool [SensorBoolean](#) (const byte port)  
*Read sensor boolean value.*
- byte [SensorDigiPinsDirection](#) (const byte port)  
*Read sensor digital pins direction.*
- byte [SensorDigiPinsOutputLevel](#) (const byte port)  
*Read sensor digital pins output level.*
- byte [SensorDigiPinsStatus](#) (const byte port)  
*Read sensor digital pins status.*
- bool [SensorInvalid](#) (const byte &port)  
*Read sensor invalid data flag.*
- byte [SensorMode](#) (const byte &port)  
*Read sensor mode.*
- unsigned int [SensorNormalized](#) (const byte &port)  
*Read sensor normalized value.*
- unsigned int [SensorRaw](#) (const byte &port)  
*Read sensor raw value.*
- unsigned int [SensorScaled](#) (const byte &port)  
*Read sensor scaled value.*
- byte [SensorType](#) (const byte &port)  
*Read sensor type.*
- unsigned int [SensorValue](#) (const byte &port)  
*Read sensor scaled value.*
- bool [SensorValueBool](#) (const byte port)  
*Read sensor boolean value.*

- unsigned int [SensorValueRaw](#) (const byte &port)  
*Read sensor raw value.*
- byte [CustomSensorActiveStatus](#) (byte port)  
*Get the custom sensor active status.*
- byte [CustomSensorPercentFullScale](#) (byte port)  
*Get the custom sensor percent full scale.*
- unsigned int [CustomSensorZeroOffset](#) (byte port)  
*Get the custom sensor zero offset.*
- void [SetCustomSensorActiveStatus](#) (byte port, byte activeStatus)  
*Set active status.*
- void [SetCustomSensorPercentFullScale](#) (byte port, byte pctFullScale)  
*Set percent full scale.*
- void [SetCustomSensorZeroOffset](#) (byte port, int zeroOffset)  
*Set custom zero offset.*
- void [SetSensorBoolean](#) (byte port, bool value)  
*Set sensor boolean value.*
- void [SetSensorDigiPinsDirection](#) (byte port, byte direction)  
*Set digital pins direction.*
- void [SetSensorDigiPinsOutputLevel](#) (byte port, byte outputLevel)  
*Set digital pins output level.*
- void [SetSensorDigiPinsStatus](#) (byte port, byte status)  
*Set digital pins status.*
- void [SysColorSensorRead](#) ([ColorSensorReadType](#) &args)  
*Read LEGO color sensor.*
- int [ReadSensorColorEx](#) (const byte &port, int &colorval, unsigned int &raw[ ], unsigned int &norm[ ], int &scaled[ ])  
*Read LEGO color sensor extra.*
- int [ReadSensorColorRaw](#) (const byte &port, unsigned int &rawVals[ ])  
*Read LEGO color sensor raw values.*

- unsigned int [ColorADRaw](#) (byte port, byte color)  
*Read a LEGO color sensor AD raw value.*
- bool [ColorBoolean](#) (byte port, byte color)  
*Read a LEGO color sensor boolean value.*
- long [ColorCalibration](#) (byte port, byte point, byte color)  
*Read a LEGO color sensor calibration point value.*
- byte [ColorCalibrationState](#) (byte port)  
*Read LEGO color sensor calibration state.*
- unsigned int [ColorCalLimits](#) (byte port, byte point)  
*Read a LEGO color sensor calibration limit value.*
- unsigned int [ColorSensorRaw](#) (byte port, byte color)  
*Read a LEGO color sensor raw value.*
- unsigned int [ColorSensorValue](#) (byte port, byte color)  
*Read a LEGO color sensor scaled value.*
- void [SysInputPinFunction](#) ([InputPinFunctionType](#) &args)  
*Execute the Input module pin function.*
- void [SetMotorPwnFreq](#) (byte n)  
*Set motor regulation frequency.*
- void [SetMotorRegulationTime](#) (byte n)  
*Set regulation time.*
- void [SetMotorRegulationOptions](#) (byte n)  
*Set regulation options.*
- void [OnFwdSyncPID](#) (byte outputs, char pwr, char turnpct, byte p, byte i, byte d)  
*Run motors forward synchronised with PID factors.*
- void [OnFwdSyncExPID](#) (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d)  
*Run motors forward synchronised and reset counters with PID factors.*

- void [OnRevSyncPID](#) (byte outputs, char pwr, char turnpct, byte p, byte i, byte d)  
*Run motors backward synchronised with PID factors.*
- void [OnRevSyncExPID](#) (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d)  
*Run motors backward synchronised and reset counters with PID factors.*
- void [OnFwdRegPID](#) (byte outputs, char pwr, byte regmode, byte p, byte i, byte d)  
*Run motors forward regulated with PID factors.*
- void [OnFwdRegExPID](#) (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d)  
*Run motors forward regulated and reset counters with PID factors.*
- void [OnRevRegPID](#) (byte outputs, char pwr, byte regmode, byte p, byte i, byte d)  
*Run motors reverse regulated with PID factors.*
- void [OnRevRegExPID](#) (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d)  
*Run motors backward regulated and reset counters with PID factors.*
- void [Off](#) (byte outputs)  
*Turn motors off.*
- void [OffEx](#) (byte outputs, const byte reset)  
*Turn motors off and reset counters.*
- void [Coast](#) (byte outputs)  
*Coast motors.*
- void [CoastEx](#) (byte outputs, const byte reset)  
*Coast motors and reset counters.*
- void [Float](#) (byte outputs)  
*Float motors.*
- void [OnFwd](#) (byte outputs, char pwr)  
*Run motors forward.*
- void [OnFwdEx](#) (byte outputs, char pwr, const byte reset)

*Run motors forward and reset counters.*

- void **OnRev** (byte outputs, char pwr)  
*Run motors backward.*
- void **OnRevEx** (byte outputs, char pwr, const byte reset)  
*Run motors backward and reset counters.*
- void **OnFwdReg** (byte outputs, char pwr, byte regmode)  
*Run motors forward regulated.*
- void **OnFwdRegEx** (byte outputs, char pwr, byte regmode, const byte reset)  
*Run motors forward regulated and reset counters.*
- void **OnRevReg** (byte outputs, char pwr, byte regmode)  
*Run motors forward regulated.*
- void **OnRevRegEx** (byte outputs, char pwr, byte regmode, const byte reset)  
*Run motors backward regulated and reset counters.*
- void **OnFwdSync** (byte outputs, char pwr, char turnpct)  
*Run motors forward synchronised.*
- void **OnFwdSyncEx** (byte outputs, char pwr, char turnpct, const byte reset)  
*Run motors forward synchronised and reset counters.*
- void **OnRevSync** (byte outputs, char pwr, char turnpct)  
*Run motors backward synchronised.*
- void **OnRevSyncEx** (byte outputs, char pwr, char turnpct, const byte reset)  
*Run motors backward synchronised and reset counters.*
- void **RotateMotor** (byte outputs, char pwr, long angle)  
*Rotate motor.*
- void **RotateMotorPID** (byte outputs, char pwr, long angle, byte p, byte i, byte d)  
*Rotate motor with PID factors.*
- void **RotateMotorEx** (byte outputs, char pwr, long angle, char turnpct, bool sync, bool stop)  
*Rotate motor.*

- void [RotateMotorExpID](#) (byte outputs, char pwr, long angle, char turnpct, bool sync, bool stop, byte p, byte i, byte d)  
*Rotate motor.*
- void [ResetTachoCount](#) (byte outputs)  
*Reset tachometer counter.*
- void [ResetBlockTachoCount](#) (byte outputs)  
*Reset block-relative counter.*
- void [ResetRotationCount](#) (byte outputs)  
*Reset program-relative counter.*
- void [ResetAllTachoCounts](#) (byte outputs)  
*Reset all tachometer counters.*
- void [SetOutput](#) (byte outputs, byte field1, variant val1,..., byte fieldN, variant valN)  
*Set output fields.*
- variant [GetOutput](#) (byte output, const byte field)  
*Get output field value.*
- byte [MotorMode](#) (byte output)  
*Get motor mode.*
- char [MotorPower](#) (byte output)  
*Get motor power level.*
- char [MotorActualSpeed](#) (byte output)  
*Get motor actual speed.*
- long [MotorTachoCount](#) (byte output)  
*Get motor tachometer counter.*
- long [MotorTachoLimit](#) (byte output)  
*Get motor tachometer limit.*
- byte [MotorRunState](#) (byte output)  
*Get motor run state.*
- char [MotorTurnRatio](#) (byte output)  
*Get motor turn ratio.*

- byte [MotorRegulation](#) (byte output)  
*Get motor regulation mode.*
- bool [MotorOverload](#) (byte output)  
*Get motor overload status.*
- byte [MotorRegPValue](#) (byte output)  
*Get motor P value.*
- byte [MotorRegIValue](#) (byte output)  
*Get motor I value.*
- byte [MotorRegDValue](#) (byte output)  
*Get motor D value.*
- long [MotorBlockTachoCount](#) (byte output)  
*Get motor block-relative counter.*
- long [MotorRotationCount](#) (byte output)  
*Get motor program-relative counter.*
- byte [MotorOutputOptions](#) (byte output)  
*Get motor options.*
- byte [MotorMaxSpeed](#) (byte output)  
*Get motor max speed.*
- byte [MotorMaxAcceleration](#) (byte output)  
*Get motor max acceleration.*
- byte [MotorPwnFreq](#) ()  
*Get motor regulation frequency.*
- byte [MotorRegulationTime](#) ()  
*Get motor regulation time.*
- byte [MotorRegulationOptions](#) ()  
*Get motor regulation options.*
- void [ResetScreen](#) ()  
*Reset LCD screen.*

- char **CircleOut** (int x, int y, byte radius, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a circle.*
- char **LineOut** (int x1, int y1, int x2, int y2, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a line.*
- char **PointOut** (int x, int y, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a point.*
- char **RectOut** (int x, int y, int width, int height, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a rectangle.*
- char **TextOut** (int x, int y, string str, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw text.*
- char **NumOut** (int x, int y, variant value, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a number.*
- char **EllipseOut** (int x, int y, byte radiusX, byte radiusY, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw an ellipse.*
- char **PolyOut** (**LocationType** points[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a polygon.*
- char **FontTextOut** (int x, int y, string filename, string str, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw text with font.*
- char **FontNumOut** (int x, int y, string filename, variant value, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a number with font.*
- char **GraphicOut** (int x, int y, string filename, unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image.*

- char [GraphicArrayOut](#) (int x, int y, byte data[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image from byte array.*
- char [GraphicOutEx](#) (int x, int y, string filename, byte vars[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image with parameters.*
- char [GraphicArrayOutEx](#) (int x, int y, byte data[ ], byte vars[ ], unsigned long options=DRAW\_OPT\_NORMAL)  
*Draw a graphic image from byte array with parameters.*
- void [GetDisplayNormal](#) (const byte x, const byte line, unsigned int cnt, byte &data[ ])  
*Read pixel data from the normal display buffer.*
- void [SetDisplayNormal](#) (const byte x, const byte line, unsigned int cnt, byte data[ ])  
*Write pixel data to the normal display buffer.*
- void [GetDisplayPopup](#) (const byte x, const byte line, unsigned int cnt, byte &data[ ])  
*Read pixel data from the popup display buffer.*
- void [SetDisplayPopup](#) (const byte x, const byte line, unsigned int cnt, byte data[ ])  
*Write pixel data to the popup display buffer.*
- unsigned long [DisplayEraseMask](#) ()  
*Read the display erase mask value.*
- unsigned long [DisplayUpdateMask](#) ()  
*Read the display update mask value.*
- unsigned long [DisplayFont](#) ()  
*Read the display font memory address.*
- unsigned long [DisplayDisplay](#) ()  
*Read the display memory address.*
- byte [DisplayFlags](#) ()  
*Read the display flags.*

- byte `DisplayTextLinesCenterFlags` ()  
*Read the display text lines center flags.*
- void `SysDrawText` (`DrawTextType` &args)  
*Draw text.*
- void `SysDrawPoint` (`DrawPointType` &args)  
*Draw a point.*
- void `SysDrawLine` (`DrawLineType` &args)  
*Draw a line.*
- void `SysDrawCircle` (`DrawCircleType` &args)  
*Draw a circle.*
- void `SysDrawRect` (`DrawRectType` &args)  
*Draw a rectangle.*
- void `SysDrawGraphic` (`DrawGraphicType` &args)  
*Draw a graphic (RIC file).*
- void `SysSetScreenMode` (`SetScreenModeType` &args)  
*Set the screen mode.*
- void `SysDisplayExecuteFunction` (`DisplayExecuteFunctionType` &args)  
*Execute any Display module command.*
- byte `DisplayContrast` ()  
*Read the display contrast setting.*
- void `SysDrawGraphicArray` (`DrawGraphicArrayType` &args)  
*Draw a graphic image from a byte array.*
- void `SysDrawPolygon` (`DrawPolygonType` &args)  
*Draw a polygon.*
- void `SysDrawEllipse` (`DrawEllipseType` &args)  
*Draw an ellipse.*
- void `SysDrawFont` (`DrawFontType` &args)  
*Draw text using a custom font.*

- void [ClearScreen](#) ()  
*Clear LCD screen.*
- void [ClearLine](#) (byte line)  
*Clear a line on the LCD screen.*
- void [SetDisplayFont](#) (unsigned long fontaddr)  
*Set the display font memory address.*
- void [SetDisplayDisplay](#) (unsigned long dispaddr)  
*Set the display memory address.*
- void [SetDisplayEraseMask](#) (unsigned long eraseMask)  
*Set the display erase mask.*
- void [SetDisplayFlags](#) (byte flags)  
*Set the display flags.*
- void [SetDisplayTextLinesCenterFlags](#) (byte ctrFlags)  
*Set the display text lines center flags.*
- void [SetDisplayUpdateMask](#) (unsigned long updateMask)  
*Set the display update mask.*
- void [SetDisplayContrast](#) (byte contrast)  
*Set the display contrast.*
- char [PlayFile](#) (string filename)  
*Play a file.*
- char [PlayFileEx](#) (string filename, byte volume, bool loop)  
*Play a file with extra options.*
- char [PlayTone](#) (unsigned int frequency, unsigned int duration)  
*Play a tone.*
- char [PlayToneEx](#) (unsigned int frequency, unsigned int duration, byte volume, bool loop)  
*Play a tone with extra options.*
- byte [SoundState](#) ()

*Get sound module state.*

- byte [SoundFlags](#) ()  
*Get sound module flags.*
- byte [StopSound](#) ()  
*Stop sound.*
- unsigned int [SoundFrequency](#) ()  
*Get sound frequency.*
- unsigned int [SoundDuration](#) ()  
*Get sound duration.*
- unsigned int [SoundSampleRate](#) ()  
*Get sample rate.*
- byte [SoundMode](#) ()  
*Get sound mode.*
- byte [SoundVolume](#) ()  
*Get volume.*
- void [SetSoundDuration](#) (unsigned int duration)  
*Set sound duration.*
- void [SetSoundFlags](#) (byte flags)  
*Set sound module flags.*
- void [SetSoundFrequency](#) (unsigned int frequency)  
*Set sound frequency.*
- void [SetSoundMode](#) (byte mode)  
*Set sound mode.*
- void [SetSoundModuleState](#) (byte state)  
*Set sound module state.*
- void [SetSoundSampleRate](#) (unsigned int sampleRate)  
*Set sample rate.*
- void [SetSoundVolume](#) (byte volume)

*Set sound volume.*

- void [SysSoundPlayFile](#) ([SoundPlayFileType](#) &args)  
*Play sound file.*
- void [SysSoundPlayTone](#) ([SoundPlayToneType](#) &args)  
*Play tone.*
- void [SysSoundGetState](#) ([SoundGetStateType](#) &args)  
*Get sound state.*
- void [SysSoundSetState](#) ([SoundSetStateType](#) &args)  
*Set sound state.*
- void [PlaySound](#) (const int &aCode)  
*Play a system sound.*
- void [PlayTones](#) ([Tone](#) tones[ ])  
*Play multiple tones.*
- byte [SensorUS](#) (const byte port)  
*Read ultrasonic sensor value.*
- char [ReadSensorUSEx](#) (const byte port, byte &values[ ])  
*Read multiple ultrasonic sensor values.*
- char [ReadSensorEMeter](#) (const byte &port, float &vIn, float &aIn, float &vOut, float &aOut, int &joules, float &wIn, float &wOut)  
*Read the LEGO EMeter values.*
- char [ConfigureTemperatureSensor](#) (const byte &port, const byte &config)  
*Configure LEGO Temperature sensor options.*
- float [SensorTemperature](#) (const byte &port)  
*Read the LEGO Temperature sensor value.*
- long [LowspeedStatus](#) (const byte port, byte &bytesready)  
*Get lowspeed status.*
- long [LowspeedCheckStatus](#) (const byte port)  
*Check lowspeed status.*
- byte [LowspeedBytesReady](#) (const byte port)

*Get lowspeed bytes ready.*

- long [LowspeedWrite](#) (const byte port, byte retlen, byte buffer[ ])  
*Write lowspeed data.*
- long [LowspeedRead](#) (const byte port, byte buflen, byte &buffer[ ])  
*Read lowspeed data.*
- long [I2CStatus](#) (const byte port, byte &bytesready)  
*Get I2C status.*
- long [I2CCheckStatus](#) (const byte port)  
*Check I2C status.*
- byte [I2CBytesReady](#) (const byte port)  
*Get I2C bytes ready.*
- long [I2CWrite](#) (const byte port, byte retlen, byte buffer[ ])  
*Write I2C data.*
- long [I2CRead](#) (const byte port, byte buflen, byte &buffer[ ])  
*Read I2C data.*
- long [I2CBytes](#) (const byte port, byte inbuf[ ], byte &count, byte &outbuf[ ])  
*Perform an I2C write/read transaction.*
- char [ReadI2CRegister](#) (byte port, byte i2caddr, byte reg, byte &out)  
*Read I2C register.*
- char [WriteI2CRegister](#) (byte port, byte i2caddr, byte reg, byte val)  
*Write I2C register.*
- string [I2CDeviceInfo](#) (byte port, byte i2caddr, byte info)  
*Read I2C device information.*
- string [I2CVersion](#) (byte port, byte i2caddr)  
*Read I2C device version.*
- string [I2CVendorId](#) (byte port, byte i2caddr)  
*Read I2C device vendor.*
- string [I2CDeviceId](#) (byte port, byte i2caddr)

*Read I2C device identifier.*

- long [I2CSendCommand](#) (byte port, byte i2caddr, byte cmd)  
*Send an I2C command.*
- void [GetLSInputBuffer](#) (const byte port, const byte offset, byte cnt, byte &data[ ])  
*Get I2C input buffer data.*
- void [GetLSOutputBuffer](#) (const byte port, const byte offset, byte cnt, byte &data[ ])  
*Get I2C output buffer data.*
- byte [LSInputBufferInPtr](#) (const byte port)  
*Get I2C input buffer in-pointer.*
- byte [LSInputBufferOutPtr](#) (const byte port)  
*Get I2C input buffer out-pointer.*
- byte [LSInputBufferBytesToRx](#) (const byte port)  
*Get I2C input buffer bytes to rx.*
- byte [LSOutputBufferInPtr](#) (const byte port)  
*Get I2C output buffer in-pointer.*
- byte [LSOutputBufferOutPtr](#) (const byte port)  
*Get I2C output buffer out-pointer.*
- byte [LSOutputBufferBytesToRx](#) (const byte port)  
*Get I2C output buffer bytes to rx.*
- byte [LSMode](#) (const byte port)  
*Get I2C mode.*
- byte [LSChannelState](#) (const byte port)  
*Get I2C channel state.*
- byte [LSErrorType](#) (const byte port)  
*Get I2C error type.*
- byte [LSSState](#) ()  
*Get I2C state.*

- byte [LSSpeed](#) ()  
*Get I2C speed.*
- byte [LSNoRestartOnRead](#) ()  
*Get I2C no restart on read setting.*
- void [SetI2COptions](#) (byte port, byte options)  
*Set I2C options.*
- void [SysCommLSWrite](#) ([CommLSWriteType](#) &args)  
*Write to a Lowspeed sensor.*
- void [SysCommLSRead](#) ([CommLSReadType](#) &args)  
*Read from a Lowspeed sensor.*
- void [SysCommLSCheckStatus](#) ([CommLSCheckStatusType](#) &args)  
*Check Lowspeed sensor status.*
- void [SysCommLSWriteEx](#) ([CommLSWriteExType](#) &args)  
*Write to a Lowspeed sensor (extra).*
- unsigned long [CurrentTick](#) ()  
*Read the current system tick.*
- unsigned long [FirstTick](#) ()  
*Get the first tick.*
- long [ResetSleepTimer](#) ()  
*Reset the sleep timer.*
- void [SysCall](#) (byte funcID, variant &args)  
*Call any system function.*
- void [SysGetStartTick](#) ([GetStartTickType](#) &args)  
*Get start tick.*
- void [SysKeepAlive](#) ([KeepAliveType](#) &args)  
*Keep alive.*
- void [SysIOMapRead](#) ([IOMapReadType](#) &args)  
*Read from IOMap by name.*

- void [SysIOMapWrite](#) ([IOMapWriteType](#) &args)  
*Write to IOMap by name.*
- void [SysIOMapReadByID](#) ([IOMapReadByIDType](#) &args)  
*Read from IOMap by identifier.*
- void [SysIOMapWriteByID](#) ([IOMapWriteByIDType](#) &args)  
*Write to IOMap by identifier.*
- void [SysDatalogWrite](#) ([DatalogWriteType](#) &args)  
*Write to the datalog.*
- void [SysDatalogGetTimes](#) ([DatalogGetTimesType](#) &args)  
*Get datalog times.*
- void [SysReadSemData](#) ([ReadSemDataType](#) &args)  
*Read semaphore data.*
- void [SysWriteSemData](#) ([WriteSemDataType](#) &args)  
*Write semaphore data.*
- void [SysUpdateCalibCacheInfo](#) ([UpdateCalibCacheInfoType](#) &args)  
*Update calibration cache information.*
- void [SysComputeCalibValue](#) ([ComputeCalibValueType](#) &args)  
*Compute calibration values.*
- char [GetMemoryInfo](#) (bool Compact, unsigned int &PoolSize, unsigned int &DataspaceSize)  
*Read memory information.*
- void [SysMemoryManager](#) ([MemoryManagerType](#) &args)  
*Read memory information.*
- char [GetLastResponseInfo](#) (bool Clear, byte &Length, byte &Command, byte &Buffer[ ])  
*Read last response information.*
- void [SysReadLastResponse](#) ([ReadLastResponseType](#) &args)  
*Read last response information.*
- void [Wait](#) (unsigned long ms)  
*Wait some milliseconds.*

- void [Yield](#) ()  
*Yield to another task.*
- void [StopAllTasks](#) ()  
*Stop all tasks.*
- void [Stop](#) (bool bvalue)  
*Stop the running program.*
- void [ExitTo](#) (task newTask)  
*Exit to another task.*
- void [Precedes](#) (task task1, task task2,..., task taskN)  
*Declare tasks that this task precedes.*
- void [Follows](#) (task task1, task task2,..., task taskN)  
*Declare tasks that this task follows.*
- void [Acquire](#) (mutex m)  
*Acquire a mutex.*
- void [Release](#) (mutex m)  
*Acquire a mutex.*
- void [StartTask](#) (task t)  
*Start a task.*
- void [StopTask](#) (task t)  
*Stop a task.*
- void [BranchTest](#) (const byte cmp, constant void lbl, variant value)  
*Branch if test is true.*
- void [BranchComp](#) (const byte cmp, constant void lbl, variant v1, variant v2)  
*Branch if compare is true.*
- void [ArrayBuild](#) (variant &aout[ ], variant src1, variant src2,..., variant srcN)  
*Build an array.*
- unsigned int [ArrayLen](#) (variant data[ ])  
*Get array length.*

- void [ArrayInit](#) (variant &aout[ ], variant value, unsigned int count)  
*Initialize an array.*
- void [ArraySubset](#) (variant &aout[ ], variant asrc[ ], unsigned int idx, unsigned int len)  
*Copy an array subset.*
- void [ArrayIndex](#) (variant &out, variant asrc[ ], unsigned int idx)  
*Extract item from an array.*
- void [ArrayReplace](#) (variant &asrc[ ], unsigned int idx, variant value)  
*Replace items in an array.*
- variant [ArraySum](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the sum of the elements in a numeric array.*
- variant [ArrayMean](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the mean of the elements in a numeric array.*
- variant [ArraySumSqr](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the sum of the squares of the elements in a numeric array.*
- variant [ArrayStd](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the standard deviation of the elements in a numeric array.*
- variant [ArrayMin](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the minimum of the elements in a numeric array.*
- variant [ArrayMax](#) (const variant &src[ ], unsigned int idx, unsigned int len)  
*Calculate the maximum of the elements in a numeric array.*
- void [ArraySort](#) (variant &dest[ ], const variant &src[ ], unsigned int idx, unsigned int len)  
*Sort the elements in a numeric array.*
- void [ArrayOp](#) (const byte op, variant &dest, const variant &src[ ], unsigned int idx, unsigned int len)  
*Operate on numeric arrays.*
- void [SetIOMapBytes](#) (string moduleName, unsigned int offset, unsigned int count, byte data[ ])  
*Set IOMap bytes by name.*

- void [SetIOMapValue](#) (string moduleName, unsigned int offset, variant value)  
*Set IOMap value by name.*
- void [GetIOMapBytes](#) (string moduleName, unsigned int offset, unsigned int count, byte &data[ ])  
*Get IOMap bytes by name.*
- void [GetIOMapValue](#) (string moduleName, unsigned int offset, variant &value)  
*Get IOMap value by name.*
- void [GetLowSpeedModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])  
*Get Lowspeed module IOMap bytes.*
- void [GetDisplayModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])  
*Get Display module IOMap bytes.*
- void [GetCommModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])  
*Get Comm module IOMap bytes.*
- void [GetCommandModuleBytes](#) (unsigned int offset, unsigned int count, byte &data[ ])  
*Get Command module IOMap bytes.*
- void [SetCommandModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])  
*Set Command module IOMap bytes.*
- void [SetLowSpeedModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])  
*Set Lowspeed module IOMap bytes.*
- void [SetDisplayModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])  
*Set Display module IOMap bytes.*
- void [SetCommModuleBytes](#) (unsigned int offset, unsigned int count, byte data[ ])  
*Set Comm module IOMap bytes.*

- void [SetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte data[ ])  
*Set IOMap bytes by ID.*
- void [SetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant value)  
*Set IOMap value by ID.*
- void [GetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte &data[ ])  
*Get IOMap bytes by ID.*
- void [GetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant &value)  
*Get IOMap value by ID.*
- void [SetCommandModuleValue](#) (unsigned int offset, variant value)  
*Set Command module IOMap value.*
- void [SetIOCtrlModuleValue](#) (unsigned int offset, variant value)  
*Set IOCtrl module IOMap value.*
- void [SetLoaderModuleValue](#) (unsigned int offset, variant value)  
*Set Loader module IOMap value.*
- void [SetUIModuleValue](#) (unsigned int offset, variant value)  
*Set Ui module IOMap value.*
- void [SetSoundModuleValue](#) (unsigned int offset, variant value)  
*Set Sound module IOMap value.*
- void [SetButtonModuleValue](#) (unsigned int offset, variant value)  
*Set Button module IOMap value.*
- void [SetInputModuleValue](#) (unsigned int offset, variant value)  
*Set Input module IOMap value.*
- void [SetOutputModuleValue](#) (unsigned int offset, variant value)  
*Set Output module IOMap value.*
- void [SetLowSpeedModuleValue](#) (unsigned int offset, variant value)  
*Set Lowspeed module IOMap value.*

- void [SetDisplayModuleValue](#) (unsigned int offset, variant value)  
*Set Display module IOMap value.*
- void [SetCommModuleValue](#) (unsigned int offset, variant value)  
*Set Comm module IOMap value.*
- void [GetCommandModuleValue](#) (unsigned int offset, variant &value)  
*Get Command module IOMap value.*
- void [GetLoaderModuleValue](#) (unsigned int offset, variant &value)  
*Get Loader module IOMap value.*
- void [GetSoundModuleValue](#) (unsigned int offset, variant &value)  
*Get Sound module IOMap value.*
- void [GetButtonModuleValue](#) (unsigned int offset, variant &value)  
*Get Button module IOMap value.*
- void [GetUIModuleValue](#) (unsigned int offset, variant &value)  
*Get Ui module IOMap value.*
- void [GetInputModuleValue](#) (unsigned int offset, variant &value)  
*Get Input module IOMap value.*
- void [GetOutputModuleValue](#) (unsigned int offset, variant &value)  
*Get Output module IOMap value.*
- void [GetLowSpeedModuleValue](#) (unsigned int offset, variant &value)  
*Get LowSpeed module IOMap value.*
- void [GetDisplayModuleValue](#) (unsigned int offset, variant &value)  
*Get Display module IOMap value.*
- void [GetCommModuleValue](#) (unsigned int offset, variant &value)  
*Get Comm module IOMap value.*
- void [PowerDown](#) ()  
*Power down the NXT.*
- void [SleepNow](#) ()  
*Put the brick to sleep immediately.*

- void [RebootInFirmwareMode](#) ()  
*Reboot the NXT in firmware download mode.*
- char [JoystickMessageRead](#) (byte queue, [JoystickMessageType](#) &msg)  
*Read a joystick message from a queue/mailbox.*
- char [SendMessage](#) (byte queue, string msg)  
*Send a message to a queue/mailbox.*
- char [ReceiveMessage](#) (byte queue, bool clear, string &msg)  
*Read a message from a queue/mailbox.*
- char [BluetoothStatus](#) (byte conn)  
*Check bluetooth status.*
- char [BluetoothWrite](#) (byte conn, byte buffer[ ])  
*Write to a bluetooth connection.*
- char [RemoteConnectionWrite](#) (byte conn, byte buffer[ ])  
*Write to a remote connection.*
- bool [RemoteConnectionIdle](#) (byte conn)  
*Check if remote connection is idle.*
- char [SendRemoteBool](#) (byte conn, byte queue, bool bval)  
*Send a boolean value to a remote mailbox.*
- char [SendRemoteNumber](#) (byte conn, byte queue, long val)  
*Send a numeric value to a remote mailbox.*
- char [SendRemoteString](#) (byte conn, byte queue, string str)  
*Send a string value to a remote mailbox.*
- char [SendResponseBool](#) (byte queue, bool bval)  
*Write a boolean value to a local response mailbox.*
- char [SendResponseNumber](#) (byte queue, long val)  
*Write a numeric value to a local response mailbox.*
- char [SendResponseString](#) (byte queue, string str)  
*Write a string value to a local response mailbox.*

- char [ReceiveRemoteBool](#) (byte queue, bool clear, bool &bval)  
*Read a boolean value from a queue/mailbox.*
- char [ReceiveRemoteMessageEx](#) (byte queue, bool clear, string &str, long &val, bool &bval)  
*Read a value from a queue/mailbox.*
- char [ReceiveRemoteNumber](#) (byte queue, bool clear, long &val)  
*Read a numeric value from a queue/mailbox.*
- char [ReceiveRemoteString](#) (byte queue, bool clear, string &str)  
*Read a string value from a queue/mailbox.*
- char [RemoteKeepAlive](#) (byte conn)  
*Send a KeepAlive message.*
- char [RemoteMessageRead](#) (byte conn, byte queue)  
*Send a MessageRead message.*
- char [RemoteMessageWrite](#) (byte conn, byte queue, string msg)  
*Send a MessageWrite message.*
- char [RemotePlaySoundFile](#) (byte conn, string filename, bool bloop)  
*Send a PlaySoundFile message.*
- char [RemotePlayTone](#) (byte conn, unsigned int frequency, unsigned int duration)  
*Send a PlayTone message.*
- char [RemoteResetMotorPosition](#) (byte conn, byte port, bool brelative)  
*Send a ResetMotorPosition message.*
- char [RemoteResetScaledValue](#) (byte conn, byte port)  
*Send a ResetScaledValue message.*
- char [RemoteSetInputMode](#) (byte conn, byte port, byte type, byte mode)  
*Send a SetInputMode message.*
- char [RemoteSetOutputState](#) (byte conn, byte port, char speed, byte mode, byte regmode, char turnpct, byte runstate, unsigned long tacholimit)  
*Send a SetOutputMode message.*

- char [RemoteStartProgram](#) (byte conn, string filename)  
*Send a StartProgram message.*
- char [RemoteStopProgram](#) (byte conn)  
*Send a StopProgram message.*
- char [RemoteStopSound](#) (byte conn)  
*Send a StopSound message.*
- char [RemoteGetOutputState](#) (byte conn, [OutputStateType](#) &params)  
*Send a GetOutputState message.*
- char [RemoteGetInputValues](#) (byte conn, [InputValuesType](#) &params)  
*Send a GetInputValues message.*
- char [RemoteGetBatteryLevel](#) (byte conn, int &value)  
*Send a GetBatteryLevel message.*
- char [RemoteLowspeedGetStatus](#) (byte conn, byte &value)  
*Send a LowspeedGetStatus message.*
- char [RemoteLowspeedRead](#) (byte conn, byte port, byte &bread, byte &data[ ])  
*Send a LowspeedRead message.*
- char [RemoteGetCurrentProgramName](#) (byte conn, string &name)  
*Send a GetCurrentProgramName message.*
- char [RemoteDatalogRead](#) (byte conn, bool remove, byte &cnt, byte &log[ ])  
*Send a DatalogRead message.*
- char [RemoteGetContactCount](#) (byte conn, byte &cnt)  
*Send a GetContactCount message.*
- char [RemoteGetContactName](#) (byte conn, byte idx, string &name)  
*Send a GetContactName message.*
- char [RemoteGetConnectionCount](#) (byte conn, byte &cnt)  
*Send a GetConnectionCount message.*
- char [RemoteGetConnectionName](#) (byte conn, byte idx, string &name)  
*Send a GetConnectionName message.*

- char [RemoteGetProperty](#) (byte conn, byte property, variant &value)  
*Send a GetProperty message.*
- char [RemoteResetTachoCount](#) (byte conn, byte port)  
*Send a ResetTachoCount message.*
- char [RemoteDatalogSetTimes](#) (byte conn, long synctime)  
*Send a DatalogSetTimes message.*
- char [RemoteSetProperty](#) (byte conn, byte prop, variant value)  
*Send a SetProperty message.*
- char [RemoteLowSpeedWrite](#) (byte conn, byte port, byte txlen, byte rxlen, byte data[ ])  
*Send a LowSpeedWrite message.*
- char [RemoteOpenRead](#) (byte conn, string filename, byte &handle, long &size)  
*Send an OpenRead message.*
- char [RemoteOpenAppendData](#) (byte conn, string filename, byte &handle, long &size)  
*Send an OpenAppendData message.*
- char [RemoteDeleteFile](#) (byte conn, string filename)  
*Send a DeleteFile message.*
- char [RemoteFindFirstFile](#) (byte conn, string mask, byte &handle, string &name, long &size)  
*Send a FindFirstFile message.*
- char [RemoteGetFirmwareVersion](#) (byte conn, byte &pmin, byte &pmaj, byte &fmin, byte &fmaj)  
*Send a GetFirmwareVersion message.*
- char [RemoteGetBluetoothAddress](#) (byte conn, byte &btaddr[ ])  
*Send a GetBluetoothAddress message.*
- char [RemoteGetDeviceInfo](#) (byte conn, string &name, byte &btaddr[ ], byte &btsignal[ ], long &freemem)  
*Send a GetDeviceInfo message.*
- char [RemoteDeleteUserFlash](#) (byte conn)  
*Send a DeleteUserFlash message.*

- char [RemoteOpenWrite](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWrite message.*
- char [RemoteOpenWriteLinear](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWriteLinear message.*
- char [RemoteOpenWriteData](#) (byte conn, string filename, long size, byte &handle)  
*Send an OpenWriteData message.*
- char [RemoteCloseFile](#) (byte conn, byte handle)  
*Send a CloseFile message.*
- char [RemoteFindNextFile](#) (byte conn, byte &handle, string &name, long &size)  
*Send a FindNextFile message.*
- char [RemotePollCommandLength](#) (byte conn, byte bufnum, byte &length)  
*Send a PollCommandLength message.*
- char [RemoteWrite](#) (byte conn, byte &handle, int &numbytes, byte data[ ])  
*Send a Write message.*
- char [RemoteRead](#) (byte conn, byte &handle, int &numbytes, byte &data[ ])  
*Send a Read message.*
- char [RemoteIOMapRead](#) (byte conn, long id, int offset, int &numbytes, byte &data[ ])  
*Send an IOMapRead message.*
- char [RemotePollCommand](#) (byte conn, byte bufnum, byte &len, byte &data[ ])  
*Send a PollCommand message.*
- char [RemoteRenameFile](#) (byte conn, string oldname, string newname)  
*Send a RenameFile message.*
- char [RemoteBluetoothFactoryReset](#) (byte conn)  
*Send a BluetoothFactoryReset message.*
- char [RemoteIOMapWriteValue](#) (byte conn, long id, int offset, variant value)  
*Send an IOMapWrite value message.*

- char [RemoteIOMapWriteBytes](#) (byte conn, long id, int offset, byte data[ ])  
*Send an IOMapWrite bytes message.*
- char [RemoteSetBrickName](#) (byte conn, string name)  
*Send a SetBrickName message.*
- void [UseRS485](#) (void)  
*Use the RS485 port.*
- char [RS485Control](#) (byte cmd, byte baud, unsigned int mode)  
*Control the RS485 port.*
- byte [RS485DataAvailable](#) (void)  
*Check for RS485 available data.*
- char [RS485Initialize](#) (void)  
*Initialize RS485 port.*
- char [RS485Disable](#) (void)  
*Disable RS485.*
- char [RS485Enable](#) (void)  
*Enable RS485.*
- char [RS485Read](#) (byte &buffer[ ])  
*Read RS485 data.*
- char [RS485ReadEx](#) (byte &buffer[ ], byte buflen)  
*Read limited RS485 data.*
- byte [RS485SendingData](#) (void)  
*Is RS485 sending data.*
- void [RS485Status](#) (byte &sendingData, byte &dataAvail)  
*Check RS485 status.*
- char [RS485Uart](#) (byte baud, unsigned int mode)  
*Configure RS485 UART.*
- char [RS485Write](#) (byte buffer[ ])  
*Write RS485 data.*

- char [SendRS485Bool](#) (bool bval)  
*Write RS485 boolean.*
- char [SendRS485Number](#) (long val)  
*Write RS485 numeric.*
- char [SendRS485String](#) (string str)  
*Write RS485 string.*
- void [GetBTInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get bluetooth input buffer data.*
- void [GetBTOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get bluetooth output buffer data.*
- void [GetHSInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get hi-speed port input buffer data.*
- void [GetHSOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get hi-speed port output buffer data.*
- void [GetUSBInputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb input buffer data.*
- void [GetUSBOutputBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb output buffer data.*
- void [GetUSBPollBuffer](#) (const byte offset, byte cnt, byte &data[ ])  
*Get usb poll buffer data.*
- string [BTDeviceName](#) (const byte devidx)  
*Get bluetooth device name.*
- string [BTConnectionName](#) (const byte conn)  
*Get bluetooth device name.*
- string [BTConnectionPinCode](#) (const byte conn)  
*Get bluetooth device pin code.*
- string [BrickDataName](#) (void)  
*Get NXT name.*

- void [GetBTDeviceAddress](#) (const byte devidx, byte &data[ ])   
*Get bluetooth device address.*
- void [GetBTConnectionAddress](#) (const byte conn, byte &data[ ])   
*Get bluetooth device address.*
- void [GetBrickDataAddress](#) (byte &data[ ])   
*Get NXT address.*
- long [BTDeviceClass](#) (const byte devidx)   
*Get bluetooth device class.*
- byte [BTDeviceStatus](#) (const byte devidx)   
*Get bluetooth device status.*
- long [BTConnectionClass](#) (const byte conn)   
*Get bluetooth device class.*
- byte [BTConnectionHandleNum](#) (const byte conn)   
*Get bluetooth device handle number.*
- byte [BTConnectionStreamStatus](#) (const byte conn)   
*Get bluetooth device stream status.*
- byte [BTConnectionLinkQuality](#) (const byte conn)   
*Get bluetooth device link quality.*
- int [BrickDataBluecoreVersion](#) (void)   
*Get NXT bluecore version.*
- byte [BrickDataBtStateStatus](#) (void)   
*Get NXT bluetooth state status.*
- byte [BrickDataBtHardwareStatus](#) (void)   
*Get NXT bluetooth hardware status.*
- byte [BrickDataTimeoutValue](#) (void)   
*Get NXT bluetooth timeout value.*
- byte [BTInputBufferInPtr](#) (void)   
*Get bluetooth input buffer in-pointer.*

- byte [BTInputBufferOutPtr](#) (void)  
*Get bluetooth input buffer out-pointer.*
- byte [BTOutputBufferInPtr](#) (void)  
*Get bluetooth output buffer in-pointer.*
- byte [BTOutputBufferOutPtr](#) (void)  
*Get bluetooth output buffer out-pointer.*
- byte [HSInputBufferInPtr](#) (void)  
*Get hi-speed port input buffer in-pointer.*
- byte [HSInputBufferOutPtr](#) (void)  
*Get hi-speed port input buffer out-pointer.*
- byte [HSOutputBufferInPtr](#) (void)  
*Get hi-speed port output buffer in-pointer.*
- byte [HSOutputBufferOutPtr](#) (void)  
*Get hi-speed port output buffer out-pointer.*
- byte [USBInputBufferInPtr](#) (void)  
*Get usb port input buffer in-pointer.*
- byte [USBInputBufferOutPtr](#) (void)  
*Get usb port input buffer out-pointer.*
- byte [USBOutputBufferInPtr](#) (void)  
*Get usb port output buffer in-pointer.*
- byte [USBOutputBufferOutPtr](#) (void)  
*Get usb port output buffer out-pointer.*
- byte [USBPollBufferInPtr](#) (void)  
*Get usb port poll buffer in-pointer.*
- byte [USBPollBufferOutPtr](#) (void)  
*Get usb port poll buffer out-pointer.*
- byte [BTDeviceCount](#) (void)  
*Get bluetooth device count.*

- byte [BTDeviceNameCount](#) (void)  
*Get bluetooth device name count.*
- byte [HSFlags](#) (void)  
*Get hi-speed port flags.*
- byte [HSSpeed](#) (void)  
*Get hi-speed port speed.*
- byte [HSState](#) (void)  
*Get hi-speed port state.*
- byte [HSAddress](#) (void)  
*Get hi-speed port address.*
- int [HSMode](#) (void)  
*Get hi-speed port mode.*
- int [BTDataMode](#) (void)  
*Get Bluetooth data mode.*
- int [HSDDataMode](#) (void)  
*Get hi-speed port datamode.*
- byte [USBState](#) (void)  
*Get USB state.*
- void [SetBTInputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set bluetooth input buffer data.*
- void [SetBTInputBufferInPtr](#) (byte n)  
*Set bluetooth input buffer in-pointer.*
- void [SetBTInputBufferOutPtr](#) (byte n)  
*Set bluetooth input buffer out-pointer.*
- void [SetBTOutputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set bluetooth output buffer data.*
- void [SetBTOutputBufferInPtr](#) (byte n)  
*Set bluetooth output buffer in-pointer.*

- void [SetBTOutputBufferOutPtr](#) (byte n)  
*Set bluetooth output buffer out-pointer.*
- void [SetHSInputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set hi-speed port input buffer data.*
- void [SetHSInputBufferInPtr](#) (byte n)  
*Set hi-speed port input buffer in-pointer.*
- void [SetHSInputBufferOutPtr](#) (byte n)  
*Set hi-speed port input buffer out-pointer.*
- void [SetHSOutputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set hi-speed port output buffer data.*
- void [SetHSOutputBufferInPtr](#) (byte n)  
*Set hi-speed port output buffer in-pointer.*
- void [SetHSOutputBufferOutPtr](#) (byte n)  
*Set hi-speed port output buffer out-pointer.*
- void [SetUSBInputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB input buffer data.*
- void [SetUSBInputBufferInPtr](#) (byte n)  
*Set USB input buffer in-pointer.*
- void [SetUSBInputBufferOutPtr](#) (byte n)  
*Set USB input buffer out-pointer.*
- void [SetUSBOutputBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB output buffer data.*
- void [SetUSBOutputBufferInPtr](#) (byte n)  
*Set USB output buffer in-pointer.*
- void [SetUSBOutputBufferOutPtr](#) (byte n)  
*Set USB output buffer out-pointer.*
- void [SetUSBPollBuffer](#) (const byte offset, byte cnt, byte data[ ])  
*Set USB poll buffer data.*

- void [SetUSBPollBufferInPtr](#) (byte n)  
*Set USB poll buffer in-pointer.*
- void [SetUSBPollBufferOutPtr](#) (byte n)  
*Set USB poll buffer out-pointer.*
- void [SetHSFlags](#) (byte hsFlags)  
*Set hi-speed port flags.*
- void [SetHSSpeed](#) (byte hsSpeed)  
*Set hi-speed port speed.*
- void [SetHSState](#) (byte hsState)  
*Set hi-speed port state.*
- void [SetHSAddress](#) (byte hsAddress)  
*Set hi-speed port address.*
- void [SetHSMode](#) (unsigned int hsMode)  
*Set hi-speed port mode.*
- void [SetBTDataMode](#) (const byte dataMode)  
*Set Bluetooth data mode.*
- void [SetHSDataMode](#) (const byte dataMode)  
*Set hi-speed port data mode.*
- void [SetUSBState](#) (byte usbState)  
*Set USB state.*
- void [SysMessageWrite](#) ([MessageWriteType](#) &args)  
*Write message.*
- void [SysMessageRead](#) ([MessageReadType](#) &args)  
*Read message.*
- void [SysCommBTWrite](#) ([CommBTWriteType](#) &args)  
*Write data to a Bluetooth connection.*
- void [SysCommBTCheckStatus](#) ([CommBTCheckStatusType](#) &args)  
*Check Bluetooth connection status.*

- void [SysCommExecuteFunction](#) ([CommExecuteFunctionType](#) &args)  
*Execute any Comm module command.*
- void [SysCommHSControl](#) ([CommHSControlType](#) &args)  
*Control the hi-speed port.*
- void [SysCommHSCheckStatus](#) ([CommHSCheckStatusType](#) &args)  
*Check the hi-speed port status.*
- void [SysCommHSRead](#) ([CommHSReadWriteType](#) &args)  
*Read from the hi-speed port.*
- void [SysCommHSWrite](#) ([CommHSReadWriteType](#) &args)  
*Write to the hi-speed port.*
- void [SysCommBTOnOff](#) ([CommBTOnOffType](#) &args)  
*Turn on or off the bluetooth subsystem.*
- void [SysCommBTConnection](#) ([CommBTConnectionType](#) &args)  
*Connect or disconnect a bluetooth device.*
- bool [ButtonPressed](#) (const byte btn, bool resetCount)  
*Check for button press.*
- byte [ButtonCount](#) (const byte btn, bool resetCount)  
*Get button press count.*
- char [ReadButtonEx](#) (const byte btn, bool reset, bool &pressed, unsigned int &count)  
*Read button information.*
- byte [ButtonPressCount](#) (const byte btn)  
*Get button press count.*
- byte [ButtonLongPressCount](#) (const byte btn)  
*Get button long press count.*
- byte [ButtonShortReleaseCount](#) (const byte btn)  
*Get button short release count.*
- byte [ButtonLongReleaseCount](#) (const byte btn)

*Get button long release count.*

- byte [ButtonReleaseCount](#) (const byte btn)  
*Get button release count.*
- byte [ButtonState](#) (const byte btn)  
*Get button state.*
- void [SetButtonLongPressCount](#) (const byte btn, const byte n)  
*Set button long press count.*
- void [SetButtonLongReleaseCount](#) (const byte btn, const byte n)  
*Set button long release count.*
- void [SetButtonPressCount](#) (const byte btn, const byte n)  
*Set button press count.*
- void [SetButtonReleaseCount](#) (const byte btn, const byte n)  
*Set button release count.*
- void [SetButtonShortReleaseCount](#) (const byte btn, const byte n)  
*Set button short release count.*
- void [SetButtonState](#) (const byte btn, const byte state)  
*Set button state.*
- void [SysReadButton](#) ([ReadButtonType](#) &args)  
*Read button.*
- byte [CommandFlags](#) (void)  
*Get command flags.*
- byte [UIState](#) (void)  
*Get UI module state.*
- byte [UIButton](#) (void)  
*Read UI button.*
- byte [VMRunState](#) (void)  
*Read VM run state.*
- byte [BatteryState](#) (void)

*Get battery state.*

- byte [BluetoothState](#) (void)  
*Get bluetooth state.*
- byte [UsbState](#) (void)  
*Get UI module USB state.*
- byte [SleepTimeout](#) (void)  
*Read sleep timeout.*
- byte [SleepTime](#) (void)  
*Read sleep time.*
- byte [SleepTimer](#) (void)  
*Read sleep timer.*
- bool [RechargeableBattery](#) (void)  
*Read battery type.*
- byte [Volume](#) (void)  
*Read volume.*
- byte [OnBrickProgramPointer](#) (void)  
*Read the on brick program pointer value.*
- byte [AbortFlag](#) (void)  
*Read abort flag.*
- byte [LongAbort](#) (void)  
*Read long abort setting.*
- unsigned int [BatteryLevel](#) (void)  
*Get battery Level.*
- void [SetCommandFlags](#) (const byte cmdFlags)  
*Set command flags.*
- void [SetUIButton](#) (byte btn)  
*Set UI button.*
- void [SetUIState](#) (byte state)

*Set UI state.*

- void [SetVMRunState](#) (const byte vmRunState)  
*Set VM run state.*
- void [SetBatteryState](#) (byte state)  
*Set battery state.*
- void [SetBluetoothState](#) (byte state)  
*Set bluetooth state.*
- void [SetSleepTimeout](#) (const byte n)  
*Set sleep timeout.*
- void [SetSleepTime](#) (const byte n)  
*Set sleep time.*
- void [SetSleepTimer](#) (const byte n)  
*Set the sleep timer.*
- void [SetVolume](#) (byte volume)  
*Set volume.*
- void [SetOnBrickProgramPointer](#) (byte obpStep)  
*Set on-brick program pointer.*
- void [ForceOff](#) (byte num)  
*Turn off NXT.*
- void [SetAbortFlag](#) (byte abortFlag)  
*Set abort flag.*
- void [SetLongAbort](#) (bool longAbort)  
*Set long abort.*
- void [SysSetSleepTimeout](#) ([SetSleepTimeoutType](#) &args)  
*Set system sleep timeout.*
- unsigned int [FreeMemory](#) (void)  
*Get free flash memory.*
- unsigned int [CreateFile](#) (string fname, unsigned int fsize, byte &handle)

*Create a file.*

- unsigned int [OpenFileAppend](#) (string fname, unsigned int &fsize, byte &handle)

*Open a file for appending.*

- unsigned int [OpenFileRead](#) (string fname, unsigned int &fsize, byte &handle)

*Open a file for reading.*

- unsigned int [CloseFile](#) (byte handle)

*Close a file.*

- unsigned int [ResolveHandle](#) (string filename, byte &handle, bool &writeable)

*Resolve a handle.*

- unsigned int [RenameFile](#) (string oldname, string newname)

*Rename a file.*

- unsigned int [DeleteFile](#) (string fname)

*Delete a file.*

- unsigned int [ResizeFile](#) (string fname, const unsigned int newsize)

*Resize a file.*

- unsigned int [CreateFileLinear](#) (string fname, unsigned int fsize, byte &handle)

*Create a linear file.*

- unsigned int [CreateFileNonLinear](#) (string fname, unsigned int fsize, byte &handle)

*Create a non-linear file.*

- unsigned int [OpenFileReadLinear](#) (string fname, unsigned int &fsize, byte &handle)

*Open a linear file for reading.*

- unsigned int [FindFirstFile](#) (string &fname, byte &handle)

*Start searching for files.*

- unsigned int [FindNextFile](#) (string &fname, byte &handle)

*Continue searching for files.*

- unsigned int [SizeOf](#) (variant &value)

*Calculate the size of a variable.*

- unsigned int [Read](#) (byte handle, variant &value)  
*Read a value from a file.*
- unsigned int [ReadLn](#) (byte handle, variant &value)  
*Read a value from a file plus line ending.*
- unsigned int [ReadBytes](#) (byte handle, unsigned int &length, byte &buf[ ])  
*Read bytes from a file.*
- unsigned int [ReadLnString](#) (byte handle, string &output)  
*Read a string from a file plus line ending.*
- unsigned int [Write](#) (byte handle, const variant &value)  
*Write value to file.*
- unsigned int [WriteBytes](#) (byte handle, const byte &buf[ ], unsigned int &cnt)  
*Write bytes to file.*
- unsigned int [WriteBytesEx](#) (byte handle, unsigned int &len, const byte &buf[ ])  
*Write bytes to a file with limit.*
- unsigned int [WriteLn](#) (byte handle, const variant &value)  
*Write a value and new line to a file.*
- unsigned int [WriteLnString](#) (byte handle, const string &str, unsigned int &cnt)  
*Write string and new line to a file.*
- unsigned int [WriteString](#) (byte handle, const string &str, unsigned int &cnt)  
*Write string to a file.*
- void [SysFileOpenRead](#) (FileOpenType &args)  
*Open file for reading.*
- void [SysFileOpenWrite](#) (FileOpenType &args)  
*Open and create file for writing.*
- void [SysFileOpenAppend](#) (FileOpenType &args)  
*Open file for writing at end of file.*
- void [SysFileRead](#) (FileReadWriteType &args)  
*Read from file.*

- void `SysFileWrite` (`FileReadWriteType` &args)  
*File write.*
- void `SysFileClose` (`FileCloseType` &args)  
*Close file handle.*
- void `SysFileResolveHandle` (`FileResolveHandleType` &args)  
*File resolve handle.*
- void `SysFileRename` (`FileRenameType` &args)  
*Rename file.*
- void `SysFileDelete` (`FileDeleteType` &args)  
*Delete file.*
- void `SysLoaderExecuteFunction` (`LoaderExecuteFunctionType` &args)  
*Execute any Loader module command.*
- void `SysFileFindFirst` (`FileFindType` &args)  
*Start finding files.*
- void `SysFileFindNext` (`FileFindType` &args)  
*Continue finding files.*
- void `SysFileOpenWriteLinear` (`FileOpenType` &args)  
*Open and create linear file for writing.*
- void `SysFileOpenWriteNonLinear` (`FileOpenType` &args)  
*Open and create non-linear file for writing.*
- void `SysFileOpenReadLinear` (`FileOpenType` &args)  
*Open linear file for reading.*
- void `SysFileSeek` (`FileSeekType` &args)  
*Seek to file position.*
- void `SysFileResize` (`FileResizeType` &args)  
*Resize a file.*
- void `SysFileTell` (`FileTellType` &args)  
*Return the file position.*

- void [SysListFiles](#) ([ListFileType](#) &args)  
*List files.*
- int [SensorHTGyro](#) (const byte &port, int offset=0)  
*Read HiTechnic Gyro sensor.*
- int [SensorHTMagnet](#) (const byte &port, int offset=0)  
*Read HiTechnic Magnet sensor.*
- int [SensorHTEOPD](#) (const byte &port)  
*Read HiTechnic EOPD sensor.*
- void [SetSensorHTEOPD](#) (const byte &port, bool bStandard)  
*Set sensor as HiTechnic EOPD.*
- void [SetSensorHTGyro](#) (const byte &port)  
*Set sensor as HiTechnic Gyro.*
- void [SetSensorHTMagnet](#) (const byte &port)  
*Set sensor as HiTechnic Magnet.*
- int [SensorHTColorNum](#) (const byte &port)  
*Read HiTechnic color sensor color number.*
- int [SensorHTCompass](#) (const byte &port)  
*Read HiTechnic compass.*
- int [SensorHTIRSeekerDir](#) (const byte &port)  
*Read HiTechnic IRSeeker direction.*
- int [SensorHTIRSeeker2Addr](#) (const byte &port, const byte reg)  
*Read HiTechnic IRSeeker2 register.*
- int [SensorHTIRSeeker2DCDir](#) (const byte &port)  
*Read HiTechnic IRSeeker2 DC direction.*
- int [SensorHTIRSeeker2ACDir](#) (const byte &port)  
*Read HiTechnic IRSeeker2 AC direction.*
- char [SetHTColor2Mode](#) (const byte &port, byte mode)  
*Set HiTechnic Color2 mode.*

- char [SetHTIRSeeker2Mode](#) (const byte &port, const byte mode)  
*Set HiTechnic IRSeeker2 mode.*
- bool [ReadSensorHTAccel](#) (const byte port, int &x, int &y, int &z)  
*Read HiTechnic acceleration values.*
- bool [ReadSensorHTColor](#) (const byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color values.*
- bool [ReadSensorHTIRSeeker](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)  
*Read HiTechnic IRSeeker values.*
- bool [ReadSensorHTNormalizedColor](#) (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color normalized values.*
- bool [ReadSensorHTRawColor](#) (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue)  
*Read HiTechnic Color raw values.*
- bool [ReadSensorHTColor2Active](#) (byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue, byte &White)  
*Read HiTechnic Color2 active values.*
- bool [ReadSensorHTNormalizedColor2Active](#) (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)  
*Read HiTechnic Color2 normalized active values.*
- bool [ReadSensorHTRawColor2](#) (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue, unsigned int &White)  
*Read HiTechnic Color2 raw values.*
- bool [ReadSensorHTIRReceiver](#) (const byte port, char &pfdata[ ])  
*Read HiTechnic IRReceiver Power Function bytes.*
- bool [ReadSensorHTIRReceiverEx](#) (const byte port, const byte offset, char &pfchar)  
*Read HiTechnic IRReceiver Power Function value.*

- bool [ReadSensorHTIRSeeker2AC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)  
*Read HiTechnic IRSeeker2 AC values.*
- bool [ReadSensorHTIRSeeker2DC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9, byte &avg)  
*Read HiTechnic IRSeeker2 DC values.*
- char [ResetSensorHTAngle](#) (const byte port, const byte mode)  
*Reset HiTechnic Angle sensor.*
- bool [ReadSensorHTAngle](#) (const byte port, int &Angle, long &AccAngle, int &RPM)  
*Read HiTechnic Angle sensor values.*
- bool [ResetHTBarometricCalibration](#) (byte port)  
*Reset HiTechnic Barometric sensor calibration.*
- bool [SetHTBarometricCalibration](#) (byte port, unsigned int cal)  
*Set HiTechnic Barometric sensor calibration.*
- bool [ReadSensorHTBarometric](#) (const byte port, int &temp, unsigned int &press)  
*Read HiTechnic Barometric sensor values.*
- int [SensorHTProtoAnalog](#) (const byte port, const byte input)  
*Read HiTechnic Prototype board analog input value.*
- bool [ReadSensorHTProtoAllAnalog](#) (const byte port, int &a0, int &a1, int &a2, int &a3, int &a4)  
*Read all HiTechnic Prototype board analog input values.*
- bool [SetSensorHTProtoDigitalControl](#) (const byte port, byte value)  
*Control HiTechnic Prototype board digital pin direction.*
- byte [SensorHTProtoDigitalControl](#) (const byte port)  
*Read HiTechnic Prototype board digital control values.*
- bool [SetSensorHTProtoDigital](#) (const byte port, byte value)  
*Set HiTechnic Prototype board digital output values.*
- byte [SensorHTProtoDigital](#) (const byte port)  
*Read HiTechnic Prototype board digital input values.*

- int [SensorHTSuperProAnalog](#) (const byte port, const byte input)  
*Read HiTechnic SuperPro board analog input value.*
- bool [ReadSensorHTSuperProAllAnalog](#) (const byte port, int &a0, int &a1, int &a2, int &a3)  
*Read all HiTechnic SuperPro board analog input values.*
- bool [SetSensorHTSuperProDigitalControl](#) (const byte port, byte value)  
*Control HiTechnic SuperPro board digital pin direction.*
- byte [SensorHTSuperProDigitalControl](#) (const byte port)  
*Read HiTechnic SuperPro board digital control values.*
- bool [SetSensorHTSuperProDigital](#) (const byte port, byte value)  
*Set HiTechnic SuperPro board digital output values.*
- byte [SensorHTSuperProDigital](#) (const byte port)  
*Read HiTechnic SuperPro board digital input values.*
- bool [SetSensorHTSuperProLED](#) (const byte port, byte value)  
*Set HiTechnic SuperPro LED value.*
- byte [SensorHTSuperProLED](#) (const byte port)  
*Read HiTechnic SuperPro LED value.*
- bool [SetSensorHTSuperProStrobe](#) (const byte port, byte value)  
*Set HiTechnic SuperPro strobe value.*
- byte [SensorHTSuperProStrobe](#) (const byte port)  
*Read HiTechnic SuperPro strobe value.*
- bool [SetSensorHTSuperProProgramControl](#) (const byte port, byte value)  
*Set HiTechnic SuperPro program control value.*
- byte [SensorHTSuperProProgramControl](#) (const byte port)  
*Read HiTechnic SuperPro program control value.*
- bool [SetSensorHTSuperProAnalogOut](#) (const byte port, const byte dac, byte mode, int freq, int volt)  
*Set HiTechnic SuperPro board analog output parameters.*

- bool [ReadSensorHTSuperProAnalogOut](#) (const byte port, const byte dac, byte &mode, int &freq, int &volt)  
*Read HiTechnic SuperPro board analog output parameters.*
- void [ReadSensorHTTouchMultiplexer](#) (const byte port, byte &t1, byte &t2, byte &t3, byte &t4)  
*Read HiTechnic touch multiplexer.*
- char [HTIRTrain](#) (const byte port, const byte channel, const byte func)  
*HTIRTrain function.*
- char [HTPFComboDirect](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*HTPFComboDirect function.*
- char [HTPFComboPWM](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*HTPFComboPWM function.*
- char [HTPFRawOutput](#) (const byte port, const byte nibble0, const byte nibble1, const byte nibble2)  
*HTPFRawOutput function.*
- char [HTPFRepeat](#) (const byte port, const byte count, const unsigned int delay)  
*HTPFRepeat function.*
- char [HTPFSingleOutputCST](#) (const byte port, const byte channel, const byte out, const byte func)  
*HTPFSingleOutputCST function.*
- char [HTPFSingleOutputPWM](#) (const byte port, const byte channel, const byte out, const byte func)  
*HTPFSingleOutputPWM function.*
- char [HTPFSinglePin](#) (const byte port, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*HTPFSinglePin function.*
- char [HTPFTrain](#) (const byte port, const byte channel, const byte func)  
*HTPFTrain function.*
- void [HTRCXSetIRLinkPort](#) (const byte port)  
*HTRCXSetIRLinkPort function.*

- int [HTRCXBatteryLevel](#) (void)  
*HTRCXBatteryLevel function.*
- int [HTRCXPoll](#) (const byte src, const byte value)  
*HTRCXPoll function Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.*
- int [HTRCXPollMemory](#) (const unsigned int address)  
*HTRCXPollMemory function.*
- void [HTRCXAddToDatalog](#) (const byte src, const unsigned int value)  
*HTRCXAddToDatalog function.*
- void [HTRCXCLEARAllEvents](#) (void)  
*HTRCXCLEARAllEvents function.*
- void [HTRCXCLEARCounter](#) (const byte counter)  
*HTRCXCLEARCounter function.*
- void [HTRCXCLEARMsg](#) (void)  
*HTRCXCLEARMsg function.*
- void [HTRCXCLEARSensor](#) (const byte port)  
*HTRCXCLEARSensor function.*
- void [HTRCXCLEARSound](#) (void)  
*HTRCXCLEARSound function.*
- void [HTRCXCLEARTimer](#) (const byte timer)  
*HTRCXCLEARTimer function.*
- void [HTRCXCreateDatalog](#) (const unsigned int size)  
*HTRCXCreateDatalog function.*
- void [HTRCXDecCounter](#) (const byte counter)  
*HTRCXDecCounter function.*
- void [HTRCXDeleteSub](#) (const byte s)  
*HTRCXDeleteSub function.*
- void [HTRCXDeleteSubs](#) (void)

*HTRCXDeleteSubs function.*

- void [HTRCXDeleteTask](#) (const byte t)  
*HTRCXDeleteTask function.*
- void [HTRCXDeleteTasks](#) (void)  
*HTRCXDeleteTasks function.*
- void [HTRCXDisableOutput](#) (const byte outputs)  
*HTRCXDisableOutput function.*
- void [HTRCXEnableOutput](#) (const byte outputs)  
*HTRCXEnableOutput function.*
- void [HTRCXEvent](#) (const byte src, const unsigned int value)  
*HTRCXEvent function.*
- void [HTRCXFloat](#) (const byte outputs)  
*HTRCXFloat function.*
- void [HTRCXFwd](#) (const byte outputs)  
*HTRCXFwd function.*
- void [HTRCXIncCounter](#) (const byte counter)  
*HTRCXIncCounter function.*
- void [HTRCXInvertOutput](#) (const byte outputs)  
*HTRCXInvertOutput function.*
- void [HTRCXMuteSound](#) (void)  
*HTRCXMuteSound function.*
- void [HTRCXObvertOutput](#) (const byte outputs)  
*HTRCXObvertOutput function.*
- void [HTRCXOff](#) (const byte outputs)  
*HTRCXOff function.*
- void [HTRCXOn](#) (const byte outputs)  
*HTRCXOn function.*
- void [HTRCXOnFor](#) (const byte outputs, const unsigned int ms)

*HTRCXOnFor function.*

- void [HTRCXOnFwd](#) (const byte outputs)  
*HTRCXOnFwd function.*
- void [HTRCXOnRev](#) (const byte outputs)  
*HTRCXOnRev function.*
- void [HTRCXPBTurnOff](#) (void)  
*HTRCXPBTurnOff function.*
- void [HTRCXPing](#) (void)  
*HTRCXPing function.*
- void [HTRCXPlaySound](#) (const byte snd)  
*HTRCXPlaySound function.*
- void [HTRCXPlayTone](#) (const unsigned int freq, const byte duration)  
*HTRCXPlayTone function.*
- void [HTRCXPlayToneVar](#) (const byte varnum, const byte duration)  
*HTRCXPlayToneVar function.*
- void [HTRCXRemote](#) (unsigned int cmd)  
*HTRCXRemote function.*
- void [HTRCXRev](#) (const byte outputs)  
*HTRCXRev function.*
- void [HTRCXSelectDisplay](#) (const byte src, const unsigned int value)  
*HTRCXSelectDisplay function.*
- void [HTRCXSelectProgram](#) (const byte prog)  
*HTRCXSelectProgram function.*
- void [HTRCXSendSerial](#) (const byte first, const byte count)  
*HTRCXSendSerial function.*
- void [HTRCXSetDirection](#) (const byte outputs, const byte dir)  
*HTRCXSetDirection function.*
- void [HTRCXSetEvent](#) (const byte evt, const byte src, const byte type)

*HTRCXSetEvent function.*

- void [HTRCXSetGlobalDirection](#) (const byte outputs, const byte dir)  
*HTRCXSetGlobalDirection function.*
- void [HTRCXSetGlobalOutput](#) (const byte outputs, const byte mode)  
*HTRCXSetGlobalOutput function.*
- void [HTRCXSetMaxPower](#) (const byte outputs, const byte pwrsrc, const byte pwrval)  
*HTRCXSetMaxPower function.*
- void [HTRCXSetMessage](#) (const byte msg)  
*HTRCXSetMessage function.*
- void [HTRCXSetOutput](#) (const byte outputs, const byte mode)  
*HTRCXSetOutput function.*
- void [HTRCXSetPower](#) (const byte outputs, const byte pwrsrc, const byte pwrval)  
*HTRCXSetPower function.*
- void [HTRCXSetPriority](#) (const byte p)  
*HTRCXSetPriority function.*
- void [HTRCXSetSensorMode](#) (const byte port, const byte mode)  
*HTRCXSetSensorMode function.*
- void [HTRCXSetSensorType](#) (const byte port, const byte type)  
*HTRCXSetSensorType function.*
- void [HTRCXSetSleepTime](#) (const byte t)  
*HTRCXSetSleepTime function.*
- void [HTRCXSetTxPower](#) (const byte pwr)  
*HTRCXSetTxPower function.*
- void [HTRCXSetWatch](#) (const byte hours, const byte minutes)  
*HTRCXSetWatch function.*
- void [HTRCXStartTask](#) (const byte t)  
*HTRCXStartTask function.*

- void [HTRCXStopAllTasks](#) (void)  
*HTRCXStopAllTasks function.*
- void [HTRCXStopTask](#) (const byte t)  
*HTRCXStopTask function.*
- void [HTRCXToggle](#) (const byte outputs)  
*HTRCXToggle function.*
- void [HTRCXUnmuteSound](#) (void)  
*HTRCXUnmuteSound function.*
- void [HTScoutCalibrateSensor](#) (void)  
*HTScoutCalibrateSensor function.*
- void [HTScoutMuteSound](#) (void)  
*HTScoutMuteSound function.*
- void [HTScoutSelectSounds](#) (const byte grp)  
*HTScoutSelectSounds function.*
- void [HTScoutSendVLL](#) (const byte src, const unsigned int value)  
*HTScoutSendVLL function.*
- void [HTScoutSetEventFeedback](#) (const byte src, const unsigned int value)  
*HTScoutSetEventFeedback function.*
- void [HTScoutSetLight](#) (const byte x)  
*HTScoutSetLight function.*
- void [HTScoutSetScoutMode](#) (const byte mode)  
*HTScoutSetScoutMode function.*
- void [HTScoutSetSensorClickTime](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorClickTime function.*
- void [HTScoutSetSensorHysteresis](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorHysteresis function.*
- void [HTScoutSetSensorLowerLimit](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorLowerLimit function.*

- void [HTScoutSetSensorUpperLimit](#) (const byte src, const unsigned int value)  
*HTScoutSetSensorUpperLimit function.*
- void [HTScoutUnmuteSound](#) (void)  
*HTScoutUnmuteSound function.*
- void [SetSensorMSPressure](#) (const byte &port)  
*Configure a mindsensors pressure sensor.*
- void [SetSensorMSDROD](#) (const byte &port, bool bActive)  
*Configure a mindsensors DROD sensor.*
- void [SetSensorNXTSumoEyes](#) (const byte &port, bool bLong)  
*Configure a mindsensors SumoEyes sensor.*
- int [SensorMSPressure](#) (const byte &port)  
*Read mindsensors pressure sensor.*
- char [SensorNXTSumoEyes](#) (const byte &port)  
*Read mindsensors NXTSumoEyes obstacle zone.*
- int [SensorMSCompass](#) (const byte &port, const byte i2caddr)  
*Read mindsensors compass value.*
- int [SensorMSDROD](#) (const byte &port)  
*Read mindsensors DROD value.*
- int [SensorNXTSumoEyesRaw](#) (const byte &port)  
*Read mindsensors NXTSumoEyes raw value.*
- int [SensorMSPressureRaw](#) (const byte &port)  
*Read mindsensors raw pressure value.*
- bool [ReadSensorMSAccel](#) (const byte port, const byte i2caddr, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSPlayStation](#) (const byte port, const byte i2caddr, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)  
*Read mindsensors playstation controller values.*
- bool [ReadSensorMSRTClock](#) (const byte port, byte &sec, byte &min, byte &hrs, byte &dow, byte &date, byte &month, byte &year)

*Read mindsensors RTClock values.*

- bool [ReadSensorMSTilt](#) (const byte &port, const byte &i2caddr, byte &x, byte &y, byte &z)

*Read mindsensors tilt values.*

- bool [PFMateSend](#) (const byte &port, const byte &i2caddr, const byte &channel, const byte &motors, const byte &cmdA, const byte &spdA, const byte &cmdB, const byte &spdB)

*Send PFMate command.*

- bool [PFMateSendRaw](#) (const byte &port, const byte &i2caddr, const byte &channel, const byte &b1, const byte &b2)

*Send raw PFMate command.*

- int [MSReadValue](#) (const byte port, const byte i2caddr, const byte reg, const byte numbytes)

*Read a mindsensors device value.*

- char [MSEnergize](#) (const byte port, const byte i2caddr)

*Turn on power to device.*

- char [MSDeenergize](#) (const byte port, const byte i2caddr)

*Turn off power to device.*

- char [MSADPAOn](#) (const byte port, const byte i2caddr)

*Turn on mindsensors ADPA mode.*

- char [MSADPAOff](#) (const byte port, const byte i2caddr)

*Turn off mindsensors ADPA mode.*

- char [DISTNxGP2D12](#) (const byte port, const byte i2caddr)

*Configure DISTNx as GP2D12.*

- char [DISTNxGP2D120](#) (const byte port, const byte i2caddr)

*Configure DISTNx as GP2D120.*

- char [DISTNxGP2YA02](#) (const byte port, const byte i2caddr)

*Configure DISTNx as GP2YA02.*

- char [DISTNxGP2YA21](#) (const byte port, const byte i2caddr)

*Configure DISTNx as GP2YA21.*

- int [DISTNxDistance](#) (const byte port, const byte i2caddr)  
*Read DISTNx distance value.*
- int [DISTNxMaxDistance](#) (const byte port, const byte i2caddr)  
*Read DISTNx maximum distance value.*
- int [DISTNxMinDistance](#) (const byte port, const byte i2caddr)  
*Read DISTNx minimum distance value.*
- byte [DISTNxModuleType](#) (const byte port, const byte i2caddr)  
*Read DISTNx module type value.*
- byte [DISTNxNumPoints](#) (const byte port, const byte i2caddr)  
*Read DISTNx num points value.*
- int [DISTNxVoltage](#) (const byte port, const byte i2caddr)  
*Read DISTNx voltage value.*
- char [ACCLNxCalibrateX](#) (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx X-axis.*
- char [ACCLNxCalibrateXEnd](#) (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx X-axis.*
- char [ACCLNxCalibrateY](#) (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx Y-axis.*
- char [ACCLNxCalibrateYEnd](#) (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx Y-axis.*
- char [ACCLNxCalibrateZ](#) (const byte port, const byte i2caddr)  
*Calibrate ACCL-Nx Z-axis.*
- char [ACCLNxCalibrateZEnd](#) (const byte port, const byte i2caddr)  
*Stop calibrating ACCL-Nx Z-axis.*
- char [ACCLNxResetCalibration](#) (const byte port, const byte i2caddr)  
*Reset ACCL-Nx calibration.*
- char [SetACCLNxSensitivity](#) (const byte port, const byte i2caddr, byte slevel)  
*Set ACCL-Nx sensitivity.*

- byte [ACCLNxSensitivity](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx sensitivity value.*
- int [ACCLNxXOffset](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx X offset value.*
- int [ACCLNxXRange](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx X range value.*
- int [ACCLNxYOffset](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx Y offset value.*
- int [ACCLNxYRange](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx Y range value.*
- int [ACCLNxZOffset](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx Z offset value.*
- int [ACCLNxZRange](#) (const byte port, const byte i2caddr)  
*Read ACCL-Nx Z range value.*
- char [PSPNxDigital](#) (const byte &port, const byte &i2caddr)  
*Configure PSPNx in digital mode.*
- char [PSPNxAnalog](#) (const byte &port, const byte &i2caddr)  
*Configure PSPNx in analog mode.*
- unsigned int [NXTServoPosition](#) (const byte &port, const byte &i2caddr, const byte servo)  
*Read NXTServo servo position value.*
- byte [NXTServoSpeed](#) (const byte &port, const byte &i2caddr, const byte servo)  
*Read NXTServo servo speed value.*
- byte [NXTServoBatteryVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTServo battery voltage value.*
- char [SetNXTServoSpeed](#) (const byte &port, const byte &i2caddr, const byte servo, const byte &speed)  
*Set NXTServo servo motor speed.*

- char [SetNXTServoQuickPosition](#) (const byte &port, const byte &i2caddr, const byte servo, const byte &qpos)  
*Set NXTServo servo motor quick position.*
- char [SetNXTServoPosition](#) (const byte &port, const byte &i2caddr, const byte servo, const byte &pos)  
*Set NXTServo servo motor position.*
- char [NXTServoReset](#) (const byte &port, const byte &i2caddr)  
*Reset NXTServo properties.*
- char [NXTServoHaltMacro](#) (const byte &port, const byte &i2caddr)  
*Halt NXTServo macro.*
- char [NXTServoResumeMacro](#) (const byte &port, const byte &i2caddr)  
*Resume NXTServo macro.*
- char [NXTServoPauseMacro](#) (const byte &port, const byte &i2caddr)  
*Pause NXTServo macro.*
- char [NXTServoInit](#) (const byte &port, const byte &i2caddr, const byte servo)  
*Initialize NXTServo servo properties.*
- char [NXTServoGotoMacroAddress](#) (const byte &port, const byte &i2caddr, const byte &macro)  
*Goto NXTServo macro address.*
- char [NXTServoEditMacro](#) (const byte &port, const byte &i2caddr)  
*Edit NXTServo macro.*
- char [NXTServoQuitEdit](#) (const byte &port)  
*Quit NXTServo macro edit mode.*
- char [NXTHIDAsciiMode](#) (const byte &port, const byte &i2caddr)  
*Set NXTHID into ASCII data mode.*
- char [NXTHIDDirectMode](#) (const byte &port, const byte &i2caddr)  
*Set NXTHID into direct data mode.*
- char [NXTHIDTransmit](#) (const byte &port, const byte &i2caddr)  
*Transmit NXTHID character.*

- char [NXTHIDLoadCharacter](#) (const byte &port, const byte &i2caddr, const byte &modifier, const byte &character)  
*Load NXTHID character.*
- char [NXTPowerMeterResetCounters](#) (const byte &port, const byte &i2caddr)  
*Reset NXTPowerMeter counters.*
- int [NXTPowerMeterPresentCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present current.*
- int [NXTPowerMeterPresentVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present voltage.*
- int [NXTPowerMeterCapacityUsed](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter capacity used.*
- int [NXTPowerMeterPresentPower](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter present power.*
- long [NXTPowerMeterTotalPowerConsumed](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter total power consumed.*
- int [NXTPowerMeterMaxCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter maximum current.*
- int [NXTPowerMeterMinCurrent](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter minimum current.*
- int [NXTPowerMeterMaxVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter maximum voltage.*
- int [NXTPowerMeterMinVoltage](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter minimum voltage.*
- long [NXTPowerMeterElapsedTime](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter elapsed time.*
- int [NXTPowerMeterErrorCount](#) (const byte &port, const byte &i2caddr)  
*Read NXTPowerMeter error count.*
- char [NXTLLineLeaderPowerDown](#) (const byte &port, const byte &i2caddr)  
*Powerdown NXTLLineLeader device.*

- char [NXtLineLeaderPowerUp](#) (const byte &port, const byte &i2caddr)  
*Powerup NXtLineLeader device.*
- char [NXtLineLeaderInvert](#) (const byte &port, const byte &i2caddr)  
*Invert NXtLineLeader colors.*
- char [NXtLineLeaderReset](#) (const byte &port, const byte &i2caddr)  
*Reset NXtLineLeader color inversion.*
- char [NXtLineLeaderSnapshot](#) (const byte &port, const byte &i2caddr)  
*Take NXtLineLeader line snapshot.*
- char [NXtLineLeaderCalibrateWhite](#) (const byte &port, const byte &i2caddr)  
*Calibrate NXtLineLeader white color.*
- char [NXtLineLeaderCalibrateBlack](#) (const byte &port, const byte &i2caddr)  
*Calibrate NXtLineLeader black color.*
- char [NXtLineLeaderSteering](#) (const byte &port, const byte &i2caddr)  
*Read NXtLineLeader steering.*
- char [NXtLineLeaderAverage](#) (const byte &port, const byte &i2caddr)  
*Read NXtLineLeader average.*
- byte [NXtLineLeaderResult](#) (const byte &port, const byte &i2caddr)  
*Read NXtLineLeader result.*
- char [SetNXtLineLeaderSetpoint](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXtLineLeader setpoint.*
- char [SetNXtLineLeaderKpValue](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXtLineLeader Kp value.*
- char [SetNXtLineLeaderKiValue](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXtLineLeader Ki value.*
- char [SetNXtLineLeaderKdValue](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXtLineLeader Kd value.*

- char [SetNXTLineLeaderKpFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kp factor.*
- char [SetNXTLineLeaderKiFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Ki factor.*
- char [SetNXTLineLeaderKdFactor](#) (const byte &port, const byte &i2caddr, const byte &value)  
*Write NXTLineLeader Kd factor.*
- char [NRLink2400](#) (const byte port, const byte i2caddr)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink4800](#) (const byte port, const byte i2caddr)  
*Configure NRLink in 4800 baud mode.*
- char [NRLinkFlush](#) (const byte port, const byte i2caddr)  
*Flush NRLink buffers.*
- char [NRLinkIRLong](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRShort](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR short mode.*
- char [NRLinkSetPF](#) (const byte port, const byte i2caddr)  
*Configure NRLink in power function mode.*
- char [NRLinkSetRCX](#) (const byte port, const byte i2caddr)  
*Configure NRLink in RCX mode.*
- char [NRLinkSetTrain](#) (const byte port, const byte i2caddr)  
*Configure NRLink in IR train mode.*
- char [NRLinkTxRaw](#) (const byte port, const byte i2caddr)  
*Configure NRLink in raw IR transmit mode.*
- byte [NRLinkStatus](#) (const byte port, const byte i2caddr)  
*Read NRLink status.*

- char [RunNRLinkMacro](#) (const byte port, const byte i2caddr, const byte macro)  
*Run NRLink macro.*
- char [WriteNRLinkBytes](#) (const byte port, const byte i2caddr, const byte data[ ])  
*Write data to NRLink.*
- bool [ReadNRLinkBytes](#) (const byte port, const byte i2caddr, byte &data[ ])  
*Read data from NRLink.*
- char [MSIRTrain](#) (const byte port, const byte i2caddr, const byte channel, const byte func)  
*MSIRTrain function.*
- char [MSPFComboDirect](#) (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirect function.*
- char [MSPFComboPWM](#) (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWM function.*
- char [MSPFRawOutput](#) (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutput function.*
- char [MSPFRepeat](#) (const byte port, const byte i2caddr, const byte count, const unsigned int delay)  
*MSPFRepeat function.*
- char [MSPFSingleOutputCST](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCST function.*
- char [MSPFSingleOutputPWM](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWM function.*
- char [MSPFSinglePin](#) (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePin function.*
- char [MSPFTrain](#) (const byte port, const byte i2caddr, const byte channel, const byte func)

*MSPFTrain function.*

- void [MSRCXSetNRLinkPort](#) (const byte port, const byte i2caddr)  
*MSRCXSetIRLinkPort function.*
- int [MSRCXBatteryLevel](#) (void)  
*MSRCXBatteryLevel function.*
- int [MSRCXPoll](#) (const byte src, const byte value)  
*MSRCXPoll function.*
- int [MSRCXPollMemory](#) (const unsigned int address)  
*MSRCXPollMemory function.*
- void [MSRCXAbsVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXAbsVar function.*
- void [MSRCXAddToDatalog](#) (const byte src, const unsigned int value)  
*MSRCXAddToDatalog function.*
- void [MSRCXAndVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXAndVar function.*
- void [MSRCXBoot](#) (void)  
*MSRCXBoot function.*
- void [MSRCXCalibrateEvent](#) (const byte evt, const byte low, const byte hi, const byte hyst)  
*MSRCXCalibrateEvent function.*
- void [MSRCXClearAllEvents](#) (void)  
*MSRCXClearAllEvents function.*
- void [MSRCXClearCounter](#) (const byte counter)  
*MSRCXClearCounter function.*
- void [MSRCXClearMsg](#) (void)  
*MSRCXClearMsg function.*
- void [MSRCXClearSensor](#) (const byte port)  
*MSRCXClearSensor function.*

- void [MSRCXClearSound](#) (void)  
*MSRCXClearSound function.*
- void [MSRCXClearTimer](#) (const byte timer)  
*MSRCXClearTimer function.*
- void [MSRCXCreateDatalog](#) (const unsigned int size)  
*MSRCXCreateDatalog function.*
- void [MSRCXDecCounter](#) (const byte counter)  
*MSRCXDecCounter function.*
- void [MSRCXDeleteSub](#) (const byte s)  
*MSRCXDeleteSub function.*
- void [MSRCXDeleteSubs](#) (void)  
*MSRCXDeleteSubs function.*
- void [MSRCXDeleteTask](#) (const byte t)  
*MSRCXDeleteTask function.*
- void [MSRCXDeleteTasks](#) (void)  
*MSRCXDeleteTasks function.*
- void [MSRCXDisableOutput](#) (const byte outputs)  
*MSRCXDisableOutput function.*
- void [MSRCXDivVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXDivVar function.*
- void [MSRCXEnableOutput](#) (const byte outputs)  
*MSRCXEnableOutput function.*
- void [MSRCXEvent](#) (const byte src, const unsigned int value)  
*MSRCXEvent function.*
- void [MSRCXFloat](#) (const byte outputs)  
*MSRCXFloat function.*
- void [MSRCXFwd](#) (const byte outputs)

*MSRCXFwd function.*

- void [MSRCXIncCounter](#) (const byte counter)  
*MSRCXIncCounter function.*
- void [MSRCXInvertOutput](#) (const byte outputs)  
*MSRCXInvertOutput function.*
- void [MSRCXMulVar](#) (const byte varnum, const byte src, unsigned int value)  
*MSRCXMulVar function.*
- void [MSRCXMuteSound](#) (void)  
*MSRCXMuteSound function.*
- void [MSRCXObvertOutput](#) (const byte outputs)  
*MSRCXObvertOutput function.*
- void [MSRCXOff](#) (const byte outputs)  
*MSRCXOff function.*
- void [MSRCXOn](#) (const byte outputs)  
*MSRCXOn function.*
- void [MSRCXOnFor](#) (const byte outputs, const unsigned int ms)  
*MSRCXOnFor function.*
- void [MSRCXOnFwd](#) (const byte outputs)  
*MSRCXOnFwd function.*
- void [MSRCXOnRev](#) (const byte outputs)  
*MSRCXOnRev function.*
- void [MSRCXOrVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXOrVar function.*
- void [MSRCXPBTurnOff](#) (void)  
*MSRCXPBTurnOff function.*
- void [MSRCXPing](#) (void)  
*MSRCXPing function.*
- void [MSRCXPlaySound](#) (const byte snd)

*MSRCXPlaySound function.*

- void [MSRCXPlayTone](#) (const unsigned int freq, const byte duration)  
*MSRCXPlayTone function.*
- void [MSRCXPlayToneVar](#) (const byte varnum, const byte duration)  
*MSRCXPlayToneVar function.*
- void [MSRCXRemote](#) (unsigned int cmd)  
*MSRCXRemote function.*
- void [MSRCXReset](#) (void)  
*MSRCXReset function.*
- void [MSRCXRev](#) (const byte outputs)  
*MSRCXRev function.*
- void [MSRCXSelectDisplay](#) (const byte src, const unsigned int value)  
*MSRCXSelectDisplay function.*
- void [MSRCXSelectProgram](#) (const byte prog)  
*MSRCXSelectProgram function.*
- void [MSRCXSendSerial](#) (const byte first, const byte count)  
*MSRCXSendSerial function.*
- void [MSRCXSet](#) (const byte dstsrc, const byte dstval, const byte src, unsigned int value)  
*MSRCXSet function.*
- void [MSRCXSetDirection](#) (const byte outputs, const byte dir)  
*MSRCXSetDirection function.*
- void [MSRCXSetEvent](#) (const byte evt, const byte src, const byte type)  
*MSRCXSetEvent function.*
- void [MSRCXSetGlobalDirection](#) (const byte outputs, const byte dir)  
*MSRCXSetGlobalDirection function.*
- void [MSRCXSetGlobalOutput](#) (const byte outputs, const byte mode)  
*MSRCXSetGlobalOutput function.*

- void [MSRCXSetMaxPower](#) (const byte outputs, const byte pwsrc, const byte pwrval)  
*MSRCXSetMaxPower function.*
- void [MSRCXSetMessage](#) (const byte msg)  
*MSRCXSetMessage function.*
- void [MSRCXSetOutput](#) (const byte outputs, const byte mode)  
*MSRCXSetOutput function.*
- void [MSRCXSetPower](#) (const byte outputs, const byte pwsrc, const byte pwrval)  
*MSRCXSetPower function.*
- void [MSRCXSetPriority](#) (const byte p)  
*MSRCXSetPriority function.*
- void [MSRCXSetSensorMode](#) (const byte port, const byte mode)  
*MSRCXSetSensorMode function.*
- void [MSRCXSetSensorType](#) (const byte port, const byte type)  
*MSRCXSetSensorType function.*
- void [MSRCXSetSleepTime](#) (const byte t)  
*MSRCXSetSleepTime function.*
- void [MSRCXSetTxPower](#) (const byte pwr)  
*MSRCXSetTxPower function.*
- void [MSRCXSetUserDisplay](#) (const byte src, const unsigned int value, const byte precision)  
*MSRCXSetUserDisplay function.*
- void [MSRCXSetVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSetVar function.*
- void [MSRCXSetWatch](#) (const byte hours, const byte minutes)  
*MSRCXSetWatch function.*
- void [MSRCXSgnVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSgnVar function.*

- void [MSRCXStartTask](#) (const byte t)  
*MSRCXStartTask function.*
- void [MSRCXStopAllTasks](#) (void)  
*MSRCXStopAllTasks function.*
- void [MSRCXStopTask](#) (const byte t)  
*MSRCXStopTask function.*
- void [MSRCXSubVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSubVar function.*
- void [MSRCXSumVar](#) (const byte varnum, const byte src, const unsigned int value)  
*MSRCXSumVar function.*
- void [MSRCXToggle](#) (const byte outputs)  
*MSRCXToggle function.*
- void [MSRCXUnlock](#) (void)  
*MSRCXUnlock function.*
- void [MSRCXUnmuteSound](#) (void)  
*MSRCXUnmuteSound function.*
- void [MSScoutCalibrateSensor](#) (void)  
*MSScoutCalibrateSensor function.*
- void [MSScoutMuteSound](#) (void)  
*MSScoutMuteSound function.*
- void [MSScoutSelectSounds](#) (const byte grp)  
*MSScoutSelectSounds function.*
- void [MSScoutSendVLL](#) (const byte src, const unsigned int value)  
*MSScoutSendVLL function.*
- void [MSScoutSetCounterLimit](#) (const byte ctr, const byte src, const unsigned int value)  
*MSScoutSetCounterLimit function.*

- void [MSScoutSetEventFeedback](#) (const byte src, const unsigned int value)  
*MSScoutSetEventFeedback function.*
- void [MSScoutSetLight](#) (const byte x)  
*MSScoutSetLight function.*
- void [MSScoutSetScoutMode](#) (const byte mode)  
*MSScoutSetScoutMode function.*
- void [MSScoutSetScoutRules](#) (const byte m, const byte t, const byte l, const byte tm, const byte fx)  
*MSScoutSetScoutRules function.*
- void [MSScoutSetSensorClickTime](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorClickTime function.*
- void [MSScoutSetSensorHysteresis](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorHysteresis function.*
- void [MSScoutSetSensorLowerLimit](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorLowerLimit function.*
- void [MSScoutSetSensorUpperLimit](#) (const byte src, const unsigned int value)  
*MSScoutSetSensorUpperLimit function.*
- void [MSScoutSetTimerLimit](#) (const byte tmr, const byte src, const unsigned int value)  
*MSScoutSetTimerLimit function.*
- void [MSScoutUnmuteSound](#) (void)  
*MSScoutUnmuteSound function.*
- bool [RFIDInit](#) (const byte &port)  
*RFIDInit function.*
- bool [RFIDMode](#) (const byte &port, const byte &mode)  
*RFIDMode function.*
- byte [RFIDStatus](#) (const byte &port)  
*RFIDStatus function.*
- bool [RFIDRead](#) (const byte &port, byte &output[ ])  
*RFIDRead function.*

- bool [RFIDStop](#) (const byte &port)  
*RFIDStop function.*
- bool [RFIDReadSingle](#) (const byte &port, byte &output[ ])  
*RFIDReadSingle function.*
- bool [RFIDReadContinuous](#) (const byte &port, byte &output[ ])  
*RFIDReadContinuous function.*
- bool [SensorDIGPSStatus](#) (byte port)  
*SensorDIGPSStatus function.*
- long [SensorDIGPSTime](#) (byte port)  
*SensorDIGPSTime function.*
- long [SensorDIGPSLatitude](#) (byte port)  
*SensorDIGPSLatitude function.*
- long [SensorDIGPSLongitude](#) (byte port)  
*SensorDIGPSLongitude function.*
- long [SensorDIGPSVelocity](#) (byte port)  
*SensorDIGPSVelocity function.*
- int [SensorDIGPSHeading](#) (byte port)  
*SensorDIGPSHeading function.*
- long [SensorDIGPSDistanceToWaypoint](#) (byte port)  
*SensorDIGPSDistanceToWaypoint function.*
- int [SensorDIGPSHeadingToWaypoint](#) (byte port)  
*SensorDIGPSHeadingToWaypoint function.*
- int [SensorDIGPSRelativeHeading](#) (byte port)  
*SensorDIGPSRelativeHeading function.*
- bool [SetSensorDIGPSWaypoint](#) (byte port, long latitude, long longitude)  
*SetSensorDIGPSWaypoint function.*
- bool [SetSensorDIGyroEx](#) (const byte port, byte scale, byte odr, byte bw)  
*SetSensorDIGyroEx function.*

- bool [SetSensorDIGyro](#) (const byte port)  
*SetSensorDIGyro function.*
- bool [ReadSensorDIGyroRaw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIGyroRaw function.*
- bool [ReadSensorDIGyro](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIGyro function.*
- int [SensorDIGyroTemperature](#) (const byte port)  
*SensorDIGyroTemperature function.*
- byte [SensorDIGyroStatus](#) (const byte port)  
*SensorDIGyroStatus function.*
- bool [SetSensorDIAccEx](#) (const byte port, byte mode)  
*SetSensorDIAccEx function.*
- bool [SetSensorDIAcc](#) (const byte port)  
*SetSensorDIAcc function.*
- bool [ReadSensorDIAccRaw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAccRaw function.*
- bool [ReadSensorDIAcc](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAcc function.*
- bool [ReadSensorDIAcc8Raw](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAcc8Raw function.*
- bool [ReadSensorDIAcc8](#) (const byte port, [VectorType](#) &vector)  
*ReadSensorDIAcc8 function.*
- byte [SensorDIAccStatus](#) (const byte port)  
*SensorDIAccStatus function.*
- bool [ReadSensorDIAccDrift](#) (const byte port, int &x, int &y, int &z)  
*ReadSensorDIAccDrift function.*
- bool [SetSensorDIAccDrift](#) (const byte port, int x, int y, int z)  
*SetSensorDIAccDrift function.*

- bool [ResetMIXG1300L](#) (byte port)  
*ResetMIXG1300L function.*
- int [SensorMIXG1300LScale](#) (byte port)  
*SensorMIXG1300LScale function.*
- bool [SetSensorMIXG1300LScale](#) (byte port, const byte scale)  
*SetSensorMIXG1300LScale function.*
- bool [ReadSensorMIXG1300L](#) (byte port, [XGPacketType](#) &packet)  
*ReadSensorMIXG1300L function.*
- float [sqrt](#) (float x)  
*Compute square root.*
- float [cos](#) (float x)  
*Compute cosine.*
- float [sin](#) (float x)  
*Compute sine.*
- float [tan](#) (float x)  
*Compute tangent.*
- float [acos](#) (float x)  
*Compute arc cosine.*
- float [asin](#) (float x)  
*Compute arc sine.*
- float [atan](#) (float x)  
*Compute arc tangent.*
- float [atan2](#) (float y, float x)  
*Compute arc tangent with 2 parameters.*
- float [cosh](#) (float x)  
*Compute hyperbolic cosine.*
- float [sinh](#) (float x)  
*Compute hyperbolic sine.*

- float **tanh** (float x)  
*Compute hyperbolic tangent.*
- float **exp** (float x)  
*Compute exponential function.*
- float **log** (float x)  
*Compute natural logarithm.*
- float **log10** (float x)  
*Compute common logarithm.*
- long **trunc** (float x)  
*Compute integral part.*
- float **frac** (float x)  
*Compute fractional part.*
- float **pow** (float base, float exponent)  
*Raise to power.*
- float **ceil** (float x)  
*Round up value.*
- float **floor** (float x)  
*Round down value.*
- long **muldiv32** (long a, long b, long c)  
*Multiply and divide.*
- float **cosd** (float x)  
*Compute cosine (degrees).*
- float **sind** (float x)  
*Compute sine (degrees).*
- float **tand** (float x)  
*Compute tangent (degrees).*
- float **acosd** (float x)  
*Compute arc cosine (degrees).*

- float [asind](#) (float x)  
*Compute arc sine (degrees).*
- float [atand](#) (float x)  
*Compute arc tangent (degrees).*
- float [atan2d](#) (float y, float x)  
*Compute arc tangent with 2 parameters (degrees).*
- float [coshd](#) (float x)  
*Compute hyperbolic cosine (degrees).*
- float [sinhd](#) (float x)  
*Compute hyperbolic sine (degrees).*
- float [tanhd](#) (float x)  
*Compute hyperbolic tangent (degrees).*
- byte [bcd2dec](#) (byte bcd)  
*Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.*
- bool [isNAN](#) (float value)  
*Is the value NaN.*
- char [sign](#) (variant num)  
*Sign value.*
- void [VectorCross](#) ([VectorType](#) a, [VectorType](#) b, [VectorType](#) &out)  
*VectorCross function.*
- float [VectorDot](#) ([VectorType](#) a, [VectorType](#) b)  
*VectorDot function.*
- void [VectorNormalize](#) ([VectorType](#) &a)  
*VectorNormalize function.*
- int [fclose](#) (byte handle)  
*Close file.*
- int [remove](#) (string filename)

*Remove file.*

- int `rename` (string old, string new)  
*Rename file.*
- char `fgetc` (byte handle)  
*Get character from file.*
- string `fgets` (string &str, int num, byte handle)  
*Get string from file.*
- int `feof` (byte handle)  
*Check End-of-file indicator.*
- void `set_fopen_size` (unsigned long fsize)  
*Set the default fopen file size.*
- byte `fopen` (string filename, const string mode)  
*Open file.*
- int `fflush` (byte handle)  
*Flush file.*
- unsigned long `ftell` (byte handle)  
*Get current position in file.*
- char `fputc` (char ch, byte handle)  
*Write character to file.*
- int `fputs` (string str, byte handle)  
*Write string to file.*
- void `printf` (string format, variant value)  
*Print formatted data to stdout.*
- void `fprintf` (byte handle, string format, variant value)  
*Write formatted data to file.*
- void `sprintf` (string &str, string format, variant value)  
*Write formatted data to string.*
- int `fseek` (byte handle, long offset, int origin)

*Reposition file position indicator.*

- void [rewind](#) (byte handle)  
*Set position indicator to the beginning.*
- int [getchar](#) ()  
*Get character from stdin.*
- void [abort](#) ()  
*Abort current process.*
- variant [abs](#) (variant num)  
*Absolute value.*
- long [srand](#) (long seed)  
*Seed the random number generator.*
- unsigned long [rand](#) ()  
*Generate random number.*
- int [Random](#) (unsigned int n=0)  
*Generate random number.*
- void [SysRandomNumber](#) ([RandomNumberType](#) &args)  
*Draw a random number.*
- void [SysRandomEx](#) ([RandomExType](#) &args)  
*Call the enhanced random number function.*
- int [atoi](#) (const string &str)  
*Convert string to integer.*
- long [atol](#) (const string &str)  
*Convert string to long integer.*
- long [labs](#) (long n)  
*Absolute value.*
- float [atof](#) (const string &str)  
*Convert string to float.*
- float [strtod](#) (const string &str, string &endptr)

*Convert string to float.*

- long [strtol](#) (const string &str, string &endptr, int base=10)  
*Convert string to long integer.*
- long [strtoul](#) (const string &str, string &endptr, int base=10)  
*Convert string to unsigned long integer.*
- [div\\_t div](#) (int numer, int denom)  
*Integral division.*
- [ldiv\\_t ldiv](#) (long numer, long denom)  
*Integral division.*
- variant [StrToNum](#) (string str)  
*Convert string to number.*
- unsigned int [StrLen](#) (string str)  
*Get string length.*
- byte [StrIndex](#) (string str, unsigned int idx)  
*Extract a character from a string.*
- string [NumToStr](#) (variant num)  
*Convert number to string.*
- string [StrCat](#) (string str1, string str2, string strN)  
*Concatenate strings.*
- string [SubStr](#) (string str, unsigned int idx, unsigned int len)  
*Extract a portion of a string.*
- string [Flatten](#) (variant num)  
*Flatten a number to a string.*
- string [StrReplace](#) (string str, unsigned int idx, string strnew)  
*Replace a portion of a string.*
- string [FormatNum](#) (string fmt, variant num)  
*Format a number.*
- string [FlattenVar](#) (variant x)

*Flatten any data to a string.*

- int [UnflattenVar](#) (string str, variant &x)  
*Unflatten a string into a data type.*
- int [Pos](#) (string Substr, string S)  
*Find substring position.*
- string [ByteArrayToStr](#) (byte data[ ])  
*Convert a byte array to a string.*
- void [ByteArrayToStrEx](#) (byte data[ ], string &str)  
*Convert a byte array to a string.*
- void [StrToByteArray](#) (string str, byte &data[ ])  
*Convert a string to a byte array.*
- string [Copy](#) (string str, unsigned int idx, unsigned int len)  
*Copy a portion of a string.*
- string [MidStr](#) (string str, unsigned int idx, unsigned int len)  
*Copy a portion from the middle of a string.*
- string [RightStr](#) (string str, unsigned int size)  
*Copy a portion from the end of a string.*
- string [LeftStr](#) (string str, unsigned int size)  
*Copy a portion from the start of a string.*
- int [strlen](#) (const string &str)  
*Get string length.*
- string [strcat](#) (string &dest, const string &src)  
*Concatenate strings.*
- string [strncat](#) (string &dest, const string &src, unsigned int num)  
*Append characters from string.*
- string [strcpy](#) (string &dest, const string &src)  
*Copy string.*
- string [strncpy](#) (string &dest, const string &src, unsigned int num)

*Copy characters from string.*

- int **strcmp** (const string &str1, const string &str2)  
*Compare two strings.*
- int **strncmp** (const string &str1, const string &str2, unsigned int num)  
*Compare characters of two strings.*
- void **memcpy** (variant dest, variant src, byte num)  
*Copy memory.*
- void **memmove** (variant dest, variant src, byte num)  
*Move memory.*
- char **memcmp** (variant ptr1, variant ptr2, byte num)  
*Compare two blocks of memory.*
- unsigned long **addressOf** (variant data)  
*Get the absolute address of a variable.*
- unsigned long **reladdressOf** (variant data)  
*Get the relative address of a variable.*
- unsigned long **addressOfEx** (variant data, bool relative)  
*Get the absolute or relative address of a variable.*
- int **isupper** (int c)  
*Check if character is uppercase letter.*
- int **islower** (int c)  
*Check if character is lowercase letter.*
- int **isalpha** (int c)  
*Check if character is alphabetic.*
- int **isdigit** (int c)  
*Check if character is decimal digit.*
- int **isalnum** (int c)  
*Check if character is alphanumeric.*
- int **isspace** (int c)

*Check if character is a white-space.*

- int [isctrl](#) (int c)  
*Check if character is a control character.*
- int [isprint](#) (int c)  
*Check if character is printable.*
- int [isgraph](#) (int c)  
*Check if character has graphical representation.*
- int [ispunct](#) (int c)  
*Check if character is a punctuation.*
- int [isxdigit](#) (int c)  
*Check if character is hexadecimal digit.*
- int [toupper](#) (int c)  
*Convert lowercase letter to uppercase.*
- int [tolower](#) (int c)  
*Convert uppercase letter to lowercase.*
- void [glInit](#) ()  
*Initialize graphics library.*
- void [glSet](#) (int glType, int glValue)  
*Set graphics library options.*
- int [glBeginObject](#) ()  
*Begin defining an object.*
- void [glEndObject](#) ()  
*Stop defining an object.*
- void [glObjectAction](#) (int glObjectId, int glAction, int glValue)  
*Perform an object action.*
- void [glAddVertex](#) (int glX, int glY, int glZ)  
*Add a vertex to an object.*
- void [glBegin](#) (int glBeginMode)

*Begin a new polygon for the current object.*

- void `glEnd ()`  
*Finish a polygon for the current object.*
- void `glBeginRender ()`  
*Begin a new render.*
- void `glCallObject (int glObjectId)`  
*Call a graphic object.*
- void `glFinishRender ()`  
*Finish the current render.*
- void `glSetAngleX (int glValue)`  
*Set the X axis angle.*
- void `glAddToAngleX (int glValue)`  
*Add to the X axis angle.*
- void `glSetAngleY (int glValue)`  
*Set the Y axis angle.*
- void `glAddToAngleY (int glValue)`  
*Add to the Y axis angle.*
- void `glSetAngleZ (int glValue)`  
*Set the Z axis angle.*
- void `glAddToAngleZ (int glValue)`  
*Add to the Z axis angle.*
- int `glSin32768 (int glAngle)`  
*Table-based sine scaled by 32768.*
- int `glCos32768 (int glAngle)`  
*Table-based cosine scaled by 32768.*
- int `glBox (int glMode, int glSizeX, int glSizeY, int glSizeZ)`  
*Create a 3D box.*
- int `glCube (int glMode, int glSize)`

*Create a 3D cube.*

- int [glPyramid](#) (int glMode, int glSizeX, int glSizeY, int glSizeZ)  
*Create a 3D pyramid.*
- void [PosRegEnable](#) (byte output, byte p=PID\_3, byte i=PID\_1, byte d=PID\_1)  
*Enable absolute position regulation with PID factors.*
- void [PosRegSetAngle](#) (byte output, long angle)  
*Change the current value for set angle.*
- void [PosRegAddAngle](#) (byte output, long angle\_add)  
*Add to the current value for set angle.*
- void [PosRegSetMax](#) (byte output, byte max\_speed, byte max\_acceleration)  
*Set maximum limits.*

### Variables

- unsigned long `__fopen_default_size` = 1024

### 8.3.1 Detailed Description

Constants, macros, and API functions for NXC. [NXCDefs.h](#) contains declarations for the NXC NXT API resources

License:

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of this code is John Hansen. Portions created by John Hansen are Copyright (C) 2009-2011 John Hansen. All Rights Reserved.

-----  
**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2011-10-16

**Version:**

104

**8.3.2 Define Documentation****8.3.2.1 #define \_SENSOR\_CFG(\_type, \_mode) (((\_type)<<8)+(\_mode))**

Macro for defining [SetSensor](#) combined type and mode constants

**8.3.2.2 #define Acos(\_X) asm { acos \_\_FLTRETVAL\_\_, \_X }**

Compute arc cosine. Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [acos\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc cosine of `_X`.

**8.3.2.3 #define AcosD(\_X) asm { acosd \_\_FLTRETVAL\_\_, \_X }**

Compute arc cosine (degrees). Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [acosd\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc cosine of `_X`.

**8.3.2.4 #define Asin(\_X) asm { asin \_\_FLTRETVAL\_\_, \_X }**

Compute arc sine. Computes the arc sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `asin()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc sine of `_X`.

**8.3.2.5 #define AsinD(\_X) asm { asind \_\_FLTRETVAL\_\_, \_X }**

Compute arch sine (degrees). Computes the arc sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `asind()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc sine of `_X`.

**8.3.2.6 #define Atan(\_X) asm { atan \_\_FLTRETVAL\_\_, \_X }**

Compute arc tangent. Computes the arc tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc tangent of `_X`.

**8.3.2.7 #define Atan2(\_Y, \_X) asm { atan2 \_\_FLTRETVAL\_\_, \_Y, \_X }**

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of `_Y/_X`, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan2()` instead.

**Parameters:**

`_Y` Floating point value representing a y coordinate.

`_X` Floating point value representing an x coordinate.

**Returns:**

Arc tangent of `_Y/_X`, in the interval  $[-\pi, +\pi]$  radians.

**8.3.2.8 #define Atan2D(\_Y, \_X) asm { atan2d \_\_FLTRETVAL\_\_, \_Y, \_X }**

Compute arc tangent with two parameters (degrees). Computes the arc tangent of `_Y/_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atan2d()` instead.

**Parameters:**

`_Y` Floating point value.

`_X` Floating point value.

**Returns:**

Arc tangent of `_Y/_X`, in the interval `[-180,+180]` degrees.

**8.3.2.9 #define AtanD(\_X) asm { atand \_\_FLTRETVAL\_\_, \_X }**

Compute arc tangent (degrees). Computes the arc tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `atand()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Arc tangent of `_X`.

**8.3.2.10 #define Ceil(\_X) asm { ceil \_\_FLTRETVAL\_\_, \_X }**

Round up value. Computes the smallest integral value that is not less than `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `ceil()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

The smallest integral value not less than `_X`.

**8.3.2.11 #define Cos(\_X) asm { cos \_\_FLTRETVAL\_\_, \_X }**

Compute cosine. Computes the cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cos()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Cosine of `_X`.

**8.3.2.12 #define CosD(\_X) asm { cosd \_\_FLTRETVAL\_\_, \_X }**

Compute cosine (degrees). Computes the cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cosd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Cosine of `_X`.

**8.3.2.13 #define Cosh(\_X) asm { cosh \_\_FLTRETVAL\_\_, \_X }**

Compute hyperbolic cosine. Computes the hyperbolic cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `cosh()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic cosine of `_X`.

**8.3.2.14 #define CoshD(\_X) asm { coshd \_\_FLTRETVAL\_\_, \_X }**

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [coshd\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic cosine of `_X`.

**8.3.2.15 #define EQ 0x04**

The first value is equal to the second.

**8.3.2.16 #define Exp(\_X) asm { exp \_\_FLTRETVAL\_\_, \_X }**

Compute exponential function . Computes the base-e exponential function of `_X`, which is the e number raised to the power `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use [exp\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Exponential value of `_X`.

**8.3.2.17 #define Floor(\_X) asm { floor \_\_FLTRETVAL\_\_, \_X }**

Round down value. Computes the largest integral value that is not greater than `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `floor()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

The largest integral value not greater than `_X`.

**8.3.2.18 #define Frac(\_X) asm { frac \_\_FLTRETVAL\_\_, \_X }**

Compute fractional part. Computes the fractional part of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `frac()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Fractional part of `_X`.

### 8.3.2.19 #define getc(\_handle) fgetc(\_handle)

Get character from file. Returns the character currently pointed to by the internal file position indicator of the file specified by the handle. The internal file position indicator is then advanced by one character to point to the next character. The functions fgetc and getc are equivalent.

#### Parameters:

*\_handle* The handle of the file from which the character is read.

#### Returns:

The character read from the file.

#### Examples:

[ex\\_getc.nxc](#).

### 8.3.2.20 #define GT 0x01

The first value is greater than the second.

#### Examples:

[ex\\_nbcopt.nxc](#).

### 8.3.2.21 #define GTEQ 0x03

The first value is greater than or equal to the second.

### 8.3.2.22 #define Log(\_X) asm { log \_\_FLTRETVAL\_\_, \_X }

Compute natural logarithm. Computes the natural logarithm of *\_X*. The natural logarithm is the base-e logarithm, the inverse of the natural exponential function (exp). For base-10 logarithms, a specific function [Log10\(\)](#) exists. Only constants or variables allowed (no expressions).

#### Deprecated

Use [log\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Natural logarithm of `_X`.

**8.3.2.23 #define Log10(\_X) asm { log10 \_\_FLTRETVAL\_\_, \_X }**

Compute common logarithm. Computes the common logarithm of `_X`. The common logarithm is the base-10 logarithm. For base-e logarithms, a specific function [Log\(\)](#) exists. Only constants or variables allowed (no expressions).

**Deprecated**

Use [log10\(\)](#) instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Common logarithm of `_X`.

**8.3.2.24 #define LT 0x00**

The first value is less than the second.

**8.3.2.25 #define LTEQ 0x02**

The first value is less than or equal to the second.

**8.3.2.26 #define MulDiv32(\_A, \_B, \_C) asm { muldiv \_\_RETVAL\_\_, \_A, \_B, \_C }**

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value. Only constants or variables allowed (no expressions).

### Deprecated

Use `muldiv32()` instead.

#### Parameters:

*\_A* 32-bit long value.

*\_B* 32-bit long value.

*\_C* 32-bit long value.

#### Returns:

The result of multiplying *\_A* times *\_B* and dividing by *\_C*.

#### 8.3.2.27 #define NEQ 0x05

The first value is not equal to the second.

#### 8.3.2.28 #define Pow(\_Base, \_Exponent) asm { pow \_\_FLTRETVAL\_\_, \_Base, \_Exponent }

Raise to power. Computes *\_Base* raised to the power *\_Exponent*. Only constants or variables allowed (no expressions).

### Deprecated

Use `pow()` instead.

#### Parameters:

*\_Base* Floating point value.

*\_Exponent* Floating point value.

#### Returns:

The result of raising *\_Base* to the power *\_Exponent*.

#### 8.3.2.29 #define putc(\_ch, \_handle) fputc(\_ch, \_handle)

Write character to file. Writes a character to the file and advances the position indicator. The character is written at the current position of the file as indicated by the internal position indicator, which is then advanced one character. If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned.

**Parameters:**

- \_ch* The character to be written.
- \_handle* The handle of the file where the character is to be written.

**Returns:**

The character written to the file.

**Examples:**

[ex\\_putc.nxc](#).

### 8.3.2.30 #define RICSetValue(\_data, \_idx, \_newval) \_data[(\_idx)] = (\_newval)&0xFF; \_data[(\_idx)+1] = (\_newval)>>8

Set the value of an element in an RIC data array.

**Parameters:**

- \_data* The RIC data array
- \_idx* The array index to update
- \_newval* The new value to write into the RIC data array

### 8.3.2.31 #define S1 0

Input port 1

**Examples:**

[ex\\_ACCLNxCalibrateX.nxc](#), [ex\\_ACCLNxCalibrateXEnd.nxc](#), [ex\\_ACCLNxCalibrateY.nxc](#), [ex\\_ACCLNxCalibrateYEnd.nxc](#), [ex\\_ACCLNxCalibrateZ.nxc](#), [ex\\_ACCLNxCalibrateZEnd.nxc](#), [ex\\_ACCLNxResetCalibration.nxc](#), [ex\\_ACCLNxSensitivity.nxc](#), [ex\\_ACCLNxXOffset.nxc](#), [ex\\_ACCLNxXRange.nxc](#), [ex\\_ACCLNxYOffset.nxc](#), [ex\\_ACCLNxYRange.nxc](#), [ex\\_ACCLNxZOffset.nxc](#), [ex\\_ACCLNxZRange.nxc](#), [ex\\_ClearSensor.nxc](#), [ex\\_ColorADRaw.nxc](#), [ex\\_ColorBoolean.nxc](#), [ex\\_ColorCalibration.nxc](#), [ex\\_ColorCalibrationState.nxc](#), [ex\\_ColorCalLimits.nxc](#), [ex\\_ColorSensorRaw.nxc](#), [ex\\_ColorSensorValue.nxc](#), [ex\\_ConfigureTemperatureSensor.nxc](#), [ex\\_CustomSensorActiveStatus.nxc](#), [ex\\_CustomSensorPercentFullScale.nxc](#), [ex\\_CustomSensorZeroOffset.nxc](#), [ex\\_diacl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_DISTNxDistance.nxc](#), [ex\\_DISTNxGP2D12.nxc](#), [ex\\_DISTNxGP2D120.nxc](#), [ex\\_DISTNxGP2YA02.nxc](#),

ex\_DISTNxGP2YA21.nxc,           ex\_DISTNxMaxDistance.nxc,           ex\_  
 DISTNxMinDistance.nxc,           ex\_DISTNxModuleType.nxc,           ex\_  
 DISTNxNumPoints.nxc,   ex\_DISTNxVoltage.nxc,   ex\_GetInput.nxc,   ex\_  
 GetLSInputBuffer.nxc,   ex\_GetLSOutputBuffer.nxc,   ex\_HTIRTrain.nxc,  
 ex\_HTPFComboDirect.nxc,           ex\_HTPFComboPWM.nxc,           ex\_  
 HTPFRawOutput.nxc,   ex\_HTPFRepeat.nxc,   ex\_HTPFSingleOutputCST.nxc,  
 ex\_HTPFSingleOutputPWM.nxc,   ex\_HTPFSinglePin.nxc,   ex\_HTPFTrain.nxc,  
 ex\_HTRCXAddToDatalog.nxc,           ex\_HTRCXClearSensor.nxc,           ex\_  
 HTRCXSetIRLinkPort.nxc,           ex\_HTRCXSetSensorMode.nxc,           ex\_  
 HTRCXSetSensorType.nxc,   ex\_I2CBytesReady.nxc,   ex\_I2CCheckStatus.nxc,  
 ex\_i2cdeviceid.nxc,   ex\_i2cdeviceinfo.nxc,   ex\_I2CRead.nxc,   ex\_  
 I2CSendCommand.nxc,   ex\_I2CStatus.nxc,   ex\_i2cvendorid.nxc,   ex\_  
 i2cversion.nxc,   ex\_I2CWrite.nxc,   ex\_LowspeedBytesReady.nxc,   ex\_  
 LowspeedCheckStatus.nxc,   ex\_LowspeedRead.nxc,   ex\_LowspeedStatus.nxc,  
 ex\_LowspeedWrite.nxc,   ex\_LSChannelState.nxc,   ex\_LSErrorType.nxc,  
 ex\_LSInputBufferBytesToRx.nxc,           ex\_LSInputBufferInPtr.nxc,           ex\_  
 LSInputBufferOutPtr.nxc,   ex\_LSMODE.nxc,   ex\_LSOutputBufferBytesToRx.nxc,  
 ex\_LSOutputBufferInPtr.nxc,           ex\_LSOutputBufferOutPtr.nxc,           ex\_  
 MSADPAOff.nxc,   ex\_MSADPAOn.nxc,   ex\_MSDeenergize.nxc,   ex\_  
 MSEnergize.nxc,   ex\_MSIRTrain.nxc,   ex\_MSPFComboDirect.nxc,   ex\_  
 MSPFComboPWM.nxc,   ex\_MSPFRawOutput.nxc,   ex\_MSPFRepeat.nxc,  
 ex\_MSPFSingleOutputCST.nxc,           ex\_MSPFSingleOutputPWM.nxc,           ex\_  
 MSPFSinglePin.nxc,   ex\_MSPFTrain.nxc,   ex\_MSRCXAddToDatalog.nxc,  
 ex\_MSRCXClearSensor.nxc,           ex\_MSRCXSetIRLinkPort.nxc,           ex\_  
 MSRCXSetSensorMode.nxc,           ex\_MSRCXSetSensorType.nxc,           ex\_  
 MSRCXSumVar.nxc,   ex\_MSReadValue.nxc,   ex\_NRLink2400.nxc,   ex\_  
 NRLink4800.nxc,   ex\_NRLinkFlush.nxc,   ex\_NRLinkIRLong.nxc,   ex\_  
 NRLinkIRShort.nxc,   ex\_NRLinkSetPF.nxc,   ex\_NRLinkSetRCX.nxc,  
 ex\_NRLinkSetTrain.nxc,   ex\_NRLinkStatus.nxc,   ex\_NRLinkTxRaw.nxc,  
 ex\_NXTHID.nxc,   ex\_NXTLineLeader.nxc,   ex\_NXTPowerMeter.nxc,  
 ex\_NXTServo.nxc,   ex\_NXTSumoEyes.nxc,   ex\_PFMate.nxc,   ex\_  
 proto.nxc,   ex\_PSPNxAnalog.nxc,   ex\_PSPNxDigital.nxc,   ex\_  
 readi2cregister.nxc,   ex\_ReadNRLinkBytes.nxc,   ex\_ReadSensorColorEx.nxc,  
 ex\_ReadSensorColorRaw.nxc,           ex\_ReadSensorEMeter.nxc,           ex\_  
 ReadSensorHTAccel.nxc,           ex\_ReadSensorHTColor.nxc,           ex\_  
 ReadSensorHTColor2Active.nxc,   ex\_ReadSensorHTIRReceiver.nxc,   ex\_  
 ReadSensorHTIRReceiverEx.nxc,   ex\_ReadSensorHTIRSeeker2AC.nxc,   ex\_  
 ReadSensorHTIRSeeker2DC.nxc,   ex\_ReadSensorHTNormalizedColor.nxc,   ex\_  
 ReadSensorHTNormalizedColor2Active.nxc,   ex\_ReadSensorHTRawColor.nxc,  
 ex\_ReadSensorHTRawColor2.nxc,   ex\_ReadSensorHTTouchMultiplexer.nxc,  
 ex\_ReadSensorMSAccel.nxc,           ex\_ReadSensorMSPlayStation.nxc,  
 ex\_ReadSensorMSRTClock.nxc,           ex\_ReadSensorMSTilt.nxc,           ex\_  
 ReadSensorUSEx.nxc,   ex\_RemoteLowspeedRead.nxc,           ex\_  
 RemoteLowspeedWrite.nxc,   ex\_RemoteResetScaledValue.nxc,   ex\_  
 RemoteSetInputMode.nxc,   ex\_RFIDInit.nxc,   ex\_RFIDMode.nxc,   ex\_  
 RFIDRead.nxc,   ex\_RFIDReadContinuous.nxc,   ex\_RFIDReadSingle.nxc,

ex\_RFIDStatus.nxc, ex\_RFIDStop.nxc, ex\_RunNRLinkMacro.nxc, ex\_Sensor.nxc, ex\_SensorBoolean.nxc, ex\_SensorDigiPinsDirection.nxc, ex\_SensorDigiPinsOutputLevel.nxc, ex\_SensorDigiPinsStatus.nxc, ex\_SensorHTColorNum.nxc, ex\_SensorHTCompass.nxc, ex\_SensorHTEOPD.nxc, ex\_SensorHTGyro.nxc, ex\_SensorHTIRSeeker2ACDir.nxc, ex\_SensorHTIRSeeker2Addr.nxc, ex\_SensorHTIRSeeker2DCDir.nxc, ex\_SensorHTIRSeekerDir.nxc, ex\_SensorHTMagnet.nxc, ex\_SensorInvalid.nxc, ex\_SensorMode.nxc, ex\_SensorMSCompass.nxc, ex\_SensorMSDROD.nxc, ex\_SensorMSPressure.nxc, ex\_SensorMSPressureRaw.nxc, ex\_SensorNormalized.nxc, ex\_SensorRaw.nxc, ex\_SensorScaled.nxc, ex\_SensorTemperature.nxc, ex\_SensorType.nxc, ex\_SensorValue.nxc, ex\_SensorValueBool.nxc, ex\_SensorValueRaw.nxc, ex\_SetACCLNxSensitivity.nxc, ex\_SetCustomSensorActiveStatus.nxc, ex\_SetCustomSensorPercentFullScale.nxc, ex\_SetCustomSensorZeroOffset.nxc, ex\_sethtcolor2mode.nxc, ex\_sethtirseeker2mode.nxc, ex\_SetInput.nxc, ex\_SetSensor.nxc, ex\_setsensorboolean.nxc, ex\_setsensorcolorblue.nxc, ex\_setsensorcolorfull.nxc, ex\_setsensorcolorgreen.nxc, ex\_setsensorcolornone.nxc, ex\_setsensorcolorred.nxc, ex\_SetSensorDigiPinsDirection.nxc, ex\_SetSensorDigiPinsOutputLevel.nxc, ex\_SetSensorDigiPinsStatus.nxc, ex\_SetSensorEMeter.nxc, ex\_setsensorhteopd.nxc, ex\_SetSensorHTGyro.nxc, ex\_SetSensorHTMagnet.nxc, ex\_SetSensorLight.nxc, ex\_SetSensorLowspeed.nxc, ex\_SetSensorMode.nxc, ex\_setsensormsdrod.nxc, ex\_setsensormspressure.nxc, ex\_SetSensorSound.nxc, ex\_SetSensorTemperature.nxc, ex\_SetSensorTouch.nxc, ex\_SetSensorType.nxc, ex\_SetSensorUltrasonic.nxc, ex\_superpro.nxc, ex\_SysColorSensorRead.nxc, ex\_syscommmlscheckstatus.nxc, ex\_syscommmlsread.nxc, ex\_syscommmlswrite.nxc, ex\_syscommmlswriteex.nxc, ex\_SysComputeCalibValue.nxc, ex\_sysinputpinfunction.nxc, ex\_writei2cregister.nxc, ex\_writenrlinkbytes.nxc, and ex\_xg1300.nxc.

### 8.3.2.32 #define s16 int

Signed 16 bit type

### 8.3.2.33 #define S2 1

Input port 2

### 8.3.2.34 #define S3 2

Input port 3

#### Examples:

`ex_ReadSensorHTBarometric.nxc.`

**8.3.2.35 #define s32 long**

Signed 32 bit type

**8.3.2.36 #define S4 3**

Input port 4

**Examples:**

[ex\\_I2CBytes.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ResetSensorHTAngle.nxc](#),  
and [ex\\_SensorUS.nxc](#).

**8.3.2.37 #define s8 char**

Signed 8 bit type

**8.3.2.38 #define SEEK\_CUR 1**

Seek from the current file position

**Examples:**

[ex\\_fseek.nxc](#).

**8.3.2.39 #define SEEK\_END 2**

Seek from the end of the file

**8.3.2.40 #define SEEK\_SET 0**

Seek from the beginning of the file

**Examples:**

[ex\\_sysfileseek.nxc](#).

**8.3.2.41 #define SENSOR\_1 Sensor(S1)**

Read the value of the analog sensor on port S1

**8.3.2.42 #define SENSOR\_2 Sensor(S2)**

Read the value of the analog sensor on port S2

**8.3.2.43 #define SENSOR\_3 Sensor(S3)**

Read the value of the analog sensor on port S3

**8.3.2.44 #define SENSOR\_4 Sensor(S4)**

Read the value of the analog sensor on port S4

**8.3.2.45 #define SENSOR\_CELSIUS \_SENSOR\_CFG(SENSOR\_TYPE\_-  
TEMPERATURE, SENSOR\_MODE\_CELSIUS)**

RCX temperature sensor in celcius mode

**8.3.2.46 #define SENSOR\_COLORBLUE \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORBLUE, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (blue) in percent mode

**8.3.2.47 #define SENSOR\_COLORFULL \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORFULL, SENSOR\_MODE\_RAW)**

NXT 2.0 color sensor (full) in raw mode

**8.3.2.48 #define SENSOR\_COLORGREEN \_SENSOR\_CFG(SENSOR\_-  
TYPE\_COLORGREEN, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

**8.3.2.49 #define SENSOR\_COLORNONE \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORNONE, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

**8.3.2.50** `#define SENSOR_COLORRED _SENSOR_CFG(SENSOR_TYPE_-  
COLORRED, SENSOR_MODE_PERCENT)`

NXT 2.0 color sensor (red) in percent mode

**8.3.2.51** `#define SENSOR_EDGE _SENSOR_CFG(SENSOR_TYPE_TOUCH,  
SENSOR_MODE_EDGE)`

Touch sensor in edge mode

**8.3.2.52** `#define SENSOR_FAHRENHEIT _SENSOR_CFG(SENSOR_TYPE_-  
TEMPERATURE, SENSOR_MODE_FAHRENHEIT)`

RCX temperature sensor in fahrenheit mode

**8.3.2.53** `#define SENSOR_LIGHT _SENSOR_CFG(SENSOR_TYPE_LIGHT,  
SENSOR_MODE_PERCENT)`

RCX Light sensor in percent mode

**8.3.2.54** `#define SENSOR_LOWSPEED _SENSOR_CFG(SENSOR_TYPE_-  
LOWSPEED, SENSOR_MODE_RAW)`

NXT I2C sensor without 9V power in raw mode

**8.3.2.55** `#define SENSOR_LOWSPEED_9V _SENSOR_CFG(SENSOR_-  
TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)`

NXT I2C sensor with 9V power in raw mode

**8.3.2.56** `#define SENSOR_MODE_BOOL IN_MODE_BOOLEAN`

Boolean value (0 or 1)

**Examples:**

[ex\\_HTRCXSetSensorMode.nxc](#), and [ex\\_MSRCXSetSensorMode.nxc](#).

**8.3.2.57** `#define SENSOR_MODE_CELSIUS IN_MODE_CELSIUS`

RCX temperature sensor value in degrees celcius

**8.3.2.58 #define SENSOR\_MODE\_EDGE IN\_MODE\_TRANSITIONCNT**

Counts the number of boolean transitions

**8.3.2.59 #define SENSOR\_MODE\_FAHRENHEIT IN\_MODE\_FAHRENHEIT**

RCX temperature sensor value in degrees fahrenheit

**8.3.2.60 #define SENSOR\_MODE\_PERCENT IN\_MODE\_PCTFULLSCALE**

Scaled value from 0 to 100

**8.3.2.61 #define SENSOR\_MODE\_PULSE IN\_MODE\_PERIODCOUNTER**

Counts the number of boolean periods

**8.3.2.62 #define SENSOR\_MODE\_RAW IN\_MODE\_RAW**

Raw value from 0 to 1023

**Examples:**

[ex\\_RemoteSetInputMode.nxc](#), and [ex\\_SetSensorMode.nxc](#).

**8.3.2.63 #define SENSOR\_MODE\_ROTATION IN\_MODE\_ANGLESTEP**

RCX rotation sensor (16 ticks per revolution)

**8.3.2.64 #define SENSOR\_NXTLIGHT \_SENSOR\_CFG(SENSOR\_TYPE\_-  
LIGHT\_ACTIVE, SENSOR\_MODE\_PERCENT)**

NXT light sensor in active mode

**8.3.2.65 #define SENSOR\_PULSE \_SENSOR\_CFG(SENSOR\_TYPE\_TOUCH,  
SENSOR\_MODE\_PULSE)**

Touch sensor in pulse mode

**8.3.2.66** `#define SENSOR_ROTATION _SENSOR_CFG(SENSOR_TYPE_-  
ROTATION, SENSOR_MODE_ROTATION)`

RCX rotation sensor in rotation mode

**8.3.2.67** `#define SENSOR_SOUND _SENSOR_CFG(SENSOR_TYPE_-  
SOUND_DB, SENSOR_MODE_PERCENT)`

NXT sound sensor (dB) in percent mode

**8.3.2.68** `#define SENSOR_TOUCH _SENSOR_CFG(SENSOR_TYPE_-  
TOUCH, SENSOR_MODE_BOOL)`

Touch sensor in boolean mode

**Examples:**

[ex\\_SetSensor.nxc](#).

**8.3.2.69** `#define SENSOR_TYPE_COLORBLUE IN_TYPE_COLORBLUE`

NXT 2.0 color sensor with blue light

**8.3.2.70** `#define SENSOR_TYPE_COLORFULL IN_TYPE_COLORFULL`

NXT 2.0 color sensor in full color mode

**8.3.2.71** `#define SENSOR_TYPE_COLORGREEN IN_TYPE_-  
COLORGREEN`

NXT 2.0 color sensor with green light

**8.3.2.72** `#define SENSOR_TYPE_COLORNONE IN_TYPE_COLORNONE`

NXT 2.0 color sensor with no light

**8.3.2.73** `#define SENSOR_TYPE_COLORRED IN_TYPE_COLORRED`

NXT 2.0 color sensor with red light

**8.3.2.74 #define SENSOR\_TYPE\_CUSTOM IN\_TYPE\_CUSTOM**

NXT custom sensor

**8.3.2.75 #define SENSOR\_TYPE\_HIGHSPEED IN\_TYPE\_HISPEED**

NXT Hi-speed port (only S4)

**8.3.2.76 #define SENSOR\_TYPE\_LIGHT IN\_TYPE\_REFLECTION**

RCX light sensor

**8.3.2.77 #define SENSOR\_TYPE\_LIGHT\_ACTIVE IN\_TYPE\_LIGHT\_ACTIVE**

NXT light sensor with light

**8.3.2.78 #define SENSOR\_TYPE\_LIGHT\_INACTIVE IN\_TYPE\_LIGHT\_INACTIVE**

NXT light sensor without light

**8.3.2.79 #define SENSOR\_TYPE\_LOWSPEED IN\_TYPE\_LOWSPEED**

NXT I2C digital sensor

**Examples:**[ex\\_RemoteSetInputMode.nxc.](#)**8.3.2.80 #define SENSOR\_TYPE\_LOWSPEED\_9V IN\_TYPE\_LOWSPEED\_9V**

NXT I2C digital sensor with 9V power

**8.3.2.81 #define SENSOR\_TYPE\_NONE IN\_TYPE\_NO\_SENSOR**

No sensor configured

**8.3.2.82 #define SENSOR\_TYPE\_ROTATION IN\_TYPE\_ANGLE**

RCX rotation sensor

**8.3.2.83 #define SENSOR\_TYPE\_SOUND\_DB IN\_TYPE\_SOUND\_DB**

NXT sound sensor with dB scaling

**Examples:**[ex\\_SetInput.nxc](#).**8.3.2.84 #define SENSOR\_TYPE\_SOUND\_DBA IN\_TYPE\_SOUND\_DBA**

NXT sound sensor with dBA scaling

**8.3.2.85 #define SENSOR\_TYPE\_TEMPERATURE IN\_TYPE\_-  
TEMPERATURE**

RCX temperature sensor

**8.3.2.86 #define SENSOR\_TYPE\_TOUCH IN\_TYPE\_SWITCH**

NXT or RCX touch sensor

**Examples:**[ex\\_HTRCXSetSensorType.nxc](#), [ex\\_MSRCXSetSensorType.nxc](#), and [ex\\_SetSensorType.nxc](#).**8.3.2.87 #define Sin(\_X) asm { sin \_\_FLTRETVAL\_\_, \_X }**

Compute sine. Computes the sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**Use `sin()` instead.**Parameters:**`_X` Floating point value.

**Returns:**

Sine of `_X`.

**8.3.2.88** `#define SinD(_X) asm { sind __FLTRETVAL__, _X }`

Compute sine (degrees). Computes the sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sind()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Sine of `_X`.

**8.3.2.89** `#define Sinh(_X) asm { sinh __FLTRETVAL__, _X }`

Compute hyperbolic sine. Computes the hyperbolic sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sinh()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic sine of `_X`.

**8.3.2.90** `#define SinhD(_X) asm { sinhd __FLTRETVAL__, _X }`

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sinhd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic sine of `_X`.

**8.3.2.91 #define Sqrt(\_X) asm { sqrt \_\_FLTRETVAL\_\_, \_X }**

Compute square root. Computes the square root of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `sqrt()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Square root of `_X`.

**8.3.2.92 #define Tan(\_X) asm { tan \_\_FLTRETVAL\_\_, \_X }**

Compute tangent. Computes the tangent of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `tan()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Tangent of `_X`.

**8.3.2.93** `#define TanD(_X) asm { tand __FLTRETVAL__, _X }`

Compute tangent (degrees). Computes the sine of `_X`. Only constants or variables allowed (no expressions).

#### Deprecated

Use `tand()` instead.

#### Parameters:

`_X` Floating point value.

#### Returns:

Tangent of `_X`.

**8.3.2.94** `#define Tanh(_X) asm { tanh __FLTRETVAL__, _X }`

Compute hyperbolic tangent. Computes the hyperbolic tangent of `_X`. Only constants or variables allowed (no expressions).

#### Deprecated

Use `tanh()` instead.

#### Parameters:

`_X` Floating point value.

#### Returns:

Hyperbolic tangent of `_X`.

**8.3.2.95** `#define TanhD(_X) asm { tanhd __FLTRETVAL__, _X }`

Compute hyperbolic tangent (degrees). Computes the hyperbolic tangent of `_X`. Only constants or variables allowed (no expressions).

#### Deprecated

Use `tanhd()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Hyperbolic tangent of `_X`.

**8.3.2.96 #define Trunc(\_X) asm { trunc \_\_RETVAL\_\_, \_X }**

Compute integral part. Computes the integral part of `_X`. Only constants or variables allowed (no expressions).

**Deprecated**

Use `trunc()` instead.

**Parameters:**

`_X` Floating point value.

**Returns:**

Integral part of `_X`.

**8.3.2.97 #define u16 unsigned int**

Unsigned 16 bit type

**8.3.2.98 #define u32 unsigned long**

Unsigned 32 bit type

**8.3.2.99 #define u8 unsigned char**

Unsigned 8 bit type

**8.3.3 Function Documentation****8.3.3.1 void abort () [inline]**

Abort current process. Aborts the process with an abnormal program termination. The function never returns to its caller.

**Examples:**

[ex\\_abort.nxc](#).

**8.3.3.2 byte AbortFlag (void) [inline]**

Read abort flag. Return the enhanced NBC/NXC firmware's abort flag.

**Returns:**

The current abort flag value. See [ButtonState constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_AbortFlag.nxc](#).

**8.3.3.3 variant abs (variant num) [inline]**

Absolute value. Return the absolute value of the value argument. Any scalar type can be passed into this function.

**Parameters:**

*num* The numeric value.

**Returns:**

The absolute value of num. The return type matches the input type.

**Examples:**

[ex\\_abs.nxc](#).

#### 8.3.3.4 `char ACCLNxCalibrateX (const byte port, const byte i2caddr)` `[inline]`

Calibrate ACCL-Nx X-axis. Calibrate the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Low-speed port before using this function.

##### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

##### Returns:

The function call result.

##### Examples:

[ex\\_ACCLNxCalibrateX.nxc](#).

#### 8.3.3.5 `char ACCLNxCalibrateXEnd (const byte port, const byte i2caddr)` `[inline]`

Stop calibrating ACCL-Nx X-axis. Stop calibrating the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Low-speed port before using this function.

##### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

##### Returns:

The function call result.

##### Examples:

[ex\\_ACCLNxCalibrateXEnd.nxc](#).

#### 8.3.3.6 `char ACCLNxCalibrateY (const byte port, const byte i2caddr)` `[inline]`

Calibrate ACCL-Nx Y-axis. Calibrate the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateY.nxc](#).

**8.3.3.7 char ACCLNxCalibrateYEnd (const byte *port*, const byte *i2caddr*)  
[inline]**

Stop calibrating ACCL-Nx Y-axis. Stop calibrating the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateYEnd.nxc](#).

**8.3.3.8 char ACCLNxCalibrateZ (const byte *port*, const byte *i2caddr*)  
[inline]**

Calibrate ACCL-Nx Z-axis. Calibrate the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateZ.nxc](#).

**8.3.3.9 char ACCLNxCalibrateZEnd (const byte *port*, const byte *i2caddr*)  
[inline]**

Stop calibrating ACCL-Nx Z-axis. Stop calibrating the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxCalibrateZEnd.nxc](#).

**8.3.3.10 char ACCLNxResetCalibration (const byte *port*, const byte *i2caddr*)  
[inline]**

Reset ACCL-Nx calibration. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ACCLNxResetCalibration.nxc](#).

### 8.3.3.11 `byte ACCLNxSensitivity (const byte port, const byte i2caddr) [inline]`

Read ACCL-Nx sensitivity value. Read the mindsensors ACCL-Nx sensitivity value. The port must be configured as a Low-speed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The sensitivity value.

#### Examples:

[ex\\_ACCLNxSensitivity.nxc](#).

### 8.3.3.12 `int ACCLNxXOffset (const byte port, const byte i2caddr) [inline]`

Read ACCL-Nx X offset value. Read the mindsensors ACCL-Nx sensor's X offset value. The port must be configured as a Low-speed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The X offset value.

#### Examples:

[ex\\_ACCLNxXOffset.nxc](#).

### 8.3.3.13 `int ACCLNxXRange (const byte port, const byte i2caddr) [inline]`

Read ACCL-Nx X range value. Read the mindsensors ACCL-Nx sensor's X range value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The X range value.

**Examples:**

[ex\\_ACCLNxXRange.nxc](#).

**8.3.3.14 int ACCLNxYOffset (const byte port, const byte i2caddr) [inline]**

Read ACCL-Nx Y offset value. Read the mindsensors ACCL-Nx sensor's Y offset value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Y offset value.

**Examples:**

[ex\\_ACCLNxYOffset.nxc](#).

**8.3.3.15 int ACCLNxYRange (const byte port, const byte i2caddr) [inline]**

Read ACCL-Nx Y range value. Read the mindsensors ACCL-Nx sensor's Y range value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Y range value.

**Examples:**

[ex\\_ACCLNxYRange.nxc](#).

**8.3.3.16 int ACCLNxZOffset (const byte *port*, const byte *i2caddr*) [inline]**

Read ACCL-Nx Z offset value. Read the mindsensors ACCL-Nx sensor's Z offset value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Z offset value.

**Examples:**

[ex\\_ACCLNxZOffset.nxc](#).

**8.3.3.17 int ACCLNxZRange (const byte *port*, const byte *i2caddr*) [inline]**

Read ACCL-Nx Z range value. Read the mindsensors ACCL-Nx sensor's Z range value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The Z range value.

**Examples:**

[ex\\_ACCLNxZRange.nxc](#).

**8.3.3.18 float acos (float x) [inline]**

Compute arc cosine. Computes the principal value of the arc cosine of  $x$ , expressed in radians. In trigonometrics, arc cosine is the inverse operation of cosine.

**Parameters:**

$x$  Floating point value in the interval  $[-1,+1]$ .

**Returns:**

Arc cosine of  $x$ , in the interval  $[0,\pi]$  radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acos.nxc](#).

**8.3.3.19 float acosd (float x) [inline]**

Compute arc cosine (degrees). Computes the principal value of the arc cosine of  $x$ , expressed in degrees. In trigonometrics, arc cosine is the inverse operation of cosine.

**Parameters:**

$x$  Floating point value in the interval  $[-1,+1]$ .

**Returns:**

Arc cosine of  $x$ , in the interval  $[0,180]$  degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acosd.nxc](#).

### 8.3.3.20 void Acquire (mutex *m*) [inline]

Acquire a mutex. Acquire the specified mutex variable. If another task already has acquired the mutex then the current task will be suspended until the mutex is released by the other task. This function is used to ensure that the current task has exclusive access to a shared resource, such as the display or a motor. After the current task has finished using the shared resource the program should call Release to allow other tasks to acquire the mutex.

#### Parameters:

*m* The mutex to acquire.

#### Examples:

[ex\\_Acquire.nxc](#), and [ex\\_Release.nxc](#).

### 8.3.3.21 unsigned long addressOf (variant *data*) [inline]

Get the absolute address of a variable. Get the absolute address of a variable and return it to the calling routine as an unsigned long value.

#### Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

#### Parameters:

*data* A variable whose address you wish to get.

#### Returns:

The absolute address of the variable.

#### Examples:

[ex\\_addressof.nxc](#).

### 8.3.3.22 unsigned long addressOfEx (variant *data*, bool *relative*) [inline]

Get the absolute or relative address of a variable. Get the absolute or relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*data* A variable whose address you wish to get.

*relative* A boolean flag indicating whether you want to get the relative or absolute address.

**Returns:**

The absolute or relative address of the variable.

**Examples:**

[ex\\_addressofex.nxc](#).

**8.3.3.23 void ArrayBuild (variant & aout[ ], variant src1, variant src2, ..., variant srcN) [inline]**

Build an array. Build a new array from the specified source(s). The sources can be of any type so long as the number of dimensions is equal to or one less than the number of dimensions in the output array and the type is compatible with the type of the output array. If a source is an array with the same number of dimensions as the output array then all of its elements are added to the output array.

**Parameters:**

*aout* The output array to build.

*src1* The first source to build into the output array.

*src2* The second source to build into the output array.

*srcN* The first source to build into the output array.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayBuild.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_SysCommHSWrite.nxc](#), [ex\\_SysDatalogWrite.nxc](#), and [ex\\_sysmemorymanager.nxc](#).

### 8.3.3.24 void ArrayIndex (variant & *out*, variant *asrc*[], unsigned int *idx*) [inline]

Extract item from an array. Extract one element from an array. The output type depends on the type of the source array.

#### Parameters:

- out* The output value.
- asrc* The input array from which to extract an item.
- idx* The index of the item to extract.

#### Warning:

You cannot use NXC expressions with this function

#### Examples:

[ex\\_nbcopt.nxc](#).

### 8.3.3.25 void ArrayInit (variant & *aout*[], variant *value*, unsigned int *count*) [inline]

Initialize an array. Initialize the array to contain count elements with each element equal to the value provided. To initialize a multi-dimensional array, the value should be an array of N-1 dimensions, where N is the number of dimensions in the array being initialized.

#### Parameters:

- aout* The output array to initialize.
- value* The value to initialize each element to.
- count* The number of elements to create in the output array.

#### Warning:

You cannot use NXC expressions with this function

#### Examples:

[ex\\_ArrayInit.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_nbcopt.nxc](#), [ex\\_sysdrawgraphic.nxc](#), and [ex\\_sysmemorymanager.nxc](#).

### 8.3.3.26 unsigned int ArrayLen (variant *data*[ ]) [inline]

Get array length. Return the length of the specified array. Any type of array of up to four dimensions can be passed into this function.

#### Parameters:

*data* The array whose length you need to read.

#### Returns:

The length of the specified array.

#### Warning:

You cannot use NXC expressions with this function

#### Examples:

[ex\\_ArrayLen.nxc](#), [ex\\_atan2.nxc](#), [ex\\_atan2d.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_systlistfiles.nxc](#), [ex\\_tan.nxc](#), and [ex\\_tand.nxc](#).

### 8.3.3.27 variant ArrayMax (const variant & *src*[ ], unsigned int *idx*, unsigned int *len*) [inline]

Calculate the maximum of the elements in a numeric array. This function calculates the maximum of all or a subset of the elements in the numeric *src* array.

#### Warning:

This function requires the enhanced NBC/NXC firmware.

#### Parameters:

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the *src* array (from *idx* to the end of the array).

#### Returns:

The maximum of *len* elements from the *src* numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMax.nxc](#), and [ex\\_ArraySort.nxc](#).

**8.3.3.28 variant ArrayMean (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the mean of the elements in a numeric array. This function calculates the mean of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from idx to the end of the array).

**Returns:**

The mean value of len elements from the src numeric array (starting from idx).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMean.nxc](#).

**8.3.3.29 variant ArrayMin (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the minimum of the elements in a numeric array. This function calculates the minimum of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The minimum of *len* elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayMin.nxc](#), and [ex\\_ArraySort.nxc](#).

**8.3.3.30 void ArrayOp (const byte *op*, variant & *dest*, const variant & *src* [], unsigned int *idx*, unsigned int *len*) [[inline](#)]**

Operate on numeric arrays. This function lets you perform various operations on numeric arrays.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*op* The array operation. See [Array operation constants](#).

*dest* The destination variant type (scalar or array, depending on the operation).

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the specified process. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayOp.nxc](#).

**8.3.3.31 void ArrayReplace (variant & *asrc*[ ], unsigned int *idx*, variant *value*)  
[inline]**

Replace items in an array. Replace one or more items in the specified source array. The items are replaced starting at the specified index. If the value provided has the same number of dimensions as the source array then multiple items in the source are replaced. If the value provided has one less dimension than the source array then one item will be replaced. Other differences between the source array and the new value dimensionality are not supported.

**Parameters:**

*asrc* The input array to be modified

*idx* The index of the item to replace.

*value* The new value or values to put into the source array.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_nbcopt.nxc](#).

**8.3.3.32 void ArraySort (variant & *dest*[ ], const variant & *src*[ ], unsigned int *idx*, unsigned int *len*) [inline]**

Sort the elements in a numeric array. This function sorts all or a subset of the elements in the numeric *src* array in ascending order and saves the results in the numeric *dest* array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*dest* The destination numeric array.

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the sorting process. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySort.nxc](#).

**8.3.3.33 variant ArrayStd (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the standard deviation of the elements in a numeric array. This function calculates the standard deviation of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The standard deviation of len elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArrayStd.nxc](#).

**8.3.3.34 void ArraySubset (variant & aout[ ], variant asrc[ ], unsigned int idx, unsigned int len) [inline]**

Copy an array subset. Copy a subset of the source array starting at the specified index and containing the specified number of elements into the destination array.

**Parameters:**

- aout* The output array containing the subset.
- asrc* The input array from which to copy a subset.
- idx* The start index of the array subset.
- len* The length of the array subset.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySubset.nxc](#).

**8.3.3.35 variant ArraySum (const variant & src[ ], unsigned int idx, unsigned int len) [inline]**

Calculate the sum of the elements in a numeric array. This function calculates the sum of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

- src* The source numeric array.
- idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The sum of *len* elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySum.nxc](#).

**8.3.3.36 variant ArraySumSqr (const variant & src[], unsigned int idx, unsigned int len) [inline]**

Calculate the sum of the squares of the elements in a numeric array. This function calculates the sum of the squares of all or a subset of the elements in the numeric src array.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Parameters:**

*src* The source numeric array.

*idx* The index of the start of the array subset to process. Pass [NA](#) to start with the first element.

*len* The number of elements to include in the calculation. Pass [NA](#) to include the rest of the elements in the src array (from *idx* to the end of the array).

**Returns:**

The sum of the squares of *len* elements from the src numeric array (starting from *idx*).

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_ArraySumSqr.nxc](#).

**8.3.3.37 float asin (float *x*) [inline]**

Compute arc sine. Computes the principal value of the arc sine of *x*, expressed in radians. In trigonometrics, arc sine is the inverse operation of sine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc sine of *x*, in the interval [-pi/2,+pi/2] radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_asin.nxc](#).

**8.3.3.38 float asind (float *x*) [inline]**

Compute arc sine (degrees). Computes the principal value of the arc sine of *x*, expressed in degrees. In trigonometrics, arc sine is the inverse operation of sine.

**Parameters:**

*x* Floating point value in the interval [-1,+1].

**Returns:**

Arc sine of *x*, in the interval [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_asind.nxc](#).

### 8.3.3.39 float atan(float x) [inline]

Compute arc tangent. Computes the principal value of the arc tangent of  $x$ , expressed in radians. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use [atan2\(\)](#) if you need to determine the quadrant.

**See also:**

[atan2\(\)](#)

**Parameters:**

$x$  Floating point value.

**Returns:**

Arc tangent of  $x$ , in the interval  $[-\pi/2, +\pi/2]$  radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan.nxc](#).

### 8.3.3.40 float atan2(float y, float x) [inline]

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of  $y/x$ , expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant.

**See also:**

[atan\(\)](#)

**Parameters:**

$y$  Floating point value representing a  $y$  coordinate.

$x$  Floating point value representing an  $x$  coordinate.

**Returns:**

Arc tangent of  $y/x$ , in the interval  $[-\pi, +\pi]$  radians.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2.nxc](#).

**8.3.3.41 float atan2d (float y, float x) [inline]**

Compute arc tangent with 2 parameters (degrees). Computes the principal value of the arc tangent of  $y/x$ , expressed in degrees. To compute the value, the function uses the sign of both arguments to determine the quadrant.

**Parameters:**

- y* Floating point value representing a y coordinate.
- x* Floating point value representing an x coordinate.

**Returns:**

Arc tangent of  $y/x$ , in the interval  $[-180, +180]$  degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2d.nxc](#).

**8.3.3.42 float atand (float x) [inline]**

Compute arc tangent (degrees). Computes the principal value of the arc tangent of  $x$ , expressed in degrees. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use `atan2d` if you need to determine the quadrant.

**Parameters:**

*x* Floating point value.

**Returns:**

Arc tangent of *x*, in the interval [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atand.nxc](#).

**8.3.3.43 float atof (const string & *str*) [inline]**

Convert string to float. Parses the string *str* interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. The rest of the string after the last valid character is ignored and has no effect on the behavior of this function.

A valid floating point number for *atof* is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in *str* does not form a valid floating-point number as just defined, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of a floating-point number.

**Returns:**

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

**Examples:**

[ex\\_atof.nxc](#).

**8.3.3.44 int atoi (const string & str) [inline]**

Convert string to integer. Parses the string *str* interpreting its content as an integral number, which is returned as an int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in *str* does not form a valid integral number, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of an integral number.

**Returns:**

On success, the function returns the converted integral number as an int value. If no valid conversion could be performed a zero value is returned.

**Examples:**

[ex\\_atoi.nxc](#).

**8.3.3.45 long atol (const string & str) [inline]**

Convert string to long integer. Parses the string *str* interpreting its content as an integral number, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in *str* does not form a valid integral number, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of an integral number.

**Returns:**

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

**Examples:**

[ex\\_atol.nxc](#).

**8.3.3.46 unsigned int BatteryLevel (void) [inline]**

Get battery Level. Return the battery level in millivolts.

**Returns:**

The battery level

**Examples:**

[util\\_battery\\_1.nxc](#), and [util\\_battery\\_2.nxc](#).

**8.3.3.47 byte BatteryState (void) [inline]**

Get battery state. Return battery state information (0..4).

**Returns:**

The battery state (0..4)

**Examples:**

[ex\\_BatteryState.nxc](#).

### 8.3.3.48 byte bcd2dec (byte *bcd*) [**inline**]

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

**Parameters:**

*bcd* The value you want to convert from bcd to decimal.

**Returns:**

The decimal equivalent of the binary coded decimal byte.

**Examples:**

[ex\\_bcd2dec.nxc](#).

### 8.3.3.49 byte BluetoothState (void) [**inline**]

Get bluetooth state. Return the bluetooth state.

**Returns:**

The bluetooth state. See [BluetoothState constants](#).

**Examples:**

[ex\\_BluetoothState.nxc](#).

### 8.3.3.50 char BluetoothStatus (byte *conn*) [**inline**]

Check bluetooth status. Check the status of the bluetooth subsystem for the specified connection slot.

**Parameters:**

*conn* The connection slot (0..3). Connections 0 through 3 are for bluetooth connections. See [Remote connection constants](#).

**Returns:**

The bluetooth status for the specified connection.

**Examples:**

[ex\\_BluetoothStatus.nxc](#), and [ex\\_syscommbtconnection.nxc](#).

**8.3.3.51 char BluetoothWrite (byte *conn*, byte *buffer*[]) [inline]**

Write to a bluetooth connection. This method tells the NXT firmware to write the data in the buffer to the device on the specified Bluetooth connection. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3). Connections 0 through 3 are for bluetooth connections. See [Remote connection constants](#).

*buffer* The data to be written (up to 128 bytes)

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_BluetoothWrite.nxc](#).

**8.3.3.52 void BranchComp (const byte *cmp*, constant void *lbl*, variant *v1*, variant *v2*) [inline]**

Branch if compare is true. Branch to the specified label if the two values compare with a true result.

**Parameters:**

*cmp* The constant comparison code. See the [Comparison Constants](#) for valid values.

*lbl* The name of the label where code should continue executing if the comparison is true.

*v1* The first value that you want to compare.

*v2* The second value that you want to compare.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_nbcopt.nxc](#).

**8.3.3.53 void BranchTest (const byte *cmp*, constant void *lbl*, variant *value*)  
[inline]**

Branch if test is true. Branch to the specified label if the variable compares to zero with a true result.

**Parameters:**

*cmp* The constant comparison code. See the [Comparison Constants](#) for valid values.

*lbl* The name of the label where code should continue executing if the test is true.

*value* The value that you want to compare against zero.

**Warning:**

You cannot use NXC expressions with this function

**Examples:**

[ex\\_nbcopt.nxc](#).

**8.3.3.54 int BrickDataBluecoreVersion (void) [inline]**

Get NXT bluecore version. This method returns the bluecore version of the NXT.

**Returns:**

The NXT's bluecore version number.

**Examples:**

[ex\\_BrickDataBluecoreVersion.nxc](#).

**8.3.3.55 byte BrickDataBtHardwareStatus (void) [inline]**

Get NXT bluetooth hardware status. This method returns the Bluetooth hardware status of the NXT.

**Returns:**

The NXT's bluetooth hardware status.

**Examples:**

[ex\\_BrickDataBtHardwareStatus.nxc](#).

**8.3.3.56 byte BrickDataBtStateStatus (void) [inline]**

Get NXT bluetooth state status. This method returns the Bluetooth state status of the NXT.

**Returns:**

The NXT's bluetooth state status.

**Examples:**

[ex\\_BrickDataBtStateStatus.nxc](#).

**8.3.3.57 string BrickDataName (void) [inline]**

Get NXT name. This method returns the name of the NXT.

**Returns:**

The NXT's bluetooth name.

**Examples:**

[ex\\_BrickDataName.nxc](#).

**8.3.3.58 byte BrickDataTimeoutValue (void) [inline]**

Get NXT bluetooth timeout value. This method returns the Bluetooth timeout value of the NXT.

**Returns:**

The NXT's bluetooth timeout value.

**Examples:**

[ex\\_BrickDataTimeoutValue.nxc](#).

**8.3.3.59 long BTConnectionClass (const byte *conn*) [inline]**

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The class of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionClass.nxc](#).

**8.3.3.60 byte BTConnectionHandleNum (const byte *conn*) [inline]**

Get bluetooth device handle number. This method returns the handle number of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The handle number of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionHandleNum.nxc](#).

**8.3.3.61 byte BTConnectionLinkQuality (const byte *conn*) [inline]**

Get bluetooth device link quality. This method returns the link quality of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The link quality of the specified connection slot (unimplemented).

**Warning:**

This function is not implemented at the firmware level.

**Examples:**

[ex\\_BTConnectionLinkQuality.nxc](#).

**8.3.3.62 string BTConnectionName (const byte *conn*) [inline]**

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The name of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionName.nxc](#).

**8.3.3.63 string BTConnectionPinCode (const byte *conn*) [inline]**

Get bluetooth device pin code. This method returns the pin code of the device at the specified index in the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The pin code for the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionPinCode.nxc](#).

**8.3.3.64 byte BTConnectionStreamStatus (const byte *conn*) [inline]**

Get bluetooth device stream status. This method returns the stream status of the device at the specified index within the Bluetooth connection table.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

The stream status of the bluetooth device at the specified connection slot.

**Examples:**

[ex\\_BTConnectionStreamStatus.nxc](#).

**8.3.3.65 int BTDataMode (void) [inline]**

Get Bluetooth data mode. This method returns the value of the Bluetooth data mode.

**Returns:**

The Bluetooth data mode. See [Data mode constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**8.3.3.66 long BTDeviceClass (const byte *devidx*) [inline]**

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The device class of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceClass.nxc](#).

**8.3.3.67 byte BTDeviceCount (void) [inline]**

Get bluetooth device count. This method returns the number of devices defined within the Bluetooth device table.

**Returns:**

The count of known bluetooth devices.

**Examples:**

[ex\\_BTDeviceCount.nxc](#).

**8.3.3.68 string BTDeviceName (const byte *devidx*) [inline]**

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The device name of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceName.nxc](#).

### 8.3.3.69 byte BTDeviceNameCount (void) [inline]

Get bluetooth device name count. This method returns the number of device names defined within the Bluetooth device table. This usually has the same value as BTDeviceCount but it can differ in some instances.

**Returns:**

The count of known bluetooth device names.

**Examples:**

[ex\\_BTDeviceNameCount.nxc](#).

### 8.3.3.70 byte BTDeviceStatus (const byte *devidx*) [inline]

Get bluetooth device status. This method returns the status of the device at the specified index within the Bluetooth device table.

**Parameters:**

*devidx* The device table index.

**Returns:**

The status of the specified bluetooth device.

**Examples:**

[ex\\_BTDeviceStatus.nxc](#).

### 8.3.3.71 byte BTInputBufferIntPtr (void) [inline]

Get bluetooth input buffer in-pointer. This method returns the value of the input pointer of the Bluetooth input buffer.

**Returns:**

The bluetooth input buffer's in-pointer value.

**Examples:**

[ex\\_BTInputBufferIntPtr.nxc](#).

**8.3.3.72** byte BTInputBufferOutPtr (void) [`inline`]

Get bluetooth input buffer out-pointer. This method returns the value of the output pointer of the Bluetooth input buffer.

**Returns:**

The bluetooth input buffer's out-pointer value.

**Examples:**

[ex\\_BTInputBufferOutPtr.nxc](#).

**8.3.3.73** byte BTOutputBufferInPtr (void) [`inline`]

Get bluetooth output buffer in-pointer. This method returns the value of the input pointer of the Bluetooth output buffer.

**Returns:**

The bluetooth output buffer's in-pointer value.

**Examples:**

[ex\\_BTOutputBufferInPtr.nxc](#).

**8.3.3.74** byte BTOutputBufferOutPtr (void) [`inline`]

Get bluetooth output buffer out-pointer. This method returns the value of the output pointer of the Bluetooth output buffer.

**Returns:**

The bluetooth output buffer's out-pointer value.

**Examples:**

[ex\\_BTOutputBufferOutPtr.nxc](#).

**8.3.3.75** byte ButtonCount (const byte *btn*, bool *resetCount*) [inline]

Get button press count. Return the number of times the specified button has been pressed since the last time the button press count was reset. Optionally clear the count after reading it.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*resetCount* Whether or not to reset the press counter.

**Returns:**

The button press count.

**Examples:**

[ex\\_ButtonCount.nxc](#).

**8.3.3.76** byte ButtonLongPressCount (const byte *btn*) [inline]

Get button long press count. Return the long press count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button long press count.

**Examples:**

[ex\\_ButtonLongPressCount.nxc](#).

**8.3.3.77** byte ButtonLongReleaseCount (const byte *btn*) [inline]

Get button long release count. Return the long release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button long release count.

**Examples:**

[ex\\_ButtonLongReleaseCount.nxc](#).

**8.3.3.78 byte ButtonPressCount (const byte *btn*) [inline]**

Get button press count. Return the press count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button press count.

**Examples:**

[ex\\_ButtonPressCount.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.3.3.79 bool ButtonPressed (const byte *btn*, bool *resetCount*) [inline]**

Check for button press. This function checks whether the specified button is pressed or not. You may optionally reset the press count.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*resetCount* Whether or not to reset the press counter.

**Returns:**

A boolean value indicating whether the button is pressed or not.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.3.3.80** byte ButtonReleaseCount (const byte *btn*) [**inline**]

Get button release count. Return the release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button release count.

**Examples:**

[ex\\_ButtonReleaseCount.nxc](#).

**8.3.3.81** byte ButtonShortReleaseCount (const byte *btn*) [**inline**]

Get button short release count. Return the short release count of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button short release count.

**Examples:**

[ex\\_ButtonShortReleaseCount.nxc](#).

**8.3.3.82** byte ButtonState (const byte *btn*) [**inline**]

Get button state. Return the state of the specified button. See [ButtonState constants](#).

**Parameters:**

*btn* The button to check. See [Button name constants](#).

**Returns:**

The button state.

**Examples:**

[ex\\_ButtonState.nxc](#).

**8.3.3.83 string ByteArrayToStr (byte *data*[]) [inline]**

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

**See also:**

[StrToByteArray](#), [ByteArrayToStrEx](#)

**Parameters:**

*data* A byte array.

**Returns:**

A string containing data and a null terminator byte.

**Examples:**

[ex\\_ByteArrayToStr.nxc](#), and [ex\\_string.nxc](#).

**8.3.3.84 void ByteArrayToStrEx (byte *data*[], string & *str*) [inline]**

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

**See also:**

[StrToByteArray](#), [ByteArrayToStr](#)

**Parameters:**

*data* A byte array.

*str* A string variable reference which, on output, will contain data and a null terminator byte.

**Examples:**

[ex\\_ByteArrayToStrEx.nxc](#), and [ex\\_string.nxc](#).

### 8.3.3.85 float ceil (float *x*) [`inline`]

Round up value. Computes the smallest integral value that is not less than *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

The smallest integral value not less than *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_ceil.nxc](#).

### 8.3.3.86 char CircleOut (int *x*, int *y*, byte *radius*, unsigned long *options* = DRAW\_OPT\_NORMAL) [`inline`]

Draw a circle. This function lets you draw a circle on the screen with its center at the specified *x* and *y* location, using the specified radius. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawCircle](#), [DrawCircleType](#)

**Parameters:**

*x* The *x* value for the center of the circle.

*y* The *y* value for the center of the circle.

*radius* The radius of the circle.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_CircleOut.nxc](#), and [ex\\_file\\_system.nxc](#).

**8.3.3.87 void ClearLine (byte *line*) [inline]**

Clear a line on the LCD screen. This function lets you clear a single line on the NXT LCD.

**Parameters:**

*line* The line you want to clear. See [Line number constants](#).

**Examples:**

[ex\\_clearline.nxc](#), and [ex\\_joystickmsg.nxc](#).

**8.3.3.88 void ClearScreen () [inline]**

Clear LCD screen. This function lets you clear the NXT LCD to a blank screen.

**Examples:**

[ex\\_ClearScreen.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispfout.nxc](#),  
[ex\\_dispgout.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_-](#)  
[ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_-](#)  
[SetAbortFlag.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_string.nxc](#), [ex\\_sysdrawpolygon.nxc](#),  
[ex\\_sysmemorymanager.nxc](#), and [ex\\_xg1300.nxc](#).

**8.3.3.89 void ClearSensor (const byte & *port*) [inline]**

Clear a sensor value. Clear the value of a sensor - only affects sensors that are configured to measure a cumulative quantity such as rotation or a pulse count.

**Parameters:**

*port* The port to clear. See [Input port constants](#).

**Examples:**

[ex\\_ClearSensor.nxc](#).

**8.3.3.90 unsigned int CloseFile (byte *handle*) [inline]**

Close a file. Close the file associated with the specified file handle. The loader result code is returned as the value of the function call. The handle parameter must be a constant or a variable.

**Parameters:**

*handle* The file handle.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_CloseFile.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

**8.3.3.91 void Coast (byte *outputs*) [inline]**

Coast motors. Turn off the specified outputs, making them coast to a stop.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

**Examples:**

[ex\\_coast.nxc](#).

**8.3.3.92 void CoastEx (byte *outputs*, const byte *reset*) [inline]**

Coast motors and reset counters. Turn off the specified outputs, making them coast to a stop.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_coastex.nxc](#).

**8.3.3.93 unsigned int ColorADRaw (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor AD raw value. This function lets you directly access a specific LEGO color sensor AD raw value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The AD raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorADRaw.nxc](#).

**8.3.3.94 bool ColorBoolean (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor boolean value. This function lets you directly access a specific LEGO color sensor boolean value. Both the port and the color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The boolean value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorBoolean.nxc](#).

**8.3.3.95 long ColorCalibration (byte *port*, byte *point*, byte *color*) [inline]**

Read a LEGO color sensor calibration point value. This function lets you directly access a specific LEGO color calibration point value. The port, point, and color index must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*point* The calibration point. See [Color calibration constants](#).

*color* The color index. See [Color sensor array indices](#).

**Returns:**

The calibration point value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalibration.nxc](#).

**8.3.3.96** byte `ColorCalibrationState` (byte *port*) [`inline`]

Read LEGO color sensor calibration state. This function lets you directly access the LEGO color calibration state. The port must be a constant.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The calibration state.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalibrationState.nxc](#).

**8.3.3.97** unsigned int `ColorCalLimits` (byte *port*, byte *point*) [`inline`]

Read a LEGO color sensor calibration limit value. This function lets you directly access a specific LEGO color calibration limit value. The port and the point must be constants.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*point* The calibration point. See [Color calibration constants](#).

**Returns:**

The calibration limit value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorCalLimits.nxc](#).

**8.3.3.98 unsigned int ColorSensorRaw (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor raw value. This function lets you directly access a specific LEGO color sensor raw value. Both the port and the color index must be constants.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- color* The color index. See [Color sensor array indices](#).

**Returns:**

The raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorSensorRaw.nxc](#).

**8.3.3.99 unsigned int ColorSensorValue (byte *port*, byte *color*) [inline]**

Read a LEGO color sensor scaled value. This function lets you directly access a specific LEGO color sensor scaled value. Both the port and the color index must be constants.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- color* The color index. See [Color sensor array indices](#).

**Returns:**

The scaled value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ColorSensorValue.nxc](#).

### 8.3.3.100 byte CommandFlags (void) [inline]

Get command flags. Return the command flags.

**Returns:**

Command flags. See [CommandFlags constants](#)

**Examples:**

[ex\\_CommandFlags.nxc](#).

### 8.3.3.101 char ConfigureTemperatureSensor (const byte &port, const byte &config) [inline]

Configure LEGO Temperature sensor options. Set various LEGO Temperature sensor options.

**Parameters:**

*port* The port to which the temperature sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*config* The temperature sensor configuration settings. See [LEGO temperature sensor constants](#) for configuration constants that can be ORed or added together.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible Result values.

**Examples:**

[ex\\_ConfigureTemperatureSensor.nxc](#).

### 8.3.3.102 string Copy (string str, unsigned int idx, unsigned int len) [inline]

Copy a portion of a string. Returns a substring of a string.

**Parameters:**

- str* A string
- idx* The starting index of the substring.
- len* The length of the substring.

**Returns:**

The specified substring.

**Examples:**

[ex\\_copy.nxc](#).

**8.3.3.103 float cos (float *x*) [inline]**

Compute cosine. Computes the cosine of an angle of *x* radians.

**Parameters:**

- x* Floating point value representing an angle expressed in radians.

**Returns:**

Cosine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#).

**8.3.3.104 float cosd (float *x*) [inline]**

Compute cosine (degrees). Computes the cosine of an angle of *x* degrees.

**Parameters:**

- x* Floating point value representing an angle expressed in degrees.

**Returns:**

Cosine of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sind\\_cosd.nxc](#).

**8.3.3.105 float cosh (float  $x$ ) [inline]**

Compute hyperbolic cosine. Computes the hyperbolic cosine of  $x$ , expressed in radians.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic cosine of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_cosh.nxc](#).

**8.3.3.106 float coshd (float  $x$ ) [inline]**

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of  $x$ , expressed in degrees.

**Parameters:**

$x$  Floating point value.

**Returns:**

Hyperbolic cosine of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.107 unsigned int CreateFile (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]**

Create a file. Create a new file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.

*fsize* The size of the file.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_CreateFile.nxc](#), and [ex\\_file\\_system.nxc](#).

**8.3.3.108 unsigned int CreateFileLinear (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]**

Create a linear file. Create a new linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.

*fsize* The size of the file.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_CreateFileLinear.nxc](#).

**8.3.3.109 unsigned int CreateFileNonLinear (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]**

Create a non-linear file. Create a new non-linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

**Parameters:**

*fname* The name of the file to create.

*fsize* The size of the file.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_CreateFileNonLinear.nxc](#).

**8.3.3.110 unsigned long CurrentTick () [inline]**

Read the current system tick. This function lets you current system tick count.

**Returns:**

The current system tick count.

**Examples:**

[ex\\_CurrentTick.nxc](#), [ex\\_dispgout.nxc](#), and [util\\_rpm.nxc](#).

**8.3.3.111 byte CustomSensorActiveStatus (byte port) [inline]**

Get the custom sensor active status. Return the custom sensor active status value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor active status.

**Examples:**

[ex\\_CustomSensorActiveStatus.nxc](#).

**8.3.3.112 byte CustomSensorPercentFullScale (byte port) [inline]**

Get the custom sensor percent full scale. Return the custom sensor percent full scale value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor percent full scale.

**Examples:**

[ex\\_CustomSensorPercentFullScale.nxc](#).

**8.3.3.113 unsigned int CustomSensorZeroOffset (byte *port*) [inline]**

Get the custom sensor zero offset. Return the custom sensor zero offset value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The custom sensor zero offset.

**Examples:**

[ex\\_CustomSensorZeroOffset.nxc](#).

**8.3.3.114 unsigned int DeleteFile (string *fname*) [inline]**

Delete a file. Delete the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to delete.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_delete\\_data\\_file.nxc](#), and [ex\\_DeleteFile.nxc](#).

**8.3.3.115 byte DisplayContrast () [inline]**

Read the display contrast setting. This function lets you read the current display contrast setting.

**Returns:**

The current display contrast (byte).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_contrast.nxc](#).

**8.3.3.116 unsigned long DisplayDisplay () [inline]**

Read the display memory address. This function lets you read the current display memory address.

**Returns:**

The current display memory address.

**Examples:**

[ex\\_DisplayDisplay.nxc](#), and [ex\\_dispmisc.nxc](#).

**8.3.3.117 unsigned long DisplayEraseMask () [inline]**

Read the display erase mask value. This function lets you read the current display erase mask value.

**Returns:**

The current display erase mask value.

**Examples:**

[ex\\_DisplayEraseMask.nxc](#), and [ex\\_dispmisc.nxc](#).

**8.3.3.118 byte DisplayFlags () [inline]**

Read the display flags. This function lets you read the current display flags. Valid flag values are listed in the [Display flags](#) group.

**Returns:**

The current display flags.

**Examples:**

[ex\\_DisplayFlags.nxc](#), and [ex\\_dispmisc.nxc](#).

**8.3.3.119 unsigned long DisplayFont () [inline]**

Read the display font memory address. This function lets you read the current display font memory address.

**Returns:**

The current display font memory address.

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_displayfont.nxc](#), and [ex\\_setdisplayfont.nxc](#).

**8.3.3.120 byte DisplayTextLinesCenterFlags () [inline]**

Read the display text lines center flags. This function lets you read the current display text lines center flags.

**Returns:**

The current display text lines center flags.

**Examples:**

[ex\\_DisplayTextLinesCenterFlags.nxc](#), and [ex\\_dispmisc.nxc](#).

**8.3.3.121 unsigned long DisplayUpdateMask () [inline]**

Read the display update mask value. This function lets you read the current display update mask value.

**Returns:**

The current display update mask.

**Examples:**

[ex\\_DisplayUpdateMask.nxc](#), and [ex\\_dispmisc.nxc](#).

**8.3.3.122 int DISTNxDistance (const byte *port*, const byte *i2caddr*) [inline]**

Read DISTNx distance value. Read the mindsensors DISTNx sensor's distance value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

[ex\\_DISTNxDistance.nxc](#).

**8.3.3.123 char DISTNxGP2D12 (const byte *port*, const byte *i2caddr*) [inline]**

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2D12.nxc](#).

### 8.3.3.124 `char DISTNxGP2D120 (const byte port, const byte i2caddr)` `[inline]`

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The function call result.

#### Examples:

[ex\\_DISTNxGP2D120.nxc](#).

### 8.3.3.125 `char DISTNxGP2YA02 (const byte port, const byte i2caddr)` `[inline]`

Configure DISTNx as GP2YA02. Configure the mindsensors DISTNx sensor as GP2YA02. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The function call result.

#### Examples:

[ex\\_DISTNxGP2YA02.nxc](#).

### 8.3.3.126 `char DISTNxGP2YA21 (const byte port, const byte i2caddr)` [inline]

Configure DISTNx as GP2YA21. Configure the mindsensors DISTNx sensor as GP2YA21. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The function call result.

#### Examples:

[ex\\_DISTNxGP2YA21.nxc](#).

### 8.3.3.127 `int DISTNxMaxDistance (const byte port, const byte i2caddr)` [inline]

Read DISTNx maximum distance value. Read the mindsensors DISTNx sensor's maximum distance value. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The maximum distance value.

#### Examples:

[ex\\_DISTNxMaxDistance.nxc](#).

**8.3.3.128 int DISTNxMinDistance (const byte *port*, const byte *i2caddr*)  
[inline]**

Read DISTNx minimum distance value. Read the mindsensors DISTNx sensor's minimum distance value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

[ex\\_DISTNxMinDistance.nxc](#).

**8.3.3.129 byte DISTNxModuleType (const byte *port*, const byte *i2caddr*)  
[inline]**

Read DISTNx module type value. Read the mindsensors DISTNx sensor's module type value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The module type value.

**Examples:**

[ex\\_DISTNxModuleType.nxc](#).

**8.3.3.130 byte DISTNxNumPoints (const byte *port*, const byte *i2caddr*)  
[inline]**

Read DISTNx num points value. Read the mindsensors DISTNx sensor's num points value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The num points value.

**Examples:**

[ex\\_DISTNxNumPoints.nxc](#).

**8.3.3.131 int DISTNxVoltage (const byte port, const byte i2caddr) [inline]**

Read DISTNx voltage value. Read the mindsensors DISTNx sensor's voltage value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The voltage value.

**Examples:**

[ex\\_DISTNxVoltage.nxc](#).

**8.3.3.132 div\_t div (int numer, int denom) [inline]**

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type `div_t`, which has two members: `quot` and `rem`.

**Parameters:**

*numer* Numerator.

*denom* Denominator.

**Returns:**

The result is returned by value in a structure defined in `cstdlib`, which has two members. For `div_t`, these are, in either order: `int quot`; `int rem`.

**Examples:**

[ex\\_div.nxc](#).

**8.3.3.133 char EllipseOut (int *x*, int *y*, byte *radiusX*, byte *radiusY*, unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw an ellipse. This function lets you draw an ellipse on the screen with its center at the specified *x* and *y* location, using the specified radii. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawEllipse](#), [DrawEllipseType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*x* The *x* value for the center of the ellipse.

*y* The *y* value for the center of the ellipse.

*radiusX* The *x* axis radius.

*radiusY* The *y* axis radius.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_EllipseOut.nxc](#).

**8.3.3.134 void ExitTo (task *newTask*) [inline]**

Exit to another task. Immediately exit the current task and start executing the specified task.

**Parameters:**

*newTask* The task to start executing after exiting the current task.

**Examples:**

[alternating\\_tasks.nxc](#).

**8.3.3.135 float exp (float *x*) [inline]**

Compute exponential function. Computes the base-e exponential function of *x*, which is the e number raised to the power *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

Exponential value of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_exp.nxc](#).

**8.3.3.136 int fclose (byte *handle*) [inline]**

Close file. Close the file associated with the specified file handle. The loader result code is returned as the value of the function call.

**Parameters:**

*handle* The handle of the file to be closed.

**Returns:**

The loader result code.

**Examples:**

[ex\\_fclose.nxc](#).

**8.3.3.137 int feof (byte *handle*) [*inline*]**

Check End-of-file indicator. Checks whether the End-of-File indicator associated with the handle is set, returning a value different from zero if it is.

**Parameters:**

*handle* The handle of the file to check.

**Returns:**

Currently always returns 0.

**Examples:**

[ex\\_feof.nxc](#).

**8.3.3.138 int fflush (byte *handle*) [*inline*]**

Flush file. Writes any buffered data to the file. A zero value indicates success.

**Parameters:**

*handle* The handle of the file to be flushed.

**Returns:**

Currently always returns 0.

**Examples:**

[ex\\_fflush.nxc](#).

**8.3.3.139 char fgetc (byte *handle*) [inline]**

Get character from file. Returns the character currently pointed to by the internal file position indicator of the file specified by the handle. The internal file position indicator is then advanced by one character to point to the next character. The functions `fgetc` and `getc` are equivalent.

**Parameters:**

*handle* The handle of the file from which the character is read.

**Returns:**

The character read from the file.

**Examples:**

[ex\\_fgetc.nxc](#).

**8.3.3.140 string fgets (string & *str*, int *num*, byte *handle*) [inline]**

Get string from file. Reads characters from a file and stores them as a string into `str` until `(num-1)` characters have been read or either a newline or a the End-of-File is reached, whichever comes first. A newline character makes `fgets` stop reading, but it is considered a valid character and therefore it is included in the string copied to `str`. A null character is automatically appended in `str` after the characters read to signal the end of the string. Returns the string parameter.

**Parameters:**

*str* The string where the characters are stored.

*num* The maximum number of characters to be read.

*handle* The handle of the file from which the characters are read.

**Returns:**

The string read from the file.

**Examples:**

[ex\\_fgets.nxc](#).

### 8.3.3.141 unsigned int FindFirstFile (string & *fname*, byte & *handle*) [inline]

Start searching for files. This function lets you begin iterating through files stored on the NXT.

#### Parameters:

*fname* On input this contains the filename pattern you are searching for. On output this contains the name of the first file found that matches the pattern.

*handle* The search handle input to and output from the function call.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Warning:

This function requires the enhanced NBC/NXC firmware.

#### Examples:

[ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

### 8.3.3.142 unsigned int FindNextFile (string & *fname*, byte & *handle*) [inline]

Continue searching for files. This function lets you continue iterating through files stored on the NXT.

#### Parameters:

*fname* On output this contains the name of the next file found that matches the pattern used when the search began by calling [FindFirstFile](#).

*handle* The search handle input to and output from the function call.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Warning:

This function requires the enhanced NBC/NXC firmware.

#### Examples:

[ex\\_findfirstfile.nxc](#), and [ex\\_findnextfile.nxc](#).

**8.3.3.143 unsigned long FirstTick () [inline]**

Get the first tick. Return an unsigned 32-bit value, which is the system timing value (called a "tick") in milliseconds at the time that the program began running.

**Returns:**

The tick count at the start of program execution.

**Examples:**

[ex\\_FirstTick.nxc](#).

**8.3.3.144 string Flatten (variant *num*) [inline]**

Flatten a number to a string. Return a string containing the byte representation of the specified value.

**Parameters:**

*num* A number.

**Returns:**

A string containing the byte representation of the parameter *num*.

**Examples:**

[ex\\_Flatten.nxc](#), and [ex\\_string.nxc](#).

**8.3.3.145 string FlattenVar (variant *x*) [inline]**

Flatten any data to a string. Return a string containing the byte representation of the specified value.

**See also:**

[UnflattenVar](#)

**Parameters:**

*x* Any NXC datatype.

**Returns:**

A string containing the byte representation of the parameter *x*.

**Examples:**

[ex\\_FlattenVar.nxc](#), [ex\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

**8.3.3.146 void Float (byte *outputs*) [inline]**

Float motors. Make outputs float. Float is an alias for Coast.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

**Examples:**

[ex\\_float.nxc](#).

**8.3.3.147 float floor (float *x*) [inline]**

Round down value. Computes the largest integral value that is not greater than *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

The largest integral value not greater than *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_floor.nxc](#).

**8.3.3.148 void Follows (task *task1*, task *task2*, ..., task *taskN*) [inline]**

Declare tasks that this task follows. Schedule this task to follow the specified tasks so that it will execute once any of the specified tasks has completed executing. This statement should occur once within a task - preferably at the start of the task definition. If multiple tasks declare that they follow the same task then they will all execute simultaneously unless other dependencies prevent them from doing so. Any number of tasks may be listed in the Follows statement.

**Parameters:**

*task1* The first task that this task follows.

*task2* The second task that this task follows.

*taskN* The last task that this task follows.

**Examples:**

[ex\\_Follows.nxc](#).

**8.3.3.149 char FontNumOut (int *x*, int *y*, string *filename*, variant *value*, unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw a number with font. Draw a numeric value on the screen at the specified x and y location using a custom RIC font. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. See the [Font drawing option constants](#) group for options specific to the font drawing functions.

**See also:**

[FontTextOut](#), [SysDrawFont](#), [DrawFontType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*x* The x value for the start of the number output.

*y* The y value for the start of the number output.

*filename* The filename of the RIC font.

*value* The value to output to the LCD screen. Any numeric type is supported.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispfnout.nxc](#).

**8.3.3.150 char FontTextOut (int x, int y, string filename, string str, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw text with font. Draw a text value on the screen at the specified x and y location using a custom RIC font. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. See the [Font drawing option constants](#) for options specific to the font drawing functions.

**See also:**

[FontNumOut](#), [SysDrawFont](#), [DrawFontType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*x* The x value for the start of the text output.

*y* The y value for the start of the text output.

*filename* The filename of the RIC font.

*str* The text to output to the LCD screen.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispftout.nxc](#).

### 8.3.3.151 byte fopen (string *filename*, const string *mode*)

Open file. Opens the file whose name is specified in the parameter *filename* and associates it with a file handle that can be identified in future operations by the handle that is returned. The operations that are allowed on the stream and how these are performed are defined by the mode parameter.

#### Parameters:

*filename* The name of the file to be opened.

*mode* The file access mode. Valid values are "r" - opens an existing file for reading, "w" - creates a new file and opens it for writing, and "a" - opens an existing file for appending to the end of the file.

#### Returns:

The handle to the opened file.

#### Examples:

[ex\\_fopen.nxc](#).

### 8.3.3.152 void ForceOff (byte *num*) [inline]

Turn off NXT. Force the NXT to turn off if the specified value is greater than zero.

#### Parameters:

*num* If greater than zero the NXT will turn off.

#### Examples:

[ex\\_ForceOff.nxc](#).

### 8.3.3.153 string FormatNum (string *fmt*, variant *num*) [inline]

Format a number. Return the formatted string using the format and value. Use a standard numeric sprintf format specifier within the format string. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*fmt* The string format containing a sprintf numeric format specifier.

*num* A number.

**Returns:**

A string containing the formatted numeric value.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_acos.nxc](#), [ex\\_acosd.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_asin.nxc](#), [ex\\_asind.nxc](#), [ex\\_atan.nxc](#), [ex\\_atan2.nxc](#), [ex\\_atan2d.nxc](#), [ex\\_atand.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_FormatNum.nxc](#), [ex\\_GetBrickDataAddress.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_string.nxc](#), [ex\\_tan.nxc](#), [ex\\_tand.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**8.3.3.154 void fprintf (byte *handle*, string *format*, variant *value*) [inline]**

Write formatted data to file. Writes a sequence of data formatted as the format argument specifies to a file. After the format parameter, the function expects one value argument.

**Parameters:**

*handle* The handle of the file to write to.

*format* A string specifying the desired format.

*value* A value to be formatted for writing to the file.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_fprintf.nxc](#).

**8.3.3.155 char fputc (char *ch*, byte *handle*) [inline]**

Write character to file. Writes a character to the file and advances the position indicator. The character is written at the current position of the file as indicated by the internal position indicator, which is then advanced one character. If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned.

**Parameters:**

*ch* The character to be written.

*handle* The handle of the file where the character is to be written.

**Returns:**

The character written to the file.

**Examples:**

[ex\\_fputc.nxc](#).

**8.3.3.156 int fputs (string *str*, byte *handle*) [inline]**

Write string to file. Writes the string to the file specified by the handle. The null terminating character at the end of the string is not written to the file. If there are no errors, a non-negative value is returned. If an error occurs, EOF is returned.

**Parameters:**

*str* The string of characters to be written.

*handle* The handle of the file where the string is to be written.

**Returns:**

The number of characters written to the file.

**Examples:**

[ex\\_fputs.nxc](#).

**8.3.3.157 float frac (float *x*) [inline]**

Compute fractional part. Computes the fractional part of *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

Fractional part of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_frac.nxc](#).

**8.3.3.158 unsigned int FreeMemory (void) [inline]**

Get free flash memory. Get the number of bytes of flash memory that are available for use.

**Returns:**

The number of bytes of unused flash memory.

**Examples:**

[ex\\_FreeMemory.nxc](#).

**8.3.3.159 int fseek (byte handle, long offset, int origin) [inline]**

Reposition file position indicator. Sets the position indicator associated with the file to a new position defined by adding offset to a reference position specified by origin.

**Parameters:**

*handle* The handle of the file.

*offset* The number of bytes to offset from origin.

*origin* Position from where offset is added. It is specified by one of the following constants: SEEK\_SET - beginning of file, SEEK\_CUR - current position of the file pointer, or SEEK\_END - end of file. [fseek origin constants](#)

**Returns:**

A value of zero if successful or non-zero otherwise. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_fseek.nxc](#).

**8.3.3.160 unsigned long ftell (byte *handle*) [inline]**

Get current position in file. Returns the current value of the file position indicator of the specified handle.

**Parameters:**

*handle* The handle of the file.

**Returns:**

The current file position in the open file.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_ftell.nxc](#).

**8.3.3.161 void GetBrickDataAddress (byte & *data*[]) [inline]**

Get NXT address. This method reads the address of the NXT and stores it in the data buffer provided.

**Parameters:**

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBrickDataAddress.nxc](#).

**8.3.3.162 void GetBTConnectionAddress (const byte *conn*, byte & *data*[ ])  
[inline]**

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth connection table and stores it in the data buffer provided.

**Parameters:**

*conn* The connection slot (0..3).

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBTConnectionAddress.nxc.](#)

**8.3.3.163 void GetBTDeviceAddress (const byte *devidx*, byte & *data*[ ])  
[inline]**

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth device table and stores it in the data buffer provided.

**Parameters:**

*devidx* The device table index.

*data* The byte array reference that will contain the device address.

**Examples:**

[ex\\_GetBTDeviceAddress.nxc.](#)

**8.3.3.164 void GetBTInputBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ])  
[inline]**

Get bluetooth input buffer data. This method reads count bytes of data from the Bluetooth input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the bluetooth input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the bluetooth input buffer.

**Examples:**

[ex\\_GetBTInputBuffer.nxc](#).

**8.3.3.165 void GetBTOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[])**  
**[inline]**

Get bluetooth output buffer data. This method reads count bytes of data from the Bluetooth output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the bluetooth output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the bluetooth output buffer.

**Examples:**

[ex\\_GetBTOutputBuffer.nxc](#).

**8.3.3.166 void GetButtonModuleValue (unsigned int *offset*, variant & *value*)**  
**[inline]**

Get Button module IOMap value. Read a value from the Button module IOMap structure. You provide the offset into the Button module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Button module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.167 int getchar () [inline]**

Get character from stdin. Returns the next character from the standard input (stdin). It is equivalent to `getc` with `stdin` as its argument. On the NXT this means wait for a button press and return the value of the button pressed.

**Returns:**

The pressed button. See [Button name constants](#).

**Examples:**

[ex\\_getchar.nxc](#).

**8.3.3.168 void GetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Command module IOMap bytes. Read one or more bytes of data from Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the data should be read. See [Command module IOMAP offsets](#).

*count* The number of bytes to read from the specified Command module IOMap offset.

*data* A byte array that will contain the data read from the Command module IOMap.

**8.3.3.169 void GetCommandModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Command module IOMap value. Read a value from the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Command module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.170 void GetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Comm module IOMap bytes. Read one or more bytes of data from Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the data should be read. See [Comm module IOMAP offsets](#).

*count* The number of bytes to read from the specified Comm module IOMap offset.

*data* A byte array that will contain the data read from the Comm module IOMap.

**8.3.3.171 void GetCommModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Comm module IOMap value. Read a value from the Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Comm module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.172 void GetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get Display module IOMap bytes. Read one or more bytes of data from Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the data should be read. See [Display module IOMAP offsets](#).

*count* The number of bytes to read from the specified Display module IOMap offset.

*data* A byte array that will contain the data read from the Display module IOMap.

**8.3.3.173 void GetDisplayModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Display module IOMap value. Read a value from the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Display module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.174 void GetDisplayNormal (const byte *x*, const byte *line*, unsigned int *cnt*, byte & *data*[ ]) [inline]**

Read pixel data from the normal display buffer. Read "cnt" bytes from the normal display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

*x* The desired x position from which to read pixel data.

*line* The desired line from which to read pixel data.

*cnt* The number of bytes of pixel data to read.

*data* The array of bytes into which pixel data is read.

**Examples:**

[ex\\_GetDisplayNormal.nxc](#).

**8.3.3.175 void GetDisplayPopup (const byte *x*, const byte *line*, unsigned int *cnt*, byte & *data*[ ]) [inline]**

Read pixel data from the popup display buffer. Read "cnt" bytes from the popup display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

*x* The desired x position from which to read pixel data.

*line* The desired line from which to read pixel data.

*cnt* The number of bytes of pixel data to read.

*data* The array of bytes into which pixel data is read.

**Examples:**

[ex\\_GetDisplayPopup.nxc](#).

**8.3.3.176 void GetHSInputBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ]) [inline]**

Get hi-speed port input buffer data. This method reads count bytes of data from the hi-speed port input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the hi-speed port input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the hi-speed port input buffer.

**Examples:**

[ex\\_GetHSInputBuffer.nxc](#).

**8.3.3.177 void GetHSOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ]) [inline]**

Get hi-speed port output buffer data. This method reads count bytes of data from the hi-speed port output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the hi-speed port output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the hi-speed port output buffer.

**Examples:**

[ex\\_GetHSOutputBuffer.nxc](#).

**8.3.3.178 variant GetInput (const byte & *port*, const byte *field*) [inline]**

Get an input field value. Return the value of the specified field of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

*field* An input field constant. See [Input field constants](#).

**Returns:**

The input field value.

**Examples:**

[ex\\_GetInput.nxc](#).

**8.3.3.179 void GetInputModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Input module IOMap value. Read a value from the Input module IOMap structure. You provide the offset into the Input module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Input module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.180 void GetIOMapBytes (string *moduleName*, unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by name. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*moduleName* The module name of the IOMap. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be read

*count* The number of bytes to read from the specified IOMap offset.

*data* A byte array that will contain the data read from the IOMap

**8.3.3.181 void GetIOMapBytesByID (unsigned long *moduleId*, unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by ID. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

- moduleId* The module ID of the IOMap. See [NXT firmware module IDs](#).
- offset* The number of bytes offset from the start of the IOMap structure where the data should be read.
- count* The number of bytes to read from the specified IOMap offset.
- data* A byte array that will contain the data read from the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.182 void GetIOMapValue (string *moduleName*, unsigned int *offset*, variant & *value*) [inline]**

Get IOMap value by name. Read a value from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

**Parameters:**

- moduleName* The module name of the IOMap. See [NXT firmware module names](#).
- offset* The number of bytes offset from the start of the IOMap structure where the value should be read
- value* A variable that will contain the value read from the IOMap

**8.3.3.183 void GetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant & *value*) [inline]**

Get IOMap value by ID. Read a value from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

**Parameters:**

- moduleId* The module ID of the IOMap. See [NXT firmware module IDs](#).
- offset* The number of bytes offset from the start of the IOMap structure where the value should be read.

*value* A variable that will contain the value read from the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.184 char GetLastResponseInfo (bool *Clear*, byte & *Length*, byte & *Command*, byte & *Buffer*[]) [inline]**

Read last response information. Read the last direct or system command response packet received by the NXT. Optionally clear the response after retrieving the information.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Parameters:**

*Clear* A boolean value indicating whether to clear the response or not.

*Length* The response packet length.

*Command* The original command byte.

*Buffer* The response packet buffer.

**Returns:**

The response status code.

**Examples:**

[ex\\_GetLastResponseInfo.nxc](#).

**8.3.3.185 void GetLoaderModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get Loader module IOMap value. Read a value from the Loader module IOMap structure. You provide the offset into the Loader module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Loader module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.186 void GetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[ ]) [inline]**

Get Lowspeed module IOMap bytes. Read one or more bytes of data from Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

**Parameters:**

*offset* The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be read. See [Low speed module IOMAP offsets](#).

*count* The number of bytes to read from the specified Lowspeed module IOMap offset.

*data* A byte array that will contain the data read from the Lowspeed module IOMap.

**8.3.3.187 void GetLowSpeedModuleValue (unsigned int *offset*, variant & *value*) [inline]**

Get LowSpeed module IOMap value. Read a value from the LowSpeed module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Low speed module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.188** void GetLSInputBuffer (const byte *port*, const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get I2C input buffer data. This method reads count bytes of data from the I2C input buffer for the specified port and writes it to the buffer provided.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

*offset* A constant offset into the I2C input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the I2C input buffer.

**Examples:**

[ex\\_GetLSInputBuffer.nxc](#).

**8.3.3.189** void GetLSOutputBuffer (const byte *port*, const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get I2C output buffer data. This method reads cnt bytes of data from the I2C output buffer for the specified port and writes it to the buffer provided.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

*offset* A constant offset into the I2C output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the I2C output buffer.

**Examples:**

[ex\\_GetLSOutputBuffer.nxc](#).

**8.3.3.190** char GetMemoryInfo (bool *Compact*, unsigned int & *PoolSize*, unsigned int & *DataspaceSize*) [inline]

Read memory information. Read the current pool size and dataspace size. Optionally compact the dataspace before returning the information. Running programs have a maximum of 32k bytes of memory available. The amount of free RAM can be calculated by subtracting the value returned by this function from [POOL\\_MAX\\_SIZE](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*Compact* A boolean value indicating whether to compact the dataspace or not.

*PoolSize* The current pool size.

*DataspaceSize* The current dataspace size.

**Returns:**

The function call result. It will be [NO\\_ERR](#) if the compact operation is not performed. Otherwise it will be the result of the compact operation.

**Examples:**

[ex\\_getmemoryinfo.nxc](#).

**8.3.3.191 variant GetOutput (byte *output*, const byte *field*) [inline]**

Get output field value. Get the value of the specified field for the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

*field* Output port field to access, this should be a constant, see [Output field constants](#).

**Returns:**

The requested output field value.

**Examples:**

[ex\\_getoutput.nxc](#).

**8.3.3.192 void GetOutputModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Output module IOMap value. Read a value from the Output module IOMap structure. You provide the offset into the Output module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Output module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.193 void GetSoundModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Sound module IOMap value. Read a value from the Sound module IOMap structure. You provide the offset into the Sound module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Sound module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.194 void GetUIModuleValue (unsigned int *offset*, variant & *value*)  
[inline]**

Get Ui module IOMap value. Read a value from the Ui module IOMap structure. You provide the offset into the Ui module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

**Parameters:**

*offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Ui module IOMAP offsets](#).

*value* A variable that will contain the value read from the IOMap.

**8.3.3.195** void GetUSBInputBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ]) [inline]

Get usb input buffer data. This method reads count bytes of data from the usb input buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb input buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb input buffer.

**Examples:**

[ex\\_GetUSBInputBuffer.nxc.](#)

**8.3.3.196** void GetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ]) [inline]

Get usb output buffer data. This method reads count bytes of data from the usb output buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb output buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb output buffer.

**Examples:**

[ex\\_GetUSBOutputBuffer.nxc.](#)

**8.3.3.197** void GetUSBPollBuffer (const byte *offset*, byte *cnt*, byte & *data*[ ]) [inline]

Get usb poll buffer data. This method reads count bytes of data from the usb poll buffer and writes it to the buffer provided.

**Parameters:**

*offset* A constant offset into the usb poll buffer.

*cnt* The number of bytes to read.

*data* The byte array reference which will contain the data read from the usb poll buffer.

**Examples:**

[ex\\_GetUSBPollBuffer.nxc](#).

**8.3.3.198 void glAddToAngleX (int *glValue*) [inline]**

Add to the X axis angle. Add the specified value to the existing X axis angle.

**Parameters:**

*glValue* The value to add to the X axis angle.

**Examples:**

[glBoxDemo.nxc](#), and [glCircleDemo.nxc](#).

**8.3.3.199 void glAddToAngleY (int *glValue*) [inline]**

Add to the Y axis angle. Add the specified value to the existing Y axis angle.

**Parameters:**

*glValue* The value to add to the Y axis angle.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.200 void glAddToAngleZ (int *glValue*) [inline]**

Add to the Z axis angle. Add the specified value to the existing Z axis angle.

**Parameters:**

*glValue* The value to add to the Z axis angle.

**8.3.3.201 void glAddVertex (int glX, int glY, int glZ) [inline]**

Add a vertex to an object. Add a vertex to an object currently being defined. This function should only be used between [glBegin](#) and [glEnd](#) which are themselves nested within a [glBeginObject](#) and [glEndObject](#) pair.

**Parameters:**

*glX* The X axis coordinate.

*glY* The Y axis coordinate.

*glZ* The Z axis coordinate.

**8.3.3.202 void glBegin (int glBeginMode) [inline]**

Begin a new polygon for the current object. Start defining a polygon surface for the current graphics object using the specified begin mode.

**Parameters:**

*glBeginMode* The desired mode. See [Graphics library begin modes](#).

**8.3.3.203 int glBeginObject () [inline]**

Begin defining an object. Start the process of defining a graphics library object using low level functions such as [glBegin](#), [glAddVertex](#), and [glEnd](#).

**Returns:**

The object index of the new object being created.

**8.3.3.204 void glBeginRender () [inline]**

Begin a new render. Start the process of rendering the existing graphic objects.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.205 int glBox (int glMode, int glSizeX, int glSizeY, int glSizeZ) [inline]**

Create a 3D box. Define a 3D box using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with width, height, and depth specified via the glSizeX, glSizeY, and glSizeZ parameters.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSizeX* The X axis size (width).

*glSizeY* The Y axis size (height).

*glSizeZ* The Z axis size (depth).

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.206 void glCallObject (int glObjectId) [inline]**

Call a graphic object. Tell the graphics library that you want it to include the specified object in the render.

**Parameters:**

*glObjectId* The desired object id.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.207 int glCos32768 (int *glAngle*) [inline]**

Table-based cosine scaled by 32768. Return the cosine of the specified angle in degrees. The result is scaled by 32768.

**Parameters:**

*glAngle* The angle in degrees.

**Returns:**

The cosine value scaled by 32768.

**8.3.3.208 int glCube (int *glMode*, int *glSize*) [inline]**

Create a 3D cube. Define a 3D cube using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with equal width, height, and depth specified via the *glSize* parameter.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSize* The cube's width, height, and depth.

**Examples:**

[glBoxDemo.nxc](#).

**8.3.3.209 void glEnd () [inline]**

Finish a polygon for the current object. Stop defining a polygon surface for the current graphics object.

**8.3.3.210 void glEndObject () [inline]**

Stop defining an object. Finish the process of defining a graphics library object. Call this function after you have completed the object definition.

**8.3.3.211 void glFinishRender () [inline]**

Finish the current render. Rotate the vertex list, clear the screen, and draw the rendered objects to the LCD.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.212 void glInit () [inline]**

Initialize graphics library. Setup all the necessary data for the graphics library to function. Call this function before any other graphics library routine.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.213 void glObjectAction (int glObjectId, int glAction, int glValue) [inline]**

Perform an object action. Execute the specified action on the specified object.

**Parameters:**

*glObjectId* The object id.

*glAction* The action to perform on the object. See [Graphics library actions](#).

*glValue* The setting value.

**Examples:**

[glBoxDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.214** `int glPyramid (int glMode, int glSizeX, int glSizeY, int glSizeZ)`  
`[inline]`

Create a 3D pyramid. Define a 3D pyramid using the specified begin mode for all faces. The center of the pyramid is at the origin of the XYZ axis with width, height, and depth specified via the *glSizeX*, *glSizeY*, and *glSizeZ* parameters.

**Parameters:**

*glMode* The begin mode for each surface. See [Graphics library begin modes](#).

*glSizeX* The X axis size (width).

*glSizeY* The Y axis size (height).

*glSizeZ* The Z axis size (depth).

**8.3.3.215** `void glSet (int glType, int glValue)` `[inline]`

Set graphics library options. Adjust graphic library settings for circle size and cull mode.

**Parameters:**

*glType* The setting type. See [Graphics library settings](#).

*glValue* The setting value. For culling modes see [Graphics library cull mode](#).

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.216** `void glSetAngleX (int glValue)` `[inline]`

Set the X axis angle. Set the X axis angle to the specified value.

**Parameters:**

*glValue* The new X axis angle.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**8.3.3.217 void glSetAngleY (int *glValue*) [inline]**

Set the Y axis angle. Set the Y axis angle to the specified value.

**Parameters:**

*glValue* The new Y axis angle.

**8.3.3.218 void glSetAngleZ (int *glValue*) [inline]**

Set the Z axis angle. Set the Z axis angle to the specified value.

**Parameters:**

*glValue* The new Z axis angle.

**8.3.3.219 int glSin32768 (int *glAngle*) [inline]**

Table-based sine scaled by 32768. Return the sine of the specified angle in degrees. The result is scaled by 32768.

**Parameters:**

*glAngle* The angle in degrees.

**Returns:**

The sine value scaled by 32768.

**8.3.3.220 char GraphicArrayOut (int *x*, int *y*, byte *data*[], unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image from byte array. Draw a graphic image byte array on the screen at the specified *x* and *y* location. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

**Parameters:**

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgaout.nxc](#).

**8.3.3.221** `char GraphicArrayOutEx (int x, int y, byte data[], byte vars[], unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a graphic image from byte array with parameters. Draw a graphic image byte array on the screen at the specified x and y location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

**Parameters:**

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- vars* The byte array of parameters.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgaoutex.nxc](#).

**8.3.3.222 char GraphicOut (int x, int y, string filename, unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image. Draw a graphic image file on the screen at the specified x and y location. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphic](#), [DrawGraphicType](#)

**Parameters:**

*x* The x value for the position of the graphic image.

*y* The y value for the position of the graphic image.

*filename* The filename of the RIC graphic image.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_GraphicOut.nxc](#).

**8.3.3.223 char GraphicOutEx (int x, int y, string filename, byte vars[], unsigned long options = DRAW\_OPT\_NORMAL) [inline]**

Draw a graphic image with parameters. Draw a graphic image file on the screen at the specified x and y location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

**See also:**

[SysDrawGraphic](#), [DrawGraphicType](#)

**Parameters:**

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- filename* The filename of the RIC graphic image.
- vars* The byte array of parameters.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_dispgoutex.nxc](#), and [ex\\_GraphicOutEx.nxc](#).

**8.3.3.224 byte HSAddress (void) [inline]**

Get hi-speed port address. This method returns the value of the hi-speed port address.

**Returns:**

The hi-speed port address. See [Hi-speed port address constants](#).

**8.3.3.225 int HSDataMode (void) [inline]**

Get hi-speed port datamode. This method returns the value of the hi-speed port data mode.

**Returns:**

The hi-speed port data mode. See [Data mode constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**8.3.3.226 byte HSFlags (void) [inline]**

Get hi-speed port flags. This method returns the value of the hi-speed port flags.

**Returns:**

The hi-speed port flags. See [Hi-speed port flags constants](#).

**Examples:**

[ex\\_HSFlags.nxc](#).

**8.3.3.227 byte HSInputBufferInPtr (void) [inline]**

Get hi-speed port input buffer in-pointer. This method returns the value of the input pointer of the hi-speed port input buffer.

**Returns:**

The hi-speed port input buffer's in-pointer value.

**Examples:**

[ex\\_HSInputBufferInPtr.nxc](#).

**8.3.3.228 byte HSInputBufferOutPtr (void) [inline]**

Get hi-speed port input buffer out-pointer. This method returns the value of the output pointer of the hi-speed port input buffer.

**Returns:**

The hi-speed port input buffer's out-pointer value.

**Examples:**

[ex\\_HSInputBufferOutPtr.nxc](#).

**8.3.3.229 int HSMMode (void) [inline]**

Get hi-speed port mode. This method returns the value of the hi-speed port mode.

**Returns:**

The hi-speed port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_HSMode.nxc](#).

**8.3.3.230 byte HSOutputBufferInPtr (void) [inline]**

Get hi-speed port output buffer in-pointer. This method returns the value of the input pointer of the hi-speed port output buffer.

**Returns:**

The hi-speed port output buffer's in-pointer value.

**Examples:**

[ex\\_HSOutputBufferInPtr.nxc](#).

**8.3.3.231 byte HSOutputBufferOutPtr (void) [inline]**

Get hi-speed port output buffer out-pointer. This method returns the value of the output pointer of the hi-speed port output buffer.

**Returns:**

The hi-speed port output buffer's out-pointer value.

**Examples:**

[ex\\_HSOutputBufferOutPtr.nxc](#).

### 8.3.3.232 byte HSSpeed (void) [inline]

Get hi-speed port speed. This method returns the value of the hi-speed port speed (baud rate).

**Returns:**

The hi-speed port speed (baud rate). See [Hi-speed port baud rate constants](#).

**Examples:**

[ex\\_HSSpeed.nxc](#).

### 8.3.3.233 byte HSState (void) [inline]

Get hi-speed port state. This method returns the value of the hi-speed port state.

**Returns:**

The hi-speed port state. See [Hi-speed port state constants](#).

**Examples:**

[ex\\_HSState.nxc](#).

### 8.3.3.234 char HTIRTrain (const byte *port*, const byte *channel*, const byte *func*) [inline]

HTIRTrain function. Control an IR Train receiver set to the specified channel using the HiTechnic iRLink device. Valid func values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channel values are [TRAIN\\_CHANNEL\\_1](#) through [TRAIN\\_CHANNEL\\_3](#) and [TRAIN\\_CHANNEL\\_ALL](#). The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The IR Train channel. See [IR Train channel constants](#).

*func* The IR Train function. See [PF/IR Train function constants](#)

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTIRTrain.nxc](#).

**8.3.3.235 char HTPFComboDirect (const byte *port*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

HTPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the HiTechnic iRLink device. Commands for outa and outb are [PF\\_CMD\\_STOP](#), [PF\\_CMD\\_REV](#), [PF\\_CMD\\_FWD](#), and [PF\\_CMD\\_BRAKE](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*outa* The Power Function command for output A. See [Power Function command constants](#).

*outb* The Power Function command for output B. See [Power Function command constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFComboDirect.nxc](#).

**8.3.3.236 char HTPFComboPWM (const byte *port*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

HTPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Valid output values are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#),

[PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*outa* The Power Function PWM command for output A. See [Power Function PWM option constants](#).

*outb* The Power Function PWM command for output B. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFComboPWM.nxc](#).

**8.3.3.237 char HTPFRawOutput (const byte *port*, const byte *nibble0*, const byte *nibble1*, const byte *nibble2*) [inline]**

HTPFRawOutput function. Control a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Build the raw data stream using the 3 nibbles (4 bit values). The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*nibble0* The first raw data nibble.

*nibble1* The second raw data nibble.

*nibble2* The third raw data nibble.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFRawOutput.nxc](#).

**8.3.3.238** `char HTPFRepeat (const byte port, const byte count, const unsigned int delay) [inline]`

HTPFRepeat function. Repeat sending the last Power Function command using the HiTechnic iRLink device. Specify the number of times to repeat the command and the number of milliseconds of delay between each repetition. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- count* The number of times to repeat the command.
- delay* The number of milliseconds to delay between each repetition.

**Returns:**

- The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFRepeat.nxc](#).

**8.3.3.239** `char HTPFSingleOutputCST (const byte port, const byte channel, const byte out, const byte func) [inline]`

HTPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_CST\\_CLEAR1\\_CLEAR2](#), [PF\\_CST\\_SET1\\_CLEAR2](#), [PF\\_CST\\_CLEAR1\\_SET2](#), [PF\\_CST\\_SET1\\_SET2](#), [PF\\_CST\\_INCREMENT\\_PWM](#), [PF\\_CST\\_DECREMENT\\_PWM](#), [PF\\_CST\\_FULL\\_FWD](#), [PF\\_CST\\_FULL\\_REV](#), and [PF\\_CST\\_TOGGLE\\_DIR](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- func* The Power Function CST function. See [Power Function CST options constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#).

**8.3.3.240 char HTPFSingleOutputPWM (const byte *port*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

HTPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#), [PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*channel* The Power Function channel. See [Power Function channel constants](#).

*out* The Power Function output. See [Power Function output constants](#).

*func* The Power Function PWM function. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFSingleOutputPWM.nxc](#).

**8.3.3.241 char HTPFSinglePin (const byte *port*, const byte *channel*, const byte *out*, const byte *pin*, const byte *func*, bool *cont*) [inline]**

HTPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using

`PF_OUT_A` or `PF_OUT_B`. Select the desired pin using `PF_PIN_C1` or `PF_PIN_C2`. Valid functions are `PF_FUNC_NOCHANGE`, `PF_FUNC_CLEAR`, `PF_FUNC_SET`, and `PF_FUNC_TOGGLE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. Specify whether the mode by passing true (continuous) or false (timeout) as the final parameter. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- pin* The Power Function pin. See [Power Function pin constants](#).
- func* The Power Function single pin function. See [Power Function single pin function constants](#).
- cont* Control whether the mode is continuous or timeout.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFSinglePin.nxc](#).

### 8.3.3.242 char HTPFTrain (const byte *port*, const byte *channel*, const byte *func*) [[inline](#)]

HTPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device as if it were an IR Train receiver. Valid function values are `TRAIN_FUNC_STOP`, `TRAIN_FUNC_INCR_SPEED`, `TRAIN_FUNC_DECR_SPEED`, and `TRAIN_FUNC_TOGGLE_LIGHT`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- channel* The Power Function channel. See [Power Function channel constants](#).
- func* The Power Function train function. See [PF/IR Train function constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_HTPFTrain.nxc](#).

**8.3.3.243 void HTRCXAddToDatalog (const byte *src*, const unsigned int *value*) [inline]**

HTRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#).

**8.3.3.244 int HTRCXBatteryLevel (void) [inline]**

HTRCXBatteryLevel function. Send the BatteryLevel command to an RCX to read the current battery level.

**Returns:**

The RCX battery level.

**Examples:**

[ex\\_HTRCXBatteryLevel.nxc](#).

**8.3.3.245 void HTRCXCLEARAllEvents (void) [inline]**

HTRCXCLEARAllEvents function. Send the CLEARAllEvents command to an RCX.

**Examples:**

[ex\\_HTRCXCLEARAllEvents.nxc](#).

**8.3.3.246 void HTRCXCclearCounter (const byte *counter*) [inline]**

HTRCXCclearCounter function. Send the ClearCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

[ex\\_HTRCXCclearCounter.nxc](#).

**8.3.3.247 void HTRCXCclearMsg (void) [inline]**

HTRCXCclearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

[ex\\_HTRCXCclearMsg.nxc](#).

**8.3.3.248 void HTRCXCclearSensor (const byte *port*) [inline]**

HTRCXCclearSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

[ex\\_HTRCXCclearSensor.nxc](#).

**8.3.3.249 void HTRCXCclearSound (void) [inline]**

HTRCXCclearSound function. Send the ClearSound command to an RCX.

**Examples:**

[ex\\_HTRCXCclearSound.nxc](#).

**8.3.3.250 void HTRCXClearTimer (const byte *timer*) [inline]**

HTRCXClearTimer function. Send the ClearTimer command to an RCX.

**Parameters:**

*timer* The timer to clear.

**Examples:**

[ex\\_HTRCXClearTimer.nxc](#).

**8.3.3.251 void HTRCXCreateDatalog (const unsigned int *size*) [inline]**

HTRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

**Parameters:**

*size* The new datalog size.

**Examples:**

[ex\\_HTRCXCreateDatalog.nxc](#).

**8.3.3.252 void HTRCXDecCounter (const byte *counter*) [inline]**

HTRCXDecCounter function. Send the DecCounter command to an RCX.

**Parameters:**

*counter* The counter to decrement.

**Examples:**

[ex\\_HTRCXDecCounter.nxc](#).

**8.3.3.253 void HTRCXDeleteSub (const byte *s*) [inline]**

HTRCXDeleteSub function. Send the DeleteSub command to an RCX.

**Parameters:**

*s* The subroutine number to delete.

**Examples:**

[ex\\_HTRCXDeleteSub.nxc](#).

**8.3.3.254 void HTRCXDeleteSubs (void) [inline]**

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

[ex\\_HTRCXDeleteSubs.nxc](#).

**8.3.3.255 void HTRCXDeleteTask (const byte *t*) [inline]**

HTRCXDeleteTask function. Send the DeleteTask command to an RCX.

**Parameters:**

*t* The task number to delete.

**Examples:**

[ex\\_HTRCXDeleteTask.nxc](#).

**8.3.3.256 void HTRCXDeleteTasks (void) [inline]**

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

[ex\\_HTRCXDeleteTasks.nxc](#).

**8.3.3.257 void HTRCXDisableOutput (const byte *outputs*) [inline]**

HTRCXDisableOutput function. Send the DisableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to disable. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXDisableOutput.nxc](#).

**8.3.3.258 void HTRCXEnableOutput (const byte *outputs*) [inline]**

HTRCXEnableOutput function. Send the EnableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to enable. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXEnableOutput.nxc](#).

**8.3.3.259 void HTRCXEvent (const byte *src*, const unsigned int *value*) [inline]**

HTRCXEvent function. Send the Event command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXEvent.nxc](#).

**8.3.3.260 void HTRCXFloat (const byte *outputs*) [inline]**

HTRCXFloat function. Send commands to an RCX to float the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to float. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXFloat.nxc](#).

**8.3.3.261 void HTRCXFwd (const byte *outputs*) [inline]**

HTRCXFwd function. Send commands to an RCX to set the specified outputs to the forward direction.

**Parameters:**

*outputs* The RCX output(s) to set forward. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXFwd.nxc](#).

**8.3.3.262 void HTRCXIncCounter (const byte *counter*) [inline]**

HTRCXIncCounter function. Send the IncCounter command to an RCX.

**Parameters:**

*counter* The counter to increment.

**Examples:**

[ex\\_HTRCXIncCounter.nxc](#).

**8.3.3.263 void HTRCXInvertOutput (const byte *outputs*) [inline]**

HTRCXInvertOutput function. Send the InvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to invert. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXInvertOutput.nxc](#).

**8.3.3.264 void HTRCXMuteSound (void) [inline]**

HTRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

[ex\\_HTRCXMuteSound.nxc](#).

**8.3.3.265 void HTRCXObvertOutput (const byte *outputs*) [inline]**

HTRCXObvertOutput function. Send the ObvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to obvert. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXObvertOutput.nxc](#).

**8.3.3.266 void HTRCXOff (const byte *outputs*) [inline]**

HTRCXOff function. Send commands to an RCX to turn off the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn off. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOff.nxc](#).

**8.3.3.267 void HTRCXOn (const byte *outputs*) [inline]**

HTRCXOn function. Send commands to an RCX to turn on the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOn.nxc](#).

**8.3.3.268 void HTRCXOnFor (const byte *outputs*, const unsigned int *ms*) [inline]**

HTRCXOnFor function. Send commands to an RCX to turn on the specified outputs in the forward direction for the specified duration.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

*ms* The number of milliseconds to leave the outputs on

**Examples:**

[ex\\_HTRCXOnFor.nxc](#).

**8.3.3.269 void HTRCXOnFwd (const byte *outputs*) [inline]**

HTRCXOnFwd function. Send commands to an RCX to turn on the specified outputs in the forward direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the forward direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOnFwd.nxc](#).

**8.3.3.270 void HTRCXOnRev (const byte *outputs*) [inline]**

HTRCXOnRev function. Send commands to an RCX to turn on the specified outputs in the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXOnRev.nxc](#).

**8.3.3.271 void HTRCXPBTurnOff (void) [inline]**

HTRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

[ex\\_HTRCXPBTurnOff.nxc](#).

**8.3.3.272 void HTRCXPing (void) [inline]**

HTRCXPing function. Send the Ping command to an RCX.

**Examples:**

[ex\\_HTRCXPing.nxc](#).

**8.3.3.273 void HTRCXPlaySound (const byte *snd*) [inline]**

HTRCXPlaySound function. Send the PlaySound command to an RCX.

**Parameters:**

*snd* The sound number to play.

**Examples:**

[ex\\_HTRCXPlaySound.nxc](#).

**8.3.3.274** void HTRCXPlayTone (const unsigned int *freq*, const byte *duration*)  
[inline]

HTRCXPlayTone function. Send the PlayTone command to an RCX.

**Parameters:**

*freq* The frequency of the tone to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_HTRCXPlayTone.nxc](#).

**8.3.3.275** void HTRCXPlayToneVar (const byte *varnum*, const byte *duration*)  
[inline]

HTRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

**Parameters:**

*varnum* The variable containing the tone frequency to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_HTRCXPlayToneVar.nxc](#).

**8.3.3.276** int HTRCXPoll (const byte *src*, const byte *value*) [inline]

HTRCXPoll function Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Returns:**

The value read from the specified port and value.

**Examples:**

[ex\\_HTRCXPoll.nxc](#).

**8.3.3.277 int HTRCXPollMemory (const unsigned int *address*) [inline]**

HTRCXPollMemory function. Send the PollMemory command to an RCX.

**Parameters:**

*address* The RCX memory address.

**Returns:**

The value read from the specified address.

**Examples:**

[ex\\_HTRCXPollMemory.nxc](#).

**8.3.3.278 void HTRCXRemote (unsigned int *cmd*) [inline]**

HTRCXRemote function. Send the Remote command to an RCX.

**Parameters:**

*cmd* The RCX IR remote command to send. See [RCX IR remote constants](#).

**Examples:**

[ex\\_HTRCXRemote.nxc](#).

**8.3.3.279 void HTRCXRev (const byte *outputs*) [inline]**

HTRCXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXRev.nxc](#).

**8.3.3.280 void HTRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]**

HTRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_HTRCXSelectDisplay.nxc](#).

**8.3.3.281 void HTRCXSelectProgram (const byte *prog*) [inline]**

HTRCXSelectProgram function. Send the SelectProgram command to an RCX.

**Parameters:**

*prog* The program number to select.

**Examples:**

[ex\\_HTRCXSelectProgram.nxc](#).

**8.3.3.282** void HTRCXSendSerial (const byte *first*, const byte *count*)  
[inline]

HTRCXSendSerial function. Send the SendSerial command to an RCX.

**Parameters:**

- first* The first byte address.
- count* The number of bytes to send.

**Examples:**

[ex\\_HTRCXSendSerial.nxc](#).

**8.3.3.283** void HTRCXSetDirection (const byte *outputs*, const byte *dir*)  
[inline]

HTRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

**Parameters:**

- outputs* The RCX output(s) to set direction. See [RCX output constants](#).
- dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_HTRCXSetDirection.nxc](#).

**8.3.3.284** void HTRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)  
[inline]

HTRCXSetEvent function. Send the SetEvent command to an RCX.

**Parameters:**

- evt* The event number to set.
- src* The RCX source. See [RCX and Scout source constants](#).
- type* The event type.

**Examples:**

[ex\\_HTRCXSetEvent.nxc](#).

**8.3.3.285** void HTRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)  
[inline]

HTRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_HTRCXSetGlobalDirection.nxc](#).

**8.3.3.286** void HTRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)  
[inline]

HTRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_HTRCXSetGlobalOutput.nxc](#).

**8.3.3.287** void HTRCXSetIRLinkPort (const byte *port*) [inline]

HTRCXSetIRLinkPort function. Set the global port in advance of using the HTRCX\* and HTScout\* API functions for sending RCX and Scout messages over the HiTechnic iRLink device. The port must be configured as a Low-speed port before using any of the HiTechnic RCX and Scout iRLink functions.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**8.3.3.288** void HTRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [**inline**]

HTRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set max power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_HTRCXSetMaxPower.nxc](#).

**8.3.3.289** void HTRCXSetMessage (const byte *msg*) [**inline**]

HTRCXSetMessage function. Send the SetMessage command to an RCX.

**Parameters:**

*msg* The numeric message to send.

**Examples:**

[ex\\_HTRCXSetMessage.nxc](#).

**8.3.3.290** void HTRCXSetOutput (const byte *outputs*, const byte *mode*) [**inline**]

HTRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

**Parameters:**

*outputs* The RCX output(s) to set mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_HTRCXSetOutput.nxc](#).

**8.3.3.291 void HTRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]**

HTRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_HTRCXSetPower.nxc](#).

**8.3.3.292 void HTRCXSetPriority (const byte *p*) [inline]**

HTRCXSetPriority function. Send the SetPriority command to an RCX.

**Parameters:**

*p* The new task priority.

**Examples:**

[ex\\_HTRCXSetPriority.nxc](#).

**8.3.3.293 void HTRCXSetSensorMode (const byte *port*, const byte *mode*) [inline]**

HTRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

**Parameters:**

- port* The RCX sensor port.  
*mode* The RCX sensor mode.

**Examples:**

[ex\\_HTRCXSetSensorMode.nxc](#).

**8.3.3.294** void HTRCXSetSensorType (const byte *port*, const byte *type*)  
[inline]

HTRCXSetSensorType function. Send the SetSensorType command to an RCX.

**Parameters:**

- port* The RCX sensor port.  
*type* The RCX sensor type.

**Examples:**

[ex\\_HTRCXSetSensorType.nxc](#).

**8.3.3.295** void HTRCXSetSleepTime (const byte *t*) [inline]

HTRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

**Parameters:**

- t* The new sleep time value.

**Examples:**

[ex\\_HTRCXSetSleepTime.nxc](#).

**8.3.3.296** void HTRCXSetTxPower (const byte *pwr*) [inline]

HTRCXSetTxPower function. Send the SetTxPower command to an RCX.

**Parameters:**

*pwr* The IR transmit power level.

**Examples:**

[ex\\_HTRCXSetTxPower.nxc](#).

**8.3.3.297 void HTRCXSetWatch (const byte *hours*, const byte *minutes*)  
[inline]**

HTRCXSetWatch function. Send the SetWatch command to an RCX.

**Parameters:**

*hours* The new watch time hours value.

*minutes* The new watch time minutes value.

**Examples:**

[ex\\_HTRCXSetWatch.nxc](#).

**8.3.3.298 void HTRCXStartTask (const byte *t*) [inline]**

HTRCXStartTask function. Send the StartTask command to an RCX.

**Parameters:**

*t* The task number to start.

**Examples:**

[ex\\_HTRCXStartTask.nxc](#).

**8.3.3.299 void HTRCXStopAllTasks (void) [inline]**

HTRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

[ex\\_HTRCXStopAllTasks.nxc](#).

**8.3.3.300 void HTRCXStopTask (const byte *t*) [inline]**

HTRCXStopTask function. Send the StopTask command to an RCX.

**Parameters:**

*t* The task number to stop.

**Examples:**

[ex\\_HTRCXStopTask.nxc](#).

**8.3.3.301 void HTRCXToggle (const byte *outputs*) [inline]**

HTRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to toggle. See [RCX output constants](#).

**Examples:**

[ex\\_HTRCXToggle.nxc](#).

**8.3.3.302 void HTRCXUnmuteSound (void) [inline]**

HTRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

[ex\\_HTRCXUnmuteSound.nxc](#).

**8.3.3.303 void HTScoutCalibrateSensor (void) [inline]**

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

[ex\\_HTScoutCalibrateSensor.nxc](#).

**8.3.3.304 void HTScoutMuteSound (void) [inline]**

HTScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

[ex\\_HTScoutMuteSound.nxc](#).

**8.3.3.305 void HTScoutSelectSounds (const byte grp) [inline]**

HTScoutSelectSounds function. Send the SelectSounds command to a Scout.

**Parameters:**

*grp* The Scout sound group to select.

**Examples:**

[ex\\_HTScoutSelectSounds.nxc](#).

**8.3.3.306 void HTScoutSendVLL (const byte src, const unsigned int value) [inline]**

HTScoutSendVLL function. Send the SendVLL command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSendVLL.nxc](#).

**8.3.3.307 void HTScoutSetEventFeedback (const byte src, const unsigned int value) [inline]**

HTScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetEventFeedback.nxc](#).

**8.3.3.308 void HTScoutSetLight (const byte *x*) [inline]**

HTScoutSetLight function. Send the SetLight command to a Scout.

**Parameters:**

*x* Set the light on or off using this value. See [Scout light constants](#).

**Examples:**

[ex\\_HTScoutSetLight.nxc](#).

**8.3.3.309 void HTScoutSetScoutMode (const byte *mode*) [inline]**

HTScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

**Parameters:**

*mode* Set the scout mode. See [Scout mode constants](#).

**Examples:**

[ex\\_HTScoutSetScoutMode.nxc](#).

**8.3.3.310 void HTScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorClickTime.nxc](#).

**8.3.3.311 void HTScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorHysteresis.nxc](#).

**8.3.3.312 void HTScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorLowerLimit.nxc](#).

**8.3.3.313 void HTScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]**

HTScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_HTScoutSetSensorUpperLimit.nxc](#).

**8.3.3.314 void HTScoutUnmuteSound (void) [inline]**

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

[ex\\_HTScoutUnmuteSound.nxc](#).

**8.3.3.315 long I2CBytes (const byte *port*, byte *inbuf*[], byte & *count*, byte & *outbuf*[]) [inline]**

Perform an I2C write/read transaction. This method writes the bytes contained in the input buffer (*inbuf*) to the I2C device on the specified port, checks for the specified number of bytes to be ready for reading, and then tries to read the specified number (*count*) of bytes from the I2C device into the output buffer (*outbuf*).

This is a higher-level wrapper around the three main I2C functions. It also maintains a "last good read" buffer and returns values from that buffer if the I2C communication transaction fails.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*inbuf* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

*count* The number of bytes that should be returned by the I2C device. On output count is set to the number of bytes in outbuf.

*outbuf* A byte array that contains the data read from the internal I2C buffer.

**Returns:**

Returns true or false indicating whether the I2C transaction succeeded or failed.

**See also:**

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [I2CRead](#), [LowSpeedRead](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CBytes.nxc](#).

**8.3.3.316 byte I2CBytesReady (const byte port) [inline]**

Get I2C bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [LowSpeedBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CBytesReady.nxc](#).

**8.3.3.317 long I2CCheckStatus (const byte *port*) [inline]**

Check I2C status. This method checks the status of the I2C communication on the specified port.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while this function returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [LowSpeedStatus](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedCheckStatus](#)

**Examples:**

[ex\\_I2CCheckStatus.nxc](#).

**8.3.3.318 string I2CDeviceId (byte *port*, byte *i2caddr*) [inline]**

Read I2C device identifier. Read standard I2C device identifier. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device identifier.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

**8.3.3.319 string I2CDeviceInfo (byte *port*, byte *i2caddr*, byte *info*) [inline]**

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

*info* A value indicating the type of device information you are requesting. See [Standard I2C constants](#).

**Returns:**

A string containing the requested device information.

**Examples:**

[ex\\_i2cdeviceinfo.nxc](#).

**8.3.3.320 long I2CRead (const byte *port*, byte *buflen*, byte & *buffer*[ ]) [inline]**

Read I2C data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*buflen* The initial size of the output buffer.

*buffer* A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is `NO_ERR` then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_I2CRead.nxc](#).

**8.3.3.321 long I2CSendCommand (byte port, byte i2caddr, byte cmd)  
[inline]**

Send an I2C command. Send a command to an I2C device at the standard command register: [I2C\\_REG\\_CMD](#). The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

*cmd* The command to send to the I2C device.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_I2CSendCommand.nxc](#).

**8.3.3.322 long I2CStatus (const byte port, byte & bytesready) [inline]**

Get I2C status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*bytesready* The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible return values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while I2CStatus returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [LowspeedStatus](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

**Examples:**

[ex\\_I2CStatus.nxc](#).

**8.3.3.323 string I2CVendorId (byte port, byte i2caddr) [inline]**

Read I2C device vendor. Read standard I2C device vendor. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device vendor.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

**8.3.3.324 string I2CVersion (byte port, byte i2caddr) [inline]**

Read I2C device version. Read standard I2C device version. The I2C device uses the specified address.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*i2caddr* The I2C device address.

**Returns:**

A string containing the device version.

**Examples:**

[ex\\_i2cdeviceid.nxc](#), [ex\\_i2cvendorid.nxc](#), and [ex\\_i2cversion.nxc](#).

**8.3.3.325 long I2CWrite (const byte port, byte retlen, byte buffer[]) [inline]**

Write I2C data. This method starts a transaction to write the bytes contained in the array buffer to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*retlen* The number of bytes that should be returned by the I2C device.

*buffer* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is `NO_ERR` then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

**Examples:**

[ex\\_I2CWrite.nxc](#).

**8.3.3.326 int isalnum (int c) [inline]**

Check if character is alphanumeric. Checks if parameter *c* is either a decimal digit or an uppercase or lowercase letter. The result is true if either `isalpha` or `isdigit` would also return true.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is either a digit or a letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isalnum.nxc](#).

**8.3.3.327 int isalpha (int *c*) [inline]**

Check if character is alphabetic. Checks if parameter *c* is either an uppercase or lowercase letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isalpha.nxc](#).

**8.3.3.328 int iscntrl (int *c*) [inline]**

Check if character is a control character. Checks if parameter *c* is a control character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a control character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_iscntrl.nxc](#).

**8.3.3.329 int isdigit (int *c*) [inline]**

Check if character is decimal digit. Checks if parameter *c* is a decimal digit character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a decimal digit, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isdigit.nxc](#).

**8.3.3.330 int isgraph (int *c*) [inline]**

Check if character has graphical representation. Checks if parameter *c* is a character with a graphical representation.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* has a graphical representation, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isgraph.nxc](#).

**8.3.3.331 int islower (int *c*) [inline]**

Check if character is lowercase letter. Checks if parameter *c* is an lowercase alphabetic letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an lowercase alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_islower.nxc](#).

**8.3.3.332 bool isNaN (float *value*) [inline]**

Is the value NaN. Returns true if the floating point value is NaN (not a number).

**Parameters:**

*value* A floating point variable.

**Returns:**

Whether the value is NaN.

**Examples:**

[ex\\_isnan.nxc](#), and [ex\\_labs.nxc](#).

**8.3.3.333 int isprint (int *c*) [inline]**

Check if character is printable. Checks if parameter *c* is a printable character (i.e., not a control character).

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a printable character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isprint.nxc](#).

**8.3.3.334 int ispunct (int *c*) [inline]**

Check if character is a punctuation. Checks if parameter *c* is a punctuation character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a punctuation character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_ispunct.nxc](#).

**8.3.3.335 int isspace (int *c*) [inline]**

Check if character is a white-space. Checks if parameter *c* is a white-space character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a white-space character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isspace.nxc](#).

**8.3.3.336 int isupper (int *c*) [inline]**

Check if character is uppercase letter. Checks if parameter *c* is an uppercase alphabetic letter.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is an uppercase alphabetic letter, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isupper.nxc](#).

**8.3.3.337 int isxdigit (int *c*) [inline]**

Check if character is hexadecimal digit. Checks if parameter *c* is a hexadecimal digit character.

**Parameters:**

*c* Character to be checked.

**Returns:**

Returns a non-zero value (true) if *c* is a hexadecimal digit character, otherwise it returns 0 (false).

**Examples:**

[ex\\_ctype.nxc](#), and [ex\\_isxdigit.nxc](#).

**8.3.3.338 char JoystickMessageRead (byte *queue*, JoystickMessageType & *msg*) [inline]**

Read a joystick message from a queue/mailbox. Read a joystick message from a queue/mailbox.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*msg* The joystick message that is read from the mailbox. See [JoystickMessageType](#) for details.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_joystickmsg.nxc](#).

**8.3.3.339 long labs (long *n*) [inline]**

Absolute value. Return the absolute value of parameter *n*.

**Parameters:**

*n* Integral value.

**Returns:**

The absolute value of *n*.

**8.3.3.340 ldiv\_t ldiv (long numer, long denom) [inline]**

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type `ldiv_t`, which has two members: `quot` and `rem`.

**Parameters:**

*numer* Numerator.

*denom* Denominator.

**Returns:**

The result is returned by value in a structure defined in `cstdlib`, which has two members. For `ldiv_t`, these are, in either order: `long quot`; `long rem`.

**Examples:**

[ex\\_ldiv.nxc](#).

**8.3.3.341 string LeftStr (string str, unsigned int size) [inline]**

Copy a portion from the start of a string. Returns the substring of a specified length that appears at the start of a string.

**Parameters:**

*str* A string

*size* The size or length of the substring.

**Returns:**

The substring of a specified length that appears at the start of a string.

**Examples:**

[ex\\_leftstr.nxc](#).

**8.3.3.342** `char LineOut (int x1, int y1, int x2, int y2, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a line. This function lets you draw a line on the screen from x1, y1 to x2, y2. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawLine](#), [DrawLineType](#)

**Parameters:**

*x1* The x value for the start of the line.  
*y1* The y value for the start of the line.  
*x2* The x value for the end of the line.  
*y2* The y value for the end of the line.  
*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_LineOut.nxc](#).

**8.3.3.343** `float log (float x) [inline]`

Compute natural logarithm. Computes the natural logarithm of x. The natural logarithm is the base-e logarithm, the inverse of the natural exponential function (exp). For base-10 logarithms, a specific function [log10\(\)](#) exists.

**See also:**

[log10\(\)](#), [exp\(\)](#)

**Parameters:**

*x* Floating point value.

**Returns:**

Natural logarithm of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_log.nxc](#).

**8.3.3.344 float log10 (float  $x$ ) [inline]**

Compute common logarithm. Computes the common logarithm of  $x$ . The common logarithm is the base-10 logarithm. For base-e logarithms, a specific function [log\(\)](#) exists.

**See also:**

[log\(\)](#), [exp\(\)](#)

**Parameters:**

$x$  Floating point value.

**Returns:**

Common logarithm of  $x$ .

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_log10.nxc](#).

**8.3.3.345 byte LongAbort (void) [inline]**

Read long abort setting. Return the enhanced NBC/NXC firmware's long abort setting.

**See also:**

[AbortFlag](#)

**Returns:**

The current abort flag value. See [ButtonState constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_LongAbort.nxc](#).

**8.3.3.346 byte LowSpeedBytesReady (const byte *port*) [inline]**

Get lowspeed bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful `LowSpeedWrite` call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_LowSpeedBytesReady.nxc](#).

**8.3.3.347 long LowSpeedCheckStatus (const byte *port*) [inline]**

Check lowspeed status. This method checks the status of the I2C communication on the specified port.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while [LowspeedCheckStatus](#) returns [STAT\\_-COMM\\_PENDING](#).

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

**Examples:**

[ex\\_LowspeedCheckStatus.nxc](#).

**8.3.3.348 long LowspeedRead (const byte *port*, byte *buflen*, byte & *buffer*[ ])  
[inline]**

Read lowspeed data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*buflen* The initial size of the output buffer.

*buffer* A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors.

**See also:**

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowSpeedWrite](#), [LowSpeedCheckStatus](#), [LowSpeedBytesReady](#), and [LowSpeedStatus](#)

**Examples:**

[ex\\_LowSpeedRead.nxc](#).

**8.3.3.349 long LowSpeedStatus (const byte *port*, byte & *bytesready*)  
[inline]**

Get lowspeed status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful [LowSpeedWrite](#) call that requested response data from the device then *bytesready* will be set to the number of bytes in the internal read buffer.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*bytesready* The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO\\_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowSpeedRead](#) or [LowSpeedWrite](#) while [LowSpeedStatus](#) returns [STAT\\_COMM\\_PENDING](#).

**See also:**

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [I2CCheckStatus](#), [I2CBytesReady](#), [LowSpeedRead](#), [LowSpeedWrite](#), and [LowSpeedCheckStatus](#)

**Examples:**

[ex\\_LowSpeedStatus.nxc](#).

### 8.3.3.350 long LowspeedWrite (const byte *port*, byte *retlen*, byte *buffer*[ ]) [inline]

Write lowspeed data. This method starts a transaction to write the bytes contained in the array *buffer* to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

#### Parameters:

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

*retlen* The number of bytes that should be returned by the I2C device.

*buffer* A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

#### Returns:

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is `NO_ERR` then the last operation did not cause any errors.

#### See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

#### Examples:

[ex\\_LowspeedWrite.nxc](#).

### 8.3.3.351 byte LSChannelState (const byte *port*) [inline]

Get I2C channel state. This method returns the value of the I2C channel state for the specified port.

#### Parameters:

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port channel state. See [LSChannelState constants](#).

**Examples:**

[ex\\_LSChannelState.nxc](#).

**8.3.3.352 byte LSErrorType (const byte port) [inline]**

Get I2C error type. This method returns the value of the I2C error type for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port error type. See [LSErrorType constants](#).

**Examples:**

[ex\\_LSErrorType.nxc](#).

**8.3.3.353 byte LSInputBufferBytesToRx (const byte port) [inline]**

Get I2C input buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's bytes to rx value.

**Examples:**

[ex\\_LSInputBufferBytesToRx.nxc](#).

**8.3.3.354** byte `LSInputBufferInPtr` (const byte *port*) [`inline`]

Get I2C input buffer in-pointer. This method returns the value of the input pointer of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's in-pointer value.

**Examples:**

[ex\\_LSInputBufferInPtr.nxc](#).

**8.3.3.355** byte `LSInputBufferOutPtr` (const byte *port*) [`inline`]

Get I2C input buffer out-pointer. This method returns the value of the output pointer of the I2C input buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C input buffer's out-pointer value.

**Examples:**

[ex\\_LSInputBufferOutPtr.nxc](#).

**8.3.3.356** byte `LSMode` (const byte *port*) [`inline`]

Get I2C mode. This method returns the value of the I2C mode for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C port mode. See [LSMode constants](#).

**Examples:**

[ex\\_LSMODE.nxc](#).

**8.3.3.357 byte LSNoRestartOnRead () [inline]**

Get I2C no restart on read setting. This method returns the value of the I2C no restart on read field.

**Returns:**

The I2C no restart on read field. See [LSNoRestartOnRead constants](#).

**Examples:**

[ex\\_LSNoRestartOnRead.nxc](#).

**8.3.3.358 byte LSOutputBufferBytesToRx (const byte port) [inline]**

Get I2C output buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's bytes to rx value.

**Examples:**

[ex\\_LSOutputBufferBytesToRx.nxc](#).

**8.3.3.359 byte LSOutputBufferInPtr (const byte port) [inline]**

Get I2C output buffer in-pointer. This method returns the value of the input pointer of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's in-pointer value.

**Examples:**

[ex\\_LSOutputBufferInPtr.nxc](#).

**8.3.3.360 byte LSOutputBufferOutPtr (const byte *port*) [inline]**

Get I2C output buffer out-pointer. This method returns the value of the output pointer of the I2C output buffer for the specified port.

**Parameters:**

*port* A constant port number (S1..S4). See [Input port constants](#).

**Returns:**

The I2C output buffer's out-pointer value.

**Examples:**

[ex\\_LSOutputBufferOutPtr.nxc](#).

**8.3.3.361 byte LSSpeed () [inline]**

Get I2C speed. This method returns the value of the I2C speed.

**Returns:**

The I2C speed.

**Warning:**

This function is unimplemented within the firmware.

**Examples:**

[ex\\_LSSpeed.nxc](#).

**8.3.3.362** byte LSSState () [**inline**]

Get I2C state. This method returns the value of the I2C state.

**Returns:**

The I2C state. See [LSSState constants](#).

**Examples:**

[ex\\_LSSState.nxc](#).

**8.3.3.363** char memcmp (variant *ptr1*, variant *ptr2*, byte *num*) [**inline**]

Compare two blocks of memory. Compares the variant *ptr1* to the variant *ptr2*. Returns an integral value indicating the relationship between the variables. The *num* argument is ignored.

**Parameters:**

*ptr1* A variable to be compared.

*ptr2* A variable to be compared.

*num* The number of bytes to compare (ignored).

**Examples:**

[ex\\_memcmp.nxc](#).

**8.3.3.364** void memcpy (variant *dest*, variant *src*, byte *num*) [**inline**]

Copy memory. Copies memory contents from the source to the destination. The *num* argument is ignored.

**Parameters:**

*dest* The destination variable.

*src* The source variable.

*num* The number of bytes to copy (ignored).

**Examples:**

[ex\\_memcpy.nxc](#).

**8.3.3.365 void memmove (variant *dest*, variant *src*, byte *num*) [inline]**

Move memory. Moves memory contents from the source to the destination. The *num* argument is ignored.

**Parameters:**

- dest* The destination variable.
- src* The source variable.
- num* The number of bytes to copy (ignored).

**Examples:**

[ex\\_memmove.nxc](#).

**8.3.3.366 string MidStr (string *str*, unsigned int *idx*, unsigned int *len*) [inline]**

Copy a portion from the middle of a string. Returns the substring of a specified length that appears at a specified position in a string.

**Parameters:**

- str* A string
- idx* The starting index of the substring.
- len* The length of the substring.

**Returns:**

The substring of a specified length that appears at a specified position in a string.

**Examples:**

[ex\\_midstr.nxc](#).

**8.3.3.367 char MotorActualSpeed (byte *output*) [inline]**

Get motor actual speed. Get the actual speed value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The actual speed value of the specified output.

**Examples:**

[ex\\_motoractualspeed.nxc](#).

**8.3.3.368 long MotorBlockTachoCount (byte output) [inline]**

Get motor block-relative counter. Get the block-relative position counter value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The block-relative position counter value of the specified output.

**Examples:**

[ex\\_motorblocktachocount.nxc](#).

**8.3.3.369 byte MotorMaxAcceleration (byte output) [inline]**

Get motor max acceleration. Get the max acceleration value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The max acceleration value of the specified output.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.370 byte MotorMaxSpeed (byte *output*) [inline]**

Get motor max speed. Get the max speed value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The max speed value of the specified output.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.371 byte MotorMode (byte *output*) [inline]**

Get motor mode. Get the mode of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The mode of the specified output.

**Examples:**

[ex\\_motormode.nxc](#).

**8.3.3.372 byte MotorOutputOptions (byte *output*) [inline]**

Get motor options. Get the options value of the specified output.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The options value of the specified output.

**Examples:**

[ex\\_motoroutputoptions.nxc](#).

**8.3.3.373 bool MotorOverload (byte *output*) [inline]**

Get motor overload status. Get the overload value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The overload value of the specified output.

**Examples:**

[ex\\_motoroverload.nxc](#).

**8.3.3.374 char MotorPower (byte *output*) [inline]**

Get motor power level. Get the power level of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The power level of the specified output.

**Examples:**

[ex\\_motorpower.nxc](#).

**8.3.3.375 byte MotorPwnFreq () [inline]**

Get motor regulation frequency. Get the current motor regulation frequency in milliseconds.

**Returns:**

The motor regulation frequency.

**Examples:**

[ex\\_motorpwnfreq.nxc](#).

**8.3.3.376 byte MotorRegDValue (byte output) [inline]**

Get motor D value. Get the derivative PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The derivative PID value of the specified output.

**Examples:**

[ex\\_motorregdvalue.nxc](#).

**8.3.3.377** byte `MotorRegIValue` (byte *output*) [`inline`]

Get motor I value. Get the integral PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be `OUT_A`, `OUT_B`, `OUT_C` or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The integral PID value of the specified output.

**Examples:**

[ex\\_motorregivalue.nxc](#).

**8.3.3.378** byte `MotorRegPValue` (byte *output*) [`inline`]

Get motor P value. Get the proportional PID value of the specified output.

**Parameters:**

*output* Desired output port. Can be `OUT_A`, `OUT_B`, `OUT_C` or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The proportional PID value of the specified output.

**Examples:**

[ex\\_motorregpvalue.nxc](#).

**8.3.3.379** byte `MotorRegulation` (byte *output*) [`inline`]

Get motor regulation mode. Get the regulation value of the specified output.

**Parameters:**

*output* Desired output port. Can be `OUT_A`, `OUT_B`, `OUT_C` or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The regulation value of the specified output.

**Examples:**

[ex\\_motorregulation.nxc](#).

**8.3.3.380 byte MotorRegulationOptions () [inline]**

Get motor regulation options. Get the current motor regulation options.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Returns:**

The motor regulation options.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.381 byte MotorRegulationTime () [inline]**

Get motor regulation time. Get the current motor regulation time in milliseconds.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Returns:**

The motor regulation time.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.382 long MotorRotationCount (byte output) [inline]**

Get motor program-relative counter. Get the program-relative position counter value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The program-relative position counter value of the specified output.

**Examples:**

[ex\\_motorrotationcount.nxc](#), and [util\\_rpm.nxc](#).

**8.3.3.383 byte MotorRunState (byte output) [inline]**

Get motor run state. Get the RunState value of the specified output, see [Output port run state constants](#).

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The RunState value of the specified output.

**Examples:**

[ex\\_motorrynstate.nxc](#).

**8.3.3.384 long MotorTachoCount (byte output) [inline]**

Get motor tachometer counter. Get the tachometer count value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The tachometer count value of the specified output.

**Examples:**

[ex\\_motortachocount.nxc](#).

**8.3.3.385 long MotorTachoLimit (byte *output*) [inline]**

Get motor tachometer limit. Get the tachometer limit value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The tachometer limit value of the specified output.

**Examples:**

[ex\\_motortacholimit.nxc](#).

**8.3.3.386 char MotorTurnRatio (byte *output*) [inline]**

Get motor turn ratio. Get the turn ratio value of the specified output.

**Parameters:**

*output* Desired output port. Can be [OUT\\_A](#), [OUT\\_B](#), [OUT\\_C](#) or a variable containing one of these values, see [Output port constants](#).

**Returns:**

The turn ratio value of the specified output.

**Examples:**

[ex\\_motorturnratio.nxc](#).

**8.3.3.387 char MSADPAOff (const byte *port*, const byte *i2caddr*) [inline]**

Turn off mindsensors ADPA mode. Turn ADPA mode off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSADPAOff.nxc](#).

**8.3.3.388 char MSADPAOn (const byte *port*, const byte *i2caddr*) [inline]**

Turn on mindsensors ADPA mode. Turn ADPA mode on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSADPAOn.nxc](#).

**8.3.3.389 char MSDeenergize (const byte *port*, const byte *i2caddr*) [inline]**

Turn off power to device. Turn power off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSDeenergize.nxc](#).

**8.3.3.390 char MSEnergize (const byte *port*, const byte *i2caddr*) [*inline*]**

Turn on power to device. Turn the power on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSEnergize.nxc](#).

**8.3.3.391 char MSIRTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [*inline*]**

MSIRTrain function. Control an IR Train receiver set to the specified channel using the mindsensors NRLink device. Valid function values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channels are [TRAIN\\_CHANNEL\\_1](#) through [TRAIN\\_CHANNEL\\_3](#) and [TRAIN\\_CHANNEL\\_ALL](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The IR Train channel. See [IR Train channel constants](#).

*func* The IR Train function. See [PF/IR Train function constants](#)

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSIRTrain.nxc](#).

**8.3.3.392 char MSPFComboDirect (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

MSPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the mindsensors NRLink device. Commands for *outa* and *outb* are PF\_CMD\_STOP, PF\_CMD\_REV, PF\_CMD\_FWD, and [PF\\_CMD\\_BRAKE](#). Valid channels are PF\_CHANNEL\_1 through PF\_CHANNEL\_4. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The Power Function channel. See [Power Function channel constants](#).

*outa* The Power Function command for output A. See [Power Function command constants](#).

*outb* The Power Function command for output B. See [Power Function command constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFComboDirect.nxc](#).

**8.3.3.393** `char MSPFComboPWM (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb) [inline]`

MSPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Valid output values are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#), [PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- outa* The Power Function PWM command for output A. See [Power Function PWM option constants](#).
- outb* The Power Function PWM command for output B. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFComboPWM.nxc](#).

**8.3.3.394** `char MSPFRawOutput (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2) [inline]`

MSPFRawOutput function. Control a Power Function receiver set to the specified channel using the mindsensors NRLink device. Build the raw data stream using the 3 nibbles (4 bit values). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*nibble0* The first raw data nibble.

*nibble1* The second raw data nibble.

*nibble2* The third raw data nibble.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFRawOutput.nxc](#).

**8.3.3.395** `char MSPFRepeat (const byte port, const byte i2caddr, const byte count, const unsigned int delay) [inline]`

MSPFRepeat function. Repeat sending the last Power Function command using the mindsensors NRLink device. Specify the number of times to repeat the command and the number of milliseconds of delay between each repetition. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*count* The number of times to repeat the command.

*delay* The number of milliseconds to delay between each repetition.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFRepeat.nxc](#).

**8.3.3.396** `char MSPFSingleOutputCST (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func) [inline]`

MSPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_CST\\_CLEAR1\\_CLEAR2](#), [PF\\_CST\\_SET1\\_CLEAR2](#), [PF\\_CST\\_CLEAR1\\_SET2](#), [PF\\_CST\\_SET1\\_SET2](#), [PF\\_CST\\_INCREMENT\\_PWM](#), [PF\\_CST\\_DECREMENT\\_PWM](#), [PF\\_CST\\_FULL\\_FWD](#), [PF\\_CST\\_FULL\\_REV](#), and [PF\\_CST\\_TOGGLE\\_DIR](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- out* The Power Function output. See [Power Function output constants](#).
- func* The Power Function CST function. See [Power Function CST options constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFSingleOutputCST.nxc](#).

**8.3.3.397 char MSPFSingleOutputPWM (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func) [inline]**

MSPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Valid functions are [PF\\_PWM\\_FLOAT](#), [PF\\_PWM\\_FWD1](#), [PF\\_PWM\\_FWD2](#), [PF\\_PWM\\_FWD3](#), [PF\\_PWM\\_FWD4](#), [PF\\_PWM\\_FWD5](#), [PF\\_PWM\\_FWD6](#), [PF\\_PWM\\_FWD7](#), [PF\\_PWM\\_BRAKE](#), [PF\\_PWM\\_REV7](#), [PF\\_PWM\\_REV6](#), [PF\\_PWM\\_REV5](#), [PF\\_PWM\\_REV4](#), [PF\\_PWM\\_REV3](#), [PF\\_PWM\\_REV2](#), and [PF\\_PWM\\_REV1](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The Power Function channel. See [Power Function channel constants](#).  
*out* The Power Function output. See [Power Function output constants](#).  
*func* The Power Function PWM function. See [Power Function PWM option constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFSingleOutputPWM.nxc](#).

**8.3.3.398** `char MSPFSinglePin (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont) [inline]`

MSPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF\\_OUT\\_A](#) or [PF\\_OUT\\_B](#). Select the desired pin using [PF\\_PIN\\_C1](#) or [PF\\_PIN\\_C2](#). Valid functions are [PF\\_FUNC\\_NOCHANGE](#), [PF\\_FUNC\\_CLEAR](#), [PF\\_FUNC\\_SET](#), and [PF\\_FUNC\\_TOGGLE](#). Valid channels are [PF\\_CHANNEL\\_1](#) through [PF\\_CHANNEL\\_4](#). Specify whether the mode by passing true (continuous) or false (timeout) as the final parameter. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).  
*i2caddr* The sensor I2C address. See sensor documentation for this value.  
*channel* The Power Function channel. See [Power Function channel constants](#).  
*out* The Power Function output. See [Power Function output constants](#).  
*pin* The Power Function pin. See [Power Function pin constants](#).  
*func* The Power Function single pin function. See [Power Function single pin function constants](#).  
*cont* Control whether the mode is continuous or timeout.

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFSinglePin.nxc](#).

**8.3.3.399** `char MSPFTrain (const byte port, const byte i2caddr, const byte channel, const byte func) [inline]`

MSPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device as if it were an IR Train receiver. Valid function values are [TRAIN\\_FUNC\\_STOP](#), [TRAIN\\_FUNC\\_INCR\\_SPEED](#), [TRAIN\\_FUNC\\_DECR\\_SPEED](#), and [TRAIN\\_FUNC\\_TOGGLE\\_LIGHT](#). Valid channels are PF\_CHANNEL\_1 through PF\_CHANNEL\_4. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- channel* The Power Function channel. See [Power Function channel constants](#).
- func* The Power Function train function. See [PF/IR Train function constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_MSPFTrain.nxc](#).

**8.3.3.400** `void MSRCXAbsVar (const byte varnum, const byte src, const unsigned int value) [inline]`

MSRCXAbsVar function. Send the AbsVar command to an RCX.

**Parameters:**

- varnum* The variable number to change.
- src* The RCX source. See [RCX and Scout source constants](#).
- value* The RCX value.

**Examples:**

[ex\\_MSRCXAbsVar.nxc](#).

**8.3.3.401** void MSRCXAddToDatalog (const byte *src*, const unsigned int *value*)  
[inline]

MSRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXAddToDatalog.nxc](#).

**8.3.3.402** void MSRCXAndVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXAndVar function. Send the AndVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXAndVar.nxc](#).

**8.3.3.403** int MSRCXBatteryLevel (void) [inline]

MSRCXBatteryLevel function. Send the BatteryLevel command to an RCX to read the current battery level.

**Returns:**

The RCX battery level.

**Examples:**

[ex\\_MSRCXBatteryLevel.nxc](#).

**8.3.3.404 void MSRCXBoot (void) [inline]**

MSRCXBoot function. Send the Boot command to an RCX.

**Examples:**

[ex\\_MSRCXBoot.nxc](#).

**8.3.3.405 void MSRCXCalibrateEvent (const byte *evt*, const byte *low*, const byte *hi*, const byte *hyst*) [inline]**

MSRCXCalibrateEvent function. Send the CalibrateEvent command to an RCX.

**Parameters:**

- evt* The event number.
- low* The low threshold.
- hi* The high threshold.
- hyst* The hysteresis value.

**Examples:**

[ex\\_MSRCXCalibrateEvent.nxc](#).

**8.3.3.406 void MSRCXClearAllEvents (void) [inline]**

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

**Examples:**

[ex\\_MSRCXClearAllEvents.nxc](#).

**8.3.3.407 void MSRCXClearCounter (const byte *counter*) [inline]**

MSRCXClearCounter function. Send the ClearCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

[ex\\_MSRCXClearCounter.nxc](#).

**8.3.3.408 void MSRCXClearMsg (void) [inline]**

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

[ex\\_MSRCXClearMsg.nxc](#).

**8.3.3.409 void MSRCXClearSensor (const byte port) [inline]**

MSRCXClearSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

[ex\\_MSRCXClearSensor.nxc](#).

**8.3.3.410 void MSRCXClearSound (void) [inline]**

MSRCXClearSound function. Send the ClearSound command to an RCX.

**Examples:**

[ex\\_MSRCXClearSound.nxc](#).

**8.3.3.411 void MSRCXClearTimer (const byte *timer*) [inline]**

MSRCXClearTimer function. Send the ClearTimer command to an RCX.

**Parameters:**

*timer* The timer to clear.

**Examples:**

[ex\\_MSRCXClearTimer.nxc](#).

**8.3.3.412 void MSRCXCreateDatalog (const unsigned int *size*) [inline]**

MSRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

**Parameters:**

*size* The new datalog size.

**Examples:**

[ex\\_MSRCXCreateDatalog.nxc](#).

**8.3.3.413 void MSRCXDecCounter (const byte *counter*) [inline]**

MSRCXDecCounter function. Send the DecCounter command to an RCX.

**Parameters:**

*counter* The counter to decrement.

**Examples:**

[ex\\_MSRCXDecCounter.nxc](#).

**8.3.3.414 void MSRCXDeleteSub (const byte *s*) [inline]**

MSRCXDeleteSub function. Send the DeleteSub command to an RCX.

**Parameters:**

*s* The subroutine number to delete.

**Examples:**

[ex\\_MSRCXDeleteSub.nxc](#).

**8.3.3.415 void MSRCXDeleteSubs (void) [inline]**

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

[ex\\_MSRCXDeleteSubs.nxc](#).

**8.3.3.416 void MSRCXDeleteTask (const byte *t*) [inline]**

MSRCXDeleteTask function. Send the DeleteTask command to an RCX.

**Parameters:**

*t* The task number to delete.

**Examples:**

[ex\\_MSRCXDeleteTask.nxc](#).

**8.3.3.417 void MSRCXDeleteTasks (void) [inline]**

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

[ex\\_MSRCXDeleteTasks.nxc](#).

**8.3.3.418 void MSRCXDisableOutput (const byte *outputs*) [inline]**

MSRCXDisableOutput function. Send the DisableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to disable. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXDisableOutput.nxc](#).

**8.3.3.419 void MSRCXDivVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXDivVar function. Send the DivVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXDivVar.nxc](#).

**8.3.3.420 void MSRCXEnableOutput (const byte *outputs*) [inline]**

MSRCXEnableOutput function. Send the EnableOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to enable. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXEnableOutput.nxc](#).

**8.3.3.421 void MSRCXEvent (const byte *src*, const unsigned int *value*)  
[inline]**

MSRCXEvent function. Send the Event command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXEvent.nxc](#).

**8.3.3.422 void MSRCXFloat (const byte *outputs*) [inline]**

MSRCXFloat function. Send commands to an RCX to float the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to float. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXFloat.nxc](#).

**8.3.3.423 void MSRCXFwd (const byte *outputs*) [inline]**

MSRCXFwd function. Send commands to an RCX to set the specified outputs to the forward direction.

**Parameters:**

*outputs* The RCX output(s) to set forward. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXFwd.nxc](#).

**8.3.3.424 void MSRCXIncCounter (const byte *counter*) [inline]**

MSRCXIncCounter function. Send the IncCounter command to an RCX.

**Parameters:**

*counter* The counter to increment.

**Examples:**

[ex\\_MSRCXIncCounter.nxc](#).

**8.3.3.425 void MSRCXInvertOutput (const byte *outputs*) [inline]**

MSRCXInvertOutput function. Send the InvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to invert. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXInvertOutput.nxc](#).

**8.3.3.426 void MSRCXMulVar (const byte *varnum*, const byte *src*, unsigned int *value*) [inline]**

MSRCXMulVar function. Send the MulVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXMulVar.nxc](#).

**8.3.3.427 void MSRCXMuteSound (void) [inline]**

MSRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

[ex\\_MSRCXMuteSound.nxc](#).

**8.3.3.428 void MSRCXObvertOutput (const byte *outputs*) [inline]**

MSRCXObvertOutput function. Send the ObvertOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to obvert. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXObvertOutput.nxc](#).

**8.3.3.429 void MSRCXOff (const byte *outputs*) [inline]**

MSRCXOff function. Send commands to an RCX to turn off the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn off. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOff.nxc](#).

**8.3.3.430 void MSRCXOn (const byte *outputs*) [inline]**

MSRCXOn function. Send commands to an RCX to turn on the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOn.nxc](#).

**8.3.3.431 void MSRCXOnFor (const byte *outputs*, const unsigned int *ms*)  
[inline]**

MSRCXOnFor function. Send commands to an RCX to turn on the specified outputs in the forward direction for the specified duration.

**Parameters:**

*outputs* The RCX output(s) to turn on. See [RCX output constants](#).

*ms* The number of milliseconds to leave the outputs on

**Examples:**

[ex\\_MSRCXOnFor.nxc](#).

**8.3.3.432 void MSRCXOnFwd (const byte *outputs*) [inline]**

MSRCXOnFwd function. Send commands to an RCX to turn on the specified outputs in the forward direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the forward direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOnFwd.nxc](#).

**8.3.3.433 void MSRCXOnRev (const byte *outputs*) [inline]**

MSRCXOnRev function. Send commands to an RCX to turn on the specified outputs in the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to turn on in the reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXOnRev.nxc](#).

**8.3.3.434** void MSRCXOrVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [**inline**]

MSRCXOrVar function. Send the OrVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXOrVar.nxc](#).

**8.3.3.435** void MSRCXPBTurnOff (void) [**inline**]

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

[ex\\_MSRCXPBTurnOff.nxc](#).

**8.3.3.436** void MSRCXPing (void) [**inline**]

MSRCXPing function. Send the Ping command to an RCX.

**Examples:**

[ex\\_MSRCXPing.nxc](#).

**8.3.3.437 void MSRCXPlaySound (const byte *snd*) [inline]**

MSRCXPlaySound function. Send the PlaySound command to an RCX.

**Parameters:**

*snd* The sound number to play.

**Examples:**

[ex\\_MSRCXPlaySound.nxc](#).

**8.3.3.438 void MSRCXPlayTone (const unsigned int *freq*, const byte *duration*) [inline]**

MSRCXPlayTone function. Send the PlayTone command to an RCX.

**Parameters:**

*freq* The frequency of the tone to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_MSRCXPlayTone.nxc](#).

**8.3.3.439 void MSRCXPlayToneVar (const byte *varnum*, const byte *duration*) [inline]**

MSRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

**Parameters:**

*varnum* The variable containing the tone frequency to play.

*duration* The duration of the tone to play.

**Examples:**

[ex\\_MSRCXPlayToneVar.nxc](#).

**8.3.3.440 int MSRCXPoll (const byte *src*, const byte *value*) [inline]**

MSRCXPoll function. Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Returns:**

The value read from the specified port and value.

**Examples:**

[ex\\_MSRCXPoll.nxc](#).

**8.3.3.441 int MSRCXPollMemory (const unsigned int *address*) [inline]**

MSRCXPollMemory function. Send the PollMemory command to an RCX.

**Parameters:**

*address* The RCX memory address.

**Returns:**

The value read from the specified address.

**Examples:**

[ex\\_MSRCXPollMemory.nxc](#).

**8.3.3.442 void MSRCXRemote (unsigned int *cmd*) [inline]**

MSRCXRemote function. Send the Remote command to an RCX.

**Parameters:**

*cmd* The RCX IR remote command to send. See [RCX IR remote constants](#).

**Examples:**

[ex\\_MSRCXRemote.nxc](#).

**8.3.3.443 void MSRCXReset (void) [inline]**

MSRCXReset function. Send the Reset command to an RCX.

**Examples:**

[ex\\_MSRCXReset.nxc](#).

**8.3.3.444 void MSRCXRev (const byte *outputs*) [inline]**

MSRCXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

**Parameters:**

*outputs* The RCX output(s) to reverse direction. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXRev.nxc](#).

**8.3.3.445 void MSRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]**

MSRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSelectDisplay.nxc](#).

**8.3.3.446 void MSRCXSelectProgram (const byte *prog*) [inline]**

MSRCXSelectProgram function. Send the SelectProgram command to an RCX.

**Parameters:**

*prog* The program number to select.

**Examples:**

[ex\\_MSRCXSelectProgram.nxc](#).

**8.3.3.447 void MSRCXSendSerial (const byte *first*, const byte *count*) [inline]**

MSRCXSendSerial function. Send the SendSerial command to an RCX.

**Parameters:**

*first* The first byte address.

*count* The number of bytes to send.

**Examples:**

[ex\\_MSRCXSendSerial.nxc](#).

**8.3.3.448 void MSRCXSet (const byte *dstsrc*, const byte *dstval*, const byte *src*, unsigned int *value*) [inline]**

MSRCXSet function. Send the Set command to an RCX.

**Parameters:**

*dstsrc* The RCX destination source. See [RCX and Scout source constants](#).

*dstval* The RCX destination value.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSet.nxc](#).

**8.3.3.449** void MSRCXSetDirection (const byte *outputs*, const byte *dir*)  
[inline]

MSRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_MSRCXSetDirection.nxc](#).

**8.3.3.450** void MSRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)  
[inline]

MSRCXSetEvent function. Send the SetEvent command to an RCX.

**Parameters:**

*evt* The event number to set.

*src* The RCX source. See [RCX and Scout source constants](#).

*type* The event type.

**Examples:**

[ex\\_MSRCXSetEvent.nxc](#).

**8.3.3.451** void MSRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)  
[inline]

MSRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global direction. See [RCX output constants](#).

*dir* The RCX output direction. See [RCX output direction constants](#).

**Examples:**

[ex\\_MSRCXSetGlobalDirection.nxc](#).

**8.3.3.452** void MSRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)  
[inline]

MSRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set global mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_MSRCXSetGlobalOutput.nxc](#).

**8.3.3.453** void MSRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*,  
const byte *pwrval*) [inline]

MSRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

**Parameters:**

*outputs* The RCX output(s) to set max power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_MSRCXSetMaxPower.nxc](#).

**8.3.3.454** void MSRCXSetMessage (const byte *msg*) [inline]

MSRCXSetMessage function. Send the SetMessage command to an RCX.

**Parameters:**

*msg* The numeric message to send.

**Examples:**

[ex\\_MSRCXSetMessage.nxc](#).

**8.3.3.455 void MSRCXSetNRLinkPort (const byte *port*, const byte *i2caddr*)  
[inline]**

MSRCXSetNRLinkPort function. Set the global port in advance of using the MSRCX\* and MSScout\* API functions for sending RCX and Scout messages over the mindsensors NRLink device. The port must be configured as a Low-speed port before using any of the mindsensors RCX and Scout NRLink functions.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Examples:**

[ex\\_MSRCXSetNRLinkPort.nxc](#).

**8.3.3.456 void MSRCXSetOutput (const byte *outputs*, const byte *mode*)  
[inline]**

MSRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

**Parameters:**

*outputs* The RCX output(s) to set mode. See [RCX output constants](#).

*mode* The RCX output mode. See [RCX output mode constants](#).

**Examples:**

[ex\\_MSRCXSetOutput.nxc](#).

**8.3.3.457** void MSRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]

MSRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to set power. See [RCX output constants](#).

*pwrsrc* The RCX source. See [RCX and Scout source constants](#).

*pwrval* The RCX value.

**Examples:**

[ex\\_MSRCXSetPower.nxc](#).

**8.3.3.458** void MSRCXSetPriority (const byte *p*) [inline]

MSRCXSetPriority function. Send the SetPriority command to an RCX.

**Parameters:**

*p* The new task priority.

**Examples:**

[ex\\_MSRCXSetPriority.nxc](#).

**8.3.3.459** void MSRCXSetSensorMode (const byte *port*, const byte *mode*) [inline]

MSRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

**Parameters:**

*port* The RCX sensor port.

*mode* The RCX sensor mode.

**Examples:**

[ex\\_MSRCXSetSensorMode.nxc](#).

**8.3.3.460** void MSRCXSetSensorType (const byte *port*, const byte *type*)  
[inline]

MSRCXSetSensorType function. Send the SetSensorType command to an RCX.

**Parameters:**

*port* The RCX sensor port.

*type* The RCX sensor type.

**Examples:**

[ex\\_MSRCXSetSensorType.nxc](#).

**8.3.3.461** void MSRCXSetSleepTime (const byte *t*) [inline]

MSRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

**Parameters:**

*t* The new sleep time value.

**Examples:**

[ex\\_MSRCXSetSleepTime.nxc](#).

**8.3.3.462** void MSRCXSetTxPower (const byte *pwr*) [inline]

MSRCXSetTxPower function. Send the SetTxPower command to an RCX.

**Parameters:**

*pwr* The IR transmit power level.

**Examples:**

[ex\\_MSRCXSetTxPower.nxc](#).

**8.3.3.463** void MSRCXSetUserDisplay (const byte *src*, const unsigned int *value*, const byte *precision*) [**inline**]

MSRCXSetUserDisplay function. Send the SetUserDisplay command to an RCX.

**Parameters:**

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

*precision* The number of digits of precision.

**Examples:**

[ex\\_MSRCXSetUserDisplay.nxc](#).

**8.3.3.464** void MSRCXSetVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [**inline**]

MSRCXSetVar function. Send the SetVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSetVar.nxc](#).

**8.3.3.465** void MSRCXSetWatch (const byte *hours*, const byte *minutes*) [**inline**]

MSRCXSetWatch function. Send the SetWatch command to an RCX.

**Parameters:**

*hours* The new watch time hours value.

*minutes* The new watch time minutes value.

**Examples:**

[ex\\_MSRCXSetWatch.nxc](#).

**8.3.3.466** void MSRCXSgnVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [**inline**]

MSRCXSgnVar function. Send the SgnVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSgnVar.nxc](#).

**8.3.3.467** void MSRCXStartTask (const byte *t*) [**inline**]

MSRCXStartTask function. Send the StartTask command to an RCX.

**Parameters:**

*t* The task number to start.

**Examples:**

[ex\\_MSRCXStartTask.nxc](#).

**8.3.3.468** void MSRCXStopAllTasks (void) [**inline**]

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

[ex\\_MSRCXStopAllTasks.nxc](#).

**8.3.3.469 void MSRCXStopTask (const byte *t*) [inline]**

MSRCXStopTask function. Send the StopTask command to an RCX.

**Parameters:**

*t* The task number to stop.

**Examples:**

[ex\\_MSRCXStopTask.nxc](#).

**8.3.3.470 void MSRCXSubVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXSubVar function. Send the SubVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSubVar.nxc](#).

**8.3.3.471 void MSRCXSumVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]**

MSRCXSumVar function. Send the SumVar command to an RCX.

**Parameters:**

*varnum* The variable number to change.

*src* The RCX source. See [RCX and Scout source constants](#).

*value* The RCX value.

**Examples:**

[ex\\_MSRCXSumVar.nxc](#).

**8.3.3.472 void MSRCXToggle (const byte *outputs*) [inline]**

MSRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

**Parameters:**

*outputs* The RCX output(s) to toggle. See [RCX output constants](#).

**Examples:**

[ex\\_MSRCXToggle.nxc](#).

**8.3.3.473 void MSRCXUnlock (void) [inline]**

MSRCXUnlock function. Send the Unlock command to an RCX.

**Examples:**

[ex\\_MSRCXUnlock.nxc](#).

**8.3.3.474 void MSRCXUnmuteSound (void) [inline]**

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

[ex\\_MSRCXUnmuteSound.nxc](#).

**8.3.3.475 int MSReadValue (const byte *port*, const byte *i2caddr*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one, two, or four byte value from a mind-sensors sensor. The value must be stored with the least significant byte (LSB) first (i.e., little endian). Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*reg* The device register to read.

*numbytes* The number of bytes to read. Only 1, 2 or 4 byte values are supported.

**Returns:**

The function call result.

**Examples:**

[ex\\_MSReadValue.nxc](#).

**8.3.3.476 void MSScoutCalibrateSensor (void) [inline]**

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

[ex\\_MSScoutCalibrateSensor.nxc](#).

**8.3.3.477 void MSScoutMuteSound (void) [inline]**

MSScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

[ex\\_MSScoutMuteSound.nxc](#).

**8.3.3.478 void MSScoutSelectSounds (const byte grp) [inline]**

MSScoutSelectSounds function. Send the SelectSounds command to a Scout.

**Parameters:**

*grp* The Scout sound group to select.

**Examples:**

[ex\\_MSScoutSelectSounds.nxc](#).

**8.3.3.479** void MSScoutSendVLL (const byte *src*, const unsigned int *value*)  
[inline]

MSScoutSendVLL function. Send the SendVLL command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSendVLL.nxc](#).

**8.3.3.480** void MSScoutSetCounterLimit (const byte *ctr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetCounterLimit function. Send the SetCounterLimit command to a Scout.

**Parameters:**

*ctr* The counter for which to set the limit.

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetCounterLimit.nxc](#).

**8.3.3.481** void MSScoutSetEventFeedback (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetEventFeedback.nxc](#).

**8.3.3.482 void MSScoutSetLight (const byte *x*) [inline]**

MSScoutSetLight function. Send the SetLight command to a Scout.

**Parameters:**

*x* Set the light on or off using this value. See [Scout light constants](#).

**Examples:**

[ex\\_MSScoutSetLight.nxc](#).

**8.3.3.483 void MSScoutSetScoutMode (const byte *mode*) [inline]**

MSScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

**Parameters:**

*mode* Set the scout mode. See [Scout mode constants](#).

**Examples:**

[ex\\_MSScoutSetScoutMode.nxc](#).

**8.3.3.484 void MSScoutSetScoutRules (const byte *m*, const byte *t*, const byte *l*, const byte *tm*, const byte *fx*) [inline]**

MSScoutSetScoutRules function. Send the SetScoutRules command to a Scout.

**Parameters:**

*m* Scout motion rule. See [Scout motion rule constants](#).

*t* Scout touch rule. See [Scout touch rule constants](#).

*l* Scout light rule. See [Scout light rule constants](#).

*tm* Scout transmit rule. See [Scout transmit rule constants](#).

*fx* Scout special effects rule. See [Scout special effect constants](#).

**Examples:**

[ex\\_MSScoutSetScoutRules.nxc](#).

**8.3.3.485** void MSScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorClickTime.nxc](#).

**8.3.3.486** void MSScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorHysteresis.nxc](#).

**8.3.3.487** void MSScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorLowerLimit.nxc](#).

**8.3.3.488** void MSScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

**Parameters:**

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetSensorUpperLimit.nxc](#).

**8.3.3.489** void MSScoutSetTimerLimit (const byte *tmr*, const byte *src*, const unsigned int *value*) [**inline**]

MSScoutSetTimerLimit function. Send the SetTimerLimit command to a Scout.

**Parameters:**

*tmr* The timer for which to set a limit.

*src* The Scout source. See [RCX and Scout source constants](#).

*value* The Scout value.

**Examples:**

[ex\\_MSScoutSetTimerLimit.nxc](#).

**8.3.3.490** void MSScoutUnmuteSound (void) [**inline**]

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

[ex\\_MSScoutUnmuteSound.nxc](#).

**8.3.3.491 long muldiv32 (long *a*, long *b*, long *c*) [inline]**

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value.

**Parameters:**

- a* 32-bit long value.
- b* 32-bit long value.
- c* 32-bit long value.

**Returns:**

The result of multiplying *a* times *b* and dividing by *c*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_muldiv32.nxc](#).

**8.3.3.492 char NRLink2400 (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in 2400 baud mode. Configure the mindsensors NRLink device in 2400 baud mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400.nxc](#).

**8.3.3.493 char NRLink4800 (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in 4800 baud mode. Configure the mindsensors NRLink device in 4800 baud mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800.nxc](#).

**8.3.3.494 char NRLinkFlush (const byte *port*, const byte *i2caddr*) [inline]**

Flush NRLink buffers. Flush the mindsensors NRLink device buffers. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkFlush.nxc](#).

**8.3.3.495 char NRLinkIRLong (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in IR long mode. Configure the mindsensors NRLink device in IR long mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkIRLong.nxc](#).

**8.3.3.496 char NRLinkIRShort (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in IR short mode. Configure the mindsensors NRLink device in IR short mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkIRShort.nxc](#).

**8.3.3.497 char NRLinkSetPF (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in power function mode. Configure the mindsensors NRLink device in power function mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetPF.nxc](#).

**8.3.3.498 char NRLinkSetRCX (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in RCX mode. Configure the mindsensors NRLink device in RCX mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetRCX.nxc](#).

**8.3.3.499 char NRLinkSetTrain (const byte *port*, const byte *i2caddr*)  
[inline]**

Configure NRLink in IR train mode. Configure the mindsensors NRLink device in IR train mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkSetTrain.nxc](#).

**8.3.3.500 byte NRLinkStatus (const byte *port*, const byte *i2caddr*) [inline]**

Read NRLink status. Read the status of the mindsensors NRLink device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors NRLink status.

**Examples:**

[ex\\_NRLinkStatus.nxc](#).

**8.3.3.501 char NRLinkTxRaw (const byte *port*, const byte *i2caddr*) [inline]**

Configure NRLink in raw IR transmit mode. Configure the mindsensors NRLink device in raw IR transmit mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLinkTxRaw.nxc](#).

**8.3.3.502 char NumOut (int *x*, int *y*, variant *value*, unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw a number. Draw a numeric value on the screen at the specified x and y location. The y value must be a multiple of 8. Valid line number constants are listed in the [Line number constants](#) group. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawText](#), [DrawTextType](#)

**Parameters:**

- x* The x value for the start of the number output.
- y* The text line number for the number output.
- value* The value to output to the LCD screen. Any numeric type is supported.
- options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_ArrayBuild.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_buttonpressed.nxc](#), [ex\\_contrast.nxc](#), [ex\\_ctype.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getchar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_motoroutputoptions.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_Pos.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_SizeOf.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_superpro.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_SysReadLastResponse.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), [ex\\_SysWriteSemData.nxc](#), [ex\\_UnflattenVar.nxc](#), and [ex\\_xg1300.nxc](#).

**8.3.3.503 string NumToStr (variant *num*) [inline]**

Convert number to string. Return the string representation of the specified numeric value.

**Parameters:**

*num* A number.

**Returns:**

The string representation of the parameter *num*.

**Examples:**

[ex\\_NumToStr.nxc](#), [ex\\_RS485Send.nxc](#), and [ex\\_string.nxc](#).

**8.3.3.504 char NXTHIDAsciiMode (const byte & *port*, const byte & *i2caddr*)  
[inline]**

Set NXTHID into ASCII data mode. Set the NXTHID device into ASCII data mode. Only printable characters can be transmitted in this mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.3.3.505 char NXTHIDDirectMode (const byte & *port*, const byte & *i2caddr*)  
[inline]**

Set NXTHID into direct data mode. Set the NXTHID device into direct data mode. Any character can be transmitted while in this mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.3.3.506 char NXTHIDLoadCharacter (const byte & port, const byte & i2caddr, const byte & modifier, const byte & character) [inline]**

Load NXTHID character. Load a character into the NXTHID device. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*modifier* The key modifier. See the [MindSensors NXTHID modifier keys](#) group.

*character* The character.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.3.3.507 char NXTHIDTransmit (const byte & port, const byte & i2caddr) [inline]**

Transmit NXTHID character. Transmit a single character to a computer using the NXTHID device. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTHID.nxc](#).

**8.3.3.508 char NXTLineLeaderAverage (const byte & port, const byte & i2caddr) [inline]**

Read NXTLineLeader average. Read the mindsensors NXTLineLeader device's average value. The average is a weighted average of the bits set to 1 based on the position. The left most bit has a weight of 10, second bit has a weight of 20, and so forth. When all 8 sensors are over a black surface the average will be 45. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader average value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.509 char NXTLineLeaderCalibrateBlack (const byte & port, const byte & i2caddr) [inline]**

Calibrate NXTLineLeader black color. Store calibration data for the black color. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.510 char NXTLineLeaderCalibrateWhite (const byte & *port*, const byte & *i2caddr*) [inline]**

Calibrate NXTLineLeader white color. Store calibration data for the white color. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.511 char NXTLineLeaderInvert (const byte & *port*, const byte & *i2caddr*) [inline]**

Invert NXTLineLeader colors. Invert color sensing so that the device can detect a white line on a black background. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.512 char NXTLineLeaderPowerDown (const byte & *port*, const byte & *i2caddr*) [inline]**

Powerdown NXTLineLeader device. Put the NXTLineLeader to sleep so that it does not consume power when it is not required. The device wakes up on its own when any I2C communication happens or you can specifically wake it up by using the [NXTLineLeaderPowerUp](#) command. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.513 char NXTLineLeaderPowerUp (const byte & *port*, const byte & *i2caddr*) [inline]**

Powerup NXTLineLeader device. Wake up the NXTLineLeader device so that it can be used. The device can be put to sleep using the [NXTLineLeaderPowerDown](#) command. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.514 char NXTLineLeaderReset (const byte & port, const byte & i2caddr) [inline]**

Reset NXTLineLeader color inversion. Reset the NXTLineLeader color detection back to its default state (black line on a white background). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.515 byte NXTLineLeaderResult (const byte & port, const byte & i2caddr) [inline]**

Read NXTLineLeader result. Read the mindsensors NXTLineLeader device's result value. This is a single byte showing the 8 sensor's readings. Each bit corresponding to the sensor where the line is seen is set to 1, otherwise it is set to 0. When all 8 sensors are over a black surface the result will be 255 (b11111111). The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader result value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.516 char NXTLineLeaderSnapshot (const byte & *port*, const byte & *i2caddr*) [inline]**

Take NXTLineLeader line snapshot. Takes a snapshot of the line under the sensor and tracks that position in subsequent tracking operations. This function also will set color inversion if it sees a white line on a black background. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.517 char NXTLineLeaderSteering (const byte & *port*, const byte & *i2caddr*) [inline]**

Read NXTLineLeader steering. Read the mindsensors NXTLineLeader device's steering value. This is the power returned by the sensor to correct your course. Add this value to your left motor and subtract it from your right motor. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTLineLeader steering value.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.518 int NXTPowerMeterCapacityUsed (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter capacity used. Read the mindsensors NXTPowerMeter device's capacity used since the last reset command. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter capacity used value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.519 long NXTPowerMeterElapsedTime (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter elapsed time. Read the mindsensors NXTPowerMeter device's elapsed time since the last reset command. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter elapsed time value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.520 int NXTPowerMeterErrorCount (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter error count. Read the mindsensors NXTPowerMeter device's error count value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter error count value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.521 int NXTPowerMeterMaxCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter maximum current. Read the mindsensors NXTPowerMeter device's maximum current value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter maximum current value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.522 int NXTPowerMeterMaxVoltage (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter maximum voltage. Read the mindsensors NXTPowerMeter device's maximum voltage value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter maximum voltage value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.523 int NXTPowerMeterMinCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter minimum current. Read the mindsensors NXTPowerMeter device's minimum current value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter minimum current value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.524 int NXTPowerMeterMinVoltage (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter minimum voltage. Read the mindsensors NXTPowerMeter device's minimum voltage value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter minimum voltage value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.525 int NXTPowerMeterPresentCurrent (const byte & port, const byte & i2caddr) [inline]**

Read NXTPowerMeter present current. Read the mindsensors NXTPowerMeter device's present current value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present current.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.526 int NXTPowerMeterPresentPower (const byte & *port*, const byte & *i2caddr*) [inline]**

Read NXTPowerMeter present power. Read the mindsensors NXTPowerMeter device's present power value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present power value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.527 int NXTPowerMeterPresentVoltage (const byte & *port*, const byte & *i2caddr*) [inline]**

Read NXTPowerMeter present voltage. Read the mindsensors NXTPowerMeter device's present voltage value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter present voltage.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.528** `char NXTPowerMeterResetCounters (const byte & port, const byte & i2caddr) [inline]`

Reset NXTPowerMeter counters. Reset the NXTPowerMeter counters back to zero. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

**8.3.3.529** `long NXTPowerMeterTotalPowerConsumed (const byte & port, const byte & i2caddr) [inline]`

Read NXTPowerMeter total power consumed. Read the mindsensors NXTPowerMeter device's total power consumed since the last reset command. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The NXTPowerMeter total power consumed value.

**Examples:**

[ex\\_NXTPowerMeter.nxc](#).

### 8.3.3.530 byte NXTServoBatteryVoltage (const byte & *port*, const byte & *i2caddr*) [**inline**]

Read NXTServo battery voltage value. Read the mindsensors NXTServo device's battery voltage value. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The battery level.

#### Examples:

[ex\\_NXTServo.nxc](#).

### 8.3.3.531 char NXTServoEditMacro (const byte & *port*, const byte & *i2caddr*) [**inline**]

Edit NXTServo macro. Put the NXTServo device into macro edit mode. This operation changes the I2C address of the device to 0x40. Macros are written to EEPROM addresses between 0x21 and 0xFF. Use [NXTServoQuitEdit](#) to return the device to its normal operation mode. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

#### Examples:

[ex\\_NXTServo.nxc](#).

**8.3.3.532 char NXTServoGotoMacroAddress (const byte & *port*, const byte & *i2caddr*, const byte & *macro*) [inline]**

Goto NXTServo macro address. Run the macro found at the specified EEPROM macro address. This command re-initializes the macro environment. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*macro* The EEPROM macro address.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.533 char NXTServoHaltMacro (const byte & *port*, const byte & *i2caddr*) [inline]**

Halt NXTServo macro. Halt a macro executing on the NXTServo device. This command re-initializes the macro environment. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.534 char NXTServoInit (const byte & *port*, const byte & *i2caddr*, const byte *servo*) [inline]**

Initialize NXTServo servo properties. Store the initial speed and position properties of the servo motor 'n'. Current speed and position values of the nth servo is read from the servo speed register and servo position register and written to permanent memory. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.535 char NXTServoPauseMacro (const byte & *port*, const byte & *i2caddr*) [inline]**

Pause NXTServo macro. Pause a macro executing on the NXTServo device. This command will pause the currently executing macro, and save the environment for subsequent resumption. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.536 unsigned int NXTServoPosition (const byte & *port*, const byte & *i2caddr*, const byte *servo*) [inline]**

Read NXTServo servo position value. Read the mindsensors NXTServo device's servo position value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

The specified servo's position value.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.537 char NXTServoQuitEdit (const byte & *port*) [inline]**

Quit NXTServo macro edit mode. Stop editing NXTServo device macro EEPROM memory. Use [NXTServoEditMacro](#) to start editing a macro. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

### 8.3.3.538 char NXTServoReset (const byte & *port*, const byte & *i2caddr*) [inline]

Reset NXTServo properties. Reset NXTServo device properties to factory defaults. Initial position = 1500. Initial speed = 0. The port must be configured as a Low-speed port before using this function.

#### Parameters:

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

#### Examples:

[ex\\_NXTServo.nxc](#).

### 8.3.3.539 char NXTServoResumeMacro (const byte & *port*, const byte & *i2caddr*) [inline]

Resume NXTServo macro. Resume a macro executing on the NXTServo device. This command resumes executing a macro where it was paused last, using the same environment. The port must be configured as a Low-speed port before using this function.

#### Parameters:

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

#### Examples:

[ex\\_NXTServo.nxc](#).

**8.3.3.540** `byte NXTServoSpeed (const byte & port, const byte & i2caddr, const byte servo) [inline]`

Read NXTServo servo speed value. Read the mindsensors NXTServo device's servo speed value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

**Returns:**

The specified servo's speed value.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.541** `void Off (byte outputs) [inline]`

Turn motors off. Turn the specified outputs off (with braking).

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

**Examples:**

[ex\\_off.nxc](#).

**8.3.3.542** `void OffEx (byte outputs, const byte reset) [inline]`

Turn motors off and reset counters. Turn the specified outputs off (with braking).

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_offex.nxc](#).

**8.3.3.543 byte OnBrickProgramPointer (void) [inline]**

Read the on brick program pointer value. Return the current OBP (on-brick program) step

**Returns:**

On brick program pointer (step).

**Examples:**

[ex\\_OnBrickProgramPointer.nxc](#).

**8.3.3.544 void OnFwd (byte outputs, char pwr) [inline]**

Run motors forward. Set outputs to forward direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

**Examples:**

[ex\\_onfwd.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

**8.3.3.545 void OnFwdEx (byte *outputs*, char *pwr*, const byte *reset*) [inline]**

Run motors forward and reset counters. Set outputs to forward direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdex.nxc](#).

**8.3.3.546 void OnFwdReg (byte *outputs*, char *pwr*, byte *regmode*) [inline]**

Run motors forward regulated. Run the specified outputs forward using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

**Examples:**

[ex\\_onfwdreg.nxc](#).

**8.3.3.547 void OnFwdRegEx (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*) [inline]**

Run motors forward regulated and reset counters. Run the specified outputs forward using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdregex.nxc](#).

**8.3.3.548** `void OnFwdRegExPID (byte outputs, char pwr, byte regmode, const byte reset, byte p, byte i, byte d) [inline]`

Run motors forward regulated and reset counters with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdregexpid.nxc](#).

**8.3.3.549** void OnFwdRegPID (byte *outputs*, char *pwr*, byte *regmode*, byte *p*, byte *i*, byte *d*) [**inline**]

Run motors forward regulated with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdregpid.nxc](#).

**8.3.3.550** void OnFwdSync (byte *outputs*, char *pwr*, char *turnpct*) [**inline**]

Run motors forward synchronised. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

**Examples:**

[ex\\_onfwdsync.nxc](#).

**8.3.3.551** `void OnFwdSyncEx (byte outputs, char pwr, char turnpct, const byte reset) [inline]`

Run motors forward synchronised and reset counters. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onfwdsyncex.nxc](#).

**8.3.3.552** `void OnFwdSyncExPID (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d) [inline]`

Run motors forward synchronised and reset counters with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a

single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

#### Examples:

[ex\\_onfwdsyncexpid.nxc](#).

#### 8.3.3.553 void OnFwdSyncPID (byte *outputs*, char *pwr*, char *turnpct*, byte *p*, byte *i*, byte *d*) [*inline*]

Run motors forward synchronised with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onfwdsyncpid.nxc](#).

**8.3.3.554 void OnRev (byte *outputs*, char *pwr*) [inline]**

Run motors backward. Set outputs to reverse direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

**Examples:**

[ex\\_onrev.nxc](#).

**8.3.3.555 void OnRevEx (byte *outputs*, char *pwr*, const byte *reset*) [inline]**

Run motors backward and reset counters. Set outputs to reverse direction and turn them on.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onrevex.nxc](#).

**8.3.3.556 void OnRevReg (byte *outputs*, char *pwr*, byte *regmode*) [inline]**

Run motors forward regulated. Run the specified outputs in reverse using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

**Examples:**

[ex\\_onrevreg.nxc](#).

**8.3.3.557 void OnRevRegEx (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*) [inline]**

Run motors backward regulated and reset counters. Run the specified outputs in reverse using the specified regulation mode.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onrevregex.nxc](#).

**8.3.3.558** void OnRevRegExPID (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*, byte *p*, byte *i*, byte *d*) [`inline`]

Run motors backward regulated and reset counters with PID factors. Run the specified outputs in reverse using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*regmode* Regulation mode, see [Output port regulation mode constants](#).

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevregexpid.nxc](#).

**8.3.3.559** void OnRevRegPID (byte *outputs*, char *pwr*, byte *regmode*, byte *p*, byte *i*, byte *d*) [`inline`]

Run motors reverse regulated with PID factors. Run the specified outputs in reverse using the specified regulation mode. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- regmode* Regulation mode, see [Output port regulation mode constants](#).
- p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevregpid.nxc](#).

**8.3.3.560 void OnRevSync (byte *outputs*, char *pwr*, char *turnpct*) [inline]**

Run motors backward synchronised. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

**Examples:**

[ex\\_onrevsync.nxc](#).

**8.3.3.561 void OnRevSyncEx (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*) [inline]**

Run motors backward synchronised and reset counters. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

**Examples:**

[ex\\_onrevsyncex.nxc](#).

**8.3.3.562** `void OnRevSyncExPID (byte outputs, char pwr, char turnpct, const byte reset, byte p, byte i, byte d) [inline]`

Run motors backward synchronised and reset counters with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*reset* Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_onrevsyncexpid.nxc](#).

### 8.3.3.563 void OnRevSyncPID (byte *outputs*, char *pwr*, char *turnpct*, byte *p*, byte *i*, byte *d*) [inline]

Run motors backward synchronised with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

#### Examples:

[ex\\_onrevsyncpid.nxc](#).

### 8.3.3.564 unsigned int OpenFileAppend (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]

Open a file for appending. Open an existing file with the specified filename for writing. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

#### Parameters:

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_OpenFileAppend.nxc](#).

**8.3.3.565 unsigned int OpenFileRead (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]**

Open a file for reading. Open an existing file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_OpenFileRead.nxc](#).

**8.3.3.566 unsigned int OpenFileReadLinear (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]**

Open a linear file for reading. Open an existing linear file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to open.

*fsize* The size of the file returned by the function.

*handle* The file handle output from the function call.

**Returns:**

The function call result. See [Loader module error codes](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_OpenFileReadLinear.nxc](#).

**8.3.3.567** `bool PFMateSend (const byte & port, const byte & i2caddr, const byte & channel, const byte & motors, const byte & cmdA, const byte & spdA, const byte & cmdB, const byte & spdB) [inline]`

Send PFMate command. Send a PFMate command to the power function IR receiver. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The power function IR receiver channel. See the [PFMate channel constants](#) group.

*motors* The motor(s) to control. See the [PFMate motor constants](#) group.

*cmdA* The power function command for motor A.

*spdA* The power function speed for motor A.

*cmdB* The power function command for motor B.

*spdB* The power function speed for motor B.

**Returns:**

The function call result.

**Examples:**

[ex\\_PFMate.nxc](#).

**8.3.3.568** `bool PFMateSendRaw (const byte & port, const byte & i2caddr,  
const byte & channel, const byte & b1, const byte & b2) [inline]`

Send raw PFMate command. Send a raw PFMate command to the power function IR receiver. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*channel* The power function IR receiver channel. See the [PFMate channel constants](#) group.

*b1* Raw byte 1.

*b2* Raw byte 2.

**Returns:**

The function call result.

**Examples:**

[ex\\_PFMate.nxc](#).

**8.3.3.569** `char PlayFile (string filename) [inline]`

Play a file. Play the specified file. The filename may be any valid string expression. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

**Parameters:**

*filename* The name of the sound or melody file to play.

**Examples:**

[ex\\_PlayFile.nxc](#).

### 8.3.3.570 char PlayFileEx (string *filename*, byte *volume*, bool *loop*) [inline]

Play a file with extra options. Play the specified file. The filename may be any valid string expression. Volume should be a number from 0 (silent) to 4 (loudest). Play the file repeatedly if loop is true. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

#### Parameters:

*filename* The name of the sound or melody file to play.

*volume* The desired tone volume.

*loop* A boolean flag indicating whether to play the file repeatedly.

#### Examples:

[ex\\_PlayFileEx.nxc](#).

### 8.3.3.571 void PlaySound (const int & *aCode*)

Play a system sound. Play a sound that mimics the RCX system sounds using one of the [RCX and Scout sound constants](#).

| aCode                             | Resulting Sound |
|-----------------------------------|-----------------|
| <a href="#">SOUND_CLICK</a>       | key click sound |
| <a href="#">SOUND_DOUBLE_BEEP</a> | double beep     |
| <a href="#">SOUND_DOWN</a>        | sweep down      |
| <a href="#">SOUND_UP</a>          | sweep up        |
| <a href="#">SOUND_LOW_BEEP</a>    | error sound     |
| <a href="#">SOUND_FAST_UP</a>     | fast sweep up   |

#### Parameters:

*aCode* The system sound to play. See [RCX and Scout sound constants](#).

#### Examples:

[ex\\_playsound.nxc](#).

**8.3.3.572 char PlayTone (unsigned int *frequency*, unsigned int *duration*)  
[inline]**

Play a tone. Play a single tone of the specified frequency and duration. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). The tone is played at the loudest sound level supported by the firmware and it is not looped.

**Parameters:**

*frequency* The desired tone frequency, in Hz.

*duration* The desired tone duration, in ms.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_PlayTone.nxc](#), and [ex\\_yield.nxc](#).

**8.3.3.573 char PlayToneEx (unsigned int *frequency*, unsigned int *duration*,  
byte *volume*, bool *loop*) [inline]**

Play a tone with extra options. Play a single tone of the specified frequency, duration, and volume. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). Volume should be a number from 0 (silent) to 4 (loudest). Play the tone repeatedly if loop is true.

**Parameters:**

*frequency* The desired tone frequency, in Hz.

*duration* The desired tone duration, in ms.

*volume* The desired tone volume.

*loop* A boolean flag indicating whether to play the tone repeatedly.

**Examples:**

[ex\\_PlayToneEx.nxc](#).

**8.3.3.574 void PlayTones (Tone *tones*[ ])**

Play multiple tones. Play a series of tones contained in the tones array. Each element in the array is an instance of the [Tone](#) structure, containing a frequency and a duration.

**Parameters:**

*tones* The array of tones to play.

**Examples:**

[ex\\_playtones.nxc](#).

**8.3.3.575** `char PointOut (int x, int y, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a point. This function lets you draw a point on the screen at x, y. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawPoint](#), [DrawPointType](#)

**Parameters:**

*x* The x value for the point.  
*y* The y value for the point.  
*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_PointOut.nxc](#), [ex\\_sin\\_cos.nxc](#), and [ex\\_sind\\_cosd.nxc](#).

**8.3.3.576** `char PolyOut (LocationType points[], unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a polygon. This function lets you draw a polygon on the screen using an array of points. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawPolygon](#), [DrawPolygonType](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*points* An array of [LocationType](#) points that define the polygon.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_PolyOut.nxc](#).

**8.3.3.577 int Pos (string Substr, string S) [inline]**

Find substring position. Returns the index value of the first character in a specified substring that occurs in a given string. Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos is case-sensitive. If Substr is not found, Pos returns negative one.

**Parameters:**

*Substr* A substring to search for in another string.

*S* A string that might contain the specified substring.

**Returns:**

The position of the substring in the specified string or -1 if it is not found.

**Examples:**

[ex\\_Pos.nxc](#).

**8.3.3.578 void PosRegAddAngle (byte *output*, long *angle\_add*) [inline]**

Add to the current value for set angle. Add an offset to the current set position. Returns immediately, but keep regulating.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*angle\_add* Value to add to the current set position, in degree. Can be negative. Can be greater than 360 degree to make several turns.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.579 void PosRegEnable (byte *output*, byte *p* = PID\_3, byte *i* = PID\_1, byte *d* = PID\_1) [inline]**

Enable absolute position regulation with PID factors. Enable absolute position regulation on the specified output. Motor is kept regulated as long as this is enabled. Optionally specify proportional, integral, and derivative factors.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_3](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_1](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#). Default value is [PID\\_1](#).

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.580 void PosRegSetAngle (byte *output*, long *angle*) [inline]**

Change the current value for set angle. Make the absolute position regulation going toward the new provided angle. Returns immediately, but keep regulating.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*angle* New set position, in degree. The 0 angle corresponds to the position of the motor when absolute position regulation was first enabled. Can be negative. Can be greater than 360 degree to make several turns.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.581 void PosRegSetMax (byte *output*, byte *max\_speed*, byte *max\_acceleration*) [inline]**

Set maximum limits. Set maximum speed and acceleration.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*output* Desired output port. Can be a constant or a variable, see [Output port constants](#).

*max\_speed* Maximum speed, or 0 to disable speed limiting.

*max\_acceleration* Maximum acceleration, or 0 to disable acceleration limiting. The *max\_speed* parameter should not be 0 if this is not 0.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.582 float pow (float *base*, float *exponent*) [inline]**

Raise to power. Computes base raised to the power exponent.

**Parameters:**

*base* Floating point value.

*exponent* Floating point value.

**Returns:**

The result of raising base to the power exponent.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_pow.nxc](#).

**8.3.3.583 void PowerDown () [inline]**

Power down the NXT. This function powers down the NXT. The running program will terminate as a result of this action.

**Examples:**

[ex\\_PowerDown.nxc](#).

**8.3.3.584 void Precedes (task *task1*, task *task2*, ..., task *taskN*) [inline]**

Declare tasks that this task precedes. Schedule the listed tasks for execution once the current task has completed executing. The tasks will all execute simultaneously unless other dependencies prevent them from doing so. This statement should be used once within a task - preferably at the start of the task definition. Any number of tasks may be listed in the Precedes statement.

**Parameters:**

- task1* The first task to start executing after the current task ends.
- task2* The second task to start executing after the current task ends.
- taskN* The last task to start executing after the current task ends.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_Precedes.nxc](#), and [ex\\_yield.nxc](#).

**8.3.3.585 void printf (string *format*, variant *value*) [inline]**

Print formatted data to stdout. Writes to the LCD at 0, LCD\_LINE1 a sequence of data formatted as the format argument specifies. After the format parameter, the function expects one value argument.

**Parameters:**

- format* A string specifying the desired format.
- value* A value to be formatted for writing to the LCD.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_printf.nxc](#).

**8.3.3.586 char PSPNxAnalog (const byte & *port*, const byte & *i2caddr*) [inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_PSPNxAnalog.nxc](#), and [ex\\_ReadSensorMSPlayStation.nxc](#).

**8.3.3.587 char PSPNxDigital (const byte & port, const byte & i2caddr)  
[inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_PSPNxDigital.nxc](#).

**8.3.3.588 unsigned long rand () [inline]**

Generate random number. Returns a pseudo-random integral number in the range 0 to [RAND\\_MAX](#).

This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called.

**Returns:**

An integer value between 0 and [RAND\\_MAX](#) (inclusive).

**Examples:**

[ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), and [ex\\_rand.nxc](#).

**8.3.3.589 int Random (unsigned int *n* = 0) [inline]**

Generate random number. Return a signed or unsigned 16-bit random number. If the optional argument *n* is not provided the function will return a signed value. Otherwise the returned value will range between 0 and *n* (exclusive).

**Parameters:**

*n* The maximum unsigned value desired (optional).

**Returns:**

A random number

**Examples:**

[ex\\_ArrayMax.nxc](#), [ex\\_CircleOut.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_EllipseOut.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_Random.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), [ex\\_string.nxc](#), [ex\\_SysDrawEllipse.nxc](#), and [ex\\_wait.nxc](#).

**8.3.3.590 unsigned int Read (byte *handle*, variant & *value*) [inline]**

Read a value from a file. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read.

**Parameters:**

*handle* The file handle.

*value* The variable to store the data read from the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_Read.nxc](#).

**8.3.3.591 char ReadButtonEx (const byte *btn*, bool *reset*, bool & *pressed*, unsigned int & *count*) [inline]**

Read button information. Read the specified button. Set the pressed and count parameters with the current state of the button. Optionally reset the press count after reading it.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*reset* Whether or not to reset the press counter.

*pressed* The button pressed state.

*count* The button press count.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadButtonEx.nxc](#).

**8.3.3.592 unsigned int ReadBytes (byte *handle*, unsigned int & *length*, byte & *buf*[]) [inline]**

Read bytes from a file. Read the specified number of bytes from the file associated with the specified handle. The handle parameter must be a variable. The length parameter must be a variable. The buf parameter must be an array or a string variable. The actual number of bytes read is returned in the length parameter.

**Parameters:**

*handle* The file handle.

*length* The number of bytes to read. Returns the number of bytes actually read.

*buf* The byte array where the data is stored on output.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ReadBytes.nxc](#).

**8.3.3.593 char ReadI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte & *out*) [inline]**

Read I2C register. Read a single byte from an I2C device register.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*i2caddr* The I2C device address.

*reg* The I2C device register from which to read a single byte.

*out* The single byte read from the I2C device.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_readi2cregister.nxc](#).

**8.3.3.594 unsigned int ReadLn (byte *handle*, variant & *value*) [inline]**

Read a value from a file plus line ending. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read. The ReadLn function reads two additional bytes from the file which it assumes are a carriage return and line feed pair.

**Parameters:**

*handle* The file handle.

*value* The variable to store the data read from the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ReadLn.nxc](#).

**8.3.3.595 unsigned int ReadLnString (byte *handle*, string & *output*)  
[inline]**

Read a string from a file plus line ending. Read a string from the file associated with the specified handle. The handle parameter must be a variable. The output parameter must be a variable. Appends bytes to the output variable until a line ending (CRLF) is reached. The line ending is also read but it is not appended to the output parameter.

**Parameters:**

- handle* The file handle.  
*output* The variable to store the string read from the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**8.3.3.596 bool ReadNRLinkBytes (const byte *port*, const byte *i2caddr*, byte & *data*[]) [inline]**

Read data from NRLink. Read data from the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).  
*i2caddr* The sensor I2C address. See sensor documentation for this value.  
*data* A byte array that will contain the data read from the device on output.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadNRLinkBytes.nxc](#).

**8.3.3.597 int ReadSensorColorEx (const byte & *port*, int & *colorval*, unsigned int & *raw*[], unsigned int & *norm*[], int & *scaled*[]) [inline]**

Read LEGO color sensor extra. This function lets you read the LEGO color sensor. It returns the color value, and three arrays containing raw, normalized, and scaled color values for red, green, blue, and none indices.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*colorval* The color value. See [Color values](#).

*raw* An array containing four raw color values. See [Color sensor array indices](#).

*norm* An array containing four normalized color values. See [Color sensor array indices](#).

*scaled* An array containing four scaled color values. See [Color sensor array indices](#).

**Returns:**

The function call result.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ReadSensorColorEx.nxc](#).

**8.3.3.598 int ReadSensorColorRaw (const byte & port, unsigned int & rawVals[]) [inline]**

Read LEGO color sensor raw values. This function lets you read the LEGO color sensor. It returns an array containing raw color values for red, green, blue, and none indices.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*rawVals* An array containing four raw color values. See [Color sensor array indices](#).

**Returns:**

The function call result.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_ReadSensorColorRaw.nxc](#).

**8.3.3.599 bool ReadSensorDIAccl (const byte *port*, VectorType & *vector*)  
[inline]**

ReadSensorDIAccl function. Read the scaled Dexter Industries IMU Accl X, Y, and Z axis 10-bit values.

**Parameters:**

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the scaled X, Y, and Z 10-bit values.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**8.3.3.600 bool ReadSensorDIAccl8 (const byte *port*, VectorType & *vector*)  
[inline]**

ReadSensorDIAccl8 function. Read the scaled Dexter Industries IMU Accl X, Y, and Z axis 8-bit values.

**Parameters:**

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the scaled X, Y, and Z 8-bit values.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

### 8.3.3.601 `bool ReadSensorDIAccl8Raw (const byte port, VectorType & vector) [inline]`

ReadSensorDIAccl8Raw function. Read the raw Dexter Industries IMU Accl X, Y, and Z axis 8-bit values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the raw X, Y, and Z 8-bit values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_diaccl.nxc](#).

### 8.3.3.602 `bool ReadSensorDIAcclDrift (const byte port, int & x, int & y, int & z) [inline]`

ReadSensorDIAcclDrift function. Read the Dexter Industries IMU Accl X, Y, and Z axis 10-bit drift values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*x* The X axis 10-bit drift value.

*y* The Y axis 10-bit drift value.

*z* The Z axis 10-bit drift value.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_diaccl.nxc](#).

### 8.3.3.603 `bool ReadSensorDIacclRaw (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIacclRaw function. Read the raw Dexter Industries IMU Accl X, Y, and Z axis 10-bit values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the raw X, Y, and Z 10-bit values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_diaccl.nxc](#).

### 8.3.3.604 `bool ReadSensorDIGyro (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIGyro function. Read the scaled Dexter Industries IMU Gyro X, Y, and Z axis values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the scaled X, Y, and Z values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_digyro.nxc](#).

### 8.3.3.605 `bool ReadSensorDIGyroRaw (const byte port, VectorType & vector)` `[inline]`

ReadSensorDIGyroRaw function. Read the raw Dexter Industries IMU Gyro X, Y, and Z axis values.

#### Parameters:

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vector* A variable of type [VectorType](#) which will contain the raw X, Y, and Z values.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_digyro.nxc](#).

### 8.3.3.606 `char ReadSensorEMeter (const byte & port, float & vIn, float & aIn, float & vOut, float & aOut, int & joules, float & wIn, float & wOut)` `[inline]`

Read the LEGO EMeter values. Read all the LEGO EMeter register values. They must all be read at once to ensure data coherency.

#### Parameters:

*port* The port to which the LEGO EMeter sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*vIn* Input voltage

*aIn* Input current

*vOut* Output voltage

*aOut* Output current

*joules* The number of joules stored in the EMeter

*wIn* The number of watts generated

*wOut* The number of watts consumed

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_ReadSensorEMeter.nxc](#).

**8.3.3.607 bool ReadSensorHTAccel (const byte *port*, int & *x*, int & *y*, int & *z*) [inline]**

Read HiTechnic acceleration values. Read X, Y, and Z axis acceleration values from the HiTechnic Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*x* The output x-axis acceleration.

*y* The output y-axis acceleration.

*z* The output z-axis acceleration.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTAccel.nxc](#).

**8.3.3.608 bool ReadSensorHTAngle (const byte *port*, int & *Angle*, long & *AccAngle*, int & *RPM*) [inline]**

Read HiTechnic Angle sensor values. Read values from the HiTechnic Angle sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Angle* Current angle in degrees (0-359).

*AccAngle* Accumulated angle in degrees (-2147483648 to 2147483647).

*RPM* rotations per minute (-1000 to 1000).

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTAngle.nxc](#).

**8.3.3.609 bool ReadSensorHTBarometric (const byte *port*, int & *temp*, unsigned int & *press*) [inline]**

Read HiTechnic Barometric sensor values. Read values from the HiTechnic Barometric sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*temp* Current temperature in 1/10ths of degrees Celcius.

*press* Current barometric pressure in 1/1000 inches of mercury.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTBarometric.nxc](#).

**8.3.3.610 bool ReadSensorHTColor (const byte *port*, byte & *ColorNum*, byte & *Red*, byte & *Green*, byte & *Blue*) [inline]**

Read HiTechnic Color values. Read color number, red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*ColorNum* The output color number.

*Red* The red color value.

*Green* The green color value.

*Blue* The blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTColor.nxc](#).

**8.3.3.611 bool ReadSensorHTColor2Active (byte *port*, byte & *ColorNum*, byte & *Red*, byte & *Green*, byte & *Blue*, byte & *White*) [inline]**

Read HiTechnic Color2 active values. Read color number, red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*ColorNum* The output color number.

*Red* The red color value.

*Green* The green color value.

*Blue* The blue color value.

*White* The white color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTColor2Active.nxc](#).

**8.3.3.612** `bool ReadSensorHTIRReceiver (const byte port, char & pfdata[ ]) [inline]`

Read HiTechnic IRReceiver Power Function bytes. Read Power Function bytes from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*pfdata* Eight bytes of power function remote IR data.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRReceiver.nxc](#).

**8.3.3.613** `bool ReadSensorHTIRReceiverEx (const byte port, const byte offset, char & pfchar) [inline]`

Read HiTechnic IRReceiver Power Function value. Read a Power Function byte from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The power function data offset. See [HiTechnic IRReceiver constants](#).

*pfchar* A single byte of power function remote IR data.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRReceiverEx.nxc](#).

**8.3.3.614** `bool ReadSensorHTIRSeeker (const byte port, byte & dir, byte & s1, byte & s3, byte & s5, byte & s7, byte & s9) [inline]`

Read HiTechnic IRSeeker values. Read direction, and five signal strength values from the HiTechnic IRSeeker sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

*s3* The signal strength from sensor 3.

*s5* The signal strength from sensor 5.

*s7* The signal strength from sensor 7.

*s9* The signal strength from sensor 9.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker.nxc](#).

**8.3.3.615** `bool ReadSensorHTIRSeeker2AC (const byte port, byte & dir, byte & s1, byte & s3, byte & s5, byte & s7, byte & s9) [inline]`

Read HiTechnic IRSeeker2 AC values. Read direction, and five signal strength values from the HiTechnic IRSeeker2 sensor in AC mode. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

*s3* The signal strength from sensor 3.

*s5* The signal strength from sensor 5.

*s7* The signal strength from sensor 7.

*s9* The signal strength from sensor 9.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker2AC.nxc](#).

**8.3.3.616** `bool ReadSensorHTIRSeeker2DC (const byte port, byte & dir, byte & s1, byte & s3, byte & s5, byte & s7, byte & s9, byte & avg)`  
[inline]

Read HiTechnic IRSeeker2 DC values. Read direction, five signal strength, and average strength values from the HiTechnic IRSeeker2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*dir* The direction.

*s1* The signal strength from sensor 1.

*s3* The signal strength from sensor 3.

*s5* The signal strength from sensor 5.

*s7* The signal strength from sensor 7.

*s9* The signal strength from sensor 9.

*avg* The average signal strength.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTIRSeeker2DC.nxc](#).

### 8.3.3.617 `bool ReadSensorHTNormalizedColor (const byte port, byte & ColorIdx, byte & Red, byte & Green, byte & Blue) [inline]`

Read HiTechnic Color normalized values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*ColorIdx* The output color index.

*Red* The normalized red color value.

*Green* The normalized green color value.

*Blue* The normalized blue color value.

#### Returns:

The function call result.

#### Examples:

[ex\\_ReadSensorHTNormalizedColor.nxc](#).

### 8.3.3.618 `bool ReadSensorHTNormalizedColor2Active (const byte port, byte & ColorIdx, byte & Red, byte & Green, byte & Blue) [inline]`

Read HiTechnic Color2 normalized active values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).

*ColorIdx* The output color index.

*Red* The normalized red color value.

*Green* The normalized green color value.

*Blue* The normalized blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTNormalizedColor2Active.nxc](#).

**8.3.3.619 bool ReadSensorHTProtoAllAnalog (const byte *port*, int & *a0*, int & *a1*, int & *a2*, int & *a3*, int & *a4*) [inline]**

Read all HiTechnic Prototype board analog input values. Read all 5 analog input values from the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*a0* The A0 analog input value.

*a1* The A1 analog input value.

*a2* The A2 analog input value.

*a3* The A3 analog input value.

*a4* The A4 analog input value.

**Returns:**

The function call result.

**Examples:**

[ex\\_proto.nxc](#).

**8.3.3.620 bool ReadSensorHTRawColor (const byte *port*, unsigned int & *Red*, unsigned int & *Green*, unsigned int & *Blue*) [inline]**

Read HiTechnic Color raw values. Read the raw red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Red* The raw red color value.

*Green* The raw green color value.

*Blue* The raw blue color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTRawColor.nxc](#).

**8.3.3.621** `bool ReadSensorHTRawColor2 (const byte port, unsigned int & Red, unsigned int & Green, unsigned int & Blue, unsigned int & White) [inline]`

Read HiTechnic Color2 raw values. Read the raw red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*Red* The raw red color value.

*Green* The raw green color value.

*Blue* The raw blue color value.

*White* The raw white color value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorHTRawColor2.nxc](#).

### 8.3.3.622 `bool ReadSensorHTSuperProAllAnalog (const byte port, int & a0, int & a1, int & a2, int & a3) [inline]`

Read all HiTechnic SuperPro board analog input values. Read all 4 analog input values from the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

- port* The sensor port. See [Input port constants](#).
- a0* The A0 analog input value.
- a1* The A1 analog input value.
- a2* The A2 analog input value.
- a3* The A3 analog input value.

#### Returns:

The function call result.

#### Examples:

[ex\\_superpro.nxc](#).

### 8.3.3.623 `bool ReadSensorHTSuperProAnalogOut (const byte port, const byte dac, byte & mode, int & freq, int & volt) [inline]`

Read HiTechnic SuperPro board analog output parameters. Read the analog output parameters on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

- port* The sensor port. See [NBC Input port constants](#).
- dac* The analog output index. See [HiTechnic SuperPro analog output index constants](#).
- mode* The analog output mode. See [SuperPro analog output mode constants](#).
- freq* The analog output frequency. Between 1 and 8191.
- volt* The analog output voltage level. A 10 bit value (0..1023).

**Returns:**

The function call result.

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.624 void ReadSensorHTTouchMultiplexer (const byte *port*, byte & *t1*, byte & *t2*, byte & *t3*, byte & *t4*) [inline]**

Read HiTechnic touch multiplexer. Read touch sensor values from the HiTechnic touch multiplexer device.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*t1* The value of touch sensor 1.

*t2* The value of touch sensor 2.

*t3* The value of touch sensor 3.

*t4* The value of touch sensor 4.

**Examples:**

[ex\\_ReadSensorHTTouchMultiplexer.nxc](#).

**8.3.3.625 bool ReadSensorMIXG1300L (byte *port*, XGPacketType & *packet*) [inline]**

ReadSensorMIXG1300L function. Read Microinfinity CruzCore XG1300L values. Read accumulated angle, turn rate, and X, Y, and Z axis acceleration values from the Microinfinity CruzCore XG1300L sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See the [Input port constants](#) group.

*packet* The output XK1300L data structure. See [XGPacketType](#).

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.3.3.626 bool ReadSensorMSAccel (const byte *port*, const byte *i2caddr*, int & *x*, int & *y*, int & *z*) [inline]**

Read mindsensors acceleration values. Read X, Y, and Z axis acceleration values from the mindsensors Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*x* The output x-axis acceleration.

*y* The output y-axis acceleration.

*z* The output z-axis acceleration.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSAccel.nxc](#).

**8.3.3.627 bool ReadSensorMSPlayStation (const byte *port*, const byte *i2caddr*, byte & *btnset1*, byte & *btnset2*, byte & *xleft*, byte & *yleft*, byte & *xright*, byte & *yright*) [inline]**

Read mindsensors playstation controller values. Read playstation controller values from the mindsensors playstation sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- btnset1* The button set 1 values. See [MindSensors PSP-Nx button set 1 constants](#).
- btnset2* The button set 2 values. See [MindSensors PSP-Nx button set 2 constants](#).
- xleft* The left joystick x value.
- yleft* The left joystick y value.
- xright* The right joystick x value.
- yright* The right joystick y value.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSPlayStation.nxc](#).

**8.3.3.628** `bool ReadSensorMSRTClock (const byte port, byte & sec, byte & min, byte & hrs, byte & dow, byte & date, byte & month, byte & year) [inline]`

Read mindsensors RTClock values. Read real-time clock values from the Mindsensors RTClock sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- sec* The seconds.
- min* The minutes.
- hrs* The hours.
- dow* The day of week number.
- date* The day.
- month* The month.
- year* The year.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSRTClock.nxc](#).

**8.3.3.629 bool ReadSensorMSTilt (const byte & port, const byte & i2caddr, byte & x, byte & y, byte & z) [inline]**

Read mindsensors tilt values. Read X, Y, and Z axis tilt values from the mindsensors tilt sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*x* The output x-axis tilt.

*y* The output y-axis tilt.

*z* The output z-axis tilt.

**Returns:**

The function call result.

**Examples:**

[ex\\_ReadSensorMSTilt.nxc](#).

**8.3.3.630 char ReadSensorUSEx (const byte port, byte & values[]) [inline]**

Read multiple ultrasonic sensor values. Return eight ultrasonic sensor distance values.

**Parameters:**

*port* The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*values* An array of bytes that will contain the 8 distance values read from the ultrasonic sensor.

**Returns:**

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**Examples:**

[ex\\_ReadSensorUSEx.nxc](#).

**8.3.3.631 void RebootInFirmwareMode () [inline]**

Reboot the NXT in firmware download mode. This function lets you reboot the NXT into SAMBA or firmware download mode. The running program will terminate as a result of this action.

**Examples:**

[ex\\_RebootInFirmwareMode.nxc](#).

**8.3.3.632 char ReceiveMessage (byte *queue*, bool *clear*, string & *msg*) [inline]**

Read a message from a queue/mailbox. Read a message from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*clear* A flag indicating whether to remove the message from the mailbox after it has been read.

*msg* The message that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**8.3.3.633 char ReceiveRemoteBool (byte *queue*, bool *clear*, bool & *bval*) [inline]**

Read a boolean value from a queue/mailbox. Read a boolean value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- bval* The boolean value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteBool.nxc](#), and [ex\\_ReceiveRemoteNumber.nxc](#).

**8.3.3.634 char ReceiveRemoteMessageEx (byte *queue*, bool *clear*, string & *str*, long & *val*, bool & *bval*) [inline]**

Read a value from a queue/mailbox. Read a value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number. Output the value in string, number, and boolean form.

**Parameters:**

- queue* The mailbox number. See [Mailbox constants](#).
- clear* A flag indicating whether to remove the message from the mailbox after it has been read.
- str* The string value that is read from the mailbox.
- val* The numeric value that is read from the mailbox.
- bval* The boolean value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteMessageEx.nxc](#).

**8.3.3.635 char ReceiveRemoteNumber (byte *queue*, bool *clear*, long & *val*)  
[inline]**

Read a numeric value from a queue/mailbox. Read a numeric value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*clear* A flag indicating whether to remove the message from the mailbox after it has been read.

*val* The numeric value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**8.3.3.636 char ReceiveRemoteString (byte *queue*, bool *clear*, string & *str*)  
[inline]**

Read a string value from a queue/mailbox. Read a string value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*clear* A flag indicating whether to remove the message from the mailbox after it has been read.

*str* The string value that is read from the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_ReceiveRemoteString.nxc](#).

**8.3.3.637 bool RechargeableBattery (void) [inline]**

Read battery type. Return whether the NXT has a rechargeable battery installed or not.

**Returns:**

Whether the battery is rechargeable or not. (false = no, true = yes)

**Examples:**

[ex\\_RechargeableBattery.nxc](#).

**8.3.3.638 char RectOut (int *x*, int *y*, int *width*, int *height*, unsigned long *options* = DRAW\_OPT\_NORMAL) [inline]**

Draw a rectangle. This function lets you draw a rectangle on the screen at *x*, *y* with the specified width and height. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW\\_OPT\\_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

**See also:**

[SysDrawRect](#), [DrawRectType](#)

**Parameters:**

*x* The x value for the top left corner of the rectangle.

*y* The y value for the top left corner of the rectangle.

*width* The width of the rectangle.

*height* The height of the rectangle.

*options* The optional drawing options.

**Returns:**

The result of the drawing operation.

**Examples:**

[ex\\_RectOut.nxc](#).

**8.3.3.639 unsigned long reladdressOf (variant *data*) [inline]**

Get the relative address of a variable. Get the relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*data* A variable whose address you wish to get.

**Returns:**

The relative address of the variable.

**Examples:**

[ex\\_reladdressof.nxc](#).

**8.3.3.640 void Release (mutex *m*) [inline]**

Acquire a mutex. Release the specified mutex variable. Use this to relinquish a mutex so that it can be acquired by another task. Release should always be called after a matching call to Acquire and as soon as possible after a shared resource is no longer needed.

**Parameters:**

*m* The mutex to release.

**Examples:**

[ex\\_Acquire.nxc](#), and [ex\\_Release.nxc](#).

**8.3.3.641 char RemoteBluetoothFactoryReset (byte *conn*) [inline]**

Send a BluetoothFactoryReset message. This method sends a BluetoothFactoryReset system command to the device on the specified connection. Use [RemoteConnection-Idle](#) to determine when this write request is completed. This command cannot be sent over a bluetooth connection.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteBluetoothFactoryReset.nxc](#).

**8.3.3.642 char RemoteCloseFile (byte *conn*, byte *handle*) [inline]**

Send a CloseFile message. Send the CloseFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file to close.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteCloseFile.nxc](#).

**8.3.3.643 bool RemoteConnectionIdle (byte *conn*) [inline]**

Check if remote connection is idle. Check whether a Bluetooth or RS485 hi-speed port connection is idle, i.e., not currently sending data.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A boolean value indicating whether the connection is idle or busy.

**Warning:**

Checking the status of the RS485 hi-speed connection requires the enhanced NBC/NXC firmware

**Examples:**

[ex\\_RemoteConnectionIdle.nxc](#).

**8.3.3.644 char RemoteConnectionWrite (byte *conn*, byte *buffer*[ ]) [inline]**

Write to a remote connection. This method tells the NXT firmware to write the data in the buffer to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*buffer* The data to be written (up to 128 bytes)

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

Writing to the RS485 hi-speed connection requires the enhanced NBC/NXC firmware

**Examples:**

[ex\\_RemoteConnectionWrite.nxc](#).

**8.3.3.645 char RemoteDatalogRead (byte *conn*, bool *remove*, byte & *cnt*, byte & *log*[]) [inline]**

Send a DatalogRead message. Send the DatalogRead direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*remove* Remove the datalog message from the queue after reading it (true or false).

*cnt* The number of bytes read from the datalog.

*log* A byte array containing the datalog contents.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDatalogRead.nxc](#).

**8.3.3.646 char RemoteDatalogSetTimes (byte *conn*, long *synctime*) [inline]**

Send a DatalogSetTimes message. Send the DatalogSetTimes direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*synctime* The datalog sync time.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDatalogSetTimes.nxc](#).

**8.3.3.647 char RemoteDeleteFile (byte *conn*, string *filename*) [inline]**

Send a DeleteFile message. Send the DeleteFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to delete.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDeleteFile.nxc](#).

**8.3.3.648 char RemoteDeleteUserFlash (byte *conn*) [inline]**

Send a DeleteUserFlash message. This method sends a DeleteUserFlash system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteDeleteUserFlash.nxc](#).

**8.3.3.649 char RemoteFindFirstFile (byte *conn*, string *mask*, byte & *handle*, string & *name*, long & *size*) [inline]**

Send a FindFirstFile message. Send the FindFirstFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*mask* The filename mask for the files you want to find.

*handle* The handle of the found file.

*name* The name of the found file.

*size* The size of the found file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteFindFirstFile.nxc](#).

**8.3.3.650 char RemoteFindNextFile (byte *conn*, byte & *handle*, string & *name*, long & *size*) [inline]**

Send a FindNextFile message. Send the FindNextFile system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle returned by the last [FindFirstFile](#) or FindNextFile call.

*name* The name of the next found file.

*size* The size of the next found file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteFindNextFile.nxc](#).

**8.3.3.651 char RemoteGetBatteryLevel (byte conn, int & value) [inline]**

Send a GetBatteryLevel message. Send the GetBatteryLevel direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*value* The battery level value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetBatteryLevel.nxc](#).

**8.3.3.652 char RemoteGetBluetoothAddress (byte *conn*, byte & *btaddr*[ ]) [inline]**

Send a GetBluetoothAddress message. This method sends a GetBluetoothAddress system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*btaddr* The bluetooth address of the remote device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetBluetoothAddress.nxc](#).

**8.3.3.653 char RemoteGetConnectionCount (byte *conn*, byte & *cnt*) [inline]**

Send a GetConnectionCount message. This method sends a GetConnectionCount direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*cnt* The number of connections.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetConnectionCount.nxc](#).

**8.3.3.654 char RemoteGetConnectionName (byte *conn*, byte *idx*, string & *name*) [inline]**

Send a GetConnectionName message. Send the GetConnectionName direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*idx* The index of the connection.

*name* The name of the specified connection.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetConnectionName.nxc](#).

**8.3.3.655 char RemoteGetContactCount (byte *conn*, byte & *cnt*) [inline]**

Send a GetContactCount message. This method sends a GetContactCount direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*cnt* The number of contacts.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetContactCount.nxc](#).

**8.3.3.656 char RemoteGetContactName (byte *conn*, byte *idx*, string & *name*)  
[inline]**

Send a GetContactName message. Send the GetContactName direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*idx* The index of the contact.

*name* The name of the specified contact.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetContactName.nxc](#).

**8.3.3.657 char RemoteGetCurrentProgramName (byte *conn*, string & *name*)  
[inline]**

Send a GetCurrentProgramName message. This method sends a GetCurrentProgramName direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The current program name.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetCurrentProgramName.nxc](#).

**8.3.3.658 char RemoteGetDeviceInfo (byte *conn*, string & *name*, byte & *btaddr*[], byte & *btsignal*[], long & *freemem*) [inline]**

Send a GetDeviceInfo message. This method sends a GetDeviceInfo system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The name of the remote device.

*btaddr* The bluetooth address of the remote device.

*btsignal* The signal strength of each connection on the remote device.

*freemem* The number of bytes of free flash memory on the remote device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetDeviceInfo.nxc](#).

**8.3.3.659 char RemoteGetFirmwareVersion (byte *conn*, byte & *pmin*, byte & *pmaj*, byte & *fmin*, byte & *fmaj*) [inline]**

Send a GetFirmwareVersion message. This method sends a GetFirmwareVersion system command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*pmin* The protocol minor version byte.

*pmaj* The protocol major version byte.

*fmin* The firmware minor version byte.

*fmaj* The firmware major version byte.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetFirmwareVersion.nxc](#).

**8.3.3.660 char RemoteGetInputValues (byte *conn*, InputValueType & *params*) [inline]**

Send a GetInputValues message. Send the GetInputValues direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*params* The input and output parameters for the function call. See [InputValuesType](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetInputValues.nxc](#).

**8.3.3.661 char RemoteGetOutputState (byte *conn*, OutputStateType & *params*) [inline]**

Send a GetOutputState message. Send the GetOutputState direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*params* The input and output parameters for the function call. See [OutputStateType](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetOutputState.nxc](#).

**8.3.3.662 char RemoteGetProperty (byte *conn*, byte *property*, variant & *value*) [inline]**

Send a GetProperty message. Send the GetProperty direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*property* The property to read. See [Property constants](#).

*value* The property value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteGetProperty.nxc](#).

**8.3.3.663 char RemoteIOMapRead (byte *conn*, long *id*, int *offset*, int & *numbytes*, byte & *data*[]) [inline]**

Send an IOMapRead message. Send the IOMapRead system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module from which to read data.

*offset* The offset into the IOMap structure from which to read.

*numbytes* The number of bytes of data to read. Returns the number of bytes actually read.

*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteIOMapRead.nxc](#).

**8.3.3.664 char RemoteIOMapWriteBytes (byte *conn*, long *id*, int *offset*, byte *data*[]) [inline]**

Send an IOMapWrite bytes message. Send the IOMapWrite system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module to which to write data.

*offset* The offset into the IOMap structure to which to write.

*data* A byte array containing the data you are writing to the IOMap structure.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteIOMapWriteBytes.nxc](#).

**8.3.3.665 char RemoteIOMapWriteValue (byte *conn*, long *id*, int *offset*, variant *value*) [inline]**

Send an IOMapWrite value message. Send the IOMapWrite system command on the specified connection slot to write the value provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*id* The ID of the module to which to write data.

*offset* The offset into the IOMap structure to which to write.

*value* A scalar variable containing the value you are writing to the IOMap structure.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteIOMapWriteValue.nxc](#).

**8.3.3.666 char RemoteKeepAlive (byte *conn*) [inline]**

Send a KeepAlive message. This method sends a KeepAlive direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteKeepAlive.nxc](#).

**8.3.3.667 char RemoteLowSpeedGetStatus (byte *conn*, byte & *value*) [inline]**

Send a LowSpeedGetStatus message. This method sends a LowSpeedGetStatus direct command to the device on the specified connection.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*value* The count of available bytes to read.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowspeedGetStatus.nxc](#).

**8.3.3.668 char RemoteLowspeedRead (byte *conn*, byte *port*, byte & *bread*, byte & *data*[]) [inline]**

Send a LowspeedRead message. Send the LowspeedRead direct command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port from which to read I2C data. See [Input port constants](#).

*bread* The number of bytes read.

*data* A byte array containing the data read from the I2C device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowspeedRead.nxc](#).

**8.3.3.669 char RemoteLowSpeedWrite (byte *conn*, byte *port*, byte *txlen*, byte *rxlen*, byte *data*[]) [inline]**

Send a LowSpeedWrite message. Send the LowSpeedWrite direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The I2C port. See [Input port constants](#).

*txlen* The number of bytes you are writing to the I2C device.

*rxlen* The number of bytes want to read from the I2C device.

*data* A byte array containing the data you are writing to the device.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteLowSpeedWrite.nxc](#).

**8.3.3.670 char RemoteMessageRead (byte *conn*, byte *queue*) [inline]**

Send a MessageRead message. This method sends a MessageRead direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox to read. See [Mailbox constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteMessageRead.nxc](#).

**8.3.3.671 char RemoteMessageWrite (byte *conn*, byte *queue*, string *msg*)  
[inline]**

Send a MessageWrite message. This method sends a MessageWrite direct command to the device on the specified connection. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox to write. See [Mailbox constants](#).

*msg* The message to write to the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteMessageWrite.nxc](#).

**8.3.3.672 char RemoteOpenAppendData (byte *conn*, string *filename*, byte & *handle*, long & *size*) [inline]**

Send an OpenAppendData message. Send the OpenAppendData system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for appending.

*handle* The handle of the file.

*size* The size of the file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenAppendData.nxc](#).

**8.3.3.673 char RemoteOpenRead (byte *conn*, string *filename*, byte & *handle*, long & *size*) [inline]**

Send an OpenRead message. Send the OpenRead system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for reading.

*handle* The handle of the file.

*size* The size of the file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenRead.nxc](#).

**8.3.3.674 char RemoteOpenWrite (byte *conn*, string *filename*, long *size*, byte & *handle*) [inline]**

Send an OpenWrite message. Send the OpenWrite system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWrite.nxc](#).

**8.3.3.675 char RemoteOpenWriteData (byte *conn*, string *filename*, long *size*, byte & *handle*) [inline]**

Send an OpenWriteData message. Send the OpenWriteData system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWriteData.nxc](#).

**8.3.3.676 char RemoteOpenWriteLinear (byte *conn*, string *filename*, long *size*, byte & *handle*) [inline]**

Send an OpenWriteLinear message. Send the OpenWriteLinear system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the file to open for writing (i.e., create the file).

*size* The size for the new file.

*handle* The handle of the new file.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteOpenWriteLinear.nxc](#).

**8.3.3.677 char RemotePlaySoundFile (byte *conn*, string *filename*, bool *loop*) [inline]**

Send a PlaySoundFile message. Send the PlaySoundFile direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the sound file to play.

*loop* A boolean value indicating whether to loop the sound file or not.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePlaySoundFile.nxc](#).

**8.3.3.678 char RemotePlayTone (byte *conn*, unsigned int *frequency*, unsigned int *duration*) [inline]**

Send a PlayTone message. Send the PlayTone direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*frequency* The frequency of the tone.

*duration* The duration of the tone.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePlayTone.nxc](#).

**8.3.3.679 char RemotePollCommand (byte *conn*, byte *bufnum*, byte & *len*, byte & *data*[]) [inline]**

Send a PollCommand message. Send the PollCommand system command on the specified connection slot to write the data provided.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*bufnum* The buffer from which to read data (0=USBPoll, 1=HiSpeed).

*len* The number of bytes to read. Returns the number of bytes actually read.

*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePollCommand.nxc](#).

**8.3.3.680 char RemotePollCommandLength (byte *conn*, byte *bufnum*, byte & *length*) [inline]**

Send a PollCommandLength message. Send the PollCommandLength system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*bufnum* The poll buffer you want to query (0=USBPoll, 1=HiSpeed).

*length* The number of bytes available for polling.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemotePollCommandLength.nxc](#).

**8.3.3.681 char RemoteRead (byte *conn*, byte & *handle*, int & *numbytes*, byte & *data*[ ]) [inline]**

Send a Read message. Send the Read system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file you are reading from.

*numbytes* The number of bytes you want to read. Returns the number of bytes actually read.

*data* A byte array containing the response data.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteRead.nxc](#).

**8.3.3.682 char RemoteRenameFile (byte *conn*, string *oldname*, string *newname*) [inline]**

Send a RenameFile message. Send the RenameFile system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*oldname* The old filename.

*newname* The new filename.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteRenameFile.nxc](#).

**8.3.3.683 char RemoteResetMotorPosition (byte *conn*, byte *port*, bool *brelative*) [inline]**

Send a ResetMotorPosition message. Send the ResetMotorPosition direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to reset.

*brelative* A flag indicating whether the counter to reset is relative.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetMotorPosition.nxc](#).

**8.3.3.684 char RemoteResetScaledValue (byte *conn*, byte *port*) [inline]**

Send a ResetScaledValue message. Send the ResetScaledValue direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port to reset.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetScaledValue.nxc](#).

**8.3.3.685 char RemoteResetTachoCount (byte *conn*, byte *port*) [inline]**

Send a ResetTachoCount message. Send the ResetTachoCount direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to reset the tachometer count on. See [Output port constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteResetTachoCount.nxc](#).

**8.3.3.686 char RemoteSetBrickName (byte *conn*, string *name*) [inline]**

Send a SetBrickName message. Send the SetBrickName system command on the specified connection slot to write the data provided. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*name* The new brick name.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetBrickName.nxc](#).

**8.3.3.687 char RemoteSetInputMode (byte *conn*, byte *port*, byte *type*, byte *mode*) [inline]**

Send a SetInputMode message. Send the SetInputMode direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The input port to configure. See [Input port constants](#).

*type* The sensor type. See [Sensor type constants](#).

*mode* The sensor mode. See [Sensor mode constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetInputMode.nxc](#).

**8.3.3.688** `char RemoteSetOutputState (byte conn, byte port, char speed, byte mode, byte regmode, char turnpct, byte runstate, unsigned long tacholimit) [inline]`

Send a SetOutputMode message. Send the SetOutputMode direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*port* The output port to configure. See [Output port constants](#).

*speed* The motor speed. (-100..100)

*mode* The motor mode. See [Output port mode constants](#).

*regmode* The motor regulation mode. See [Output port regulation mode constants](#).

*turnpct* The motor synchronized turn percentage. (-100..100)

*runstate* The motor run state. See [Output port run state constants](#).

*tacholimit* The motor tachometer limit.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetOutputState.nxc](#).

**8.3.3.689 char RemoteSetProperty (byte *conn*, byte *prop*, variant *value*)  
[inline]**

Send a SetProperty message. Send the SetProperty direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*prop* The property to set. See [Property constants](#).

*value* The new property value.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteSetProperty.nxc](#).

**8.3.3.690 char RemoteStartProgram (byte *conn*, string *filename*) [inline]**

Send a StartProgram message. Send the StartProgram direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*filename* The name of the program to start running.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStartProgram.nxc](#).

**8.3.3.691 char RemoteStopProgram (byte *conn*) [inline]**

Send a StopProgram message. Send the StopProgram direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStopProgram.nxc](#).

**8.3.3.692 char RemoteStopSound (byte *conn*) [inline]**

Send a StopSound message. Send the StopSound direct command on the specified connection slot. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteStopSound.nxc](#).

**8.3.3.693** `char RemoteWrite (byte conn, byte & handle, int & numbytes, byte data[]) [inline]`

Send a Write message. Send the Write system command on the specified connection slot.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*handle* The handle of the file you are writing to.

*numbytes* The number of bytes actually written.

*data* A byte array containing the data you are writing.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_RemoteWrite.nxc](#).

**8.3.3.694** `int remove (string filename) [inline]`

Remove file. Delete the specified file. The loader result code is returned as the value of the function call.

**Parameters:**

*filename* The name of the file to be deleted.

**Returns:**

The loader result code.

**8.3.3.695 int rename (string *old*, string *new*) [inline]**

Rename file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call.

**Parameters:**

*old* The name of the file to be renamed.

*new* The new name for the file.

**Returns:**

The loader result code.

**Examples:**

[ex\\_rename.nxc](#).

**8.3.3.696 unsigned int RenameFile (string *oldname*, string *newname*) [inline]**

Rename a file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call. The filename parameters must be constants or variables.

**Parameters:**

*oldname* The old filename.

*newname* The new filename.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_RenameFile.nxc](#).

**8.3.3.697 void ResetAllTachoCounts (byte *outputs*) [inline]**

Reset all tachometer counters. Reset all three position counters and reset the current tachometer limit goal for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resetalltachocounts.nxc](#).

**8.3.3.698 void ResetBlockTachoCount (byte *outputs*) [inline]**

Reset block-relative counter. Reset the block-relative position counter for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resetblocktachocount.nxc](#).

**8.3.3.699 bool ResetHTBarometricCalibration (byte *port*) [inline]**

Reset HiTechnic Barometric sensor calibration. Reset the HiTechnic Barometric sensor to its factory calibration. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The function call result.

### 8.3.3.700 bool ResetMIXG1300L (byte *port*) [inline]

ResetMIXG1300L function. Reset the Microinfinity CruzCore XG1300L device.

During reset, the XG1300L will recomputed the bias drift value, therefore it must remain stationary. The bias drift value will change randomly over time due to temperature variations, however the internal algorithm in the XG1300L will compensate for these changes. We strongly recommend issuing a reset command to the XG1300L at the beginning of the program.

The reset function also resets the accumulate angle value to a zero. Since the accelerometers measurements are taken with respect to the sensor reference frame the reset function will have no effect in the accelerometer measurements.

Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See the [Input port constants](#) group.

#### Returns:

The boolean function call result.

#### Examples:

[ex\\_xg1300.nxc](#).

### 8.3.3.701 void ResetRotationCount (byte *outputs*) [inline]

Reset program-relative counter. Reset the program-relative position counter for the specified outputs.

#### Parameters:

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

#### Examples:

[ex\\_resetrotationcount.nxc](#).

**8.3.3.702 void ResetScreen () [inline]**

Reset LCD screen. This function lets you restore the standard NXT running program screen.

**Examples:**

[ex\\_ResetScreen.nxc](#).

**8.3.3.703 void ResetSensor (const byte & port) [inline]**

Reset the sensor port. Sets the invalid data flag on the specified port and waits for it to become valid again. After changing the type or the mode of a sensor port you must call this function to give the firmware time to reconfigure the sensor port.

**Parameters:**

*port* The port to reset. See [Input port constants](#).

**Examples:**

[ex\\_ResetSensor.nxc](#).

**8.3.3.704 char ResetSensorHTAngle (const byte port, const byte mode) [inline]**

Reset HiTechnic Angle sensor. Reset the HiTechnic Angle sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The Angle reset mode. See [HiTechnic Angle sensor constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_ResetSensorHTAngle.nxc](#).

**8.3.3.705 long ResetSleepTimer () [inline]**

Reset the sleep timer. This function lets you reset the sleep timer.

**Returns:**

The result of resetting the sleep timer.

**Examples:**

[ex\\_ResetSleepTimer.nxc](#).

**8.3.3.706 void ResetTachoCount (byte outputs) [inline]**

Reset tachometer counter. Reset the tachometer count and tachometer limit goal for the specified outputs.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

**Examples:**

[ex\\_resettachocount.nxc](#).

**8.3.3.707 unsigned int ResizeFile (string fname, const unsigned int newsize) [inline]**

Resize a file. Resize the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*fname* The name of the file to resize.

*newsiz*e The new size for the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_resizefile.nxc](#).

**8.3.3.708 unsigned int ResolveHandle (string *filename*, byte & *handle*, bool & *writable*) [inline]**

Resolve a handle. Resolve a file handle from the specified filename. The file handle is returned in the second parameter, which must be a variable. A boolean value indicating whether the handle can be used to write to the file or not is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

**Parameters:**

*filename* The name of the file for which to resolve a handle.

*handle* The file handle output from the function call.

*writable* A boolean flag indicating whether the handle is to a file open for writing (true) or reading (false).

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_ResolveHandle.nxc](#).

**8.3.3.709 void rewind (byte *handle*) [inline]**

Set position indicator to the beginning. Sets the position indicator associated with stream to the beginning of the file.

**Parameters:**

*handle* The handle of the file.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_rewind.nxc](#).

**8.3.3.710 bool RFIDInit (const byte & port) [inline]**

RFIDInit function. Initialize the Codatex RFID sensor.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDInit.nxc](#).

**8.3.3.711 bool RFIDMode (const byte & port, const byte & mode) [inline]**

RFIDMode function. Configure the Codatex RFID sensor mode.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*mode* The RFID sensor mode. See the [Codatex RFID sensor modes](#) group.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDMode.nxc](#).

**8.3.3.712 bool RFIDRead (const byte & port, byte & output[]) [inline]**

RFIDRead function. Read the Codatex RFID sensor value.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDRead.nxc](#).

**8.3.3.713 bool RFIDReadContinuous (const byte & port, byte & output[])  
[inline]**

RFIDReadContinuous function. Set the Codatex RFID sensor into continuous mode, if necessary, and read the RFID data.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDReadContinuous.nxc](#).

**8.3.3.714 bool RFIDReadSingle (const byte & port, byte & output[])  
[inline]**

RFIDReadSingle function. Set the Codatex RFID sensor into single mode and read the RFID data.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*output* The five bytes of RFID data.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDReadSingle.nxc](#).

**8.3.3.715 byte RFIDStatus (const byte & port) [inline]**

RFIDStatus function. Read the Codatex RFID sensor status.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The RFID sensor status.

**Examples:**

[ex\\_RFIDStatus.nxc](#).

**8.3.3.716 bool RFIDStop (const byte & port) [inline]**

RFIDStop function. Stop the Codatex RFID sensor.

**Parameters:**

*port* The port to which the Codatex RFID sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_RFIDStop.nxc](#).

**8.3.3.717 string RightStr (string *str*, unsigned int *size*) [inline]**

Copy a portion from the end of a string. Returns the substring of a specified length that appears at the end of a string.

**Parameters:**

- str* A string
- size* The size or length of the substring.

**Returns:**

The substring of a specified length that appears at the end of a string.

**Examples:**

[ex\\_rightstr.nxc](#).

**8.3.3.718 void RotateMotor (byte *outputs*, char *pwr*, long *angle*) [inline]**

Rotate motor. Run the specified outputs forward for the specified number of degrees.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- angle* Angle limit, in degree. Can be negative to reverse direction.

**Examples:**

[ex\\_rotatemotor.nxc](#).

**8.3.3.719 void RotateMotorEx (byte *outputs*, char *pwr*, long *angle*, char *turnpct*, bool *sync*, bool *stop*) [inline]**

Rotate motor. Run the specified outputs forward for the specified number of degrees.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- angle* Angle limit, in degree. Can be negative to reverse direction.
- turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.
- sync* Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.
- stop* Specify whether the motor(s) should brake at the end of the rotation.

**Examples:**

[ex\\_rotatemotorex.nxc](#).

**8.3.3.720 void RotateMotorExPID (byte *outputs*, char *pwr*, long *angle*, char *turnpct*, bool *sync*, bool *stop*, byte *p*, byte *i*, byte *d*) [**inline**]**

Rotate motor. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.
- pwr* Output power, 0 to 100. Can be negative to reverse direction.
- angle* Angle limit, in degree. Can be negative to reverse direction.
- turnpct* Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.
- sync* Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.
- stop* Specify whether the motor(s) should brake at the end of the rotation.
- p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).
- i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_rotatemotorexpid.nxc](#).

**8.3.3.721 void RotateMotorPID (byte *outputs*, char *pwr*, long *angle*, byte *p*, byte *i*, byte *d*) [inline]**

Rotate motor with PID factors. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

**Parameters:**

*outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

*pwr* Output power, 0 to 100. Can be negative to reverse direction.

*angle* Angle limit, in degree. Can be negative to reverse direction.

*p* Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*i* Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

*d* Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

**Examples:**

[ex\\_rotatemotorpid.nxc](#).

**8.3.3.722 char RS485Control (byte *cmd*, byte *baud*, unsigned int *mode*) [inline]**

Control the RS485 port. Control the RS485 hi-speed port using the specified parameters.

**Parameters:**

*cmd* The control command to send to the port. See [Hi-speed port SysCommHSCControl constants](#).

*baud* The baud rate for the RS485 port. See [Hi-speed port baud rate constants](#).

*mode* The RS485 port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.723 byte RS485DataAvailable (void) [inline]**

Check for RS485 available data. Check the RS485 hi-speed port for available data.

**Returns:**

The number of bytes of data available for reading.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.724 char RS485Disable (void) [inline]**

Disable RS485. Turn off the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**8.3.3.725 char RS485Enable (void) [inline]**

Enable RS485. Turn on the RS485 hi-speed port so that it can be used.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.726 char RS485Initialize (void) [inline]**

Initialize RS485 port. Initialize the RS485 UART port to its default values. The baud rate is set to 921600 and the mode is set to 8N1 (8 data bits, no parity, 1 stop bit). Data cannot be sent or received over the RS485 port until the port is configured as a hi-speed port, the port is turned on, and the UART is initialized.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.727 char RS485Read (byte & buffer[]) [inline]**

Read RS485 data. Read data from the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array that will contain the data read from the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.728 char RS485ReadEx (byte & *buffer*[], byte *buflen*) [inline]**

Read limited RS485 data. Read a limited number of bytes of data from the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array that will contain the data read from the RS485 port.

*buflen* The number of bytes you want to read.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_RS485Receive.nxc](#).

**8.3.3.729 byte RS485SendingData (void) [inline]**

Is RS485 sending data. Check whether the RS485 is actively sending data.

**Returns:**

The number of bytes of data being sent.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.730** `void RS485Status (byte & sendingData, byte & dataAvail)`  
`[inline]`

Check RS485 status. Check the status of the RS485 hi-speed port.

**Parameters:**

*sendingData* The number of bytes of data being sent.

*dataAvail* The number of bytes of data available for reading.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.731** `char RS485Uart (byte baud, unsigned int mode)` `[inline]`

Configure RS485 UART. Configure the RS485 UART parameters, including baud rate, data bits, stop bits, and parity.

**Parameters:**

*baud* The baud rate for the RS485 port. See [Hi-speed port baud rate constants](#).

*mode* The RS485 port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.732** `char RS485Write (byte buffer[])` `[inline]`

Write RS485 data. Write data to the RS485 hi-speed port.

**Parameters:**

*buffer* A byte array containing the data to write to the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Receive.nxc](#).

**8.3.3.733 char RunNRLinkMacro (const byte *port*, const byte *i2caddr*, const byte *macro*) [inline]**

Run NRLink macro. Run the specified mindsensors NRLink device macro. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*macro* The address of the macro to execute.

**Returns:**

The function call result.

**Examples:**

[ex\\_RunNRLinkMacro.nxc](#).

**8.3.3.734 char SendMessage (byte *queue*, string *msg*) [inline]**

Send a message to a queue/mailbox. Write a message into a local mailbox.

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#).

*msg* The message to write to the mailbox.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendMessage.nxc](#).

**8.3.3.735 char SendRemoteBool (byte *conn*, byte *queue*, bool *bval*)  
[inline]**

Send a boolean value to a remote mailbox. Send a boolean value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*bval* The boolean value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteBool.nxc](#).

**8.3.3.736 char SendRemoteNumber (byte *conn*, byte *queue*, long *val*)  
[inline]**

Send a numeric value to a remote mailbox. Send a numeric value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*val* The numeric value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteNumber.nxc](#).

**8.3.3.737 char SendRemoteString (byte *conn*, byte *queue*, string *str*) [inline]**

Send a string value to a remote mailbox. Send a string value on the specified connection to the specified remote mailbox number. Use [RemoteConnectionIdle](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..4). Connections 0 through 3 are for bluetooth connections. Connection 4 refers to the RS485 hi-speed port. See [Remote connection constants](#).

*queue* The mailbox number. See [Mailbox constants](#).

*str* The string value to send.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendRemoteString.nxc](#).

**8.3.3.738 char SendResponseBool (byte *queue*, bool *bval*) [inline]**

Write a boolean value to a local response mailbox. Write a boolean value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*bval* The boolean value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseBool.nxc](#).

**8.3.3.739 char SendResponseNumber (byte *queue*, long *val*) [inline]**

Write a numeric value to a local response mailbox. Write a numeric value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*val* The numeric value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseNumber.nxc](#).

**8.3.3.740 char SendResponseString (byte *queue*, string *str*) [inline]**

Write a string value to a local response mailbox. Write a string value to a response mailbox (the mailbox number + 10).

**Parameters:**

*queue* The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

*str* The string value to write.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

[ex\\_SendResponseString.nxc](#).

**8.3.3.741 char SendRS485Bool (bool *bval*) [inline]**

Write RS485 boolean. Write a boolean value to the RS485 hi-speed port.

**Parameters:**

*bval* A boolean value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.742 char SendRS485Number (long *val*) [inline]**

Write RS485 numeric. Write a numeric value to the RS485 hi-speed port.

**Parameters:**

*val* A numeric value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**8.3.3.743 char SendRS485String (string *str*) [inline]**

Write RS485 string. Write a string value to the RS485 hi-speed port.

**Parameters:**

*str* A string value to write over the RS485 port.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Send.nxc](#).

**8.3.3.744 unsigned int Sensor (const byte & *port*) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)).

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_Sensor.nxc](#), and [ex\\_SysComputeCalibValue.nxc](#).

**8.3.3.745 bool SensorBoolean (const byte *port*) [inline]**

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling SetSensorMode.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's boolean value.

**Examples:**

[ex\\_SensorBoolean.nxc](#).

**8.3.3.746 byte SensorDIAcclStatus (const byte *port*) [inline]**

SensorDIAcclStatus function. Read the Dexter Industries IMU Accl status value.

**Parameters:**

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The status value.

**Examples:**

[ex\\_diaccl.nxc](#).

**8.3.3.747 byte SensorDigiPinsDirection (const byte *port*) [inline]**

Read sensor digital pins direction. Return the digital pins direction value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins direction.

**Examples:**

[ex\\_SensorDigiPinsDirection.nxc](#).

**8.3.3.748 byte SensorDigiPinsOutputLevel (const byte *port*) [inline]**

Read sensor digital pins output level. Return the digital pins output level value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins output level.

**Examples:**

[ex\\_SensorDigiPinsOutputLevel.nxc](#).

**8.3.3.749 byte SensorDigiPinsStatus (const byte *port*) [inline]**

Read sensor digital pins status. Return the digital pins status value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's digital pins status.

**Examples:**

[ex\\_SensorDigiPinsStatus.nxc](#).

**8.3.3.750 long SensorDIGPSDistanceToWaypoint (byte *port*) [inline]**

SensorDIGPSDistanceToWaypoint function. Read the distance remaining to reach the current waypoint in meters.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The distance to the waypoint in meters

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.751 int SensorDIGPSHeading (byte *port*) [inline]**

SensorDIGPSHeading function. Read the current heading in degrees.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current heading in degrees

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.752 int SensorDIGPSHeadingToWaypoint (byte *port*) [inline]**

SensorDIGPSHeadingToWaypoint function. Read the heading required to reach the current waypoint.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The heading to the waypoint in degrees

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.753 long SensorDIGPSLatitude (byte *port*) [inline]**

SensorDIGPSLatitude function. Read the integer latitude reported by the GPS (ddddddd; Positive = North; Negative = South).

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The integer latitude

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.754 long SensorDIGPSLongitude (byte *port*) [inline]**

SensorDIGPSLongitude function. Read the integer longitude reported by the GPS (ddddddd; Positive = East; Negative = West).

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The integer longitude

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.755 int SensorDIGPSRelativeHeading (byte *port*) [inline]**

SensorDIGPSRelativeHeading function. Read the angle travelled since last request. Resets the request coordinates on the GPS sensor. Sends the angle of travel since the last call.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The relative heading in degrees

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.756 bool SensorDIGPSStatus (byte *port*) [inline]**

SensorDIGPSStatus function. Read the status of the GPS satellite link.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean GPS status

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.757 long SensorDIGPSTime (byte *port*) [inline]**

SensorDIGPSTime function. Read the current time reported by the GPS in UTC.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current time in UTC

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.758 long SensorDIGPSVelocity (byte *port*) [inline]**

SensorDIGPSVelocity function. Read the current velocity in cm/s.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The current velocity in cm/s

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.759 byte SensorDIGyroStatus (const byte *port*) [inline]**

SensorDIGyroStatus function. Read the Dexter Industries IMU Gyro status value.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The status value.

**Examples:**

[ex\\_digyro.nxc](#).

**8.3.3.760 int SensorDIGyroTemperature (const byte *port*) [inline]**

SensorDIGyroTemperature function. Read the Dexter Industries IMU Gyro temperature value.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The temperature value.

**Examples:**

[ex\\_digyro.nxc](#).

**8.3.3.761 int SensorHTColorNum (const byte & *port*) [inline]**

Read HiTechnic color sensor color number. Read the color number from the HiTechnic Color sensor on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The color number.

**Examples:**

[ex\\_SensorHTColorNum.nxc](#).

**8.3.3.762 int SensorHTCompass (const byte & port) [inline]**

Read HiTechnic compass. Read the compass heading value of the HiTechnic Compass sensor on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The compass heading.

**Examples:**

[ex\\_SensorHTCompass.nxc](#).

**8.3.3.763 int SensorHTEOPD (const byte & port) [inline]**

Read HiTechnic EOPD sensor. Read the HiTechnic EOPD sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The EOPD sensor reading.

**Examples:**

[ex\\_SensorHTEOPD.nxc](#).

**8.3.3.764 int SensorHTGyro (const byte & port, int offset = 0) [inline]**

Read HiTechnic Gyro sensor. Read the HiTechnic Gyro sensor on the specified port. The offset value should be calculated by averaging several readings with an offset of zero while the sensor is perfectly still.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The zero offset.

**Returns:**

The Gyro sensor reading.

**Examples:**

[ex\\_HTGyroTest.nxc](#), and [ex\\_SensorHTGyro.nxc](#).

**8.3.3.765 int SensorHTIRSeeker2ACDir (const byte & port) [inline]**

Read HiTechnic IRSeeker2 AC direction. Read the AC direction value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker2 AC direction.

**Examples:**

[ex\\_SensorHTIRSeeker2ACDir.nxc](#).

**8.3.3.766 int SensorHTIRSeeker2Addr (const byte & port, const byte reg) [inline]**

Read HiTechnic IRSeeker2 register. Read a register value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*reg* The register address. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The IRSeeker2 register value.

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**8.3.3.767 int SensorHTIRSeeker2DCDir (const byte & port) [inline]**

Read HiTechnic IRSeeker2 DC direction. Read the DC direction value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker2 DC direction.

**Examples:**

[ex\\_SensorHTIRSeeker2DCDir.nxc](#).

**8.3.3.768 int SensorHTIRSeekerDir (const byte & port) [inline]**

Read HiTechnic IRSeeker direction. Read the direction value of the HiTechnic IR Seeker on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The IRSeeker direction.

**Examples:**

[ex\\_SensorHTIRSeekerDir.nxc](#).

**8.3.3.769 int SensorHTMagnet (const byte & port, int offset = 0) [inline]**

Read HiTechnic Magnet sensor. Read the HiTechnic Magnet sensor on the specified port. The offset value should be calculated by averaging several readings with an offset of zero while the sensor is perfectly still.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*offset* The zero offset.

**Returns:**

The Magnet sensor reading.

**Examples:**

[ex\\_SensorHTMagnet.nxc](#).

**8.3.3.770 int SensorHTProtoAnalog (const byte port, const byte input) [inline]**

Read HiTechnic Prototype board analog input value. Read an analog input value from the HiTechnic prototype board. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*input* The analog input. See [HiTechnic Prototype board analog input constants](#).

**Returns:**

The analog input value.

**Examples:**

[ex\\_proto.nxc](#).

**8.3.3.771 byte SensorHTProtoDigital (const byte *port*) [inline]**

Read HiTechnic Prototype board digital input values. Read digital input values from the HiTechnic prototype board. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_proto.nxc](#).

**8.3.3.772 byte SensorHTProtoDigitalControl (const byte *port*) [inline]**

Read HiTechnic Prototype board digital control values. Read digital control values from the HiTechnic prototype board. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital control values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_proto.nxc](#).

**8.3.3.773 int SensorHTSuperProAnalog (const byte *port*, const byte *input*) [inline]**

Read HiTechnic SuperPro board analog input value. Read an analog input value from the HiTechnic SuperPro board. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*input* The analog input. See [HiTechnic SuperPro analog input index constants](#).

**Returns:**

The analog input value.

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.774 byte SensorHTSuperProDigital (const byte *port*) [inline]**

Read HiTechnic SuperPro board digital input values. Read digital input values from the HiTechnic SuperPro board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.775 byte SensorHTSuperProDigitalControl (const byte *port*) [inline]**

Read HiTechnic SuperPro board digital control values. Read digital control values from the HiTechnic SuperPro board. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The digital input values. See [SuperPro digital pin constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.776 byte SensorHTSuperProLED (const byte *port*) [inline]**

Read HiTechnic SuperPro LED value. Read the HiTechnic SuperPro LED value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The LED value. See [SuperPro LED control constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.777 byte SensorHTSuperProProgramControl (const byte *port*) [inline]**

Read HiTechnic SuperPro program control value. Read the HiTechnic SuperPro program control value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The program control value.

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.778** byte SensorHTSuperProStrobe (const byte *port*) [**inline**]

Read HiTechnic SuperPro strobe value. Read the HiTechnic SuperPro strobe value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The strobe value. See [SuperPro Strobe control constants](#).

**Examples:**

[ex\\_superpro.nxc](#).

**8.3.3.779** bool SensorInvalid (const byte & *port*) [**inline**]

Read sensor invalid data flag. Return the value of the InvalidData flag of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's invalid data flag.

**Examples:**

[ex\\_SensorInvalid.nxc](#).

**8.3.3.780** int SensorMIXG1300LScale (byte *port*) [**inline**]

SensorMIXG1300LScale function. Read the Microinfinity CruizCore XG1300L accelerometer scale. The accelerometer in the CruizCore XG1300L can be set to operate with a scale ranging from +/-2G, +/-4G, or +/-8G. Returns the scale value that the device is currently configured to use. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See the [Input port constants](#) group.

**Returns:**

The current scale value.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.3.3.781 byte SensorMode (const byte & port) [inline]**

Read sensor mode. Return the mode of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's mode. See [Sensor mode constants](#).

**Examples:**

[ex\\_SensorMode.nxc](#).

**8.3.3.782 int SensorMSCompass (const byte & port, const byte i2caddr) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors compass value

**Examples:**

[ex\\_SensorMSCompass.nxc](#).

**8.3.3.783 int SensorMSDROD (const byte & port) [inline]**

Read mindsensors DROD value. Return the Mindsensors DROD sensor value.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors DROD value

**Examples:**

[ex\\_SensorMSDROD.nxc](#).

**8.3.3.784 int SensorMSPressure (const byte & port) [inline]**

Read mindsensors pressure sensor. Read the pressure sensor value of the mindsensors pressure sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The pressure reading.

**Examples:**

[ex\\_SensorMSPressure.nxc](#).

**8.3.3.785 int SensorMSPressureRaw (const byte & port) [inline]**

Read mindsensors raw pressure value. Return the Mindsensors pressure sensor raw value.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors raw pressure value

**Examples:**

[ex\\_SensorMSPressureRaw.nxc](#).

**8.3.3.786 unsigned int SensorNormalized (const byte & port) [inline]**

Read sensor normalized value. Return the normalized value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's normalized value.

**Examples:**

[ex\\_SensorNormalized.nxc](#).

**8.3.3.787 char SensorNXTSumoEyes (const byte & port)**

Read mindsensors NXTSumoEyes obstacle zone. Return the Mindsensors NXTSumoEyes sensor obstacle zone value. The port should be configured for the NXTSumoEyes device using [SetSensorNXTSumoEyes](#) before calling this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors NXTSumoEyes obstacle zone value. See [MindSensors NXTSumoEyes constants](#).

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.3.3.788 int SensorNXTSumoEyesRaw (const byte & port) [inline]**

Read mindsensors NXTSumoEyes raw value. Return the Mindsensors NXTSumoEyes raw sensor value. The port should be configured for the NXTSumoEyes device using [SetSensorNXTSumoEyes](#) before calling this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Returns:**

The mindsensors NXTSumoEyes raw value

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.3.3.789 unsigned int SensorRaw (const byte & port) [inline]**

Read sensor raw value. Return the raw value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's raw value.

**Examples:**

[ex\\_SensorRaw.nxc](#).

**8.3.3.790 unsigned int SensorScaled (const byte & port) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)) or the [Sensor](#) function.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_SensorScaled.nxc](#).

**8.3.3.791 float SensorTemperature (const byte & port) [inline]**

Read the LEGO Temperature sensor value. Return the temperature sensor value in degrees celcius. Since a temperature sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. The port must be configured as a temperature sensor port before using this function. Use [SetSensorTemperature](#) to configure the port.

**Parameters:**

*port* The port to which the temperature sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The temperature sensor value in degrees celcius.

**Examples:**

[ex\\_SensorTemperature.nxc](#).

**8.3.3.792 byte SensorType (const byte & port) [inline]**

Read sensor type. Return the type of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's type. See [Sensor type constants](#).

**Examples:**

[ex\\_SensorType.nxc](#).

**8.3.3.793 byte SensorUS (const byte *port*) [inline]**

Read ultrasonic sensor value. Return the ultrasonic sensor distance value. Since an ultrasonic sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The ultrasonic sensor distance value (0..255)

**Examples:**

[ex\\_SensorUS.nxc](#).

**8.3.3.794 unsigned int SensorValue (const byte & *port*) [inline]**

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR\\_1](#)) or the [Sensor](#) function.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's scaled value.

**Examples:**

[ex\\_SensorValue.nxc](#).

**8.3.3.795 bool SensorValueBool (const byte *port*) [inline]**

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling SetSensorMode.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

**Returns:**

The sensor's boolean value.

**Examples:**

[ex\\_SensorValueBool.nxc](#).

**8.3.3.796 unsigned int SensorValueRaw (const byte & *port*) [inline]**

Read sensor raw value. Return the raw value of a sensor on the specified port.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

**Returns:**

The sensor's raw value.

**Examples:**

[ex\\_SensorValueRaw.nxc](#).

**8.3.3.797 void set\_fopen\_size (unsigned long *fsize*) [inline]**

Set the default fopen file size. Set the default size of a file created via a call to fopen.

**Parameters:**

*fsize* The default new file size for fopen.

**8.3.3.798 void SetAbortFlag (byte *abortFlag*) [inline]**

Set abort flag. Set the enhanced NBC/NXC firmware's program abort flag. By default the running program can be interrupted by a short press of the escape button. You can change this to any other button state flag.

**Parameters:**

*abortFlag* The new abort flag value. See [ButtonState constants](#)

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.3.3.799 char SetACCLNxSensitivity (const byte *port*, const byte *i2caddr*, byte *slevel*) [inline]**

Set ACCL-Nx sensitivity. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*slevel* The sensitivity level. See [MindSensors ACCL-Nx sensitivity level constants](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_SetACCLNxSensitivity.nxc](#).

**8.3.3.800 void SetBatteryState (byte *state*) [inline]**

Set battery state. Set battery state information.

**Parameters:**

*state* The desired battery state (0..4).

**Examples:**

[ex\\_SetBatteryState.nxc](#).

**8.3.3.801 void SetBluetoothState (byte *state*) [inline]**

Set bluetooth state. Set the Bluetooth state.

**Parameters:**

*state* The desired bluetooth state. See [BluetoothState constants](#).

**Examples:**

[ex\\_SetBluetoothState.nxc](#).

**8.3.3.802 void SetBTDataMode (const byte *dataMode*) [inline]**

Set Bluetooth data mode. This method sets the value of the Bluetooth data mode.

**Parameters:**

*dataMode* The Bluetooth data mode. See [Data mode constants](#). Must be a constant.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**8.3.3.803** void SetBTInputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]

Set bluetooth input buffer data. Write *cnt* bytes of data to the bluetooth input buffer at *offset*.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetBTInputBuffer.nxc](#).

**8.3.3.804** void SetBTInputBufferInPtr (byte *n*) [inline]

Set bluetooth input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetBTInputBufferInPtr.nxc](#).

**8.3.3.805** void SetBTInputBufferOutPtr (byte *n*) [inline]

Set bluetooth input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetBTInputBufferOutPtr.nxc](#).

**8.3.3.806** void SetBTOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]

Set bluetooth output buffer data. Write *cnt* bytes of data to the bluetooth output buffer at *offset*.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetBTOutputBuffer.nxc](#).

**8.3.3.807** void SetBTOutputBufferInPtr (byte *n*) [inline]

Set bluetooth output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetBTOutputBufferInPtr.nxc](#).

**8.3.3.808** void SetBTOutputBufferOutPtr (byte *n*) [inline]

Set bluetooth output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetBTOutputBufferOutPtr.nxc](#).

**8.3.3.809** void SetButtonLongPressCount (const byte *btn*, const byte *n*)  
[inline]

Set button long press count. Set the long press count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new long press count value.

**Examples:**

[ex\\_SetButtonLongPressCount.nxc](#).

**8.3.3.810** void SetButtonLongReleaseCount (const byte *btn*, const byte *n*)  
[inline]

Set button long release count. Set the long release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new long release count value.

**Examples:**

[ex\\_SetButtonLongReleaseCount.nxc](#).

**8.3.3.811** void SetButtonModuleValue (unsigned int *offset*, variant *value*)  
[inline]

Set Button module IOMap value. Set one of the fields of the Button module IOMap structure to a new value. You provide the offset into the Button module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Button module IOMap structure where the new value should be written. See [Button module IOMAP offsets](#).

*value* A variable containing the new value to write to the Button module IOMap.

**8.3.3.812 void SetButtonPressCount (const byte *btn*, const byte *n*) [inline]**

Set button press count. Set the press count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new press count value.

**Examples:**

[ex\\_SetButtonPressCount.nxc](#).

**8.3.3.813 void SetButtonReleaseCount (const byte *btn*, const byte *n*) [inline]**

Set button release count. Set the release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new release count value.

**Examples:**

[ex\\_SetButtonReleaseCount.nxc](#).

**8.3.3.814 void SetButtonShortReleaseCount (const byte *btn*, const byte *n*) [inline]**

Set button short release count. Set the short release count of the specified button.

**Parameters:**

*btn* The button number. See [Button name constants](#).

*n* The new short release count value.

**Examples:**

[ex\\_SetButtonShortReleaseCount.nxc](#).

**8.3.3.815 void SetButtonState (const byte *btn*, const byte *state*) [inline]**

Set button state. Set the state of the specified button.

**Parameters:**

*btn* The button to check. See [Button name constants](#).

*state* The new button state. See [ButtonState constants](#).

**Examples:**

[ex\\_SetButtonState.nxc](#).

**8.3.3.816 void SetCommandFlags (const byte *cmdFlags*) [inline]**

Set command flags. Set the command flags.

**Parameters:**

*cmdFlags* The new command flags. See [CommandFlags constants](#).

**Examples:**

[ex\\_SetCommandFlags.nxc](#).

**8.3.3.817 void SetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]**

Set Command module IOMap bytes. Modify one or more bytes of data in the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the data should be written. See [Command module IOMAP offsets](#).

*count* The number of bytes to write at the specified Command module IOMap offset.

*data* The byte array containing the data to write to the Command module IOMap.

**8.3.3.818 void SetCommandModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Command module IOMap value. Set one of the fields of the Command module IOMap structure to a new value. You provide the offset into the Command module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Command module IOMap structure where the new value should be written. See [Command module IOMAP offsets](#).

*value* A variable containing the new value to write to the Command module IOMap.

**8.3.3.819 void SetCommModuleBytes (unsigned int *offset*, unsigned int *count*,  
byte *data*[]) [inline]**

Set Comm module IOMap bytes. Modify one or more bytes of data in an IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the data should be written. See [Comm module IOMAP offsets](#).

*count* The number of bytes to write at the specified Comm module IOMap offset.

*data* The byte array containing the data to write to the Comm module IOMap.

**8.3.3.820 void SetCommModuleValue (unsigned int *offset*, variant *value*)  
[inline]**

Set Comm module IOMap value. Set one of the fields of the Comm module IOMap structure to a new value. You provide the offset into the Comm module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Comm module IOMap structure where the new value should be written. See [Comm module IOMAP offsets](#).

*value* A variable containing the new value to write to the Comm module IOMap.

**8.3.3.821** `void SetCustomSensorActiveStatus (byte port, byte activeStatus)  
[inline]`

Set active status. Sets the active status value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*activeStatus* The new active status value.

**Examples:**

[ex\\_SetCustomSensorActiveStatus.nxc](#).

**8.3.3.822** `void SetCustomSensorPercentFullScale (byte port, byte pctFullScale)  
[inline]`

Set percent full scale. Sets the percent full scale value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*pctFullScale* The new percent full scale value.

**Examples:**

[ex\\_SetCustomSensorPercentFullScale.nxc](#).

**8.3.3.823** `void SetCustomSensorZeroOffset (byte port, int zeroOffset)  
[inline]`

Set custom zero offset. Sets the zero offset value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*zeroOffset* The new zero offset value.

**Examples:**

[ex\\_SetCustomSensorZeroOffset.nxc](#).

**8.3.3.824 void SetDisplayContrast (byte *contrast*) [inline]**

Set the display contrast. This function lets you set the display contrast setting.

**Parameters:**

*contrast* The desired display contrast.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_contrast.nxc](#), and [ex\\_setdisplaycontrast.nxc](#).

**8.3.3.825 void SetDisplayDisplay (unsigned long *dispaddr*) [inline]**

Set the display memory address. This function lets you set the current display memory address.

**Parameters:**

*dispaddr* The new display memory address.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayDisplay.nxc](#).

**8.3.3.826 void SetDisplayEraseMask (unsigned long *eraseMask*) [inline]**

Set the display erase mask. This function lets you set the current display erase mask.

**Parameters:**

*eraseMask* The new display erase mask.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayEraseMask.nxc](#).

**8.3.3.827 void SetDisplayFlags (byte *flags*) [inline]**

Set the display flags. This function lets you set the current display flags.

**Parameters:**

*flags* The new display flags. See [Display flags](#).

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayFlags.nxc](#).

**8.3.3.828 void SetDisplayFont (unsigned long *fontaddr*) [inline]**

Set the display font memory address. This function lets you set the current display font memory address.

**Parameters:**

*fontaddr* The new display font memory address.

**Examples:**

[ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_displayfont.nxc](#), and [ex\\_setdisplayfont.nxc](#).

**8.3.3.829** void SetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Display module IOMap bytes. Modify one or more bytes of data in the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the data should be written. See [Display module IOMAP offsets](#).

*count* The number of bytes to write at the specified Display module IOMap offset.

*data* The byte array containing the data to write to the Display module IOMap.

**8.3.3.830** void SetDisplayModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Display module IOMap value. Set one of the fields of the Display module IOMap structure to a new value. You provide the offset into the Display module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Display module IOMap structure where the new value should be written. See [Display module IOMAP offsets](#).

*value* A variable containing the new value to write to the Display module IOMap.

**8.3.3.831** void SetDisplayNormal (const byte *x*, const byte *line*, unsigned int *cnt*, byte *data*[]) [inline]

Write pixel data to the normal display buffer. Write "cnt" bytes to the normal display memory from the data array. Start writing at the specified x, line coordinate. Each byte of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

- x* The desired x position where you wish to write pixel data.
- line* The desired line where you wish to write pixel data.
- cnt* The number of bytes of pixel data to write.
- data* The array of bytes from which pixel data is read.

**Examples:**

[ex\\_SetDisplayNormal.nxc](#).

**8.3.3.832 void SetDisplayPopup (const byte *x*, const byte *line*, unsigned int *cnt*, byte *data*[ ]) [inline]**

Write pixel data to the popup display buffer. Write "cnt" bytes to the popup display memory from the data array. Start writing at the specified x, line coordinate. Each byte of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE\_1 through TEXTLINE\_8 for the "line" parameter.

**Parameters:**

- x* The desired x position where you wish to write pixel data.
- line* The desired line where you wish to write pixel data.
- cnt* The number of bytes of pixel data to write.
- data* The array of bytes from which pixel data is read.

**Examples:**

[ex\\_SetDisplayPopup.nxc](#).

**8.3.3.833 void SetDisplayTextLinesCenterFlags (byte *ctrFlags*) [inline]**

Set the display text lines center flags. This function lets you set the current display text lines center flags.

**Parameters:**

- ctrFlags* The new display text lines center flags.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayTextLinesCenterFlags.nxc](#).

**8.3.3.834** void SetDisplayUpdateMask (unsigned long *updateMask*)  
[inline]

Set the display update mask. This function lets you set the current display update mask.

**Parameters:**

*updateMask* The new display update mask.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayUpdateMask.nxc](#).

**8.3.3.835** void SetHSAddress (byte *hsAddress*) [inline]

Set hi-speed port address. This method sets the value of the hi-speed port address.

**Parameters:**

*hsAddress* The hi-speed port address. See [Hi-speed port address constants](#).

**8.3.3.836** void SetHSDataMode (const byte *dataMode*) [inline]

Set hi-speed port data mode. This method sets the value of the hi-speed port data mode.

**Parameters:**

*dataMode* The hi-speed port data mode. See [Data mode constants](#). Must be a constant.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_DataMode.nxc](#).

**8.3.3.837 void SetHSFlags (byte *hsFlags*) [inline]**

Set hi-speed port flags. This method sets the value of the hi-speed port flags.

**Parameters:**

*hsFlags* The hi-speed port flags. See [Hi-speed port flags constants](#).

**Examples:**

[ex\\_SetHSFlags.nxc](#).

**8.3.3.838 void SetHSInputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]**

Set hi-speed port input buffer data. Write *cnt* bytes of data to the hi-speed port input buffer at *offset*.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetHSInputBuffer.nxc](#).

**8.3.3.839 void SetHSInputBufferIntPtr (byte *n*) [inline]**

Set hi-speed port input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetHSInputBufferIntPtr.nxc](#).

**8.3.3.840 void SetHSInputBufferOutPtr (byte *n*) [inline]**

Set hi-speed port input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetHSInputBufferOutPtr.nxc](#).

**8.3.3.841 void SetHSMode (unsigned int *hsMode*) [inline]**

Set hi-speed port mode. This method sets the value of the hi-speed port mode.

**Parameters:**

*hsMode* The hi-speed port mode (data bits, stop bits, parity). See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sethsmode.nxc](#).

**8.3.3.842 void SetHSOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set hi-speed port output buffer data. Write *cnt* bytes of data to the hi-speed port output buffer at *offset*.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetHSOutputBuffer.nxc](#).

**8.3.3.843 void SetHSOutputBufferInPtr (byte *n*) [inline]**

Set hi-speed port output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..127).

**Examples:**

[ex\\_SetHSOutputBufferInPtr.nxc](#).

**8.3.3.844 void SetHSOutputBufferOutPtr (byte *n*) [inline]**

Set hi-speed port output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..127).

**Examples:**

[ex\\_SetHSOutputBufferOutPtr.nxc](#).

**8.3.3.845 void SetHSSpeed (byte *hsSpeed*) [inline]**

Set hi-speed port speed. This method sets the value of the hi-speed port speed (baud rate).

**Parameters:**

*hsSpeed* The hi-speed port speed (baud rate). See [Hi-speed port baud rate constants](#).

**Examples:**

[ex\\_SetHSSpeed.nxc](#).

**8.3.3.846 void SetHSState (byte *hsState*) [inline]**

Set hi-speed port state. This method sets the value of the hi-speed port state.

**Parameters:**

*hsState* The hi-speed port state. See [Hi-speed port state constants](#).

**Examples:**

[ex\\_SetHSState.nxc](#).

**8.3.3.847 bool SetHTBarometricCalibration (byte *port*, unsigned int *cal*) [inline]**

Set HiTechnic Barometric sensor calibration. Set the HiTechnic Barometric sensor pressure calibration value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*cal* The new pressure calibration value.

**Returns:**

The function call result.

**8.3.3.848 char SetHTColor2Mode (const byte & *port*, byte *mode*) [inline]**

Set HiTechnic Color2 mode. Set the mode of the HiTechnic Color2 sensor on the specified port. The port must be configured as a Low-speed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The Color2 mode. See [HiTechnic Color2 constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_sethtcolor2mode.nxc](#).

**8.3.3.849 char SetHTIRSeeker2Mode (const byte & port, const byte mode) [inline]**

Set HiTechnic IRSeeker2 mode. Set the mode of the HiTechnic IRSeeker2 sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*mode* The IRSeeker2 mode. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The function call result. [NO\\_ERR](#) or [Communications specific errors](#).

**Examples:**

[ex\\_sethtirseeker2mode.nxc](#), and [ex\\_setsensorboolean.nxc](#).

**8.3.3.850 void SetI2COptions (byte port, byte options) [inline]**

Set I2C options. This method lets you modify I2C options. Use this function to turn on or off the fast I2C mode and also control whether the standard I2C mode performs a restart prior to the read operation.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*port* The port whose I2C options you wish to change. See the [Input port constants](#) group. You may use a constant or a variable.

*options* The new option value. See [I2C option constants](#).

**8.3.3.851** `void SetInput (const byte & port, const int field, variant value)`  
[inline]

Set an input field value. Set the specified field of the sensor on the specified port to the value provided.

**Parameters:**

*port* The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

*field* An input field constant. See [Input field constants](#).

*value* The new value, which may be any valid expression.

**Examples:**

[ex\\_SetInput.nxc](#).

**8.3.3.852** `void SetInputModuleValue (unsigned int offset, variant value)`  
[inline]

Set Input module IOMap value. Set one of the fields of the Input module IOMap structure to a new value. You provide the offset into the Input module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Input module IOMap structure where the new value should be written. See [Input module IOMAP offsets](#).

*value* A variable containing the new value to write to the Input module IOMap.

**8.3.3.853** void SetIOCtrlModuleValue (unsigned int *offset*, variant *value*)  
[inline]

Set IOCtrl module IOMap value. Set one of the fields of the IOCtrl module IOMap structure to a new value. You provide the offset into the IOCtrl module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the IOCtrl module IOMap structure where the new value should be written. See [IOCtrl module IOMAP offsets](#).

*value* A variable containing the new value to write to the IOCtrl module IOMap.

**8.3.3.854** void SetIOMapBytes (string *moduleName*, unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set IOMap bytes by name. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*moduleName* The module name of the IOMap to modify. See [NXT firmware module names](#).

*offset* The number of bytes offset from the start of the IOMap structure where the data should be written

*count* The number of bytes to write at the specified IOMap offset.

*data* The byte array containing the data to write to the IOMap

**8.3.3.855** void SetIOMapBytesByID (unsigned long *moduleId*, unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set IOMap bytes by ID. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

- moduleId* The module ID of the IOMap to modify. See [NXT firmware module IDs](#).
- offset* The number of bytes offset from the start of the IOMap structure where the data should be written.
- count* The number of bytes to write at the specified IOMap offset.
- data* The byte array containing the data to write to the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.856** `void SetIOMapValue (string moduleName, unsigned int offset, variant value) [inline]`

Set IOMap value by name. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

- moduleName* The module name of the IOMap to modify. See [NXT firmware module names](#).
- offset* The number of bytes offset from the start of the IOMap structure where the new value should be written
- value* A variable containing the new value to write to the IOMap

**8.3.3.857** `void SetIOMapValueByID (unsigned long moduleId, unsigned int offset, variant value) [inline]`

Set IOMap value by ID. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

- moduleId* The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

*offset* The number of bytes offset from the start of the IOMap structure where the new value should be written.

*value* A variable containing the new value to write to the IOMap.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.858 void SetLoaderModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Loader module IOMap value. Set one of the fields of the Loader module IOMap structure to a new value. You provide the offset into the Loader module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Loader module IOMap structure where the new value should be written. See [Loader module IOMAP offsets](#).

*value* A variable containing the new value to write to the Loader module IOMap.

**8.3.3.859 void SetLongAbort (bool *longAbort*) [inline]**

Set long abort. Set the enhanced NBC/NXC firmware's long abort setting (true or false). If set to true then a program has access the escape button. Aborting a program requires a long press of the escape button.

**Parameters:**

*longAbort* If true then require a long press of the escape button to abort a program, otherwise a short press will abort it.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_buttonpressed.nxc](#), [ex\\_getchar.nxc](#), [ex\\_SetAbortFlag.nxc](#), and [ex\\_SetLongAbort.nxc](#).

**8.3.3.860** void SetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Lowspeed module IOMap bytes. Modify one or more bytes of data in the Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

**Parameters:**

*offset* The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be written. See [Low speed module IOMAP offsets](#).

*count* The number of bytes to write at the specified Lowspeed module IOMap offset.

*data* The byte array containing the data to write to the Lowspeed module IOMap.

**8.3.3.861** void SetLowSpeedModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Lowspeed module IOMap value. Set one of the fields of the Lowspeed module IOMap structure to a new value. You provide the offset into the Lowspeed module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Lowspeed module IOMap structure where the new value should be written. See [Low speed module IOMAP offsets](#).

*value* A variable containing the new value to write to the Lowspeed module IOMap.

**8.3.3.862** void SetMotorPwnFreq (byte *n*) [inline]

Set motor regulation frequency. Set the motor regulation frequency in milliseconds. By default this is set to 100ms.

**Parameters:**

*n* The motor regulation frequency.

**Examples:**

[ex\\_SetMotorPwnFreq.nxc](#).

**8.3.3.863 void SetMotorRegulationOptions (byte *n*) [inline]**

Set regulation options. Set the motor regulation options.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Parameters:**

*n* The motor regulation options.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.864 void SetMotorRegulationTime (byte *n*) [inline]**

Set regulation time. Set the motor regulation time in milliseconds. By default this is set to 100ms.

**Parameters:**

*n* The motor regulation time.

**Examples:**

[ex\\_PosReg.nxc](#).

**8.3.3.865 char SetNXTLineLeaderKdFactor (const byte & *port*, const byte & *i2caddr*, const byte & *value*) [inline]**

Write NXTLineLeader Kd factor. Write a Kd divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Kd value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Kd factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.866 char SetNXTLineLeaderKdValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kd value. Write a Kd value to the NXTLineLeader device. This value divided by PID Factor for Kd is the Derivative value for the PID control. Suggested value is 8 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Kd value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.867 char SetNXTLineLeaderKiFactor (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Ki factor. Write a Ki divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Ki value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Ki factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.868 char SetNXTLineLeaderKiValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Ki value. Write a Ki value to the NXTLineLeader device. This value divided by PID Factor for Ki is the Integral value for the PID control. Suggested value is 0 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- value* The new Ki value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.869 char SetNXTLineLeaderKpFactor (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kp factor. Write a Kp divisor factor to the NXTLineLeader device. Value ranges between 1 and 255. Change this value if you need more granularities in Kp value. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Kp factor (1..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.870 char SetNXTLineLeaderKpValue (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader Kp value. Write a Kp value to the NXTLineLeader device. This value divided by PID Factor for Kp is the Proportional value for the PID control. Suggested value is 25 with a divisor factor of 32 (which is also a factory default), start with this value, and tune it to meet your needs. Value ranges between 0 and 255. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new Kp value (0..255).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.871 char SetNXTLineLeaderSetpoint (const byte & port, const byte & i2caddr, const byte & value) [inline]**

Write NXTLineLeader setpoint. Write a new setpoint value to the NXTLineLeader device. The Set Point is a value you can ask sensor to maintain the average to. The default value is 45, whereby the line is maintained in center of the sensor. If you need to maintain line towards left of the sensor, set the Set Point to a lower value (minimum: 10). If you need it to be towards on the right of the sensor, set it to higher value (maximum: 80). Set point is also useful while tracking an edge of dark and light areas. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*value* The new setpoint value (10..80).

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible Result values.

**Examples:**

[ex\\_NXTLineLeader.nxc](#).

**8.3.3.872 char SetNXTServoPosition (const byte & port, const byte & i2caddr, const byte servo, const byte & pos) [inline]**

Set NXTServo servo motor position. Set the position of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

*pos* The servo position. See [MindSensors NXTServo position constants](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.873 char SetNXTServoQuickPosition (const byte & port, const byte & i2caddr, const byte servo, const byte & qpos) [inline]**

Set NXTServo servo motor quick position. Set the quick position of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [NBC Input port constants](#).

*i2caddr* The sensor I2C address. See sensor documentation for this value.

*servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.

*qpos* The servo quick position. See [MindSensors NXTServo quick position constants](#) group.

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.874 char SetNXTServoSpeed (const byte & port, const byte & i2caddr, const byte servo, const byte & speed) [inline]**

Set NXTServo servo motor speed. Set the speed of a servo motor controlled by the NXTServo device. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [NBC Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- servo* The servo number. See [MindSensors NXTServo servo numbers](#) group.
- speed* The servo speed. (0..255)

**Returns:**

A status code indicating whether the operation completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_NXTServo.nxc](#).

**8.3.3.875 void SetOnBrickProgramPointer (byte *obpStep*) [inline]**

Set on-brick program pointer. Set the current OBP (on-brick program) step.

**Parameters:**

- obpStep* The new on-brick program step.

**Examples:**

[ex\\_SetOnBrickProgramPointer.nxc](#).

**8.3.3.876 void SetOutput (byte *outputs*, byte *field1*, variant *val1*, ..., byte *fieldN*, variant *valN*) [inline]**

Set output fields. Set the specified field of the outputs to the value provided. The field must be a valid output field constant. This function takes a variable number of field/value pairs.

**Parameters:**

- outputs* Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

*field1* The 1st output port field to access, this should be a constant, see [Output field constants](#).

*val1* Value to set for the 1st field.

*fieldN* The Nth output port field to access, this should be a constant, see [Output field constants](#).

*valN* The value to set for the Nth field.

**Examples:**

[ex\\_setoutput.nxc](#).

**8.3.3.877** `void SetOutputModuleValue (unsigned int offset, variant value)`  
`[inline]`

Set Output module IOMap value. Set one of the fields of the Output module IOMap structure to a new value. You provide the offset into the Output module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Output module IOMap structure where the new value should be written. See [Output module IOMAP offsets](#).

*value* A variable containing the new value to write to the Output module IOMap.

**8.3.3.878** `void SetSensor (const byte & port, const unsigned int config)`  
`[inline]`

Set sensor configuration. Set the type and mode of the given sensor to the specified configuration, which must be a special constant containing both type and mode information.

**See also:**

[SetSensorType\(\)](#), [SetSensorMode\(\)](#), and [ResetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*config* The configuration constant containing both the type and mode. See [Combined sensor type and mode constants](#).

**Examples:**

[ex\\_SetSensor.nxc](#).

**8.3.3.879 void SetSensorBoolean (byte *port*, bool *value*) [inline]**

Set sensor boolean value. Sets the boolean value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The new boolean value.

**8.3.3.880 void SetSensorColorBlue (const byte & *port*) [inline]**

Configure an NXT 2.0 blue light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in blue light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorblue.nxc](#).

**8.3.3.881 void SetSensorColorFull (const byte & *port*) [inline]**

Configure an NXT 2.0 full color sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in full color mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorfull.nxc](#), and [ex\\_SysColorSensorRead.nxc](#).

**8.3.3.882 void SetSensorColorGreen (const byte & port) [inline]**

Configure an NXT 2.0 green light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in green light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorgreen.nxc](#).

**8.3.3.883 void SetSensorColorNone (const byte & port) [inline]**

Configure an NXT 2.0 no light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in no light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolornone.nxc](#).

**8.3.3.884 void SetSensorColorRed (const byte & port) [inline]**

Configure an NXT 2.0 red light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in red light mode. Requires an NXT 2.0 compatible firmware.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_setsensorcolorred.nxc](#).

**8.3.3.885 bool SetSensorDIAccl (const byte port) [inline]**

SetSensorDIAccl function. Configure DIAccl device on the specified port with default mode of 2G.

**Parameters:**

*port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**8.3.3.886 bool SetSensorDIAcclDrift (const byte port, int x, int y, int z) [inline]**

SetSensorDIAcclDrift function. Set the Dexter Industries IMU Accl X, Y, and Z axis 10-bit drift values.

**Parameters:**

- port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- x* The X axis 10-bit drift value.
- y* The Y axis 10-bit drift value.
- z* The Z axis 10-bit drift value.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**8.3.3.887 bool SetSensorDIAcclEx (const byte *port*, byte *mode*) [inline]**

SetSensorDIAcclEx function. Configure DIAccl device on the specified port with the specified mode.

**Parameters:**

- port* The port to which the Dexter Industries IMU Accl sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- mode* The mode of the device (2G, 4G, or 8G). See the [Dexter Industries IMU Accelerometer mode control register constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_diaccl.nxc](#).

**8.3.3.888 void SetSensorDigiPinsDirection (byte *port*, byte *direction*) [inline]**

Set digital pins direction. Sets the digital pins direction value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*direction* The new digital pins direction value.

**Examples:**

[ex\\_SetSensorDigiPinsDirection.nxc](#).

**8.3.3.889** void SetSensorDigiPinsOutputLevel (byte *port*, byte *outputLevel*)  
[inline]

Set digital pins output level. Sets the digital pins output level value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*outputLevel* The new digital pins output level value.

**Examples:**

[ex\\_SetSensorDigiPinsOutputLevel.nxc](#).

**8.3.3.890** void SetSensorDigiPinsStatus (byte *port*, byte *status*) [inline]

Set digital pins status. Sets the digital pins status value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*status* The new digital pins status value.

**Examples:**

[ex\\_SetSensorDigiPinsStatus.nxc](#).

**8.3.3.891** `bool SetSensorDIGPSWaypoint (byte port, long latitude, long longitude) [inline]`

SetSensorDIGPSWaypoint function. Set the coordinates of the waypoint destination. The GPS sensor uses this to calculate the heading and distance required to reach the waypoint.

**Parameters:**

*port* The port to which the Dexter Industries GPS sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*latitude* The latitude of the waypoint.

*longitude* The longitude of the waypoint.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digps.nxc](#).

**8.3.3.892** `bool SetSensorDIGyro (const byte port) [inline]`

SetSensorDIGyro function. Configure DIGyro device on the specified port with default scale of 500dps, output data rate of 100hz, and bandwidth level 1.

**Parameters:**

*port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digyro.nxc](#).

**8.3.3.893** `bool SetSensorDIGyroEx (const byte port, byte scale, byte odr, byte bw) [inline]`

SetSensorDIGyroEx function. Configure DIGyro device on the specified port with the specified scale, output data rate, and bandwidth.

**Parameters:**

- port* The port to which the Dexter Industries IMU Gyro sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- scale* The full scale of the device (250dps, 500dps, or 2000dps). See the [Dexter Industries IMU Gyro control register 4 constants](#) group. You may use a constant or a variable.
- odr* The output data rate of the device (100hz, 200hz, 400hz, or 800hz). See the [Dexter Industries IMU Gyro control register 1 constants](#) group. You may use a constant or a variable.
- bw* The bandwidth of the device. See the [Dexter Industries IMU Gyro control register 1 constants](#) group. You may use a constant or a variable.

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_digyro.nxc](#).

**8.3.3.894** `void SetSensorEMeter (const byte & port) [inline]`

Configure an EMeter sensor. Configure the sensor on the specified port as an EMeter sensor.

**Parameters:**

- port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorEMeter.nxc](#).

**8.3.3.895** void SetSensorHTEOPD (const byte & *port*, bool *bStandard*)  
[inline]

Set sensor as HiTechnic EOPD. Configure the sensor on the specified port as a HiTechnic EOPD sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*bStandard* Configure in standard or long-range mode.

**Examples:**

[ex\\_setsensorhteopd.nxc](#).

**8.3.3.896** void SetSensorHTGyro (const byte & *port*) [inline]

Set sensor as HiTechnic Gyro. Configure the sensor on the specified port as a HiTechnic Gyro sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Examples:**

[ex\\_HTGyroTest.nxc](#), [ex\\_SensorHTGyro.nxc](#), and [ex\\_SetSensorHTGyro.nxc](#).

**8.3.3.897** void SetSensorHTMagnet (const byte & *port*) [inline]

Set sensor as HiTechnic Magnet. Configure the sensor on the specified port as a HiTechnic Magnet sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorHTMagnet.nxc](#).

**8.3.3.898 bool SetSensorHTProtoDigital (const byte *port*, byte *value*)  
[inline]**

Set HiTechnic Prototype board digital output values. Set the digital pin output values on the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The digital pin output values. See [SuperPro digital pin constants](#).

**Returns:**

The function call result.

**8.3.3.899 bool SetSensorHTProtoDigitalControl (const byte *port*, byte *value*)  
[inline]**

Control HiTechnic Prototype board digital pin direction. Control the direction of the six digital pins on the HiTechnic prototype board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The digital pin control value. See [SuperPro digital pin constants](#). OR into this value the pins that you want to be output pins. The pins not included in the value will be input pins.

**Returns:**

The function call result.

**8.3.3.900 bool SetSensorHTSuperProAnalogOut (const byte *port*, const byte *dac*, byte *mode*, int *freq*, int *volt*) [inline]**

Set HiTechnic SuperPro board analog output parameters. Set the analog output parameters on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- dac* The analog output index. See [HiTechnic SuperPro analog output index constants](#).
- mode* The analog output mode. See [SuperPro analog output mode constants](#).
- freq* The analog output frequency. Between 1 and 8191.
- volt* The analog output voltage level. A 10 bit value (0..1023).

**Returns:**

The function call result.

**8.3.3.901 bool SetSensorHTSuperProDigital (const byte *port*, byte *value*) [inline]**

Set HiTechnic SuperPro board digital output values. Set the digital pin output values on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- value* The digital pin output values. See [SuperPro digital pin constants](#).

**Returns:**

The function call result.

**8.3.3.902 bool SetSensorHTSuperProDigitalControl (const byte *port*, byte *value*) [inline]**

Control HiTechnic SuperPro board digital pin direction. Control the direction of the eight digital pins on the HiTechnic SuperPro board. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The digital pin control value. See [SuperPro digital pin constants](#). OR into this value the pins that you want to be output pins. The pins not included in the value will be input pins.

**Returns:**

The function call result.

**8.3.3.903 bool SetSensorHTSuperProLED (const byte *port*, byte *value*)  
[inline]**

Set HiTechnic SuperPro LED value. Set the HiTechnic SuperPro LED value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The LED value. See [SuperPro LED control constants](#).

**Returns:**

The function call result.

**8.3.3.904 bool SetSensorHTSuperProProgramControl (const byte *port*, byte *value*) [inline]**

Set HiTechnic SuperPro program control value. Set the HiTechnic SuperPro program control value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See [Input port constants](#).

*value* The program control value.

**Returns:**

The function call result.

**8.3.3.905** `bool SetSensorHTSuperProStrobe (const byte port, byte value)`  
[inline]

Set HiTechnic SuperPro strobe value. Set the HiTechnic SuperPro strobe value. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).  
*value* The strobe value. See [SuperPro Strobe control constants](#).

**Returns:**

The function call result.

**8.3.3.906** `void SetSensorLight (const byte & port, bool bActive = true)`  
[inline]

Configure a light sensor. Configure the sensor on the specified port as an NXT light sensor.

**Parameters:**

- port* The port to configure. See [Input port constants](#).  
*bActive* A boolean flag indicating whether to configure the port as an active or inactive light sensor. The default value for this optional parameter is true.

**Examples:**

[ex\\_SetSensorLight.nxc](#).

**8.3.3.907** `void SetSensorLowspeed (const byte & port, bool bIsPowered = true)`  
[inline]

Configure an I2C sensor. Configure the sensor on the specified port as an I2C digital sensor for either powered (9 volt) or unpowered devices.

**Parameters:**

- port* The port to configure. See [Input port constants](#).

***IsPowered*** A boolean flag indicating whether to configure the port for powered or unpowered I2C devices. The default value for this optional parameter is true.

**Examples:**

[ex\\_digps.nxc](#), [ex\\_HTRCXSetIRLinkPort.nxc](#), [ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_PFMate.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_ResetSensorHTAngle.nxc](#), [ex\\_SetSensorLowspeed.nxc](#), [ex\\_superpro.nxc](#), and [ex\\_xg1300.nxc](#).

**8.3.3.908 bool SetSensorMIXG1300LScale (byte *port*, const byte *scale*) [inline]**

SetSensorMIXG1300LScale function. Set the Microinfinity CruzCore XG1300L accelerometer scale. The accelerometer in the CruzCore XG1300L can be set to operate with a scale ranging from +/-2G, +/-4G, or +/-8G. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

*port* The sensor port. See the [Input port constants](#) group.

*scale* This value must be a constant. See [Microinfinity CruzCore XG1300L](#).

**Returns:**

The boolean function call result.

**Examples:**

[ex\\_xg1300.nxc](#).

**8.3.3.909 void SetSensorMode (const byte & *port*, byte *mode*) [inline]**

Set sensor mode. Set a sensor's mode, which should be one of the predefined sensor mode constants. A slope parameter for boolean conversion, if desired, may be added to the mode. After changing the type or the mode of a sensor port you must call [ResetSensor](#) to give the firmware time to reconfigure the sensor port.

**See also:**

[SetSensorType\(\)](#), [SetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*mode* The desired sensor mode. See [Sensor mode constants](#).

**Examples:**

[ex\\_SetSensorMode.nxc](#).

**8.3.3.910 void SetSensorMSDROD (const byte & port, bool bActive) [inline]**

Configure a mindsensors DROD sensor. Configure the specified port for a mindsensors DROD sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bActive* A flag indicating whether to configure the sensor in active or inactive mode.

**Examples:**

[ex\\_setsensormsdrod.nxc](#).

**8.3.3.911 void SetSensorMSPressure (const byte & port) [inline]**

Configure a mindsensors pressure sensor. Configure the specified port for a mindsensors pressure sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_setsensormspressure.nxc](#).

**8.3.3.912** `void SetSensorNXTSumoEyes (const byte & port, bool bLong)`  
[inline]

Configure a mindsensors SumoEyes sensor. Configure the specified port for a mind-sensors SumoEyes sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bLong* A flag indicating whether to configure the sensor in long range or short range mode.

**Examples:**

[ex\\_NXTSumoEyes.nxc](#).

**8.3.3.913** `void SetSensorSound (const byte & port, bool bdBScaling = true)`  
[inline]

Configure a sound sensor. Configure the sensor on the specified port as a sound sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*bdBScaling* A boolean flag indicating whether to configure the port as a sound sensor with dB or dBA scaling. The default value for this optional parameter is true, meaning dB scaling.

**Examples:**

[ex\\_SetSensorSound.nxc](#).

**8.3.3.914** `void SetSensorTemperature (const byte & port)` [inline]

Configure a temperature sensor. Configure the sensor on the specified port as a temperature sensor. Use this to setup the temperature sensor rather than [SetSensorLowspeed](#) so that the sensor is properly configured in 12-bit conversion mode.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorTemperature.nxc](#).

**8.3.3.915 void SetSensorTouch (const byte & port) [inline]**

Configure a touch sensor. Configure the sensor on the specified port as a touch sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_ReadSensorHTTouchMultiplexer.nxc](#), and [ex\\_SetSensorTouch.nxc](#).

**8.3.3.916 void SetSensorType (const byte & port, byte type) [inline]**

Set sensor type. Set a sensor's type, which must be one of the predefined sensor type constants. After changing the type or the mode of a sensor port you must call [Reset-Sensor](#) to give the firmware time to reconfigure the sensor port.

**See also:**

[SetSensorMode\(\)](#), [SetSensor\(\)](#)

**Parameters:**

*port* The port to configure. See [Input port constants](#).

*type* The desired sensor type. See [Sensor type constants](#).

**Examples:**

[ex\\_SetSensorType.nxc](#).

**8.3.3.917 void SetSensorUltrasonic (const byte & port) [inline]**

Configure an ultrasonic sensor. Configure the sensor on the specified port as an ultrasonic sensor.

**Parameters:**

*port* The port to configure. See [Input port constants](#).

**Examples:**

[ex\\_SetSensorUltrasonic.nxc](#).

**8.3.3.918 void SetSleepTime (const byte *n*) [inline]**

Set sleep time. Set the NXT sleep timeout value to the specified number of minutes.

**Parameters:**

*n* The minutes to wait before sleeping.

**See also:**

[SetSleepTimeout](#), [SleepTimeout](#)

**Examples:**

[ex\\_setsleeptime.nxc](#).

**8.3.3.919 void SetSleepTimeout (const byte *n*) [inline]**

Set sleep timeout. Set the NXT sleep timeout value to the specified number of minutes.

**Parameters:**

*n* The minutes to wait before sleeping.

**Examples:**

[ex\\_SetSleepTimeout.nxc](#).

**8.3.3.920 void SetSleepTimer (const byte *n*) [inline]**

Set the sleep timer. Set the system sleep timer to the specified number of minutes.

**Parameters:**

*n* The minutes left on the timer.

**Examples:**

[ex\\_SetSleepTimer.nxc](#).

**8.3.3.921 void SetSoundDuration (unsigned int *duration*) [inline]**

Set sound duration. Set the sound duration.

**See also:**

[SoundDuration\(\)](#)

**Parameters:**

*duration* The new sound duration

**Examples:**

[ex\\_SetSoundDuration.nxc](#).

**8.3.3.922 void SetSoundFlags (byte *flags*) [inline]**

Set sound module flags. Set the sound module flags. See the [SoundFlags constants](#) group.

**See also:**

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Parameters:**

*flags* The new sound module flags

**Examples:**

[ex\\_SetSoundFlags.nxc](#).

**8.3.3.923 void SetSoundFrequency (unsigned int *frequency*) [inline]**

Set sound frequency. Set the sound frequency.

**See also:**

[SoundFrequency\(\)](#)

**Parameters:**

*frequency* The new sound frequency

**Examples:**

[ex\\_SetSoundFrequency.nxc](#).

**8.3.3.924 void SetSoundMode (byte *mode*) [inline]**

Set sound mode. Set the sound mode. See the [SoundMode constants](#) group.

**See also:**

[SoundMode\(\)](#)

**Parameters:**

*mode* The new sound mode

**Examples:**

[ex\\_SetSoundMode.nxc](#).

**8.3.3.925 void SetSoundModuleState (byte *state*) [inline]**

Set sound module state. Set the sound module state. See the [SoundState constants](#) group.

**See also:**

[SoundState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Parameters:**

*state* The new sound state

**Examples:**

[ex\\_SetSoundModuleState.nxc](#).

**8.3.3.926 void SetSoundModuleValue (unsigned int *offset*, variant *value*)**  
**[inline]**

Set Sound module IOMap value. Set one of the fields of the Sound module IOMap structure to a new value. You provide the offset into the Sound module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Sound module IOMap structure where the new value should be written. See [Sound module IOMAP offsets](#).

*value* A variable containing the new value to write to the Sound module IOMap.

**8.3.3.927 void SetSoundSampleRate (unsigned int *sampleRate*)** [inline]

Set sample rate. Set the sound sample rate.

**See also:**

[SoundSampleRate\(\)](#)

**Parameters:**

*sampleRate* The new sample rate

**Examples:**

[ex\\_SetSoundSampleRate.nxc](#).

**8.3.3.928 void SetSoundVolume (byte *volume*)** [inline]

Set sound volume. Set the sound volume.

See also:

[SoundVolume\(\)](#)

**Parameters:**

*volume* The new volume

**Examples:**

[ex\\_SetSoundVolume.nxc](#).

#### 8.3.3.929 void SetUIButton (byte *btn*) [inline]

Set UI button. Set user interface button information.

**Parameters:**

*btn* A user interface button value. See [UIButton constants](#).

**Examples:**

[ex\\_SetUIButton.nxc](#).

#### 8.3.3.930 void SetUIModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Ui module IOMap value. Set one of the fields of the Ui module IOMap structure to a new value. You provide the offset into the Ui module IOMap structure where you want to write the value along with a variable containing the new value.

**Parameters:**

*offset* The number of bytes offset from the start of the Ui module IOMap structure where the new value should be written. See [Ui module IOMAP offsets](#).

*value* A variable containing the new value to write to the Ui module IOMap.

#### 8.3.3.931 void SetUIState (byte *state*) [inline]

Set UI state. Set the user interface state.

**Parameters:**

*state* A user interface state value. See [UIState constants](#).

**Examples:**

[ex\\_SetUIState.nxc](#).

**8.3.3.932 void SetUSBInputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]**

Set USB input buffer data. Write *cnt* bytes of data to the USB input buffer at *offset*.

**Parameters:**

*offset* A constant offset into the input buffer  
*cnt* The number of bytes to write  
*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBInputBuffer.nxc](#).

**8.3.3.933 void SetUSBInputBufferInPtr (byte *n*) [inline]**

Set USB input buffer in-pointer. Set the value of the input buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBInputBufferInPtr.nxc](#).

**8.3.3.934 void SetUSBInputBufferOutPtr (byte *n*) [inline]**

Set USB input buffer out-pointer. Set the value of the input buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBInputBufferOutPtr.nxc](#).

**8.3.3.935** void SetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]

Set USB output buffer data. Write cnt bytes of data to the USB output buffer at offset.

**Parameters:**

*offset* A constant offset into the output buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBOutputBuffer.nxc](#).

**8.3.3.936** void SetUSBOutputBufferInPtr (byte *n*) [inline]

Set USB output buffer in-pointer. Set the value of the output buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBOutputBufferInPtr.nxc](#).

**8.3.3.937** void SetUSBOutputBufferOutPtr (byte *n*) [inline]

Set USB output buffer out-pointer. Set the value of the output buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBOutputBufferOutPtr.nxc](#).

**8.3.3.938** void SetUSBPollBuffer (const byte *offset*, byte *cnt*, byte *data*[ ]) [inline]

Set USB poll buffer data. Write cnt bytes of data to the USB poll buffer at offset.

**Parameters:**

*offset* A constant offset into the poll buffer

*cnt* The number of bytes to write

*data* A byte array containing the data to write

**Examples:**

[ex\\_SetUSBPollBuffer.nxc](#).

**8.3.3.939** void SetUSBPollBufferInPtr (byte *n*) [inline]

Set USB poll buffer in-pointer. Set the value of the poll buffer in-pointer.

**Parameters:**

*n* The new in-pointer value (0..63).

**Examples:**

[ex\\_SetUSBPollBufferInPtr.nxc](#).

**8.3.3.940** void SetUSBPollBufferOutPtr (byte *n*) [inline]

Set USB poll buffer out-pointer. Set the value of the poll buffer out-pointer.

**Parameters:**

*n* The new out-pointer value (0..63).

**Examples:**

[ex\\_SetUSBPollBufferOutPtr.nxc](#).

**8.3.3.941 void SetUSBState (byte *usbState*) [inline]**

Set USB state. This method sets the value of the USB state.

**Parameters:**

*usbState* The USB state.

**Examples:**

[ex\\_SetUsbState.nxc](#).

**8.3.3.942 void SetVMRunState (const byte *vmRunState*) [inline]**

Set VM run state. Set VM run state information.

**Parameters:**

*vmRunState* The desired VM run state. See [VM run state constants](#).

**Warning:**

It is not a good idea to change the VM run state from within a running program unless you know what you are doing.

**Examples:**

[ex\\_SetVMRunState.nxc](#).

**8.3.3.943 void SetVolume (byte *volume*) [inline]**

Set volume. Set the user interface volume level. Valid values are from 0 to 4.

**Parameters:**

*volume* The new volume level.

**Examples:**

[ex\\_SetVolume.nxc](#).

**8.3.3.944 char sign (variant *num*) [inline]**

Sign value. Return the sign of the value argument (-1, 0, or 1). Any scalar type can be passed into this function.

**Parameters:**

*num* The numeric value for which to calculate its sign value.

**Returns:**

-1 if the parameter is negative, 0 if the parameter is zero, or 1 if the parameter is positive.

**Examples:**

[ex\\_sign.nxc](#).

**8.3.3.945 float sin (float *x*) [inline]**

Compute sine. Computes the sine of an angle of *x* radians.

**Parameters:**

*x* Floating point value representing an angle expressed in radians.

**Returns:**

Sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#).

**8.3.3.946 float sind (float *x*) [inline]**

Compute sine (degrees). Computes the sine of an angle of *x* degrees.

**Parameters:**

*x* Floating point value representing an angle expressed in degrees.

**Returns:**

Sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sind\\_cosd.nxc](#).

**8.3.3.947 float sinh (float *x*) [inline]**

Compute hyperbolic sine. Computes the hyperbolic sine of *x*, expressed in radians.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sinh.nxc](#).

**8.3.3.948 float sinhd (float *x*) [inline]**

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of *x*, expressed in degrees.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic sine of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**8.3.3.949 unsigned int SizeOf (variant & *value*) [inline]**

Calculate the size of a variable. Calculate the number of bytes required to store the contents of the variable passed into the function.

**Parameters:**

*value* The variable.

**Returns:**

The number of bytes occupied by the variable.

**Examples:**

[ex\\_SizeOf.nxc](#).

**8.3.3.950 void SleepNow () [inline]**

Put the brick to sleep immediately. This function lets you immediately put the NXT to sleep. The running program will terminate as a result of this action.

**Examples:**

[ex\\_SleepNow.nxc](#).

**8.3.3.951 byte SleepTime (void) [inline]**

Read sleep time. Return the number of minutes that the NXT will remain on before it automatically shuts down.

**Returns:**

The sleep time value

**See also:**

[SleepTimeout](#)

**Examples:**

[ex\\_sleeptime.nxc](#).

**8.3.3.952 byte SleepTimeout (void) [inline]**

Read sleep timeout. Return the number of minutes that the NXT will remain on before it automatically shuts down.

**Returns:**

The sleep timeout value

**Examples:**

[ex\\_SleepTimeout.nxc](#).

**8.3.3.953 byte SleepTimer (void) [inline]**

Read sleep timer. Return the number of minutes left in the countdown to zero from the original SleepTimeout value. When the SleepTimer value reaches zero the NXT will shutdown.

**Returns:**

The sleep timer value

**Examples:**

[ex\\_SleepTimer.nxc](#).

**8.3.3.954 unsigned int SoundDuration () [inline]**

Get sound duration. Return the current sound duration.

**See also:**

[SetSoundDuration\(\)](#)

**Returns:**

The current sound duration.

**Examples:**

[ex\\_SoundDuration.nxc](#).

**8.3.3.955 byte SoundFlags () [inline]**

Get sound module flags. Return the current sound module flags. See the [SoundFlags constants](#) group.

**See also:**

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Returns:**

The current sound module flags.

**Examples:**

[ex\\_SoundFlags.nxc](#).

**8.3.3.956 unsigned int SoundFrequency () [inline]**

Get sound frequency. Return the current sound frequency.

**See also:**

[SetSoundFrequency\(\)](#)

**Returns:**

The current sound frequency.

**Examples:**

[ex\\_SoundFrequency.nxc](#).

**8.3.3.957 byte SoundMode () [inline]**

Get sound mode. Return the current sound mode. See the [SoundMode constants](#) group.

**See also:**

[SetSoundMode\(\)](#)

**Returns:**

The current sound mode.

**Examples:**

[ex\\_SoundMode.nxc](#).

**8.3.3.958 unsigned int SoundSampleRate () [inline]**

Get sample rate. Return the current sound sample rate.

**See also:**

[SetSoundSampleRate\(\)](#)

**Returns:**

The current sound sample rate.

**Examples:**

[ex\\_SoundSampleRate.nxc](#).

**8.3.3.959** `byte SoundState () [inline]`

Get sound module state. Return the current sound module state. See the [SoundState constants](#) group.

**See also:**

[SetSoundModuleState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

**Returns:**

The current sound module state.

**Examples:**

[ex\\_SoundState.nxc](#).

**8.3.3.960** `byte SoundVolume () [inline]`

Get volume. Return the current sound volume.

**See also:**

[SetSoundVolume\(\)](#)

**Returns:**

The current sound volume.

**Examples:**

[ex\\_SoundVolume.nxc](#).

**8.3.3.961** `void sprintf (string & str, string format, variant value) [inline]`

Write formatted data to string. Writes a sequence of data formatted as the format argument specifies to a string. After the format parameter, the function expects one value argument.

**Parameters:**

*str* The string to write to.

*format* A string specifying the desired format.

*value* A value to be formatted for writing to the string.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sprintf.nxc](#).

**8.3.3.962 float sqrt (float *x*) [inline]**

Compute square root. Computes the square root of *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

Square root of *x*.

**Examples:**

[ex\\_isnan.nxc](#), [ex\\_labs.nxc](#), and [ex\\_sqrt.nxc](#).

**8.3.3.963 long srand (long *seed*) [inline]**

Seed the random number generator. Provide the random number generator with a new seed value.

**Parameters:**

*seed* The new random number generator seed. A value of zero causes the seed to be based on the current time value. A value less than zero causes the seed to be restored to the last specified seed.

**Returns:**

The new seed value (useful if you pass in 0 or -1).

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Examples:**

[ex\\_srand.nxc](#).

**8.3.3.964 void StartTask (task *t*) [inline]**

Start a task. Start the specified task.

**Parameters:**

*t* The task to start.

**Examples:**

[ex\\_StartTask.nxc](#).

**8.3.3.965 void Stop (bool *bvalue*) [inline]**

Stop the running program. Stop the running program if *bvalue* is true. This will halt the program completely, so any code following this command will be ignored.

**Parameters:**

*bvalue* If this value is true the program will stop executing.

**Examples:**

[ex\\_file\\_system.nxc](#), and [ex\\_Stop.nxc](#).

**8.3.3.966 void StopAllTasks () [inline]**

Stop all tasks. Stop all currently running tasks. This will halt the program completely, so any code following this command will be ignored.

**Examples:**

[ex\\_StopAllTasks.nxc](#).

**8.3.3.967** byte StopSound () [inline]

Stop sound. Stop playing of the current tone or file.

**Returns:**

The result

**Todo**

?

**Examples:**

[ex\\_StopSound.nxc](#).

**8.3.3.968** void StopTask (task *t*) [inline]

Stop a task. Stop the specified task.

**Parameters:**

*t* The task to stop.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_StopTask.nxc](#).

**8.3.3.969** string strcat (string & *dest*, const string & *src*) [inline]

Concatenate strings. Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_StrCat.nxc](#).

**8.3.3.970 string StrCat (string *str1*, string *str2*, string *strN*) [inline]**

Concatenate strings. Return a string which is the result of concatenating all of the string arguments together. This function accepts any number of parameters which may be string variables, constants, or expressions.

**Parameters:**

*str1* The first string.

*str2* The second string.

*strN* The Nth string.

**Returns:**

The concatenated string.

**Examples:**

[ex\\_GetBrickDataAddress.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), and [util\\_battery\\_1.nxc](#).

**8.3.3.971 int strcmp (const string & *str1*, const string & *str2*) [inline]**

Compare two strings. Compares the string *str1* to the string *str2*.

**Parameters:**

*str1* A string to be compared.

*str2* A string to be compared.

**Returns:**

Returns an integral value indicating the relationship between the strings. A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in `str1` than in `str2`. A value less than zero indicates the opposite.

**Examples:**

[ex\\_strcmp.nxc](#).

**8.3.3.972 string strcpy (string & *dest*, const string & *src*) [inline]**

Copy string. Copies the string pointed by source into the array pointed by destination, including the terminating null character. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strcpy.nxc](#).

**8.3.3.973 byte StrIndex (string *str*, unsigned int *idx*) [inline]**

Extract a character from a string. Return the numeric value of the character in the specified string at the specified index. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* A string.

*idx* The index of the character to retrieve.

**Returns:**

The numeric value of the character at the specified index.

**Examples:**

[ex\\_StrIndex.nxc](#), and [ex\\_string.nxc](#).

**8.3.3.974 int strlen (const string & str) [inline]**

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string.

**Parameters:**

*str* A string.

**Returns:**

The length of the string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrLen.nxc](#).

**8.3.3.975 unsigned int StrLen (string str) [inline]**

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* A string.

**Returns:**

The length of the string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrLenOld.nxc](#).

**8.3.3.976** `string strncat (string & dest, const string & src, unsigned int num)`  
[inline]

Append characters from string. Appends the first num characters of source to destination, plus a terminating null-character. If the length of the string in source is less than num, only the content up to the terminating null-character is copied. The destination string is returned.

**Parameters:**

- dest* The destination string.
- src* The string to be appended.
- num* The maximum number of characters to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strncat.nxc](#).

**8.3.3.977** `int strncmp (const string & str1, const string & str2, unsigned int num)`  
[inline]

Compare characters of two strings. Compares up to num characters of the string str1 to those of the string str2.

**Parameters:**

- str1* A string to be compared.
- str2* A string to be compared.
- num* The maximum number of characters to be compared.

**Returns:**

Returns an integral value indicating the relationship between the strings. A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2. A value less than zero indicates the opposite.

**Examples:**

[ex\\_strncmp.nxc](#).

**8.3.3.978** `string strncpy (string & dest, const string & src, unsigned int num)`  
[inline]

Copy characters from string. Copies the first num characters of source to destination. The destination string is returned.

**Parameters:**

*dest* The destination string.

*src* The string to be appended.

*num* The maximum number of characters to be appended.

**Returns:**

The destination string.

**Examples:**

[ex\\_strncpy.nxc](#).

**8.3.3.979** `string StrReplace (string str, unsigned int idx, string strnew)`  
[inline]

Replace a portion of a string. Return a string with the part of the string replaced (starting at the specified index) with the contents of the new string value provided in the third argument. The input string parameters may be variables, constants, or expressions.

**Parameters:**

*str* A string.

*idx* The starting point for the replace operation.

*strnew* The replacement string.

**Returns:**

The modified string.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrReplace.nxc](#).

**8.3.3.980 void StrToByteArray (string *str*, byte & *data*[ ]) [inline]**

Convert a string to a byte array. Convert the specified string to an array of byte by removing the null terminator at the end of the string. The output array variable must be a one-dimensional array of byte.

**See also:**

[ByteArrayToStr](#), [ByteArrayToStrEx](#)

**Parameters:**

*str* A string

*data* A byte array reference which, on output, will contain *str* without its null terminator.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrToByteArray.nxc](#).

**8.3.3.981 float strtod (const string & *str*, string & *endptr*) [inline]**

Convert string to float. Parses the string *str* interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. A string containing the rest of the string after the last valid character is stored in *endptr*.

A valid floating point number for *atof* is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in *str* does not form a valid floating-point number as just defined, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of a floating-point number.

*endptr* Reference to a string, whose value is set by the function to the remaining characters in *str* after the numerical value.

**Returns:**

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

**Examples:**

[ex\\_strtod.nxc](#).

**8.3.3.982 long strtol (const string & *str*, string & *endptr*, int *base* = 10) [inline]**

Convert string to long integer. Parses the C string *str* interpreting its content as an integral number of the specified base, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in *str* is stored in *endptr*.

If the first sequence of non-whitespace characters in *str* does not form a valid integral number, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String beginning with the representation of an integral number.

*endptr* Reference to a string, whose value is set by the function to the remaining characters in *str* after the numerical value.

*base* Optional and ignored if specified.

**Returns:**

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

**Warning:**

Only *base* = 10 is currently supported.

**Examples:**

[ex\\_strtol.nxc](#).

**8.3.3.983 variant StrToNum (string *str*) [inline]**

Convert string to number. Return the numeric value specified by the string passed to the function. If the content of the string is not a numeric value then this function returns zero. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* String beginning with the representation of a number.

*str* A string.

**Returns:**

A number.

**Examples:**

[ex\\_string.nxc](#), and [ex\\_StrToNum.nxc](#).

**8.3.3.984 long strtoul (const string & *str*, string & *endptr*, int *base* = 10) [inline]**

Convert string to unsigned long integer. Parses the C string *str* interpreting its content as an unsigned integral number of the specified base, which is returned as an unsigned long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in *str* is stored in *endptr*.

If the first sequence of non-whitespace characters in *str* does not form a valid integral number, or if no such sequence exists because either *str* is empty or contains only whitespace characters, no conversion is performed.

**Parameters:**

*str* String containing the representation of an unsigned integral number.

*endptr* Reference to a string, whose value is set by the function to the remaining characters in *str* after the numerical value.

*base* Optional and ignored if specified.

**Returns:**

On success, the function returns the converted integral number as an unsigned long int value. If no valid conversion could be performed a zero value is returned.

**Warning:**

Only base = 10 is currently supported.

**Examples:**

[ex\\_strtoul.nxc](#).

**8.3.3.985 string SubStr (string *str*, unsigned int *idx*, unsigned int *len*)  
[inline]**

Extract a portion of a string. Return a sub-string from the specified input string starting at *idx* and including the specified number of characters. The input string parameter may be a variable, constant, or expression.

**Parameters:**

*str* A string.

*idx* The starting point of the sub-string.

*len* The length of the sub-string.

**Returns:**

The sub-string extracted from parameter *str*.

**Examples:**

[ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), and [ex\\_SubStr.nxc](#).

**8.3.3.986 void SysCall (byte *funcID*, variant & *args*) [inline]**

Call any system function. This generic macro can be used to call any system function. No type checking is performed so you need to make sure you use the correct structure type given the selected system function ID. This is, however, the fastest possible way to call a system function in NXC.

Valid function ID constants are defined in the [System Call function constants](#) group.

**Parameters:**

*funcID* The function ID constant corresponding to the function to be called.

*args* The structure containing the needed parameters.

**Examples:**

[ex\\_dispgout.nxc](#), and [ex\\_syscall.nxc](#).

**8.3.3.987 void SysColorSensorRead (ColorSensorReadType & args)  
[inline]**

Read LEGO color sensor. This function lets you read the LEGO color sensor given the parameters you pass in via the [ColorSensorReadType](#) structure.

**Parameters:**

*args* The [ColorSensorReadType](#) structure containing the required parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysColorSensorRead.nxc](#).

**8.3.3.988 void SysCommBTCheckStatus (CommBTCheckStatusType & args)**

Check Bluetooth connection status. This function lets you check the status of a Bluetooth connection using the values specified via the [CommBTCheckStatusType](#) structure.

**Parameters:**

*args* The [CommBTCheckStatusType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommbtcheckstatus.nxc](#).

**8.3.3.989 void SysCommBTConnection (CommBTConnectionType & args) [inline]**

Connect or disconnect a bluetooth device. This function lets you connect or disconnect a bluetooth device using the values specified via the [CommBTConnectionType](#) structure.

**Parameters:**

*args* The [CommBTConnectionType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_syscommbtconnection.nxc](#).

**8.3.3.990 void SysCommBTOff (CommBTOffType & args) [inline]**

Turn on or off the bluetooth subsystem. This function lets you turn on or off the bluetooth subsystem using the values specified via the [CommBTOffType](#) structure.

**Parameters:**

*args* The [CommBTOffType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysCommBTOff.nxc](#).

**8.3.3.991 void SysCommBTWrite (CommBTWriteType & args)**

Write data to a Bluetooth connection. This function lets you write to a Bluetooth connection using the values specified via the [CommBTWriteType](#) structure.

**Parameters:**

*args* The [CommBTWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommbtwrite.nxc](#).

**8.3.3.992 void SysCommExecuteFunction (CommExecuteFunctionType & args) [inline]**

Execute any Comm module command. This function lets you directly execute the Comm module's primary function using the values specified via the [CommExecuteFunctionType](#) structure.

**Parameters:**

*args* The [CommExecuteFunctionType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_syscommexecutefunction.nxc](#).

**8.3.3.993 void SysCommHSCheckStatus (CommHSCheckStatusType & args) [inline]**

Check the hi-speed port status. This function lets you check the hi-speed port status using the values specified via the [CommHSCheckStatusType](#) structure.

**Parameters:**

*args* The [CommHSCheckStatusType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSCheckStatus.nxc](#).

**8.3.3.994 void SysCommHSControl (CommHSControlType & args)  
[inline]**

Control the hi-speed port. This function lets you control the hi-speed port using the values specified via the [CommHSControlType](#) structure.

**Parameters:**

*args* The [CommHSControlType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSControl.nxc](#).

**8.3.3.995 void SysCommHSRead (CommHSReadWriteType & args)  
[inline]**

Read from the hi-speed port. This function lets you read from the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

**Parameters:**

*args* The [CommHSReadWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSRead.nxc](#).

**8.3.3.996** void SysCommHSWrite (CommHSReadWriteType & args)  
[inline]

Write to the hi-speed port. This function lets you write to the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

**Parameters:**

*args* The [CommHSReadWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_SysCommHSWrite.nxc](#).

**8.3.3.997** void SysCommLSCheckStatus (CommLSCheckStatusType & args)  
[inline]

Check Lowspeed sensor status. This function lets you check the status of an I2C (Lowspeed) sensor transaction using the values specified via the [CommLSCheckStatusType](#) structure.

**Parameters:**

*args* The [CommLSCheckStatusType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommlscheckstatus.nxc](#).

**8.3.3.998** void SysCommLSRead (CommLSReadType & args) [inline]

Read from a Lowspeed sensor. This function lets you read from an I2C (Lowspeed) sensor using the values specified via the [CommLSReadType](#) structure.

**Parameters:**

*args* The [CommLSReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommlsread.nxc](#).

**8.3.3.999 void SysCommLSWrite (CommLSWriteType & args) [inline]**

Write to a Lowspeed sensor. This function lets you write to an I2C (Lowspeed) sensor using the values specified via the [CommLSWriteType](#) structure.

**Parameters:**

*args* The [CommLSWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syscommlswrite.nxc](#).

**8.3.3.1000 void SysCommLSWriteEx (CommLSWriteExType & args) [inline]**

Write to a Lowspeed sensor (extra). This function lets you write to an I2C (Lowspeed) sensor using the values specified via the [CommLSWriteExType](#) structure. This is the same as the SysCommLSWrite function except that you also can specify whether or not the Lowspeed module should issue a restart command to the I2C device before beginning to read data from the device.

**Parameters:**

*args* The [CommLSWriteExType](#) structure containing the desired parameters.

**Examples:**

[ex\\_syscommlswriteex.nxc](#).

**8.3.3.1001 void SysComputeCalibValue (ComputeCalibValueType & args) [inline]**

Compute calibration values. This function lets you compute calibration values using the values specified via the [ComputeCalibValueType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [ComputeCalibValueType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysComputeCalibValue.nxc](#).

**8.3.3.1002 void SysDatalogGetTimes (DatalogGetTimesType & args) [inline]**

Get datalog times. This function lets you get datalog times using the values specified via the [DatalogGetTimesType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [DatalogGetTimesType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_sysdataloggettimes.nxc](#).

**8.3.3.1003 void SysDatalogWrite (DatalogWriteType & args) [inline]**

Write to the datalog. This function lets you write to the datalog using the values specified via the [DatalogWriteType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [DatalogWriteType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysDatalogWrite.nxc](#).

**8.3.3.1004 void SysDisplayExecuteFunction (DisplayExecuteFunctionType & args) [inline]**

Execute any Display module command. This function lets you directly execute the Display module's primary drawing function using the values specified via the [DisplayExecuteFunctionType](#) structure.

**Parameters:**

*args* The [DisplayExecuteFunctionType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

**8.3.3.1005 void SysDrawCircle (DrawCircleType & args) [inline]**

Draw a circle. This function lets you draw a circle on the NXT LCD given the parameters you pass in via the [DrawCircleType](#) structure.

**Parameters:**

*args* The [DrawCircleType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawcircle.nxc](#).

**8.3.3.1006 void SysDrawEllipse (DrawEllipseType & args) [inline]**

Draw an ellipse. This function lets you draw an ellipse on the NXT LCD given the parameters you pass in via the [DrawEllipseType](#) structure.

**Parameters:**

*args* The [DrawEllipseType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_SysDrawEllipse.nxc](#).

**8.3.3.1007 void SysDrawFont (DrawFontType & args) [inline]**

Draw text using a custom font. This function lets you draw text on the NXT LCD using a custom font with parameters you pass in via the [DrawFontType](#) structure.

**Parameters:**

*args* The [DrawFontType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**8.3.3.1008 void SysDrawGraphic (DrawGraphicType & args) [inline]**

Draw a graphic (RIC file). This function lets you draw a graphic image (RIC file) on the NXT LCD given the parameters you pass in via the [DrawGraphicType](#) structure.

**Parameters:**

*args* The [DrawGraphicType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawgraphic.nxc](#).

**8.3.3.1009 void SysDrawGraphicArray (DrawGraphicArrayType & args) [inline]**

Draw a graphic image from a byte array. This function lets you draw a graphic image on the NXT LCD given the parameters you pass in via the [DrawGraphicArrayType](#) structure.

**Parameters:**

*args* The [DrawGraphicArrayType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysdrawgraphicarray.nxc](#).

**8.3.3.1010 void SysDrawLine (DrawLineType & args) [inline]**

Draw a line. This function lets you draw a line on the NXT LCD given the parameters you pass in via the [DrawLineType](#) structure.

**Parameters:**

*args* The [DrawLineType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawline.nxc](#).

**8.3.3.1011 void SysDrawPoint (DrawPointType & args) [inline]**

Draw a point. This function lets you draw a pixel on the NXT LCD given the parameters you pass in via the [DrawPointType](#) structure.

**Parameters:**

*args* The [DrawPointType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawpoint.nxc](#).

**8.3.3.1012 void SysDrawPolygon (DrawPolygonType & args) [inline]**

Draw a polygon. This function lets you draw a polygon on the NXT LCD given the parameters you pass in via the [DrawPolygonType](#) structure.

**Parameters:**

*args* The [DrawPolygonType](#) structure containing the drawing parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysdrawpolygon.nxc](#).

**8.3.3.1013 void SysDrawRect (DrawRectType & args) [inline]**

Draw a rectangle. This function lets you draw a rectangle on the NXT LCD given the parameters you pass in via the [DrawRectType](#) structure.

**Parameters:**

*args* The [DrawRectType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawrect.nxc](#).

**8.3.3.1014 void SysDrawText (DrawTextType & args) [inline]**

Draw text. This function lets you draw text on the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

**Parameters:**

*args* The [DrawTextType](#) structure containing the drawing parameters.

**Examples:**

[ex\\_sysdrawtext.nxc](#).

**8.3.3.1015 void SysFileClose (FileCloseType & args) [inline]**

Close file handle. This function lets you close a file using the values specified via the [FileCloseType](#) structure.

**Parameters:**

*args* The [FileCloseType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileclose.nxc](#).

**8.3.3.1016 void SysFileDelete (FileDeleteType & args) [inline]**

Delete file. This function lets you delete a file using the values specified via the [FileDeleteType](#) structure.

**Parameters:**

*args* The [FileDeleteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfiledelete.nxc](#).

**8.3.3.1017 void SysFileFindFirst (FileFindType & args) [inline]**

Start finding files. This function lets you begin iterating through files stored on the NXT.

**Parameters:**

*args* The [FileFindType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfilefindfirst.nxc](#).

**8.3.3.1018 void SysFileFindNext (FileFindType & args) [inline]**

Continue finding files. This function lets you continue iterating through files stored on the NXT.

**Parameters:**

*args* The [FileFindType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfilefindnext.nxc](#).

**8.3.3.1019 void SysFileOpenAppend (FileOpenType & args) [inline]**

Open file for writing at end of file. This function lets you open an existing file that you can write to using the values specified via the [FileOpenType](#) structure.

The available length remaining in the file is returned via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenappend.nxc](#).

**8.3.3.1020 void SysFileOpenRead (FileOpenType & args) [inline]**

Open file for reading. This function lets you open an existing file for reading using the values specified via the [FileOpenType](#) structure.

The number of bytes that can be read from the file is returned via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenread.nxc](#).

**8.3.3.1021 void SysFileOpenReadLinear (FileOpenType & args) [inline]**

Open linear file for reading. This function lets you open an existing linear file for reading using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenreadlinear.nxc](#).

**8.3.3.1022 void SysFileOpenWrite (FileOpenType & args) [inline]**

Open and create file for writing. This function lets you create a file that you can write to using the values specified via the [FileOpenType](#) structure.

The desired maximum file capacity in bytes is specified via the Length member.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileopenwrite.nxc](#).

**8.3.3.1023 void SysFileOpenWriteLinear (FileOpenType & args) [inline]**

Open and create linear file for writing. This function lets you create a linear file that you can write to using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenwritelinear.nxc](#).

**8.3.3.1024 void SysFileOpenWriteNonLinear (FileOpenType & args) [inline]**

Open and create non-linear file for writing. This function lets you create a non-linear linear file that you can write to using the values specified via the [FileOpenType](#) structure.

**Parameters:**

*args* The [FileOpenType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileopenwritenonlinear.nxc](#).

**8.3.3.1025 void SysFileRead (FileReadWriteType & args) [inline]**

Read from file. This function lets you read from a file using the values specified via the [FileReadWriteType](#) structure.

**Parameters:**

*args* The [FileReadWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileread.nxc](#).

**8.3.3.1026 void SysFileRename (FileRenameType & args) [inline]**

Rename file. This function lets you rename a file using the values specified via the [FileRenameType](#) structure.

**Parameters:**

*args* The [FileRenameType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfilerename.nxc](#).

**8.3.3.1027 void SysFileResize (FileResizeType & args) [inline]**

Resize a file. This function lets you resize a file using the values specified via the [FileResizeType](#) structure.

**Parameters:**

*args* The [FileResizeType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware. It has not yet been implemented at the firmware level.

**Examples:**

[ex\\_sysfileresize.nxc](#).

**8.3.3.1028 void SysFileResolveHandle (FileResolveHandleType & args) [inline]**

File resolve handle. This function lets you resolve the handle of a file using the values specified via the [FileResolveHandleType](#) structure. This will find a previously opened file handle.

**Parameters:**

*args* The [FileResolveHandleType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfileresolvehandle.nxc](#).

**8.3.3.1029 void SysFileSeek (FileSeekType & args) [inline]**

Seek to file position. This function lets you seek to a specific file position using the values specified via the [FileSeekType](#) structure.

**Parameters:**

*args* The [FileSeekType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysfileseek.nxc](#).

**8.3.3.1030 void SysFileTell (FileTellType & args) [inline]**

Return the file position. This function returns the current file position in the open file specified via the [FileTellType](#) structure.

**Parameters:**

*args* The [FileTellType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**8.3.3.1031 void SysFileWrite (FileReadWriteType & args) [inline]**

File write. This function lets you write to a file using the values specified via the [FileReadWriteType](#) structure.

**Parameters:**

*args* The [FileReadWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysfilewrite.nxc](#).

**8.3.3.1032 void SysGetStartTick (GetStartTickType & args) [inline]**

Get start tick. This function lets you obtain the tick value at the time your program began executing via the [GetStartTickType](#) structure.

**Parameters:**

*args* The [GetStartTickType](#) structure receiving results.

**Examples:**

[ex\\_sysgetstarttick.nxc](#).

**8.3.3.1033 void SysInputPinFunction (InputPinFunctionType & args)  
[inline]**

Execute the Input module pin function. This function lets you execute the Input module's pin function using the values specified via the [InputPinFunctionType](#) structure.

**Parameters:**

*args* The [InputPinFunctionType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysinputpinfunction.nxc](#).

**8.3.3.1034 void SysIOMapRead (IOMapReadType & args) [inline]**

Read from IOMap by name. This function lets you read data from a firmware module's IOMap using the values specified via the [IOMapReadType](#) structure.

**Parameters:**

*args* The [IOMapReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysiomapread.nxc](#).

**8.3.3.1035 void SysIOMapReadByID (IOMapReadByIDType & args)  
[inline]**

Read from IOMap by identifier. This function lets you read data from a firmware module's IOMap using the values specified via the [IOMapReadByIDType](#) structure. This function can be as much as three times faster than using SysIOMapRead since it does not have to do a string lookup using the ModuleName.

**Parameters:**

*args* The [IOMapReadByIDType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**8.3.3.1036 void SysIOMapWrite (IOMapWriteType & args) [inline]**

Write to IOMap by name. This function lets you write data to a firmware module's IOMap using the values specified via the [IOMapWriteType](#) structure.

**Parameters:**

*args* The [IOMapWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysiomapwrite.nxc](#).

**8.3.3.1037 void SysIOMapWriteByID (IOMapWriteByIDType & args) [inline]**

Write to IOMap by identifier. This function lets you write data to a firmware module's IOMap using the values specified via the [IOMapWriteByIDType](#) structure. This function can be as much as three times faster than using SysIOMapWrite since it does not have to do a string lookup using the ModuleName.

**Parameters:**

*args* The [IOMapWriteByIDType](#) structure containing the needed parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_reladdressof.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**8.3.3.1038 void SysKeepAlive (KeepAliveType & args) [inline]**

Keep alive. This function lets you reset the sleep timer via the [KeepAliveType](#) structure.

**Parameters:**

*args* The [KeepAliveType](#) structure receiving results.

**Examples:**

[ex\\_syskeepalive.nxc](#).

**8.3.3.1039 void SysListFiles (ListFilesType & args) [inline]**

List files. This function lets you retrieve a list of files on the NXT using the values specified via the [ListFilesType](#) structure.

**Parameters:**

*args* The [ListFilesType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syslistfiles.nxc](#).

**8.3.3.1040 void SysLoaderExecuteFunction (LoaderExecuteFunctionType & args) [inline]**

Execute any Loader module command. This function lets you directly execute the Loader module's primary function using the values specified via the [LoaderExecuteFunctionType](#) structure.

**Parameters:**

*args* The [LoaderExecuteFunctionType](#) structure containing the needed parameters.

**Warning:**

This function requires the extended firmware.

**Examples:**

[ex\\_sysloaderexecutefunction.nxc](#).

**8.3.3.1041 void SysMemoryManager (MemoryManagerType & args)  
[inline]**

Read memory information. This function lets you read memory information using the values specified via the [MemoryManagerType](#) structure.

**Parameters:**

*args* The [MemoryManagerType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.28+.

**Examples:**

[ex\\_sysmemorymanager.nxc](#).

**8.3.3.1042 void SysMessageRead (MessageReadType & args)**

Read message. This function lets you read a message from a queue (aka mailbox) using the values specified via the [MessageReadType](#) structure.

**Parameters:**

*args* The [MessageReadType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysmessageread.nxc](#).

**8.3.3.1043 void SysMessageWrite (MessageWriteType & args)**

Write message. This function lets you write a message to a queue (aka mailbox) using the values specified via the [MessageWriteType](#) structure.

**Parameters:**

*args* The [MessageWriteType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysmessagewrite.nxc](#).

**8.3.3.1044 void SysRandomEx (RandomExType & args) [inline]**

Call the enhanced random number function. This function lets you either obtain a random number or seed the random number generator via the [RandomExType](#) structure.

**Parameters:**

*args* The [RandomExType](#) structure for passing inputs and receiving output values.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+

**Examples:**

[ex\\_sysrandomex.nxc](#).

**8.3.3.1045 void SysRandomNumber (RandomNumberType & args) [inline]**

Draw a random number. This function lets you obtain a random number via the [RandomNumberType](#) structure.

**Parameters:**

*args* The [RandomNumberType](#) structure receiving results.

**Examples:**

[ex\\_sysrandomnumber.nxc](#).

**8.3.3.1046 void SysReadButton (ReadButtonType & args) [inline]**

Read button. This function lets you read button state information via the [ReadButtonType](#) structure.

**Parameters:**

*args* The [ReadButtonType](#) structure containing the needed parameters.

**Examples:**

[ex\\_sysreadbutton.nxc](#), and [ex\\_xg1300.nxc](#).

**8.3.3.1047 void SysReadLastResponse (ReadLastResponseType & args) [inline]**

Read last response information. This function lets you read the last system or direct command response received by the NXT using the values specified via the [ReadLastResponseType](#) structure.

**Parameters:**

*args* The [ReadLastResponseType](#) structure containing the required parameters.

**Warning:**

This function requires the enhanced NBC/NXC firmware version 1.31+.

**Examples:**

[ex\\_SysReadLastResponse.nxc](#).

**8.3.3.1048 void SysReadSemData (ReadSemDataType & args) [inline]**

Read semaphore data. This function lets you read global motor semaphore data using the values specified via the [ReadSemDataType](#) structure.

**Parameters:**

*args* The [ReadSemDataType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysReadSemData.nxc](#).

**8.3.3.1049 void SysSetScreenMode (SetScreenModeType & args) [inline]**

Set the screen mode. This function lets you set the screen mode of the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

**Parameters:**

*args* The [SetScreenModeType](#) structure containing the screen mode parameters.

**Examples:**

[ex\\_syssetscreenmode.nxc](#).

**8.3.3.1050 void SysSetSleepTimeout (SetSleepTimeoutType & args) [inline]**

Set system sleep timeout. This function lets you set the system sleep timeout value given the parameters you pass in via the [SetSleepTimeoutType](#) structure.

**Parameters:**

*args* The [SetSleepTimeoutType](#) structure containing the required parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysSetSleepTimeout.nxc](#).

**8.3.3.1051 void SysSoundGetState (SoundGetStateType & args) [inline]**

Get sound state. This function lets you retrieve information about the sound module state via the [SoundGetStateType](#) structure.

**Parameters:**

*args* The [SoundGetStateType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundgetstate.nxc](#).

**8.3.3.1052 void SysSoundPlayFile (SoundPlayFileType & args) [inline]**

Play sound file. This function lets you play a sound file given the parameters you pass in via the [SoundPlayFileType](#) structure. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

**Parameters:**

*args* The [SoundPlayFileType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundplayfile.nxc](#).

**8.3.3.1053 void SysSoundPlayTone (SoundPlayToneType & args) [inline]**

Play tone. This function lets you play a tone given the parameters you pass in via the [SoundPlayToneType](#) structure.

**Parameters:**

*args* The [SoundPlayToneType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundplaytone.nxc](#).

**8.3.3.1054 void SysSoundSetState (SoundSetStateType & args) [inline]**

Set sound state. This function lets you set sound module state settings via the [SoundSetStateType](#) structure.

**Parameters:**

*args* The [SoundSetStateType](#) structure containing the needed parameters.

**Examples:**

[ex\\_syssoundsetstate.nxc](#).

**8.3.3.1055 void SysUpdateCalibCacheInfo (UpdateCalibCacheInfoType & args) [inline]**

Update calibration cache information. This function lets you update calibration cache information using the values specified via the [UpdateCalibCacheInfoType](#) structure.

**Todo**

figure out what this function is intended for

**Parameters:**

*args* The [UpdateCalibCacheInfoType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysUpdateCalibCacheInfo.nxc](#).

**8.3.3.1056 void SysWriteSemData (WriteSemDataType & args) [inline]**

Write semaphore data. This function lets you write global motor semaphore data using the values specified via the [WriteSemDataType](#) structure.

**Parameters:**

*args* The [WriteSemDataType](#) structure containing the needed parameters.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

[ex\\_SysWriteSemData.nxc](#).

**8.3.3.1057 float tan (float *x*) [inline]**

Compute tangent. Computes the tangent of an angle of *x* radians.

**Parameters:**

*x* Floating point value representing an angle expressed in radians.

**Returns:**

Tangent of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tan.nxc](#).

**8.3.3.1058 float tand (float *x*) [inline]**

Compute tangent (degrees). Computes the tangent of an angle of *x* degrees.

**Parameters:**

*x* Floating point value representing an angle expressed in degrees.

**Returns:**

Tangent of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tand.nxc](#).

**8.3.3.1059 float tanh (float *x*) [inline]**

Compute hyperbolic tangent. Computes the hyperbolic tangent of *x*, expressed in radians.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic tangent of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_tanh.nxc](#).

**8.3.3.1060 float tanhd (float *x*) [inline]**

Compute hyperbolic tangent (degrees). Computes the hyperbolic tangent of *x*, expressed in degrees.

**Parameters:**

*x* Floating point value.

**Returns:**

Hyperbolic tangent of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

### 8.3.3.1061 `char TextOut (int x, int y, string str, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw text. Draw a text value on the screen at the specified x and y location. The y value must be a multiple of 8. Valid line number constants are listed in the [Line number constants](#) group. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group.

#### See also:

[SysDrawText](#), [DrawTextType](#)

#### Parameters:

- x* The x value for the start of the text output.
- y* The text line number for the text output.
- str* The text to output to the LCD screen.
- options* The optional drawing options.

#### Returns:

The result of the drawing operation.

#### Examples:

[ex\\_acos.nxc](#), [ex\\_acosd.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_asin.nxc](#), [ex\\_asind.nxc](#), [ex\\_atan.nxc](#), [ex\\_atan2.nxc](#), [ex\\_atan2d.nxc](#), [ex\\_atand.nxc](#), [ex\\_clearline.nxc](#), [ex\\_copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_GetBrickDataAddress.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_i2cdeviceid.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_i2cvendorid.nxc](#), [ex\\_i2cversion.nxc](#), [ex\\_isnan.nxc](#), [ex\\_labs.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_midstr.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorHTTouchMultiplexer.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_SetAbortFlag.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_SetLongAbort.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommBTOff.nxc](#), [ex\\_SysCommHSCheckStatus.nxc](#), [ex\\_SysCommHSControl.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_SysDatalogWrite.nxc](#), [ex\\_sysfilefindfirst.nxc](#), [ex\\_sysfilefindnext.nxc](#), [ex\\_](#)

[sysfileread.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_sysmessageread.nxc](#), [ex\\_tan.nxc](#), [ex\\_tand.nxc](#), [ex\\_TextOut.nxc](#), [ex\\_xg1300.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

#### 8.3.3.1062 `int tolower (int c) [inline]`

Convert uppercase letter to lowercase. Converts parameter *c* to its lowercase equivalent if *c* is an uppercase letter and has a lowercase equivalent. If no such conversion is possible, the value returned is *c* unchanged.

##### Parameters:

*c* Uppercase letter character to be converted.

##### Returns:

The lowercase equivalent to *c*, if such value exists, or *c* (unchanged) otherwise..

##### Examples:

[ex\\_ctype.nxc](#), and [ex\\_tolower.nxc](#).

#### 8.3.3.1063 `int toupper (int c) [inline]`

Convert lowercase letter to uppercase. Converts parameter *c* to its uppercase equivalent if *c* is a lowercase letter and has an uppercase equivalent. If no such conversion is possible, the value returned is *c* unchanged.

##### Parameters:

*c* Lowercase letter character to be converted.

##### Returns:

The uppercase equivalent to *c*, if such value exists, or *c* (unchanged) otherwise..

##### Examples:

[ex\\_ctype.nxc](#), and [ex\\_toupper.nxc](#).

**8.3.3.1064 long trunc (float *x*) [inline]**

Compute integral part. Computes the integral part of *x*.

**Parameters:**

*x* Floating point value.

**Returns:**

Integral part of *x*.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), and [ex\\_trunc.nxc](#).

**8.3.3.1065 byte UIButton (void) [inline]**

Read UI button. Return user interface button information.

**Returns:**

A UI button value. See [UIButton constants](#).

**Examples:**

[ex\\_UIButton.nxc](#).

**8.3.3.1066 byte UIState (void) [inline]**

Get UI module state. Return the user interface state.

**Returns:**

The UI module state. See [UIState constants](#).

**Examples:**

[ex\\_UIState.nxc](#).

**8.3.3.1067 int UnflattenVar (string *str*, variant & *x*) [inline]**

Unflatten a string into a data type. Convert a string containing the byte representation of the specified variable back into the original variable type.

**See also:**

[FlattenVar](#), [Flatten](#)

**Parameters:**

*str* A string containing flattened data.

*x* A variable reference where the unflattened data is stored.

**Returns:**

A boolean value indicating whether the operation succeeded or not.

**Examples:**

[ex\\_FlattenVar.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

**8.3.3.1068 byte USBInputBufferInPtr (void) [inline]**

Get usb port input buffer in-pointer. This method returns the value of the input pointer of the usb port input buffer.

**Returns:**

The USB port input buffer's in-pointer value.

**Examples:**

[ex\\_USBInputBufferInPtr.nxc](#).

**8.3.3.1069 byte USBInputBufferOutPtr (void) [inline]**

Get usb port input buffer out-pointer. This method returns the value of the output pointer of the usb port input buffer.

**Returns:**

The USB port input buffer's out-pointer value.

**Examples:**

[ex\\_USBInputBufferOutPtr.nxc](#).

**8.3.3.1070 byte USBOutputBufferInPtr (void) [inline]**

Get usb port output buffer in-pointer. This method returns the value of the input pointer of the usb port output buffer.

**Returns:**

The USB port output buffer's in-pointer value.

**Examples:**

[ex\\_USBOutputBufferInPtr.nxc](#).

**8.3.3.1071 byte USBOutputBufferOutPtr (void) [inline]**

Get usb port output buffer out-pointer. This method returns the value of the output pointer of the usb port output buffer.

**Returns:**

The USB port output buffer's out-pointer value.

**Examples:**

[ex\\_USBOutputBufferOutPtr.nxc](#).

**8.3.3.1072 byte USBPollBufferInPtr (void) [inline]**

Get usb port poll buffer in-pointer. This method returns the value of the input pointer of the usb port poll buffer.

**Returns:**

The USB port poll buffer's in-pointer value.

**Examples:**

[ex\\_USBPollBufferInPtr.nxc](#).

**8.3.3.1073 byte USBPollBufferOutPtr (void) [inline]**

Get usb port poll buffer out-pointer. This method returns the value of the output pointer of the usb port poll buffer.

**Returns:**

The USB port poll buffer's out-pointer value.

**Examples:**

[ex\\_USBPollBufferOutPtr.nxc](#), and [ex\\_UsbState.nxc](#).

**8.3.3.1074 byte UsbState (void) [inline]**

Get UI module USB state. This method returns the UI module USB state.

**Returns:**

The UI module USB state. (0=disconnected, 1=connected, 2=working)

**Examples:**

[ex\\_UiUsbState.nxc](#).

**8.3.3.1075 byte USBState (void) [inline]**

Get USB state. This method returns the value of the USB state.

**Returns:**

The USB state.

**8.3.3.1076 void UseRS485 (void) [inline]**

Use the RS485 port. Configure port 4 for RS485 usage.

**Examples:**

[ex\\_RS485Receive.nxc](#), and [ex\\_RS485Send.nxc](#).

**8.3.3.1077 void VectorCross (VectorType a, VectorType b, VectorType & out) [inline]**

VectorCross function. Calculate the cross-product of two vectors.

**Parameters:**

*a* A variable of type [VectorType](#)

*b* A variable of type [VectorType](#)

*out* The cross-product vector.

**8.3.3.1078 float VectorDot (VectorType a, VectorType b) [inline]**

VectorDot function. Calculate the dot-product of two vectors.

**Parameters:**

*a* A variable of type [VectorType](#)

*b* A variable of type [VectorType](#)

**8.3.3.1079 void VectorNormalize (VectorType & a) [inline]**

VectorNormalize function. Normalize the vector.

**Parameters:**

*a* A variable of type [VectorType](#)

**8.3.3.1080 byte VMRunState (void) [inline]**

Read VM run state. Return VM run state information.

**Returns:**

VM run state. See [VM run state constants](#).

**Examples:**

[ex\\_VMRunState.nxc](#).

**8.3.3.1081 byte Volume (void) [inline]**

Read volume. Return the user interface volume level. Valid values are from 0 to 4.

**Returns:**

The UI module volume. (0..4)

**Examples:**

[ex\\_Volume.nxc](#).

**8.3.3.1082 void Wait (unsigned long ms) [inline]**

Wait some milliseconds. Make a task sleep for specified amount of time (in 1000ths of a second).

**Parameters:**

*ms* The number of milliseconds to sleep.

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_addressof.nxc](#), [ex\\_addressofex.nxc](#), [ex\\_ArrayMax.nxc](#), [ex\\_ArrayMean.nxc](#), [ex\\_ArrayMin.nxc](#), [ex\\_ArrayOp.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ArrayStd.nxc](#), [ex\\_ArraySum.nxc](#), [ex\\_ArraySumSqr.nxc](#), [ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_CircleOut.nxc](#), [ex\\_clearline.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_contrast.nxc](#), [ex\\_copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DataMode.nxc](#), [ex\\_delete\\_data\\_file.nxc](#), [ex\\_diaccl.nxc](#), [ex\\_digps.nxc](#), [ex\\_digyro.nxc](#), [ex\\_dispftout.nxc](#),

[ex\\_dispfunc.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_displayfont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_file\\_system.nxc](#), [ex\\_findfirstfile.nxc](#), [ex\\_findnextfile.nxc](#), [ex\\_FlattenVar.nxc](#), [ex\\_getchar.nxc](#), [ex\\_getmemoryinfo.nxc](#), [ex\\_HTGyroTest.nxc](#), [ex\\_i2cdeviceinfo.nxc](#), [ex\\_isnan.nxc](#), [ex\\_joystickmsg.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_NXTHID.nxc](#), [ex\\_NXTLineLeader.nxc](#), [ex\\_NXTPowerMeter.nxc](#), [ex\\_NXTServo.nxc](#), [ex\\_NXTSumoEyes.nxc](#), [ex\\_onfwdsyncpid.nxc](#), [ex\\_onrevsyncpid.nxc](#), [ex\\_PFMate.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_PosReg.nxc](#), [ex\\_proto.nxc](#), [ex\\_ReadSensorHTAngle.nxc](#), [ex\\_ReadSensorHTBarometric.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_reladdressof.nxc](#), [ex\\_ResetSensorHTAngle.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_RS485Receive.nxc](#), [ex\\_RS485Send.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_setdisplayfont.nxc](#), [ex\\_sin\\_cos.nxc](#), [ex\\_sind\\_cosd.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_StrReplace.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_syscommbtconnection.nxc](#), [ex\\_SysCommHSControl.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_sysdataloggettimes.nxc](#), [ex\\_sysdrawfont.nxc](#), [ex\\_sysdrawgraphicarray.nxc](#), [ex\\_sysdrawpolygon.nxc](#), [ex\\_syslistfiles.nxc](#), [ex\\_sysmemorymanager.nxc](#), [ex\\_UnflattenVar.nxc](#), [ex\\_wait.nxc](#), [ex\\_xg1300.nxc](#), [ex\\_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

### 8.3.3.1083 unsigned int Write (byte *handle*, const variant & *value*) [*inline*]

Write value to file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written.

#### Parameters:

*handle* The file handle.

*value* The value to write to the file.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Examples:

[ex\\_file\\_system.nxc](#), and [ex\\_Write.nxc](#).

**8.3.3.1084 unsigned int WriteBytes (byte *handle*, const byte & *buf* [], unsigned int & *cnt*) [inline]**

Write bytes to file. Write the contents of the data array to the file associated with the specified handle. The handle parameter must be a variable. The cnt parameter must be a variable. The data parameter must be a byte array. The actual number of bytes written is returned in the cnt parameter.

**Parameters:**

- handle* The file handle.
- buf* The byte array or string containing the data to write.
- cnt* The number of bytes actually written to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteBytes.nxc](#).

**8.3.3.1085 unsigned int WriteBytesEx (byte *handle*, unsigned int & *len*, const byte & *buf* []) [inline]**

Write bytes to a file with limit. Write the specified number of bytes to the file associated with the specified handle. The handle parameter must be a variable. The len parameter must be a variable. The buf parameter must be a byte array or a string variable or string constant. The actual number of bytes written is returned in the len parameter.

**Parameters:**

- handle* The file handle.
- len* The maximum number of bytes to write on input. Returns the actual number of bytes written.
- buf* The byte array or string containing the data to write.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteBytesEx.nxc](#).

**8.3.3.1086 char WriteI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte *val*)  
[inline]**

Write I2C register. Write a single byte to an I2C device register.

**Parameters:**

*port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.

*i2caddr* The I2C device address.

*reg* The I2C device register to which to write a single byte.

*val* The byte to write to the I2C device.

**Returns:**

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

**Examples:**

[ex\\_writei2cregister.nxc](#).

**8.3.3.1087 unsigned int WriteLn (byte *handle*, const variant & *value*)  
[inline]**

Write a value and new line to a file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written. This function also writes a carriage return and a line feed to the file following the numeric data.

**Parameters:**

*handle* The file handle.

*value* The value to write to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteLn.nxc](#).

**8.3.3.1088** `unsigned int WriteLnString (byte handle, const string & str, unsigned int & cnt) [inline]`

Write string and new line to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. This function also writes a carriage return and a line feed to the file following the string data. The total number of bytes written is returned in the cnt parameter.

**Parameters:**

- handle* The file handle.
- str* The string to write to the file.
- cnt* The number of bytes actually written to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteLnString.nxc](#).

**8.3.3.1089** `char WriteNRLinkBytes (const byte port, const byte i2caddr, const byte data[]) [inline]`

Write data to NRLink. Write data to the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- data* A byte array containing the data to write.

**Returns:**

The function call result.

**Examples:**

[ex\\_writenrlinkbytes.nxc](#).

**8.3.3.1090 unsigned int WriteString (byte *handle*, const string & *str*, unsigned int & *cnt*) [inline]**

Write string to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. The actual number of bytes written is returned in the cnt parameter.

**Parameters:**

- handle* The file handle.
- str* The string to write to the file.
- cnt* The number of bytes actually written to the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

[ex\\_WriteString.nxc](#).

**8.3.3.1091 void Yield () [inline]**

Yield to another task. Make a task yield to another concurrently running task.

**Examples:**

[ex\\_yield.nxc](#).

## 9 Example Documentation

### 9.1 alternating\_tasks.nxc

This is an example of how to use the [ExitTo](#) function.

```
// When run, this program alternates between task A and task B until halted
// by pressing the gray button.
```

```
task B();
```

```
void beep(const int tone)
{
    PlayTone(tone, MS_500);
    Wait(SEC_1);
}

task A()
{
    beep(TONE_C4);
    ExitTo(B);
}

task B()
{
    beep(TONE_C6);
    ExitTo(A);
}

task main()
{
    // ExitTo(B) would work as well here.
    Precedes(B);
}
```

## 9.2 ex\_abort.nxc

This is an example of how to use the [abort](#) function.

```
abort(); // stop the program
```

## 9.3 ex\_AbortFlag.nxc

This is an example of how to use the [AbortFlag](#) function.

```
byte af = AbortFlag();
```

## 9.4 ex\_abs.nxc

This is an example of how to use the [abs](#) function.

```
float val = abs(x); // return the absolute value of x
```

## 9.5 ex\_ACCLNxCalibrateX.nxc

This is an example of how to use the [ACCLNxCalibrateX](#) function.

```
result = ACCLNxCalibrateX(S1, MS_ADDR_ACCLNX);
```

### 9.6 ex\_ACCLNxCalibrateXEnd.nxc

This is an example of how to use the [ACCLNxCalibrateXEnd](#) function.

```
result = ACCLNxCalibrateXEnd(S1, MS_ADDR_ACCLNX);
```

### 9.7 ex\_ACCLNxCalibrateY.nxc

This is an example of how to use the [ACCLNxCalibrateY](#) function.

```
result = ACCLNxCalibrateY(S1, MS_ADDR_ACCLNX);
```

### 9.8 ex\_ACCLNxCalibrateYEnd.nxc

This is an example of how to use the [ACCLNxCalibrateYEnd](#) function.

```
result = ACCLNxCalibrateYEnd(S1, MS_ADDR_ACCLNX);
```

### 9.9 ex\_ACCLNxCalibrateZ.nxc

This is an example of how to use the [ACCLNxCalibrateZ](#) function.

```
result = ACCLNxCalibrateZ(S1, MS_ADDR_ACCLNX);
```

### 9.10 ex\_ACCLNxCalibrateZEnd.nxc

This is an example of how to use the [ACCLNxCalibrateZEnd](#) function.

```
result = ACCLNxCalibrateZEnd(S1, MS_ADDR_ACCLNX);
```

### 9.11 ex\_ACCLNxResetCalibration.nxc

This is an example of how to use the [ACCLNxResetCalibration](#) function.

```
result = ACCLNxResetCalibration(S1, MS_ADDR_ACCLNX);
```

### 9.12 ex\_ACCLNxSensitivity.nxc

This is an example of how to use the [ACCLNxSensitivity](#) function.

```
result = ACCLNxSensitivity(S1, MS_ADDR_ACCLNX);
```

### 9.13 ex\_ACCLNxXOffset.nxc

This is an example of how to use the [ACCLNxXOffset](#) function.

```
result = ACCLNxXOffset(S1, MS_ADDR_ACCLNX);
```

### 9.14 ex\_ACCLNxXRange.nxc

This is an example of how to use the [ACCLNxXRange](#) function.

```
result = ACCLNxXRange(S1, MS_ADDR_ACCLNX);
```

### 9.15 ex\_ACCLNxYOffset.nxc

This is an example of how to use the [ACCLNxYOffset](#) function.

```
result = ACCLNxYOffset(S1, MS_ADDR_ACCLNX);
```

### 9.16 ex\_ACCLNxYRange.nxc

This is an example of how to use the [ACCLNxYRange](#) function.

```
result = ACCLNxYRange(S1, MS_ADDR_ACCLNX);
```

### 9.17 ex\_ACCLNxZOffset.nxc

This is an example of how to use the [ACCLNxZOffset](#) function.

```
result = ACCLNxZOffset(S1, MS_ADDR_ACCLNX);
```

### 9.18 ex\_ACCLNxZRange.nxc

This is an example of how to use the [ACCLNxZRange](#) function.

```
result = ACCLNxZRange(S1, MS_ADDR_ACCLNX);
```

## 9.19 ex\_acos.nxc

This is an example of how to use the [acos](#) function.

```
// ex_acos.nxc
// Display values of the acos API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_acos(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%7.4f RAD", acos(val)));
}

task main()
{
    show_acos(MIN_VAL, LCD_LINE1); // shows 3.1416 RAD
    show_acos(MID_VAL, LCD_LINE2); // shows 1.5708 RAD
    show_acos(MAX_VAL, LCD_LINE3); // shows 0.0000 RAD
    // An invalid value returns not-a-number (nan).
    show_acos(INVALID, LCD_LINE4); // shows -nan RAD
    while (true);
}
```

## 9.20 ex\_acosd.nxc

This is an example of how to use the [acosd](#) function.

```
// ex_acosd.nxc
// Display values of the acosd API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_acos(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", acosd(val)));
}

task main()
{
    show_acos(MIN_VAL, LCD_LINE1); // shows 180.00 DEG
    show_acos(MID_VAL, LCD_LINE2); // shows 90.00 DEG
    show_acos(MAX_VAL, LCD_LINE3); // shows 0.00 DEG
    // An invalid value returns not-a-number (nan).
}
```

```

    show_acos(INVALID, LCD_LINE4); // shows  -nan DEG
    while (true);
}

```

## 9.21 ex\_Acquire.nxc

This is an example of how to use the [Acquire](#) function.

```

mutex motorMutex;
// ...
Acquire(motorMutex); // make sure we have exclusive access
// use the motors
Release(motorMutex);

```

## 9.22 ex\_addressof.nxc

This is an example of how to use the [addressOf](#) function.

```

const byte NewFont[] =
{
    0x04,0x00, // Graphics Format
    0x02,0x40, // Graphics DataSize
    0x10,     // Graphics Count X
    0x06,     // Graphics Count Y
    0x06,     // Graphics Width
    0x08,     // Graphics Height
    0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,0x06,0x00,0x00,0x07,0x03,0x00,0x07
    ,0x03,0x00,0x24,0x7E,0x24,0x7E,0x24,0x00,0x24,0x2B,0x6A,0x12,0x00,0x00,0x63,0x13,
    0x08,0x64,0x63,0x00,0x30,0x4C,0x52,0x22,0x50,0x00,0x00,0x07,0x03,0x00,0x00,0x00,0
    x00,0x3E,0x41,0x00,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x
    08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x00,
    8,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,
    0x3E,0x51,0x49,0x45,0x3E,0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x62,0x51,0x49,0x49
    ,0x46,0x00,0x22,0x49,0x49,0x49,0x36,0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x2F,0x49,
    0x49,0x49,0x31,0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
    x36,0x49,0x49,0x49,0x36,0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00,0x6C,0x6C,0x00,0x
    00,0x00,0x00,0xEC,0x6C,0x00,0x00,0x00,0x08,0x14,0x22,0x41,0x00,0x00,0x24,0x24,0x2
    4,0x24,0x24,0x00,0x00,0x41,0x22,0x14,0x08,0x00,0x02,0x01,0x59,0x09,0x06,0x00,
    0x3E,0x41,0x5D,0x55,0x1E,0x00,0x7E,0x11,0x11,0x11,0x7E,0x00,0x7F,0x49,0x49,0x49
    ,0x36,0x00,0x3E,0x41,0x41,0x41,0x22,0x00,0x7F,0x41,0x41,0x41,0x3E,0x00,0x7F,0x49,
    0x49,0x49,0x41,0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
    x7F,0x08,0x08,0x08,0x7F,0x00,0x00,0x41,0x7F,0x41,0x00,0x00,0x30,0x40,0x40,0x40,0x
    3F,0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x7F,0x40,0x40,0x40,0x40,0x00,0x7F,0x02,0x0
    4,0x02,0x7F,0x00,0x7F,0x02,0x04,0x08,0x7F,0x00,0x3E,0x41,0x41,0x41,0x3E,0x00,
    0x7F,0x09,0x09,0x09,0x06,0x00,0x3E,0x41,0x51,0x21,0x5E,0x00,0x7F,0x09,0x09,0x19
    ,0x66,0x00,0x26,0x49,0x49,0x49,0x32,0x00,0x01,0x01,0x7F,0x01,0x01,0x00,0x3F,0x40,
    0x40,0x40,0x3F,0x00,0x1F,0x20,0x40,0x20,0x1F,0x00,0x3F,0x40,0x3C,0x40,0x3F,0x00,0
    x63,0x14,0x08,0x14,0x63,0x00,0x07,0x08,0x70,0x08,0x07,0x00,0x71,0x49,0x45,0x43,0x
    00,0x00,0x00,0x7F,0x41,0x41,0x00,0x00,0x02,0x04,0x08,0x10,0x20,0x00,0x00,0x41,0x4
    1,0x7F,0x00,0x00,0x04,0x02,0x01,0x02,0x04,0x00,0x80,0x80,0x80,0x80,0x80,0x00,
    0x00,0x02,0x05,0x02,0x00,0x00,0x20,0x54,0x54,0x54,0x78,0x00,0x7F,0x44,0x44,0x44

```

```

, 0x38, 0x00, 0x38, 0x44, 0x44, 0x44, 0x28, 0x00, 0x38, 0x44, 0x44, 0x44, 0x7F, 0x00, 0x38, 0x54,
0x54, 0x54, 0x08, 0x00, 0x08, 0x7E, 0x09, 0x09, 0x00, 0x00, 0x18, 0x24, 0xA4, 0xA4, 0xFC, 0x00, 0
x7F, 0x04, 0x04, 0x78, 0x00, 0x00, 0x00, 0x00, 0x7D, 0x40, 0x00, 0x00, 0x40, 0x80, 0x84, 0x7D, 0x
00, 0x00, 0x7F, 0x10, 0x28, 0x44, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x40, 0x00, 0x00, 0x7C, 0x04, 0x1
8, 0x04, 0x78, 0x00, 0x7C, 0x04, 0x04, 0x78, 0x00, 0x00, 0x38, 0x44, 0x44, 0x44, 0x38, 0x00,
0xFC, 0x44, 0x44, 0x44, 0x38, 0x00, 0x38, 0x44, 0x44, 0x44, 0xFC, 0x00, 0x44, 0x78, 0x44, 0x04
, 0x08, 0x00, 0x08, 0x54, 0x54, 0x54, 0x20, 0x00, 0x04, 0x3E, 0x44, 0x24, 0x00, 0x00, 0x3C, 0x40,
0x20, 0x7C, 0x00, 0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C, 0x00, 0x3C, 0x60, 0x30, 0x60, 0x3C, 0x00, 0
x6C, 0x10, 0x10, 0x6C, 0x00, 0x00, 0x9C, 0xA0, 0x60, 0x3C, 0x00, 0x00, 0x64, 0x54, 0x54, 0x4C, 0x
00, 0x00, 0x08, 0x3E, 0x41, 0x41, 0x00, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x41, 0x4
1, 0x3E, 0x08, 0x00, 0x02, 0x01, 0x02, 0x01, 0x00, 0x00, 0x10, 0x20, 0x40, 0x38, 0x07, 0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    ptr = addressOf(NewFont);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

## 9.23 ex\_addressofex.nxc

This is an example of how to use the [addressOfEx](#) function.

```

const byte NewFont[] =
{
    0x04, 0x00, // Graphics Format
    0x02, 0x40, // Graphics DataSize
    0x10,      // Graphics Count X
    0x06,      // Graphics Count Y
    0x06,      // Graphics Width
    0x08,      // Graphics Height
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x5F, 0x06, 0x00, 0x00, 0x07, 0x03, 0x00, 0x07
, 0x03, 0x00, 0x24, 0x7E, 0x24, 0x7E, 0x24, 0x00, 0x24, 0x2B, 0x6A, 0x12, 0x00, 0x00, 0x63, 0x13,
0x08, 0x64, 0x63, 0x00, 0x30, 0x4C, 0x52, 0x22, 0x50, 0x00, 0x00, 0x07, 0x03, 0x00, 0x00, 0x00, 0
x00, 0x3E, 0x41, 0x00, 0x00, 0x00, 0x00, 0x41, 0x3E, 0x00, 0x00, 0x00, 0x08, 0x3E, 0x1C, 0x3E, 0x
08, 0x00, 0x08, 0x08, 0x3E, 0x08, 0x08, 0x00, 0x80, 0x60, 0x60, 0x00, 0x00, 0x00, 0x08, 0x08, 0x0
8, 0x08, 0x08, 0x00, 0x00, 0x60, 0x60, 0x00, 0x00, 0x00, 0x20, 0x10, 0x08, 0x04, 0x02, 0x00,
0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00, 0x00, 0x42, 0x7F, 0x40, 0x00, 0x00, 0x62, 0x51, 0x49, 0x49
, 0x46, 0x00, 0x22, 0x49, 0x49, 0x49, 0x36, 0x00, 0x18, 0x14, 0x12, 0x7F, 0x10, 0x00, 0x2F, 0x49,
0x49, 0x49, 0x31, 0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30, 0x00, 0x01, 0x71, 0x09, 0x05, 0x03, 0x00, 0
x36, 0x49, 0x49, 0x49, 0x36, 0x00, 0x06, 0x49, 0x49, 0x29, 0x1E, 0x00, 0x00, 0x6C, 0x6C, 0x00, 0x
00, 0x00, 0x00, 0xEC, 0x6C, 0x00, 0x00, 0x00, 0x08, 0x14, 0x22, 0x41, 0x00, 0x00, 0x24, 0x24, 0x2
4, 0x24, 0x24, 0x00, 0x00, 0x41, 0x22, 0x14, 0x08, 0x00, 0x02, 0x01, 0x59, 0x09, 0x06, 0x00,
0x3E, 0x41, 0x5D, 0x55, 0x1E, 0x00, 0x7E, 0x11, 0x11, 0x11, 0x7E, 0x00, 0x7F, 0x49, 0x49, 0x49
, 0x36, 0x00, 0x3E, 0x41, 0x41, 0x41, 0x22, 0x00, 0x7F, 0x41, 0x41, 0x41, 0x3E, 0x00, 0x7F, 0x49,
0x49, 0x49, 0x41, 0x00, 0x7F, 0x09, 0x09, 0x09, 0x01, 0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A, 0x00, 0
x7F, 0x08, 0x08, 0x08, 0x7F, 0x00, 0x00, 0x41, 0x7F, 0x41, 0x00, 0x00, 0x30, 0x40, 0x40, 0x40, 0x
3F, 0x00, 0x7F, 0x08, 0x14, 0x22, 0x41, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x00, 0x7F, 0x02, 0x0

```

```

    4, 0x02, 0x7F, 0x00, 0x7F, 0x02, 0x04, 0x08, 0x7F, 0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E, 0x00,
0x7F, 0x09, 0x09, 0x09, 0x06, 0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E, 0x00, 0x7F, 0x09, 0x09, 0x19
, 0x66, 0x00, 0x26, 0x49, 0x49, 0x49, 0x32, 0x00, 0x01, 0x01, 0x7F, 0x01, 0x01, 0x00, 0x3F, 0x40,
0x40, 0x40, 0x3F, 0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F, 0x00, 0x3F, 0x40, 0x3C, 0x40, 0x3F, 0x00, 0
x63, 0x14, 0x08, 0x14, 0x63, 0x00, 0x07, 0x08, 0x70, 0x08, 0x07, 0x00, 0x71, 0x49, 0x45, 0x43, 0x
00, 0x00, 0x00, 0x7F, 0x41, 0x41, 0x00, 0x00, 0x02, 0x04, 0x08, 0x10, 0x20, 0x00, 0x00, 0x41, 0x4
1, 0x7F, 0x00, 0x00, 0x04, 0x02, 0x01, 0x02, 0x04, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00,
0x00, 0x02, 0x05, 0x02, 0x00, 0x00, 0x20, 0x54, 0x54, 0x54, 0x78, 0x00, 0x7F, 0x44, 0x44, 0x44
, 0x38, 0x00, 0x38, 0x44, 0x44, 0x44, 0x28, 0x00, 0x38, 0x44, 0x44, 0x44, 0x7F, 0x00, 0x38, 0x54,
0x54, 0x54, 0x08, 0x00, 0x08, 0x7E, 0x09, 0x09, 0x00, 0x00, 0x18, 0x24, 0xA4, 0xA4, 0xFC, 0x00, 0
x7F, 0x04, 0x04, 0x78, 0x00, 0x00, 0x00, 0x00, 0x7D, 0x40, 0x00, 0x00, 0x40, 0x80, 0x84, 0x7D, 0x
00, 0x00, 0x7F, 0x10, 0x28, 0x44, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x40, 0x00, 0x00, 0x7C, 0x04, 0x1
8, 0x04, 0x78, 0x00, 0x7C, 0x04, 0x04, 0x78, 0x00, 0x00, 0x38, 0x44, 0x44, 0x44, 0x38, 0x00,
0xFC, 0x44, 0x44, 0x44, 0x38, 0x00, 0x38, 0x44, 0x44, 0x44, 0xFC, 0x00, 0x44, 0x78, 0x44, 0x04
, 0x08, 0x00, 0x08, 0x54, 0x54, 0x54, 0x20, 0x00, 0x04, 0x3E, 0x44, 0x24, 0x00, 0x00, 0x3C, 0x40,
0x20, 0x7C, 0x00, 0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C, 0x00, 0x3C, 0x60, 0x30, 0x60, 0x3C, 0x00, 0
x6C, 0x10, 0x10, 0x6C, 0x00, 0x00, 0x9C, 0xA0, 0x60, 0x3C, 0x00, 0x00, 0x64, 0x54, 0x54, 0x4C, 0x
00, 0x00, 0x08, 0x3E, 0x41, 0x41, 0x00, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x41, 0x4
1, 0x3E, 0x08, 0x00, 0x02, 0x01, 0x02, 0x01, 0x00, 0x00, 0x10, 0x20, 0x40, 0x38, 0x07, 0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    ptr = addressOfEx(NewFont, false);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

## 9.24 ex\_ArrayBuild.nxc

This is an example of how to use the [ArrayBuild](#) function.

```

task main()
{
    byte myArray[];
    byte src1 = 0x45, src2 = 0x1f, srcN = 0x7a;

    ArrayBuild(myArray, src1, src2, srcN);
    // myArray = {0x45, 0x1f, 0x7a};

    int abSample[];
    int s1[] = {0, 1, 2, 3};
    int s2 = 4, s3 = 5, s4 = 6, sN[] = {7, 8};
    ArrayBuild(abSample, s1, s2, s3, s4, sN);
    // abSample = {0, 1, 2, 3, 4, 5, 6, 7, 8};
    NumOut(0, LCD_LINE4, myArray[2]);
    NumOut(0, LCD_LINE5, abSample[1]);
}

```

### 9.25 ex\_ArrayInit.nxc

This is an example of how to use the [ArrayInit](#) function.

```
ArrayInit(myArray, 0, 10); // 10 elements == zero
```

### 9.26 ex\_ArrayLen.nxc

This is an example of how to use the [ArrayLen](#) function.

```
x = ArrayLen(myArray);
```

### 9.27 ex\_ArrayMax.nxc

This is an example of how to use the [ArrayMax](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = Random();
    TextOut(0, LCD_LINE1, "Max value = ");
    int x = ArrayMax(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

### 9.28 ex\_ArrayMean.nxc

This is an example of how to use the [ArrayMean](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Mean value = ");
    int x = ArrayMean(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

### 9.29 ex\_ArrayMin.nxc

This is an example of how to use the [ArrayMin](#) function.

```
task main()
{
  int data[40];
  for (int i = 0; i < 40; i++)
    data[i] = rand();
  TextOut(0, LCD_LINE1, "Min value = ");
  int x = ArrayMin(data, NA, NA); // start at 0 and go to length(data);
  NumOut(0, LCD_LINE2, x);
  Wait(SEC_3);
}
```

### 9.30 ex\_ArrayOp.nxc

This is an example of how to use the [ArrayOp](#) function.

```
task main()
{
  int data[40];
  for (int i = 0; i < 40; i++)
    data[i] = rand();
  TextOut(0, LCD_LINE1, "Max value = ");
  int x;
  ArrayOp(OPARR_MAX, x, data, NA, NA); // start at 0 and go to length(data);
  NumOut(0, LCD_LINE2, x);
  Wait(SEC_3);
}
```

### 9.31 ex\_ArraySort.nxc

This is an example of how to use the [ArraySort](#) function.

```
task main()
{
  int data[40];
  int tmp[];
  for (int i = 0; i < 40; i++)
    data[i] = rand();
  ArraySort(tmp, data, NA, NA); // start at 0 and go to length(data);
  TextOut(0, LCD_LINE1, "Min value = ");
  NumOut(0, LCD_LINE2, tmp[0]);
  TextOut(0, LCD_LINE3, "Max value = ");
  NumOut(0, LCD_LINE4, tmp[39]);
  TextOut(0, LCD_LINE5, "Min value = ");
  NumOut(0, LCD_LINE6, ArrayMin(data, NA, NA));
  TextOut(0, LCD_LINE7, "Max value = ");
  NumOut(0, LCD_LINE8, ArrayMax(data, NA, NA));
  Wait(SEC_3);
}
```

### 9.32 ex\_ArrayStd.nxc

This is an example of how to use the [ArrayStd](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "StdDev values = ");
    long x = ArrayStd(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

### 9.33 ex\_ArraySubset.nxc

This is an example of how to use the [ArraySubset](#) function.

```
// copy 5 elements starting with the 3rd element, i.e., srcArray[2]
ArraySubset(myArray, srcArray, 2, 5);
```

### 9.34 ex\_ArraySum.nxc

This is an example of how to use the [ArraySum](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Sum of values = ");
    long x = ArraySum(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

### 9.35 ex\_ArraySumSqr.nxc

This is an example of how to use the [ArraySumSqr](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "SumSqr values = ");
```

```
    long x = ArraySumSqr(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

### 9.36 ex\_asin.nxc

This is an example of how to use the [asin](#) function.

```
// ex_asin.nxc
// Display values of the asin API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_asin(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%7.4f RAD", asin(val)));
}

task main()
{
    show_asin(MIN_VAL, LCD_LINE1); // shows -1.5708 RAD
    show_asin(MID_VAL, LCD_LINE2); // shows 0.0000 RAD
    show_asin(MAX_VAL, LCD_LINE3); // shows 1.5708 RAD
    // An invalid value returns not-a-number (nan).
    show_asin(INVALID, LCD_LINE4); // shows -nan RAD
    while (true);
}
```

### 9.37 ex\_asind.nxc

This is an example of how to use the [asind](#) function.

```
// ex_asind.nxc
// Display values of the asind API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_asin(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", asind(val)));
}
```

```

task main()
{
  show_asin(MIN_VAL, LCD_LINE1); // shows -90.00 DEG
  show_asin(MID_VAL, LCD_LINE2); // shows  0.00 DEG
  show_asin(MAX_VAL, LCD_LINE3); // shows  90.00 DEG
  // An invalid value returns not-a-number (nan).
  show_asin(INVALID, LCD_LINE4); // shows  -nan DEG
  while (true);
}

```

### 9.38 ex\_atan.nxc

This is an example of how to use the [atan](#) function.

```

// ex_atan.nxc
// Display values of the atan API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define BIG_NEG_VAL -1000.0
#define NEG_VAL -1.0
#define POS_VAL 1.0
#define BIG_POS_VAL 1000.0

inline void show_atan(const float val, const int screen_y)
{
  TextOut(0, screen_y, FormatNum("%7.4f RAD", atan(val)));
}

task main()
{
  show_atan(BIG_NEG_VAL, LCD_LINE1); // shows -1.5698 RAD
  show_atan(NEG_VAL, LCD_LINE2); // shows -0.7854 RAD
  show_atan(0.0, LCD_LINE3); // shows 0.0000 RAD
  show_atan(POS_VAL, LCD_LINE4); // shows 0.7854 RAD
  show_atan(BIG_POS_VAL, LCD_LINE5); // shows 1.5698 RAD
  while (true);
}

```

### 9.39 ex\_atan2.nxc

This is an example of how to use the [atan2](#) function.

```

// ex_atan2.nxc
// Display values of the atan2 API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

// The following two arrays comprise the x and y coordinates of the corners and
// the mid-points of the sides of a square centered at the origin and having
// sides two units long.

```

```

const float y_coord[] = {-1.0, -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0};
const float x_coord[] = {-1.0, 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0};

// Display the angles made by lines from the origin to the points on the square
// as specified above.
task main()
{
    const int pts = ArrayLen(y_coord);
    for (int i = 0; i < pts; ++i)
    {
        float angle = atan2(y_coord[i], x_coord[i]);
        TextOut(0, 56 - 8 * i, FormatNum("%7.4f RAD", angle));
    }
    while (true);
}

```

## 9.40 ex\_atan2d.nxc

This is an example of how to use the [atan2d](#) function.

```

// ex_atan2d.nxc
// Display values of the atan2d API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

// The following two arrays comprise the x and y coordinates of the corners and
// the mid-points of the sides of a square centered at the origin and having
// sides two units long.
const float y_coord[] = {-1.0, -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0};
const float x_coord[] = {-1.0, 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0};

// Display the angles made by lines from the origin to the points on the square
// as specified above.
task main()
{
    const int pts = ArrayLen(y_coord);
    for (int i = 0; i < pts; ++i)
    {
        float angle = atan2d(y_coord[i], x_coord[i]);
        TextOut(0, 56 - 8 * i, FormatNum("%7.2f DEG", angle));
    }
    while (true);
}

```

## 9.41 ex\_atand.nxc

This is an example of how to use the [atand](#) function.

```

// ex_atand.nxc
// Display values of the atand API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

```

```
#define BIG_NEG_VAL -1000.0
#define NEG_VAL -1.0
#define POS_VAL 1.0
#define BIG_POS_VAL 1000.0

inline void show_atan(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", atand(val)));
}

task main()
{
    show_atan(BIG_NEG_VAL, LCD_LINE1); // shows -89.94 DEG
    show_atan(NEG_VAL, LCD_LINE2);    // shows -45.00 DEG
    show_atan(0.0, LCD_LINE3);        // shows  0.00 DEG
    show_atan(POS_VAL, LCD_LINE4);     // shows  45.00 DEG
    show_atan(BIG_POS_VAL, LCD_LINE5); // shows  89.94 DEG
    while (true);
}
```

## 9.42 ex\_atof.nxc

This is an example of how to use the [atof](#) function.

```
task main()
{
    float f = atof("3.14159e2");
    NumOut(0, LCD_LINE1, f);
    Wait(SEC_5);
}
```

## 9.43 ex\_atoi.nxc

This is an example of how to use the [atoi](#) function.

```
task main()
{
    NumOut(0, LCD_LINE1, atoi("3.14159"));
    Wait(SEC_5);
}
```

## 9.44 ex\_atol.nxc

This is an example of how to use the [atol](#) function.

```
task main()
{
    NumOut(0, LCD_LINE1, atol("3.142e2"));
    Wait(SEC_5);
}
```

### 9.45 ex\_BatteryState.nxc

This is an example of how to use the [BatteryState](#) function.

```
x = BatteryState();
```

### 9.46 ex\_bcd2dec.nxc

This is an example of how to use the [bcd2dec](#) function.

```
// convert binary-coded decimal byte to decimal.  
byte dec = bcd2dec(0x3a);
```

### 9.47 ex\_BluetoothState.nxc

This is an example of how to use the [BluetoothState](#) function.

```
x = BluetoothState();
```

### 9.48 ex\_BluetoothStatus.nxc

This is an example of how to use the [BluetoothStatus](#) function.

```
x = BluetoothStatus(1);
```

### 9.49 ex\_BluetoothWrite.nxc

This is an example of how to use the [BluetoothWrite](#) function.

```
x = BluetoothWrite(1, data);
```

### 9.50 ex\_BrickDataBluecoreVersion.nxc

This is an example of how to use the [BrickDataBluecoreVersion](#) function.

```
int bv = BrickDataBluecoreVersion();
```

### 9.51 ex\_BrickDataBtHardwareStatus.nxc

This is an example of how to use the [BrickDataBtHardwareStatus](#) function.

```
int x = BrickDataBtHardwareStatus();
```

### 9.52 ex\_BrickDataBtStateStatus.nxc

This is an example of how to use the [BrickDataBtStateStatus](#) function.

```
int x = BrickDataBtStateStatus();
```

### 9.53 ex\_BrickDataName.nxc

This is an example of how to use the [BrickDataName](#) function.

```
string name = BrickDataName();
```

### 9.54 ex\_BrickDataTimeoutValue.nxc

This is an example of how to use the [BrickDataTimeoutValue](#) function.

```
int x = BrickDataTimeoutValue();
```

### 9.55 ex\_BTConnectionClass.nxc

This is an example of how to use the [BTConnectionClass](#) function.

```
long class = BTConnectionClass(1);
```

### 9.56 ex\_BTConnectionHandleNum.nxc

This is an example of how to use the [BTConnectionHandleNum](#) function.

```
byte handlenum = BTConnectionHandleNum(idx);
```

### 9.57 ex\_BTConnectionLinkQuality.nxc

This is an example of how to use the [BTConnectionLinkQuality](#) function.

```
byte linkquality = BTConnectionLinkQuality(1);
```

### 9.58 ex\_BTConnectionName.nxc

This is an example of how to use the [BTConnectionName](#) function.

```
string name = BTConnectionName(0);
```

### 9.59 ex\_BTConnectionPinCode.nxc

This is an example of how to use the [BTConnectionPinCode](#) function.

```
string pincode = BTConnectionPinCode(0);
```

### 9.60 ex\_BTConnectionStreamStatus.nxc

This is an example of how to use the [BTConnectionStreamStatus](#) function.

```
byte streamstatus = BTConnectionStreamStatus(1);
```

### 9.61 ex\_BTDeviceClass.nxc

This is an example of how to use the [BTDeviceClass](#) function.

```
long class = BTDeviceClass(1);
```

### 9.62 ex\_BTDeviceCount.nxc

This is an example of how to use the [BTDeviceCount](#) function.

```
byte x = BTDeviceCount();
```

### 9.63 ex\_BTDeviceName.nxc

This is an example of how to use the [BTDeviceName](#) function.

```
string name = BTDeviceName(0);
```

### 9.64 ex\_BTDeviceNameCount.nxc

This is an example of how to use the [BTDeviceNameCount](#) function.

```
byte x = BTDeviceNameCount();
```

### 9.65 ex\_BTDeviceStatus.nxc

This is an example of how to use the [BTDeviceStatus](#) function.

```
byte status = BTDeviceStatus(1);
```

### 9.66 ex\_BTInputBufferInPtr.nxc

This is an example of how to use the [BTInputBufferInPtr](#) function.

```
byte x = BTInputBufferInPtr();
```

### 9.67 ex\_BTInputBufferOutPtr.nxc

This is an example of how to use the [BTInputBufferOutPtr](#) function.

```
byte x = BTInputBufferOutPtr();
```

### 9.68 ex\_BTOutputBufferInPtr.nxc

This is an example of how to use the [BTOutputBufferInPtr](#) function.

```
byte x = BTOutputBufferInPtr();
```

### 9.69 ex\_BTOutputBufferOutPtr.nxc

This is an example of how to use the [BTOutputBufferOutPtr](#) function.

```
byte x = BTOutputBufferOutPtr();
```

### 9.70 ex\_ButtonCount.nxc

This is an example of how to use the [ButtonCount](#) function.

```
value = ButtonCount(BTN1, true);
```

### 9.71 ex\_ButtonLongPressCount.nxc

This is an example of how to use the [ButtonLongPressCount](#) function.

```
value = ButtonLongPressCount(BTN1);
```

### 9.72 ex\_ButtonLongReleaseCount.nxc

This is an example of how to use the [ButtonLongReleaseCount](#) function.

```
value = ButtonLongReleaseCount(BTN1);
```

### 9.73 ex\_ButtonPressCount.nxc

This is an example of how to use the [ButtonPressCount](#) function.

```
value = ButtonPressCount(BTN1);
```

### 9.74 ex\_buttonpressed.nxc

This is an example of how to use the [ButtonPressed](#) function.

```
task main()  
{  
  #ifdef __ENHANCED_FIRMWARE  
    SetLongAbort(true);  
  }
```

```
#endif
while(true)
{
#ifdef __ENHANCED_FIRMWARE
    NumOut(0, LCD_LINE1, ButtonPressed(BTNEXIT, false));
#endif
    NumOut(0, LCD_LINE2, ButtonPressed(BTNRIGHT, false));
    NumOut(0, LCD_LINE3, ButtonPressed(BTNLEFT, false));
    NumOut(0, LCD_LINE4, ButtonPressed(BTNCENTER, false));
}
}
```

### 9.75 ex\_ButtonReleaseCount.nxc

This is an example of how to use the [ButtonReleaseCount](#) function.

```
value = ButtonReleaseCount(BTN1);
```

### 9.76 ex\_ButtonShortReleaseCount.nxc

This is an example of how to use the [ButtonShortReleaseCount](#) function.

```
value = ButtonShortReleaseCount(BTN1);
```

### 9.77 ex\_ButtonState.nxc

This is an example of how to use the [ButtonState](#) function.

```
value = ButtonState(BTN1);
```

### 9.78 ex\_ByteArrayToStr.nxc

This is an example of how to use the [ByteArrayToStr](#) function.

```
myStr = ByteArrayToStr(myArray);
```

### 9.79 ex\_ByteArrayToStrEx.nxc

This is an example of how to use the [ByteArrayToStrEx](#) function.

```
ByteArrayToStrEx(myArray, myStr);
```

## 9.80 ex\_ceil.nxc

This is an example of how to use the [ceil](#) function.

```
float a = ceil(3.01);
// a == 4.0
float b = ceil(3.14);
// b == 4.0
float c = ceil(3.99);
// c == 4.0
float d = ceil(4.0);
// d == 4.0
```

## 9.81 ex\_CircleOut.nxc

This is an example of how to use the [CircleOut](#), [Random](#), and [Wait](#) functions.

```
task main()
{
  for (int i=0; i < 50; i++) {
    int x = Random(10)+50;
    int y = Random(10)+32;
    int r = Random(20);
    CircleOut(x, y, r, DRAW_OPT_NORMAL+DRAW_OPT_LOGICAL_XOR+DRAW_OPT_FILL_SHAPE);

    Wait(MS_50);
  }
  CircleOut(20, 50, 20);
  Wait(SEC_2);
}
```

## 9.82 ex\_clearline.nxc

This is an example of how to use the [TextOut](#), [ClearLine](#), and [Wait](#) functions.

```
task main()
{
  TextOut(0, LCD_LINE1, "testing 1234567890");
  Wait(SEC_5);
  ClearLine(LCD_LINE1);
  Wait(SEC_5);
  TextOut(0, LCD_LINE1, "testing 1234567890");
  Wait(SEC_5);
}
```

## 9.83 ex\_ClearScreen.nxc

This is an example of how to use the [ClearScreen](#) and [Wait](#) functions.

```
task main()
{
  ClearScreen();
  Wait(SEC_10);
}
```

### 9.84 ex\_ClearSensor.nxc

This is an example of how to use the [ClearSensor](#) function.

```
ClearSensor(S1);
```

### 9.85 ex\_CloseFile.nxc

This is an example of how to use the [CloseFile](#) function.

```
result = CloseFile(handle);
```

### 9.86 ex\_coast.nxc

This is an example of how to use the [Coast](#) function.

```
Coast(OUT_A); // coast output A
```

### 9.87 ex\_coastex.nxc

This is an example of how to use the [CoastEx](#) function.

```
CoastEx(OUT_A, RESET_NONE); // coast output A
```

### 9.88 ex\_ColorADRaw.nxc

This is an example of how to use the [ColorADRaw](#) function.

```
unsigned int rawRed = ColorADRaw(S1, INPUT_RED);
```

### 9.89 ex\_ColorBoolean.nxc

This is an example of how to use the [ColorBoolean](#) function.

```
bool bRed = ColorBoolean(S1, INPUT_RED);
```

### 9.90 ex\_ColorCalibration.nxc

This is an example of how to use the [ColorCalibration](#) function.

```
long value = ColorCalibration(S1, INPUT_CAL_POINT_0, INPUT_RED);
```

### 9.91 ex\_ColorCalibrationState.nxc

This is an example of how to use the [ColorCalibrationState](#) function.

```
byte value = ColorCalibrationState(S1);
```

### 9.92 ex\_ColorCalLimits.nxc

This is an example of how to use the [ColorCalLimits](#) function.

```
unsigned int limit = ColorCalLimits(S1, INPUT_CAL_POINT_0);
```

### 9.93 ex\_ColorSensorRaw.nxc

This is an example of how to use the [ColorSensorRaw](#) function.

```
unsigned int rawRed = ColorSensorRaw(S1, INPUT_RED);
```

### 9.94 ex\_ColorSensorValue.nxc

This is an example of how to use the [ColorSensorValue](#) function.

```
unsigned int valRed = ColorSensorValue(S1, INPUT_RED);
```

### 9.95 ex\_CommandFlags.nxc

This is an example of how to use the [CommandFlags](#) function.

```
x = CommandFlags();
```

## 9.96 ex\_ConfigureTemperatureSensor.nxc

This is an example of how to use the [ConfigureTemperatureSensor](#) function.

```
byte config = TEMP_RES_12BIT;
char result = ConfigureTemperatureSensor(S1, config);
```

## 9.97 ex\_contrast.nxc

This is an example of how to use the [DisplayContrast](#) and [SetDisplayContrast](#) functions.

```
task main()
{
  for (byte contrast = 0; contrast < DISPLAY_CONTRAST_MAX; contrast++)
  {
    SetDisplayContrast(contrast);
    NumOut(0, LCD_LINE1, DisplayContrast());
    Wait(100);
  }
  for (byte contrast = DISPLAY_CONTRAST_MAX; contrast > 0; contrast--)
  {
    SetDisplayContrast(contrast);
    NumOut(0, LCD_LINE1, DisplayContrast());
    Wait(100);
  }
  SetDisplayContrast(DISPLAY_CONTRAST_DEFAULT);
  NumOut(0, LCD_LINE1, DisplayContrast());
  while(true);
}
```

## 9.98 ex\_copy.nxc

This is an example of how to use the [Copy](#) function.

```
task main()
{
  string s = "Now is the winter of our discontent";
  TextOut(0, LCD_LINE1, Copy(s, 12, 5));
  Wait(SEC_4);
}
```

## 9.99 ex\_cosh.nxc

This is an example of how to use the [cosh](#) function.

```
x = cosh(y);
```

### 9.100 ex\_CreateFile.nxc

This is an example of how to use the [CreateFile](#) function.

```
result = CreateFile("data.txt", 1024, handle);
```

### 9.101 ex\_CreateFileLinear.nxc

This is an example of how to use the [CreateFileLinear](#) function.

```
result = CreateFileLinear("data.txt", 1024, handle);
```

### 9.102 ex\_CreateFileNonLinear.nxc

This is an example of how to use the [CreateFileNonLinear](#) function.

```
result = CreateFileNonLinear("data.txt", 1024, handle);
```

### 9.103 ex\_cstdio.nxc

This is an example of how to use the `cstdio` API functions: [fopen](#), [fprintf](#), [fputc](#), [fputs](#), [fseek](#), [ftell](#), [fclose](#), [feof](#), [fflush](#), [fgetc](#), [fgets](#), [getc](#), [putc](#), [rewind](#), [printf](#), [sprintf](#), [rename](#), and [remove](#).

```
task main()
{
  /*
  fclose(byte handle)
  feof(byte handle)
  fflush(byte handle)
  fgetc(byte handle)
  fgets(string & str, int num, byte handle)
  fopen(string filename, const string mode)
  fprintf(byte handle, const string & format, variant value)
  fputc(char ch, byte handle)
  fputs(string str, byte handle)
  fseek(byte handle, long offset, int origin)
  ftell(byte handle)
  getc(byte handle)
  putc(char ch, byte handle)
  remove(string fname)
  rename(string oldname, string newname)
  rewind(byte handle)
  */
}
```

```
*/
}
```

## 9.104 ex\_cstring.nxc

This is an example of how to use the cstring API functions: [strcat](#), [strcmp](#), [strcpy](#), [strlen](#), [strncat](#), [strncmp](#), [strncpy](#), [memcpy](#), [memmove](#), and [memcmp](#).

```
task main()
{
/*
inline variant StrToNum(string str);
inline unsigned int StrLen(string str);
inline byte StrIndex(string str, unsigned int idx);
inline string NumToStr(variant num);
inline string StrCat(string str1, string str2, string str3, string strN);
inline string SubStr(string str, unsigned int idx, unsigned int len);
inline string Flatten(variant num);
inline string StrReplace(string str, unsigned int idx, string strnew);
inline string FormatNum(string fmt, variant number);

inline string FlattenVar(variant x);
inline int UnflattenVar(string str, variant & variable);
inline string ByteArrayToStr(byte data[]);
inline void ByteArrayToStrEx(byte data[], string & str);
inline void StrToByteArray(string str, byte & data[]);

strcat(string & dest, const string & src)
strcmp(const string & str1, const string & str2)
strcpy(string & dest, const string & src)
strlen(const string & str)
strncat(string & dest, const string & src, const int num)
strncmp(const string & str1, const string & str2, unsigned int num)
strncpy(string & dest, const string & src, const int num)
*/
}
```

## 9.105 ex\_ctype.nxc

This is an example of how to use the ctype API functions: [isupper](#), [islower](#), [isalpha](#), [isdigit](#), [isalnum](#), [isspace](#), [iscntrl](#), [isprint](#), [isgraph](#), [ispunct](#), [isxdigit](#), [toupper](#), and [tolower](#).

```
task main()
{
string tmp = "a1B2.G% ";
TextOut(0, LCD_LINE1, tmp);
NumOut(0, LCD_LINE2, isalnum(tmp[0])); // 1
NumOut(0, LCD_LINE3, isalpha(tmp[1])); // 0
NumOut(0, LCD_LINE4, iscntrl(tmp[2])); // 0
NumOut(0, LCD_LINE5, isdigit(tmp[3])); // 1
}
```

```
NumOut(0, LCD_LINE6, isgraph(tmp[4])); // 1
NumOut(0, LCD_LINE7, islower(tmp[5])); // 0
NumOut(0, LCD_LINE8, isprint(tmp[6])); // 1

NumOut(40, LCD_LINE2, ispunct(tmp[0])); // 0
NumOut(40, LCD_LINE3, isspace(tmp[1])); // 0
NumOut(40, LCD_LINE4, isupper(tmp[2])); // 1
NumOut(40, LCD_LINE5, isxdigit(tmp[3])); // 1
NumOut(40, LCD_LINE6, tolower(tmp[4])); // 46
NumOut(40, LCD_LINE7, toupper(tmp[5])); // 71

Wait(SEC_5);
}
```

### 9.106 ex\_CurrentTick.nxc

This is an example of how to use the [CurrentTick](#) function.

```
unsigned int x = CurrentTick();
```

### 9.107 ex\_CustomSensorActiveStatus.nxc

This is an example of how to use the [CustomSensorActiveStatus](#) function.

```
x = CustomSensorActiveStatus(S1);
```

### 9.108 ex\_CustomSensorPercentFullScale.nxc

This is an example of how to use the [CustomSensorPercentFullScale](#) function.

```
x = CustomSensorPercentFullScale(S1);
```

### 9.109 ex\_CustomSensorZeroOffset.nxc

This is an example of how to use the [CustomSensorZeroOffset](#) function.

```
x = CustomSensorZeroOffset(S1);
```

### 9.110 ex\_DataMode.nxc

This is an example of how to use the [HSDDataMode](#), [BTDataMode](#), [SetHSDDataMode](#), [SetBTDataMode](#), [TextOut](#), and [Wait](#) functions.

```

task main()
{
  string DataModeNames[3] = {"NXT", "GPS", "RAW"};

  byte dm;

  // hi-speed data mode
  dm = HSDataMode();
  TextOut( 0, LCD_LINE1, "HSDataMode: ");
  TextOut(80, LCD_LINE1, DataModeNames[dm]);

  // bluetooth data mode
  dm = BTDataMode();
  TextOut( 0, LCD_LINE2, "BTDataMode: ");
  TextOut(80, LCD_LINE2, DataModeNames[dm]);

  // change hi-speed port to NXT mode
  SetHSDataMode(DATA_MODE_NXT);

  // change Bluetooth to GPS mode
  SetBTDataMode(DATA_MODE_GPS);

  dm = HSDataMode();
  TextOut( 0, LCD_LINE4, "HSDataMode: ");
  TextOut(80, LCD_LINE4, DataModeNames[dm]);

  dm = BTDataMode();
  TextOut( 0, LCD_LINE5, "BTDataMode: ");
  TextOut(80, LCD_LINE5, DataModeNames[dm]);

  Wait(SEC_5);
}

```

## 9.111 ex\_delete\_data\_file.nxc

This is an example of how to use the [DeleteFile](#), [TextOut](#), [FormatNum](#), and [Wait](#) functions. It is useful for deleting the circles.dat file created by the program described in the [ex\\_file\\_system::nxc](#) example.

```

// ex_delete_data_file.nxc
// Demonstrates the use of the DeleteFile API call.
// Useful for deleting the circles.dat file created by the program described
// in the ex_file_system.nxc example.

#define FILE_NAME "circles.dat"

// Display a return code from a file system API call on the NXT screen.
// The codes most likely to be displayed are:
//   LDR_SUCCESS      0x0000
//   LDR_FILENOTFOUND 0x8700
// See "Loader module error codes" to interpret any other code that appears.
void rtn_code_out(const unsigned int code)
{

```

```
    TextOut(0, LCD_LINE1, "code          ");
    TextOut(50, LCD_LINE1, FormatNum("%04x", code));
}

task main()
{
    unsigned int rtn_code = DeleteFile(FILE_NAME);
    rtn_code_out(rtn_code);
    Wait(SEC_5);
}
```

### 9.112 ex\_DeleteFile.nxc

This is an example of how to use the [DeleteFile](#) function.

```
result = DeleteFile("data.txt");
```

### 9.113 ex\_diaccl.nxc

This is an example of how to use the [SetSensorDIACcl](#), [SetSensorDIACclEx](#), [SetSensorDIACclDrift](#), [ReadSensorDIACclDrift](#), [SensorDIACclStatus](#), [ReadSensorDIACcl8Raw](#), [ReadSensorDIACcl8](#), [ReadSensorDIACclRaw](#), and [ReadSensorDIACcl](#) functions.

```
#define DEFAULT
//#define RAW8
#define RAW10

#ifdef RAW10
#undef RAW8
#endif

void CalibrateDIACcl(const byte port, int iter)
{
    TextOut(0, LCD_LINE1, "Calibrating...");
    Wait(SEC_1);
    SetSensorDIACclDrift(port, 0, 0, 0);
    int x = 0, y = 0, z = 0;
    VectorType raw;
    repeat(iter)
    {
        ReadSensorDIACclRaw(S1, raw);
        x += raw.X;
        y += raw.Y;
        z += raw.Z;
        Wait(MS_10);
    }
    x = (0-(x/iter))*2;
    y = (0-(y/iter))*2;
    z = (60-(z/iter))*2;
}
```

```
    NumOut(0, LCD_LINE2, x);
    NumOut(0, LCD_LINE3, y);
    NumOut(0, LCD_LINE4, z);
    Wait(SEC_1);
    SetSensorDIACclDrift(port, x, y, z);
    TextOut(0, LCD_LINE5, "Completed!");
    Wait(SEC_1);
    ClearScreen();
}

task main()
{
#ifdef DEFAULT
    SetSensorDIACcl(S1);
#else
    SetSensorDIACclEx(S1, DIACCL_MODE_GLVL8);
#endif
    VectorType val, raw;
    int dx, dy, dz;

    int i = 0;
    int temp;
    byte status;
    bool done = false;
    CalibrateDIACcl(S1, 100);
    while (!done){
        ClearScreen();
        NumOut(0, LCD_LINE8, SensorDIACclStatus(S1));
        // Read the GYROSCOPE
#ifdef RAW8
        ReadSensorDIACcl8Raw(S1, raw);
#else
        ReadSensorDIACcl8(S1, val);
#endif
#ifdef RAW10
        ReadSensorDIACclRaw(S1, raw);
#else
        ReadSensorDIACcl(S1, val);
#endif
        ReadSensorDIACclDrift(S1, dx, dy, dz);
        NumOut(0, LCD_LINE1, val.X);
        NumOut(0, LCD_LINE2, val.Y);
        NumOut(0, LCD_LINE3, val.Z);
        NumOut(0, LCD_LINE4, raw.X);
        NumOut(0, LCD_LINE5, raw.Y);
        NumOut(0, LCD_LINE6, raw.Z);
        NumOut(50, LCD_LINE4, dx);
        NumOut(50, LCD_LINE5, dy);
        NumOut(50, LCD_LINE6, dz);

        Wait(MS_50);
    }
}
```

## 9.114 ex\_digps.nxc

This is an example of how to use the [SetSensorDIGPSWaypoint](#), [SensorDIGPSStatus](#), [SensorDIGPSTime](#), [SensorDIGPSLatitude](#), [SensorDIGPSLongitude](#), [SensorDIGPSVelocity](#), [SensorDIGPSHeading](#), [SensorDIGPSDistanceToWaypoint](#), [SensorDIGPSHeadingToWaypoint](#), and [SensorDIGPSRelativeHeading](#) functions.

```
task main()
{
  SetSensorLowspeed(S1);
  // while(!SensorDIGPSStatus(S1)) Wait(10);

  SetSensorDIGPSWaypoint(S1, 36165833, -86784444);
  while (true)
  {
    // show link status
    NumOut(0, LCD_LINE1, SensorDIGPSStatus(S1), true);
    // show latitude & longitude
    float lat = SensorDIGPSLatitude(S1) / 1000000;
    float lng = SensorDIGPSLongitude(S1) / 1000000;
    NumOut(0, LCD_LINE2, lat);
    NumOut(0, LCD_LINE3, lng);
    // show heading
    NumOut(0, LCD_LINE4, SensorDIGPSHeading(S1));
    // show velocity
    NumOut(0, LCD_LINE5, SensorDIGPSVelocity(S1));
    // show time in UTC
    NumOut(0, LCD_LINE6, SensorDIGPSTime(S1));
    NumOut(0, LCD_LINE7, SensorDIGPSDistanceToWaypoint(S1));
    NumOut(0, LCD_LINE8, SensorDIGPSHeadingToWaypoint(S1));
    NumOut(50, LCD_LINE1, SensorDIGPSRelativeHeading(S1), true);
    Wait(500);
  }
}
```

## 9.115 ex\_digyro.nxc

This is an example of how to use the [SetSensorDIGyro](#), [SetSensorDIGyroEx](#), [SensorDIGyroTemperature](#), [SensorDIGyroStatus](#), [ReadSensorDIGyroRaw](#), and [ReadSensorDIGyro](#) functions.

```
//#define RAW
#define DEFAULT

task main(){
#ifdef DEFAULT
  SetSensorDIGyro(S1);
#else
  SetSensorDIGyroEx(S1, DIGYRO_CTRL4_SCALE_2000, DIGYRO_CTRL1_DATARATE_800,
    DIGYRO_CTRL1_BANDWIDTH_4);
#endif
}
```

```
VectorType val;

int i = 0;
int temp;
byte status;
bool done = false;
while (!done){
    ClearScreen();
    NumOut(0, LCD_LINE7, SensorDIGyroTemperature(S1));
    NumOut(0, LCD_LINE8, SensorDIGyroStatus(S1));
    // Read the GYROSCOPE
#ifdef RAW
    if (!ReadSensorDIGyroRaw(S1, val))
#else
    if (!ReadSensorDIGyro(S1, val))
#endif
    TextOut(0, LCD_LINE8, "fail");
    NumOut(0, LCD_LINE1, val.X);
    NumOut(0, LCD_LINE2, val.Y);
    NumOut(0, LCD_LINE3, val.Z);

    Wait(MS_50);
}
}
```

### 9.116 ex\_dispfnout.nxc

This is an example of how to use the [FontNumOut](#) function.

```
#download "PropTiny.ric"

task main()
{
    FontNumOut(0, 40, "PropTiny.RIC", PI);
    while( 1 ) ;
}
```

### 9.117 ex\_dispftout.nxc

This is an example of how to use the [FontTextOut](#), [SysDrawFont](#), [Wait](#), and [ClearScreen](#) functions.

```
#download "PropTiny.ric"

task main()
{
    DrawFontType dfArgs;
    dfArgs.Location.X = 10;
    dfArgs.Location.Y = 59;
    dfArgs.Filename = "PropTiny.ric" ;
    dfArgs.Text = "Hello" ;
}
```

```

dfArgs.Options = DRAW_OPT_NORMAL|DRAW_OPT_FONT_DIR_L2RT;
SysDrawFont(dfArgs);
FontTextOut( 35,59, "PropTiny.RIC", "World", DRAW_OPT_INVERT|
DRAW_OPT_FONT_DIR_T2BL );
FontTextOut( 10,20, "PropTiny.RIC", "Now is the winter of our discontent made g
lorious summer by this son of York. And all the clouds that lowered upon our hou
se in the deep bosom of the ocean buried.", DRAW_OPT_NORMAL|
DRAW_OPT_FONT_DIR_L2RB|DRAW_OPT_FONT_WRAP );
FontTextOut( 50,56, "PropTiny.RIC", "WiWiWiWiWiWi", DRAW_OPT_NORMAL|
DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,48, "PropTiny.RIC", "WiWiWiWiWiWi", DRAW_OPT_INVERT|
DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,40, "PropTiny.RIC", "WiWiWiWiWiWi", DRAW_OPT_LOGICAL_OR|
DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,32, "PropTiny.RIC", "WiWiWiWiWiWi", DRAW_OPT_INVERT|
DRAW_OPT_LOGICAL_AND|DRAW_OPT_FONT_DIR_L2RB );
Wait(SEC_5);
ClearScreen();
Wait(SEC_4);
}

```

### 9.118 ex\_dispfunc.nxc

This is an example of how to use the [SysDisplayExecuteFunction](#) and [Wait](#) functions along with the [DisplayExecuteFunctionType](#) structure.

```

task main()
{
DisplayExecuteFunctionType defArgs;
defArgs.Cmd = DISPLAY_HORIZONTAL_LINE;
defArgs.On = DRAW_OPT_NORMAL;
defArgs.X1 = 20;
defArgs.Y1 = 20;
defArgs.X2 = 40;
SysDisplayExecuteFunction(defArgs);
Wait(SEC_15);
}

```

### 9.119 ex\_dispgaout.nxc

This is an example of how to use the [GraphicArrayOut](#), [NumOut](#), and [Wait](#) function. It also demonstrates how to use the [RICOpSprite](#), [RICSpriteData](#), [RICOpCopyBits](#), [RICImgRect](#), and [RICImgPoint](#) macros.

```

byte ric_data[] = {
    RICOpSprite(1, 64, 2,
        RICSpriteData(0xFF, 0xFF, 0x80, 0x01, 0x80, 0x41,
            0x80, 0x21, 0x80, 0x11, 0x80, 0x09,
            0x80, 0x05, 0x80, 0x09, 0x80, 0x11,
            0x80, 0x21, 0x80, 0x41, 0x80, 0x81,
            0x81, 0x01, 0x82, 0x01, 0x84, 0x01,

```

```

        0x88, 0x01, 0x90, 0x01, 0xA0, 0x01,
        0x90, 0x01, 0x88, 0x01, 0x84, 0x01,
        0x82, 0x01, 0x81, 0x01, 0x80, 0x81,
        0x80, 0x41, 0x80, 0x21, 0x80, 0x11,
        0x80, 0x09, 0x80, 0x05, 0x80, 0x09,
        0x80, 0x11, 0x80, 0x21, 0x80, 0x41,
        0x80, 0x81, 0x81, 0x01, 0x82, 0x01,
        0x84, 0x01, 0x88, 0x01, 0x90, 0x01,
        0xA0, 0x01, 0x90, 0x01, 0x88, 0x01,
        0x84, 0x01, 0x82, 0x01, 0x81, 0x01,
        0x80, 0x81, 0x80, 0x41, 0x80, 0x21,
        0x80, 0x11, 0x80, 0x09, 0x80, 0x05,
        0x80, 0x09, 0x80, 0x11, 0x80, 0x21,
        0x80, 0x41, 0x80, 0x81, 0x81, 0x01,
        0x82, 0x01, 0x84, 0x01, 0x88, 0x01,
        0x90, 0x01, 0xA0, 0x01, 0x80, 0x01,
        0xFF, 0xFF)),
    RICOpCopyBits(0, 1,
        RICImgRect(
            RICImgPoint(0, 0), 16, 64),
        RICImgPoint(0, 0))
};

void Animate()
{
    int i;
    byte a;
    byte b;

    a = ric_data[12];
    b = ric_data[13];

    for( i=12; i<132; i++ )
        ric_data[i] = ric_data[i+2];

    ric_data[ 132 ] = a;
    ric_data[ 133 ] = b;
}

task main()
{
    int counter = 0;

    while( 1 )
    {
        Animate();

        GraphicArrayOut(0, 0, ric_data);
        NumOut( 50, LCD_LINE1, ++counter );
        Wait( MS_20 );
    }
}

```

## 9.120 ex\_dispgaoutex.nxc

This is an example of how to use the [GraphicArrayOutEx](#) and [Wait](#) functions. It also demonstrates how to use the [RICOpDescription](#), [RICOpSprite](#), [RICSpriteData](#), [RICOpCopyBits](#), [RICImgRect](#), and [RICImgPoint](#) macros.

```
// Draw the Chessboard
byte Chess1_data[] = {
RICOpDescription(0, 104, 20),
RICOpSprite(1, 14, 13,
  RICSpriteData(0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE,
    0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xBE, 0xBE,
    0xEE, 0xFE, 0xFE, 0xFE, 0x82, 0xBA, 0x86, 0xC6,
    0x86, 0x86, 0xC2, 0xBE, 0xBA, 0xC6, 0xEE, 0xFE,
    0xD6, 0xEE, 0xB6, 0xBA, 0xBA, 0xBA, 0xBA, 0xBA,
    0x86, 0xB6, 0xEE, 0xC6, 0xFE, 0xEE, 0xEE, 0x8E,
    0x86, 0xAA, 0x86, 0xBE, 0xC2, 0xBA, 0x8E, 0xEE,
    0xEE, 0xFE, 0xD6, 0xEE, 0xB6, 0xBA, 0xB6, 0xB6,
    0xBE, 0xFA, 0x86, 0xB6, 0xF6, 0xFE, 0xFE, 0xFE,
    0xEE, 0xBA, 0x86, 0xCA, 0xBA, 0xFE, 0xFE, 0xFE,
    0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE,
    0xFE, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x40, 0x40, 0x10, 0x00, 0x00, 0x00, 0x7C,
    0x44, 0x78, 0x38, 0x78, 0x78, 0x3C, 0x40, 0x44,
    0x38, 0x10, 0x00, 0x28, 0x10, 0x48, 0x44, 0x44,
    0x44, 0x44, 0x44, 0x78, 0x48, 0x10, 0x38, 0x00,
    0x10, 0x10, 0x70, 0x78, 0x54, 0x78, 0x40, 0x3C,
    0x44, 0x70, 0x10, 0x10, 0x00, 0x28, 0x10, 0x48,
    0x44, 0x48, 0x48, 0x40, 0x04, 0x78, 0x48, 0x08,
    0x00, 0x00, 0x00, 0x10, 0x44, 0x78, 0x34, 0x44,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(0), 7), 7, 7), RICImgPoint(0, 0
)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(1), 0), 7, 7), RICImgPoint(7, 0
)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(2), 7), 7, 7), RICImgPoint(14,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(3), 0), 7, 7), RICImgPoint(21,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(4), 7), 7, 7), RICImgPoint(28,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(5), 0), 7, 7), RICImgPoint(35,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(6), 7), 7, 7), RICImgPoint(42,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(7), 0), 7, 7), RICImgPoint(49,
0)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(8), 0), 7, 7), RICImgPoint(0, 7
)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(9), 7), 7, 7), RICImgPoint(7, 7
)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(10), 0), 7, 7), RICImgPoint(14,
7)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(11), 7), 7, 7), RICImgPoint(21,
```

```
7)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(12), 0), 7, 7), RiCImgPoint(28,
7)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(13), 7), 7, 7), RiCImgPoint(35,
7)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(14), 0), 7, 7), RiCImgPoint(42,
7)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(15), 7), 7, 7), RiCImgPoint(49,
7)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(16), 7), 7, 7), RiCImgPoint(0,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(17), 0), 7, 7), RiCImgPoint(7,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(18), 7), 7, 7), RiCImgPoint(14,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(19), 0), 7, 7), RiCImgPoint(21,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(20), 7), 7, 7), RiCImgPoint(28,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(21), 0), 7, 7), RiCImgPoint(35,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(22), 7), 7, 7), RiCImgPoint(42,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(23), 0), 7, 7), RiCImgPoint(49,
14)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(24), 0), 7, 7), RiCImgPoint(0,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(25), 7), 7, 7), RiCImgPoint(7,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(26), 0), 7, 7), RiCImgPoint(14,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(27), 7), 7, 7), RiCImgPoint(21,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(28), 0), 7, 7), RiCImgPoint(28,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(29), 7), 7, 7), RiCImgPoint(35,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(30), 0), 7, 7), RiCImgPoint(42,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(31), 7), 7, 7), RiCImgPoint(49,
21)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(32), 7), 7, 7), RiCImgPoint(0,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(33), 0), 7, 7), RiCImgPoint(7,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(34), 7), 7, 7), RiCImgPoint(14,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(35), 0), 7, 7), RiCImgPoint(21,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(36), 7), 7, 7), RiCImgPoint(28,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(37), 0), 7, 7), RiCImgPoint(35,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(38), 7), 7, 7), RiCImgPoint(42,
28)),
RiCOpCopyBits(0, 1, RiCImgRect(RiCImgPoint(RiCArg(39), 0), 7, 7), RiCImgPoint(49,
28)),
```

```
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(40), 0), 7, 7), RICImgPoint(0,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(41), 7), 7, 7), RICImgPoint(7,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(42), 0), 7, 7), RICImgPoint(14,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(43), 7), 7, 7), RICImgPoint(21,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(44), 0), 7, 7), RICImgPoint(28,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(45), 7), 7, 7), RICImgPoint(35,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(46), 0), 7, 7), RICImgPoint(42,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(47), 7), 7, 7), RICImgPoint(49,
35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(48), 7), 7, 7), RICImgPoint(0,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(49), 0), 7, 7), RICImgPoint(7,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(50), 7), 7, 7), RICImgPoint(14,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(51), 0), 7, 7), RICImgPoint(21,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(52), 7), 7, 7), RICImgPoint(28,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(53), 0), 7, 7), RICImgPoint(35,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(54), 7), 7, 7), RICImgPoint(42,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(55), 0), 7, 7), RICImgPoint(49,
42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(56), 0), 7, 7), RICImgPoint(0,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(57), 7), 7, 7), RICImgPoint(7,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(58), 0), 7, 7), RICImgPoint(14,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(59), 7), 7, 7), RICImgPoint(21,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(60), 0), 7, 7), RICImgPoint(28,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(61), 7), 7, 7), RICImgPoint(35,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(62), 0), 7, 7), RICImgPoint(42,
49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(63), 7), 7, 7), RICImgPoint(49,
49))
};

#define A 0
#define B 1
#define C 2
#define D 3
#define E 4
#define F 5
#define G 6
```

```
#define H 8

#define P(_file, _rank) (((_rank)-1)*8)+(_file)

#define A1 P(A, 1)
#define A2 P(A, 2)
#define A3 P(A, 3)
#define A4 P(A, 4)
#define A5 P(A, 5)
#define A6 P(A, 6)
#define A7 P(A, 7)
#define A8 P(A, 8)

#define B1 P(B, 1)
#define B2 P(B, 2)
#define B3 P(B, 3)
#define B4 P(B, 4)
#define B5 P(B, 5)
#define B6 P(B, 6)
#define B7 P(B, 7)
#define B8 P(B, 8)

#define C1 P(C, 1)
#define C2 P(C, 2)
#define C3 P(C, 3)
#define C4 P(C, 4)
#define C5 P(C, 5)
#define C6 P(C, 6)
#define C7 P(C, 7)
#define C8 P(C, 8)

#define D1 P(D, 1)
#define D2 P(D, 2)
#define D3 P(D, 3)
#define D4 P(D, 4)
#define D5 P(D, 5)
#define D6 P(D, 6)
#define D7 P(D, 7)
#define D8 P(D, 8)

#define E1 P(E, 1)
#define E2 P(E, 2)
#define E3 P(E, 3)
#define E4 P(E, 4)
#define E5 P(E, 5)
#define E6 P(E, 6)
#define E7 P(E, 7)
#define E8 P(E, 8)

#define F1 P(F, 1)
#define F2 P(F, 2)
#define F3 P(F, 3)
#define F4 P(F, 4)
#define F5 P(F, 5)
#define F6 P(F, 6)
#define F7 P(F, 7)
#define F8 P(F, 8)
```

```
#define G1 P(G, 1)
#define G2 P(G, 2)
#define G3 P(G, 3)
#define G4 P(G, 4)
#define G5 P(G, 5)
#define G6 P(G, 6)
#define G7 P(G, 7)
#define G8 P(G, 8)

#define H1 P(H, 1)
#define H2 P(H, 2)
#define H3 P(H, 3)
#define H4 P(H, 4)
#define H5 P(H, 5)
#define H6 P(H, 6)
#define H7 P(H, 7)
#define H8 P(H, 8)

int b[] =
{
    64, 72, 80, 88, 96, 80, 72, 64, // 1
    56, 56, 56, 56, 56, 56, 56, 56, // 2
    48, 48, 48, 48, 48, 48, 48, 48, // 3
    48, 48, 48, 48, 48, 48, 48, 48, // 4
    48, 48, 48, 48, 48, 48, 48, 48, // 5
    48, 48, 48, 48, 48, 48, 48, 48, // 6
    40, 40, 40, 40, 40, 40, 40, 40, // 7
    32, 24, 16, 8, 0, 16, 24, 32 // 8
};
// A B C D E F G H

#define Vacant 48

#define Move(_from, _to) \
    b[_to] = b[_from]; \
    b[_from] = Vacant; \
    GraphicArrayOutEx( 8,8,Chess1_data , b, true); \
    Wait(SEC_2);

task main()
{
    // setup board
    GraphicArrayOutEx( 8,8,Chess1_data, b, true);
    WaitSEC_2;

    Move(A2, A3); // white pawn from A2 to A3
    Move(B7, B5); // black pawn from B7 to B5
    Move(A3, A4); // white pawn from A3 to A4
    Move(B5, B4); // black pawn from B5 to B4
    Move(A4, A5); // white pawn from A4 to A5
    Move(B4, B3); // black pawn from B4 to B3
    Move(A5, A6); // white pawn from A5 to A6
    while( true );
}
```

## 9.121 ex\_dispgout.nxc

This is an example of how to use the [GraphicOut](#), [SysCall](#), [TextOut](#), [CurrentTick](#), [NumOut](#), [Wait](#), and [ClearScreen](#) functions. It also demonstrates how to use the [DrawGraphicArrayType](#) structure.

```
#download "2c.ric"

byte tmpData2[] = {
    0x0A, 0x00, 0x07, 0x00, 0x00, 0x00, 0x14, 0x00, 0x14,
    0x00, 0x0A, 0x00, 0x0A, 0x00, 0x07, 0x00, 0x00, 0x00,
    0x1E, 0x00, 0x1E, 0x00, 0x0A, 0x00};

DrawGraphicArrayType dgaArgs;

string names[] = {"2c.ric" , "2l.ric" };
task main()
{
    long tick;
    TextOut(0, LCD_LINE1, "testing");
    tick = CurrentTick();
    GraphicOut(10, 10, names[0]);
    tick = CurrentTick()-tick;
    NumOut(0, LCD_LINE8, tick);
    Wait(SEC_5);
    ClearScreen();
    Wait(MS_500);
    TextOut(0, LCD_LINE1, "testing");
    tick = CurrentTick();
    dgaArgs.Location.X = 10;
    dgaArgs.Location.Y = 10;
    dgaArgs.Options = 0;
    dgaArgs.Data = tmpData2;
    SysCall(DrawGraphicArray, dgaArgs);
    tick = CurrentTick()-tick;
    NumOut(0, LCD_LINE8, tick);
    Wait(SEC_5);
}
```

## 9.122 ex\_dispgoutex.nxc

This is an example of how to use the [GraphicOutEx](#) and [Wait](#) functions.

```
#download "letters.ric"

string fnames[] = {"letters.ric", "letter2.ric" };
int Values[] = {0};
void Display( int n )
{
    Values[0]=n*10;
    GraphicOutEx(Values[0], Random(30), fnames[0], Values,
        DRAW_OPT_CLEAR_WHOLE_SCREEN);
    Wait(MS_200);
}
```

```
}  
  
task main()  
{  
  while( true )  
  {  
    for( int i=0; i<9; i++ )  
      Display( i );  
  }  
}
```

### 9.123 ex\_DisplayDisplay.nxc

This is an example of how to use the [DisplayDisplay](#) function.

```
x = DisplayDisplay();
```

### 9.124 ex\_DisplayEraseMask.nxc

This is an example of how to use the [DisplayEraseMask](#) function.

```
x = DisplayEraseMask();
```

### 9.125 ex\_DisplayFlags.nxc

This is an example of how to use the [DisplayFlags](#) function.

```
x = DisplayFlags();
```

### 9.126 ex\_displayfont.nxc

This is an example of how to use the [DisplayFont](#) function.

```
const byte NewFont[] =  
{  
  0x04,0x00, // Graphics Format  
  0x02,0x40, // Graphics DataSize  
  0x10,      // Graphics Count X  
  0x06,      // Graphics Count Y  
  0x06,      // Graphics Width  
  0x08,      // Graphics Height
```

```

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,0x06,0x00,0x00,0x07,0x03,0x00,0x07
,0x03,0x00,0x24,0x7E,0x24,0x7E,0x24,0x00,0x24,0x2B,0x6A,0x12,0x00,0x00,0x63,0x13,
0x08,0x64,0x63,0x00,0x30,0x4C,0x52,0x22,0x50,0x00,0x00,0x07,0x03,0x00,0x00,0x00,0
x00,0x3E,0x41,0x00,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x
08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x00,0x08,0x08,0x0
8,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,
0x3E,0x51,0x49,0x45,0x3E,0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x62,0x51,0x49,0x49
,0x46,0x00,0x22,0x49,0x49,0x49,0x36,0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x2F,0x49,
0x49,0x49,0x31,0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
x36,0x49,0x49,0x49,0x36,0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00,0x6C,0x6C,0x00,0x
00,0x00,0x00,0xEC,0x6C,0x00,0x00,0x00,0x08,0x14,0x22,0x41,0x00,0x00,0x24,0x24,0x2
4,0x24,0x24,0x00,0x00,0x41,0x22,0x14,0x08,0x00,0x02,0x01,0x59,0x09,0x06,0x00,
0x3E,0x41,0x5D,0x55,0x1E,0x00,0x7E,0x11,0x11,0x11,0x7E,0x00,0x7F,0x49,0x49,0x49
,0x36,0x00,0x3E,0x41,0x41,0x41,0x22,0x00,0x7F,0x41,0x41,0x41,0x3E,0x00,0x7F,0x49,
0x49,0x49,0x41,0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
x7F,0x08,0x08,0x08,0x7F,0x00,0x00,0x41,0x7F,0x41,0x00,0x00,0x30,0x40,0x40,0x40,0x
3F,0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x7F,0x40,0x40,0x40,0x40,0x00,0x7F,0x02,0x0
4,0x02,0x7F,0x00,0x7F,0x02,0x04,0x08,0x7F,0x00,0x3E,0x41,0x41,0x41,0x3E,0x00,
0x7F,0x09,0x09,0x09,0x06,0x00,0x3E,0x41,0x51,0x21,0x5E,0x00,0x7F,0x09,0x09,0x19
,0x66,0x00,0x26,0x49,0x49,0x49,0x32,0x00,0x01,0x01,0x7F,0x01,0x01,0x00,0x3F,0x40,
0x40,0x40,0x3F,0x00,0x1F,0x20,0x40,0x20,0x1F,0x00,0x3F,0x40,0x3C,0x40,0x3F,0x00,0
x63,0x14,0x08,0x14,0x63,0x00,0x07,0x08,0x70,0x08,0x07,0x00,0x71,0x49,0x45,0x43,0x
00,0x00,0x00,0x7F,0x41,0x41,0x00,0x00,0x02,0x04,0x08,0x10,0x20,0x00,0x00,0x41,0x4
1,0x7F,0x00,0x00,0x04,0x02,0x01,0x02,0x04,0x00,0x80,0x80,0x80,0x80,0x80,0x00,
0x00,0x02,0x05,0x02,0x00,0x00,0x20,0x54,0x54,0x54,0x78,0x00,0x7F,0x44,0x44,0x44
,0x38,0x00,0x38,0x44,0x44,0x44,0x28,0x00,0x38,0x44,0x44,0x44,0x7F,0x00,0x38,0x54,
0x54,0x54,0x08,0x00,0x08,0x7E,0x09,0x09,0x00,0x00,0x18,0x24,0xA4,0xA4,0xFC,0x00,0
x7F,0x04,0x04,0x78,0x00,0x00,0x00,0x00,0x7D,0x40,0x00,0x00,0x40,0x80,0x84,0x7D,0x
00,0x00,0x7F,0x10,0x28,0x44,0x00,0x00,0x00,0x00,0x7F,0x40,0x00,0x00,0x7C,0x04,0x1
8,0x04,0x78,0x00,0x7C,0x04,0x04,0x78,0x00,0x00,0x38,0x44,0x44,0x44,0x38,0x00,
0xFC,0x44,0x44,0x44,0x38,0x00,0x38,0x44,0x44,0x44,0x44,0xFC,0x00,0x44,0x78,0x44,0x04
,0x08,0x00,0x08,0x54,0x54,0x54,0x20,0x00,0x04,0x3E,0x44,0x24,0x00,0x00,0x3C,0x40,
0x20,0x7C,0x00,0x00,0x1C,0x20,0x40,0x20,0x1C,0x00,0x3C,0x60,0x30,0x60,0x3C,0x00,0
x6C,0x10,0x10,0x6C,0x00,0x00,0x9C,0xA0,0x60,0x3C,0x00,0x00,0x64,0x54,0x54,0x4C,0x
00,0x00,0x08,0x3E,0x41,0x41,0x00,0x00,0x00,0x77,0x00,0x00,0x00,0x00,0x41,0x41,0x4
1,0x3E,0x08,0x00,0x02,0x01,0x02,0x01,0x00,0x00,0x10,0x20,0x40,0x38,0x07,0x00
};

task main()
{
  unsigned long ptr, pOldFont;
  byte myData[800];
  ptr = addr(NewFont);
  TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
  pOldFont = DisplayFont();
  SetDisplayFont(ptr);
  TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
  SetDisplayFont(pOldFont);
  TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
  Wait(SEC_10);
}

```

## 9.127 ex\_DisplayTextLinesCenterFlags.nxc

This is an example of how to use the [DisplayTextLinesCenterFlags](#) function.

```
x = DisplayTextLinesCenterFlags();
```

### 9.128 ex\_DisplayUpdateMask.nxc

This is an example of how to use the [DisplayUpdateMask](#) function.

```
x = DisplayUpdateMask();
```

### 9.129 ex\_dispmisc.nxc

This is an example of how to use the [DisplayEraseMask](#), [DisplayUpdateMask](#), [DisplayDisplay](#), [DisplayFlags](#), [DisplayTextLinesCenterFlags](#) functions, [SetDisplayEraseMask](#), [SetDisplayUpdateMask](#), [SetDisplayDisplay](#), [SetDisplayFlags](#), and [SetDisplayTextLinesCenterFlags](#) functions,

```
task main()
{
    unsigned long addr = DisplayDisplay();
    NumOut(0, LCD_LINE1, DisplayEraseMask());
    NumOut(0, LCD_LINE2, DisplayUpdateMask());
    NumOut(0, LCD_LINE3, addr);
    NumOut(0, LCD_LINE4, DisplayFlags());
    NumOut(0, LCD_LINE5, DisplayTextLinesCenterFlags());
    Wait(SEC_4);
    // setting the display address function can be ... dangerous
    SetDisplayDisplay(addr);
    // fiddling with the display flags is also dangerous
    unsigned long flags = DisplayFlags();
    flags |= DISPLAY_POPUP;
    SetDisplayFlags(flags);
    Wait(SEC_2);
    flags = flags & (~DISPLAY_POPUP);
    SetDisplayFlags(flags);
    Wait(SEC_1);
    SetDisplayEraseMask(DisplayEraseMask());
    SetDisplayUpdateMask(DisplayUpdateMask());
    SetDisplayTextLinesCenterFlags(DisplayTextLinesCenterFlags());
    Wait(SEC_2);
}
```

### 9.130 ex\_DISTNxDistance.nxc

This is an example of how to use the [DISTNxDistance](#) function.

```
int dist = DISTNxDistance(S1, MS_ADDR_DISTNX);
```

### 9.131 ex\_DISTNxGP2D12.nxc

This is an example of how to use the [DISTNxGP2D12](#) function.

```
char result = DISTNxGP2D12(S1, MS_ADDR_DISTNX);
```

### 9.132 ex\_DISTNxGP2D120.nxc

This is an example of how to use the [DISTNxGP2D120](#) function.

```
char result = DISTNxGP2D120(S1, MS_ADDR_DISTNX);
```

### 9.133 ex\_DISTNxGP2YA02.nxc

This is an example of how to use the [DISTNxGP2YA02](#) function.

```
char result = DISTNxGP2YA02(S1, MS_ADDR_DISTNX);
```

### 9.134 ex\_DISTNxGP2YA21.nxc

This is an example of how to use the [DISTNxGP2YA21](#) function.

```
char result = DISTNxGP2YA21(S1, MS_ADDR_DISTNX);
```

### 9.135 ex\_DISTNxMaxDistance.nxc

This is an example of how to use the [DISTNxMaxDistance](#) function.

```
int dist = DISTNxMaxDistance(S1, MS_ADDR_DISTNX);
```

### 9.136 ex\_DISTNxMinDistance.nxc

This is an example of how to use the [DISTNxMinDistance](#) function.

```
int dist = DISTNxMinDistance(S1, MS_ADDR_DISTNX);
```

### 9.137 ex\_DISTNxModuleType.nxc

This is an example of how to use the [DISTNxModuleType](#) function.

```
int modtype = DISTNxModuleType(S1, MS_ADDR_DISTNX);
```

### 9.138 ex\_DISTNxNumPoints.nxc

This is an example of how to use the [DISTNxNumPoints](#) function.

```
int numpoints = DISTNxNumPoints(S1, MS_ADDR_DISTNX);
```

### 9.139 ex\_DISTNxVoltage.nxc

This is an example of how to use the [DISTNxVoltage](#) function.

```
int volt = DISTNxVoltage(S1, MS_ADDR_DISTNX);
```

### 9.140 ex\_div.nxc

This is an example of how to use the [div](#) function.

```
task main()
{
  long x, y;
  x = 31464;
  y = 33;
  div_t r;
  r = div(x, y);
  NumOut(0, LCD_LINE1, r.quot);
  NumOut(0, LCD_LINE2, r.rem);
  Wait(SEC_3);
}
```

### 9.141 ex\_EllipseOut.nxc

This is an example of how to use the [EllipseOut](#) and [Random](#) functions.

```
task main()
{
  repeat (10)
    EllipseOut(50, 32, 20+Random(15), 20+Random(10), DRAW_OPT_FILL_SHAPE |
      DRAW_OPT_LOGICAL_XOR);
  while(true);
}
```

### 9.142 ex\_exp.nxc

This is an example of how to use the [exp](#) function.

```
y = exp(x);
```

**9.143 ex\_fclose.nxc**

This is an example of how to use the [fclose](#) function.

```
result = fclose(handle);
```

**9.144 ex\_feof.nxc**

This is an example of how to use the [feof](#) function.

```
int i = feof(handle);
```

**9.145 ex\_fflush.nxc**

This is an example of how to use the [fflush](#) function.

```
int i = fflush(handle);
```

**9.146 ex\_fgetc.nxc**

This is an example of how to use the [fgetc](#) function.

```
char val = fgetc(handle);
```

**9.147 ex\_fgets.nxc**

This is an example of how to use the [fgets](#) function.

```
fgets(msg, 10, handle);
```

**9.148 ex\_file\_system.nxc**

This is an example of how to use the [PlayTone](#), [Wait](#), [Stop](#), [TextOut](#), [OpenFileAppend](#), [CloseFile](#), [OpenFileRead](#), [FormatNum](#), [Write](#), [Read](#), and [CircleOut](#) functions. This program is intended to serve as an introduction to data files on the NXT. It focuses on handling the codes returned by the file system's API calls, which is an important aspect of the API that is all too often neglected by programmers. The program deals

with a data file describing circles. On each run, it adds a new circle record to the data file. Then it reads in the whole data file and displays all the circles on NXT screen. It creates the data file if doesn't already exist. If you run it several times in secession, you will fill the data file and get a file-is-full exception. The data file created by this program is not visible on the NXT. To delete the file, circles.dat, you can use the NeXT Explorer or the example program [ex\\_delete\\_data\\_file::nxc](#).

```
// ex_file_system.nxc
// This program is intended to serve as an introduction to data files on the
// NXT. It focuses on handling the codes returned by the file system's API
// calls, which is an important aspect of the API that is all too often
// neglected by programmers.
//
// The program deals with a data file describing circles. On each run, it adds
// a new circle record to the data file. Then it reads in the whole data file
// and displays all the circles on NXT screen. It creates the data file if
// doesn't already exist. If you run it several times in secession, you will
// fill the data file and get a file-is-full exception.
//
// The data file created by this program is not visible on the NXT. To delete
// the file, circles.dat, you can use the NeXT Explorer or the example program
// ex_delete_data_file.nxc.

#define MIN_R 10
#define MAX_R 30
#define MIN_X 20
#define MAX_X 80
#define MIN_Y 10
#define MAX_Y 54

byte handle = 0; // file handle

#define FILE_NAME "circles.dat"
// The file size is made small so it will fill up quickly.
#define RECORDS 4
#define RECORD_SIZE 3
#define FILE_SIZE (RECORD_SIZE * RECORDS)

// This struct defines the data records.
struct circle
{
    byte r; // radius
    byte cx; // center x-coordinate
    byte cy; // center y-coordinate
};

// Initialize a circle with random radius r and center (cx, cy).
void init_circle(circle & c)
{
    c.r = MIN_R + Random(MAX_R - MIN_R);
    c.cx = MIN_X + Random(MAX_X - MIN_X);
    c.cy = MIN_Y + Random(MAX_Y - MIN_Y);
}

// Make sure file is closed whether or not file operations succeed or fail.
void shutdown(const int delay)
```

```
{
    if (handle) CloseFile(handle);
    // Get user's attention.
    PlayTone(TONE_C5, SEC_1);
    // Give the user time to read screen messages.
    Wait(delay);
    Stop(true);
}

// Display a return code from a file sytem API call on the NXT screen.
void rtn_code_out(const unsigned int code)
{
    TextOut(0, LCD_LINE2, "code          ");
    TextOut(50, LCD_LINE2, FormatNum("%04x", code));
}

// Open the data file for writing.
void open_for_write()
{
    unsigned int file_size = FILE_SIZE;
    handle = 0;
    // Start with the assumptions the file doesn't exist and needs to be created.
    unsigned int rtn_code = CreateFile(FILE_NAME, file_size, handle);
    // If the file already exists, open it with the intent of adding to the data
    // that is already there.
    if (rtn_code == LDR_FILEEXISTS)
        rtn_code = OpenFileAppend(FILE_NAME, file_size, handle);
    // Return code handling
    switch (rtn_code)
    {
    case LDR_SUCCESS:
        return;
    case LDR_FILEISFULL:
        TextOut(0, LCD_LINE1, "file is full  ");
        break;
    default:
        // Unanticipated exception.
        TextOut(0, LCD_LINE1, "write open    ");
        rtn_code_out(rtn_code);
        break;
    }
    shutdown(SEC_8);
}

// Open the data file for reading.
void open_for_read()
{
    unsigned int file_size = FILE_SIZE;
    handle = 0;
    unsigned int rtn_code = OpenFileRead(FILE_NAME, file_size, handle);
    // Return code handling
    if (rtn_code != LDR_SUCCESS)
    {
        // Unanticipated exception.
        TextOut(0, LCD_LINE1, "read open    ");
        rtn_code_out(rtn_code);
        shutdown(SEC_8);
    }
}
```

```
    }
}

// Write one circle record to the data file.
void write_recd(const circle recd)
{
    unsigned int rtn_code = Write(handle, recd);
    // Return code handling
    if (rtn_code != LDR_SUCCESS)
    {
        switch (rtn_code)
        {
            case LDR_EOFEXPECTED:
                TextOut(0, LCD_LINE1, "no more space  ");
                break;
            default:
                // Unanticipated exception.
                TextOut(0, LCD_LINE1, "write failed  ");
                rtn_code_out(rtn_code);
                break;
        }
        shutdown(SEC_8);
    }
}

// Read all the circle records from the data file. Display each circle as it is
// read.
void read_all(circle & recd)
{
    while (true)
    {
        unsigned int rtn_code = Read(handle, recd);
        // rtn_code_out(rtn_code);
        // Return code handling
        switch (rtn_code)
        {
            case LDR_SUCCESS:
                // Record has been read. Display circle described by it.
                CircleOut(recd.cx, recd.cy, recd.r);
                Wait(SEC_2);
                break;
            case LDR_ENDOFFILE:
                // No more data to read.
                return;
            default:
                // Unanticipated exception.
                TextOut(0, LCD_LINE1, "read failed  ");
                rtn_code_out(rtn_code);
                shutdown(SEC_8);
        }
    }
}

task main()
{
    circle c;
    open_for_write();
}
```

```
    init_circle(c);
    write_recd(c);
    CloseFile(handle);
    open_for_read();
    read_all(c);
    shutdown(SEC_8);
}
```

### 9.149 ex\_findfirstfile.nxc

This is an example of how to use the [FindFirstFile](#) function.

```
task main() {
    byte handle;
    unsigned int result, fsize;
    string fname = "*.ric";
    result = FindFirstFile(fname, handle);
    NumOut(0, LCD_LINE1, result, true);
    int i=1;
    while (result == LDR_SUCCESS) {
        NumOut(0, LCD_LINE2, i, false);
        TextOut(0, LCD_LINE3, fname, false);
        Wait(1500);
        //    fname = "";
        result = FindNextFile(fname, handle);
        NumOut(0, LCD_LINE1, result, true);
        i++;
    }
    CloseFile(handle);
    Wait(3000);
}
```

### 9.150 ex\_findnextfile.nxc

This is an example of how to use the [FindNextFile](#) function.

```
task main() {
    byte handle;
    unsigned int result, fsize;
    string fname = "*.ric";
    result = FindFirstFile(fname, handle);
    NumOut(0, LCD_LINE1, result, true);
    int i=1;
    while (result == LDR_SUCCESS) {
        NumOut(0, LCD_LINE2, i, false);
        TextOut(0, LCD_LINE3, fname, false);
        Wait(1500);
        //    fname = "";
        result = FindNextFile(fname, handle);
        NumOut(0, LCD_LINE1, result, true);
        i++;
    }
}
```

```
    CloseFile(handle);
    Wait(3000);
}
```

### 9.151 ex\_FirstTick.nxc

This is an example of how to use the [FirstTick](#) function.

```
unsigned int x = FirstTick();
```

### 9.152 ex\_Flatten.nxc

This is an example of how to use the [Flatten](#) function.

```
msg = Flatten(48); // returns "0" since 48 == ascii("0")
```

### 9.153 ex\_FlattenVar.nxc

This is an example of how to use the [FlattenVar](#) function.

```
task main()
{
    long data[] = {-50123, 68142, 128176, -45123};
    long data2[4];
    float fdata[] = {12.123, 3.14159, 2.68};
    float fdata2[3];
    NumOut(0, LCD_LINE1, data[0]);
    NumOut(0, LCD_LINE2, fdata[1]);
    string sdata = FlattenVar(data);
    string tmp;
    // transfer the string to another NXT
    tmp = sdata;
    UnflattenVar(tmp, data2);
    NumOut(0, LCD_LINE3, data2[0]);
    sdata = FlattenVar(fdata);
    // transfer the string to another NXT
    tmp = sdata;
    UnflattenVar(tmp, fdata2);
    NumOut(0, LCD_LINE4, fdata2[1]);
    Wait(SEC_5);
}
```

### 9.154 ex\_float.nxc

This is an example of how to use the [Float](#) function.

```
Float(OUT_A); // float output A
```

### 9.155 ex\_floor.nxc

This is an example of how to use the [floor](#) function.

```
y = floor(x);
```

### 9.156 ex\_Follows.nxc

This is an example of how to use the [Follows](#) statement.

```
Follows(main);
```

### 9.157 ex\_fopen.nxc

This is an example of how to use the [fopen](#) function.

```
byte handle = fopen("test.txt", "r");
```

### 9.158 ex\_ForceOff.nxc

This is an example of how to use the [ForceOff](#) function.

```
ForceOff(true);
```

### 9.159 ex\_FormatNum.nxc

This is an example of how to use the [FormatNum](#) function.

```
msg = FormatNum("value = %d", x);
```

### 9.160 ex\_fprintf.nxc

This is an example of how to use the [fprintf](#) function.

```
fprintf(handle, "value = %d", value);
```

**9.161 ex\_fputc.nxc**

This is an example of how to use the [fputc](#) function.

```
fputc(ch, handle);
```

**9.162 ex\_fputs.nxc**

This is an example of how to use the [fputs](#) function.

```
fputs(msg, handle);
```

**9.163 ex\_frac.nxc**

This is an example of how to use the [frac](#) function.

```
y = frac(x);
```

**9.164 ex\_FreeMemory.nxc**

This is an example of how to use the [FreeMemory](#) function.

```
x = FreeMemory();
```

**9.165 ex\_fseek.nxc**

This is an example of how to use the [fseek](#) function.

```
fseek(handle, 10, SEEK_CUR);
```

**9.166 ex\_ftell.nxc**

This is an example of how to use the [ftell](#) function.

```
long i = ftell(handle);
```

### 9.167 ex\_GetBrickDataAddress.nxc

This is an example of how to use the [GetBrickDataAddress](#) function.

```
task main()
{
  byte data[];
  GetBrickDataAddress(data);
  // 6 bytes plus null
  TextOut(0, LCD_LINE1, StrCat(
    FormatNum("%2.2x", data[0]),
    FormatNum("%2.2x", data[1]),
    FormatNum("%2.2x", data[2]),
    FormatNum("%2.2x", data[3]),
    FormatNum("%2.2x", data[4]),
    FormatNum("%2.2x", data[5]));
  while (true);
}
```

### 9.168 ex\_GetBTConnectionAddress.nxc

This is an example of how to use the [GetBTConnectionAddress](#) function.

```
GetBTConnectionAddress(0, buffer);
```

### 9.169 ex\_GetBTDeviceAddress.nxc

This is an example of how to use the [GetBTDeviceAddress](#) function.

```
GetBTDeviceAddress(0, buffer);
```

### 9.170 ex\_GetBTInputBuffer.nxc

This is an example of how to use the [GetBTInputBuffer](#) function.

```
GetBTInputBuffer(0, 10, buffer);
```

### 9.171 ex\_GetBTOutputBuffer.nxc

This is an example of how to use the [GetBTOutputBuffer](#) function.

```
GetBTOutputBuffer(0, 10, buffer);
```

### 9.172 ex\_getc.nxc

This is an example of how to use the [getc](#) function.

```
char val = getc(handle);
```

### 9.173 ex\_getchar.nxc

This is an example of how to use the [getchar](#) function.

```
task main()
{
    SetLongAbort(true);
    while (true) {
        NumOut(0, LCD_LINE1, getchar(), true);
        Wait(MS_5);
    }
}
```

### 9.174 ex\_GetDisplayNormal.nxc

This is an example of how to use the [GetDisplayNormal](#) function.

```
GetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

### 9.175 ex\_GetDisplayPopup.nxc

This is an example of how to use the [GetDisplayPopup](#) function.

```
GetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

### 9.176 ex\_GetHSInputBuffer.nxc

This is an example of how to use the [GetHSInputBuffer](#) function.

```
GetHSInputBuffer(0, 10, buffer);
```

### 9.177 ex\_GetHSOutputBuffer.nxc

This is an example of how to use the [GetHSOutputBuffer](#) function.

```
GetHSOutputBuffer(0, 10, buffer);
```

### 9.178 ex\_GetInput.nxc

This is an example of how to use the [GetInput](#) function.

```
x = GetInput(S1, Type);
```

### 9.179 ex\_GetLastResponseInfo.nxc

This is an example of how to use the [GetLastResponseInfo](#) function.

```
byte len;  
byte cmd;  
byte buf[];  
  
char result = GetLastResponseInfo(true, len, cmd, buf);
```

### 9.180 ex\_GetLSInputBuffer.nxc

This is an example of how to use the [GetLSInputBuffer](#) function.

```
GetLSInputBuffer(S1, 0, 8, buffer);
```

### 9.181 ex\_GetLSOutputBuffer.nxc

This is an example of how to use the [GetLSOutputBuffer](#) function.

```
GetLSOutputBuffer(S1, 0, 8, outbuffer);
```

### 9.182 ex\_getmemoryinfo.nxc

This is an example of how to use the [GetMemoryInfo](#) function.

```
task main() {
    byte data[];
    byte data2[];
    int data3[];
    int ps, ds;
    char result = GetMemoryInfo(false, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    /*
    result = GetMemoryInfo(true, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    */
    /*
    ArrayInit(data, 10, 1000);
    data[10]++;
    ps = data[10];
    result = GetMemoryInfo(false, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    NumOut(0, LCD_LINE8, data[10]);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    data2 = data;
    result = GetMemoryInfo(false, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    NumOut(0, LCD_LINE8, data2[10]);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    ArrayBuild(data3, ps, ds, ps, ds, ps, ds, ps, ds);
    result = GetMemoryInfo(false, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    NumOut(0, LCD_LINE8, data3[3]);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    ArrayInit(data2, 5, 1);
    result = GetMemoryInfo(false, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    NumOut(0, LCD_LINE8, data2[0]);
    Wait(SEC_5);
    ClearScreen();
    Wait(SEC_1);
    result = GetMemoryInfo(true, ps, ds);
    NumOut(0, LCD_LINE1, ps);
    NumOut(0, LCD_LINE2, ds);
    Wait(SEC_5);
    */
}
```

```
ClearScreen();  
Wait(SEC_1);  
while(true);  
}
```

### 9.183 ex\_getoutput.nxc

This is an example of how to use the [GetOutput](#) function.

```
x = GetOutput(OUT_A, TachoLimit);
```

### 9.184 ex\_GetUSBInputBuffer.nxc

This is an example of how to use the [GetUSBInputBuffer](#) function.

```
GetUSBInputBuffer(0, 10, buffer);
```

### 9.185 ex\_GetUSBOutputBuffer.nxc

This is an example of how to use the [GetUSBOutputBuffer](#) function.

```
GetUSBOutputBuffer(0, 10, buffer);
```

### 9.186 ex\_GetUSBPollBuffer.nxc

This is an example of how to use the [GetUSBPollBuffer](#) function.

```
GetUSBPollBuffer(0, 10, buffer);
```

### 9.187 ex\_GraphicOut.nxc

This is an example of how to use the [GraphicOut](#) function.

```
GraphicOut(40, 40, "image.ric");
```

### 9.188 ex\_GraphicOutEx.nxc

This is an example of how to use the [GraphicOutEx](#) function.

```
GraphicOutEx(40, 40, "image.ric", variables);
```

### 9.189 ex\_HSFlags.nxc

This is an example of how to use the [HSFlags](#) function.

```
byte x = HSFlags();
```

### 9.190 ex\_HSInputBufferInPtr.nxc

This is an example of how to use the [HSInputBufferInPtr](#) function.

```
byte x = HSInputBufferInPtr();
```

### 9.191 ex\_HSInputBufferOutPtr.nxc

This is an example of how to use the [HSInputBufferOutPtr](#) function.

```
byte x = HSInputBufferOutPtr();
```

### 9.192 ex\_HSMMode.nxc

This is an example of how to use the [HSMMode](#) function.

```
int mode = HSMMode();
```

### 9.193 ex\_HSOutputBufferInPtr.nxc

This is an example of how to use the [HSOutputBufferInPtr](#) function.

```
byte x = HSOutputBufferInPtr();
```

### 9.194 ex\_HSOutputBufferOutPtr.nxc

This is an example of how to use the [HSOutputBufferOutPtr](#) function.

```
byte x = HSOutputBufferOutPtr();
```

### 9.195 ex\_HSSpeed.nxc

This is an example of how to use the [HSSpeed](#) function.

```
byte x = HSSpeed();
```

### 9.196 ex\_HSState.nxc

This is an example of how to use the [HSState](#) function.

```
byte x = HSState();
```

### 9.197 ex\_HTGyroTest.nxc

This is an example of how to use the [SetSensorHTGyro](#), [SensorHTGyro](#), [Wait](#), [TextOut](#), [NumOut](#), and [ButtonPressed](#) functions.

```
//=====
// HiTechnic Gyro Test
//
#define GYRO    IN_1

#define SAMPLESIZE 100

task main()
{
    int i, y, d;
    int v, offset;

    float gyroAvg, gyroSum = 0;

    int data[SAMPLESIZE];
    int cSet[7];

    SetSensorHTGyro(GYRO);

    // Let user get finger off start button before starting sampling
    Wait(1000);
```

```

for (i=0; i<SAMPLESIZE; i++) {
  v = SensorHTGyro(GYRO);
  data[i] = v;
  gyroSum += v;
  Wait(4);
}

// Display floating point gyro average
gyroAvg = gyroSum/SAMPLESIZE;
TextOut(0, LCD_LINE1, "Avg:   ");
NumOut(6*4, LCD_LINE1, gyroAvg);

// Round to nearest int
offset = gyroAvg+0.5;

// Go through sample set and see how many are
// offset-3, offset-2, offset-1, offset, offset+1, offset+2, offset+3
for (i=0; i<SAMPLESIZE; i++) {
  d = data[i] - offset;
  if (d < -3) d = -3;
  if (d > 3) d = 3;
  d += 3;
  cSet[d]++;
}

// Display on the screen now many of each value was in the sample
y = LCD_LINE2;
for (i=0; i<7; i++) {
  if (i==0)
    TextOut(0, y, "<=  :");
  else if (i==6)
    TextOut(0, y, ">=  :");
  else
    TextOut(0, y, "==  :");
  NumOut(6*2, y, offset+i-3);
  NumOut(6*6, y, cSet[i]);

  y-= 8;
}

// Keep display on screen until button pressed
until(ButtonPressed(BTNCENTER, false)) Wait(100);
}

```

## 9.198 ex\_HTIRTrain.nxc

This is an example of how to use the [HTIRTrain](#) function.

```
HTIRTrain(S1, TRAIN_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

### 9.199 ex\_HTPFComboDirect.nxc

This is an example of how to use the [HTPFComboDirect](#) function.

```
HTPFComboDirect(S1, PF_CHANNEL_1, PF_CMD_STOP, PF_CMD_FWD);
```

### 9.200 ex\_HTPFComboPWM.nxc

This is an example of how to use the [HTPFComboPWM](#) function.

```
HTPFComboPWM(S1, PF_CHANNEL_1, PF_PWM_REV4, PF_PWM_FWD5);
```

### 9.201 ex\_HTPFRawOutput.nxc

This is an example of how to use the [HTPFRawOutput](#) function.

```
HTPFRawOutput(S1, 0x0a, 0x01, 0x02);
```

### 9.202 ex\_HTPFRepeat.nxc

This is an example of how to use the [HTPFRepeat](#) function.

```
HTPFRepeat(S1, 5, 100);
```

### 9.203 ex\_HTPFSingleOutputCST.nxc

This is an example of how to use the [HTPFSingleOutputCST](#) function.

```
HTPFSingleOutputCST(S1, PF_CHANNEL_1, PF_OUT_A, PF_CST_SET1_SET2);
```

### 9.204 ex\_HTPFSingleOutputPWM.nxc

This is an example of how to use the [HTPFSingleOutputPWM](#) function.

```
HTPFSingleOutputPWM(S1, PF_CHANNEL_1, PF_OUT_A, PF_PWM_FWD5);
```

### 9.205 ex\_HTPFSinglePin.nxc

This is an example of how to use the [HTPFSinglePin](#) function.

```
HTPFSinglePin(S1, PF_CHANNEL_1, PF_OUT_A, PF_PIN_C1, PF_FUNC_SET, true);
```

### 9.206 ex\_HTPFTrain.nxc

This is an example of how to use the [HTPFTrain](#) function.

```
HTPFTrain(S1, PF_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

### 9.207 ex\_HTRCXAddToDatalog.nxc

This is an example of how to use the [HTRCXAddToDatalog](#) function.

```
HTRCXAddToDatalog(RCX_InputValueSrc, S1);
```

### 9.208 ex\_HTRCXBatteryLevel.nxc

This is an example of how to use the [HTRCXBatteryLevel](#) function.

```
x = HTRCXBatteryLevel();
```

### 9.209 ex\_HTRCXClearAllEvents.nxc

This is an example of how to use the [HTRCXClearAllEvents](#) function.

```
HTRCXClearAllEvents();
```

### 9.210 ex\_HTRCXClearCounter.nxc

This is an example of how to use the [HTRCXClearCounter](#) function.

```
HTRCXClearCounter(0);
```

### 9.211 ex\_HTRCXClearMsg.nxc

This is an example of how to use the [HTRCXClearMsg](#) function.

```
HTRCXClearMsg();
```

**9.212 ex\_HTRCXCleatSensor.nxc**

This is an example of how to use the [HTRCXCleatSensor](#) function.

```
HTRCXCleatSensor (S1);
```

**9.213 ex\_HTRCXCleatSound.nxc**

This is an example of how to use the [HTRCXCleatSound](#) function.

```
HTRCXCleatSound();
```

**9.214 ex\_HTRCXCleatTimer.nxc**

This is an example of how to use the [HTRCXCleatTimer](#) function.

```
HTRCXCleatTimer (0);
```

**9.215 ex\_HTRCXCleatDatalog.nxc**

This is an example of how to use the [HTRCXCleatDatalog](#) function.

```
HTRCXCleatDatalog (50);
```

**9.216 ex\_HTRCXCleatDecCounter.nxc**

This is an example of how to use the [HTRCXCleatDecCounter](#) function.

```
HTRCXCleatDecCounter (0);
```

**9.217 ex\_HTRCXCleatDeleteSub.nxc**

This is an example of how to use the [HTRCXCleatDeleteSub](#) function.

```
HTRCXCleatDeleteSub (2);
```

### 9.218 ex\_HTRCXDeleteSubs.nxc

This is an example of how to use the [HTRCXDeleteSubs](#) function.

```
HTRCXDeleteSubs ();
```

### 9.219 ex\_HTRCXDeleteTask.nxc

This is an example of how to use the [HTRCXDeleteTask](#) function.

```
HTRCXDeleteTask (3);
```

### 9.220 ex\_HTRCXDeleteTasks.nxc

This is an example of how to use the [HTRCXDeleteTasks](#) function.

```
HTRCXDeleteTasks ();
```

### 9.221 ex\_HTRCXDisableOutput.nxc

This is an example of how to use the [HTRCXDisableOutput](#) function.

```
HTRCXDisableOutput (RCX_OUT_A);
```

### 9.222 ex\_HTRCXEnableOutput.nxc

This is an example of how to use the [HTRCXEnableOutput](#) function.

```
HTRCXEnableOutput (RCX_OUT_A);
```

### 9.223 ex\_HTRCXEvent.nxc

This is an example of how to use the [HTRCXEvent](#) function.

```
HTRCXEvent (RCX_ConstantSrc, 2);
```

### 9.224 ex\_HTRCXFloat.nxc

This is an example of how to use the [HTRCXFloat](#) function.

```
HTRCXFloat (RCX_OUT_A);
```

### 9.225 ex\_HTRCXFwd.nxc

This is an example of how to use the [HTRCXFwd](#) function.

```
HTRCXFwd (RCX_OUT_A);
```

### 9.226 ex\_HTRCXIncCounter.nxc

This is an example of how to use the [HTRCXIncCounter](#) function.

```
HTRCXIncCounter (0);
```

### 9.227 ex\_HTRCXInvertOutput.nxc

This is an example of how to use the [HTRCXInvertOutput](#) function.

```
HTRCXInvertOutput (RCX_OUT_A);
```

### 9.228 ex\_HTRCXMuteSound.nxc

This is an example of how to use the [HTRCXMuteSound](#) function.

```
HTRCXMuteSound ();
```

### 9.229 ex\_HTRCXObvertOutput.nxc

This is an example of how to use the [HTRCXObvertOutput](#) function.

```
HTRCXObvertOutput (RCX_OUT_A);
```

**9.230 ex\_HTRCXOff.nxc**

This is an example of how to use the [HTRCXOff](#) function.

```
HTRCXOff (RCX_OUT_A);
```

**9.231 ex\_HTRCXOn.nxc**

This is an example of how to use the [HTRCXOn](#) function.

```
HTRCXOn (RCX_OUT_A);
```

**9.232 ex\_HTRCXOnFor.nxc**

This is an example of how to use the [HTRCXOnFor](#) function.

```
HTRCXOnFor (RCX_OUT_A, 100);
```

**9.233 ex\_HTRCXOnFwd.nxc**

This is an example of how to use the [HTRCXOnFwd](#) function.

```
HTRCXOnFwd (RCX_OUT_A);
```

**9.234 ex\_HTRCXOnRev.nxc**

This is an example of how to use the [HTRCXOnRev](#) function.

```
HTRCXOnRev (RCX_OUT_A);
```

**9.235 ex\_HTRCXPBTurnOff.nxc**

This is an example of how to use the [HTRCXPBTurnOff](#) function.

```
HTRCXPBTurnOff ();
```

### 9.236 ex\_HTRCXPing.nxc

This is an example of how to use the [HTRCXPing](#) function.

```
HTRCXPing();
```

### 9.237 ex\_HTRCXPlaySound.nxc

This is an example of how to use the [HTRCXPlaySound](#) function.

```
HTRCXPlaySound(RCX_SOUND_UP);
```

### 9.238 ex\_HTRCXPlayTone.nxc

This is an example of how to use the [HTRCXPlayTone](#) function.

```
HTRCXPlayTone(440, 100);
```

### 9.239 ex\_HTRCXPlayToneVar.nxc

This is an example of how to use the [HTRCXPlayToneVar](#) function.

```
HTRCXPlayToneVar(0, 50);
```

### 9.240 ex\_HTRCXPoll.nxc

This is an example of how to use the [HTRCXPoll](#) function.

```
x = HTRCXPoll(RCX_VariableSrc, 0);
```

### 9.241 ex\_HTRCXPollMemory.nxc

This is an example of how to use the [HTRCXPollMemory](#) function.

```
HTRCXPollMemory(0, 10);
```

### 9.242 ex\_HTRCXRemote.nxc

This is an example of how to use the [HTRCXRemote](#) function.

```
HTRCXRemote(RCX_RemotePlayASound);
```

### 9.243 ex\_HTRCXRev.nxc

This is an example of how to use the [HTRCXRev](#) function.

```
HTRCXRev(RCX_OUT_A);
```

### 9.244 ex\_HTRCXSelectDisplay.nxc

This is an example of how to use the [HTRCXSelectDisplay](#) function.

```
HTRCXSelectDisplay(RCX_VariableSrc, 2);
```

### 9.245 ex\_HTRCXSelectProgram.nxc

This is an example of how to use the [HTRCXSelectProgram](#) function.

```
HTRCXSelectProgram(3);
```

### 9.246 ex\_HTRCXSendSerial.nxc

This is an example of how to use the [HTRCXSendSerial](#) function.

```
HTRCXSendSerial(0, 10);
```

### 9.247 ex\_HTRCXSetDirection.nxc

This is an example of how to use the [HTRCXSetDirection](#) function.

```
HTRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

### 9.248 ex\_HTRCXSetEvent.nxc

This is an example of how to use the [HTRCXSetEvent](#) function.

```
HTRCXSetEvent(0, RCX_ConstantSrc, 5);
```

### 9.249 ex\_HTRCXSetGlobalDirection.nxc

This is an example of how to use the [HTRCXSetGlobalDirection](#) function.

```
HTRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

### 9.250 ex\_HTRCXSetGlobalOutput.nxc

This is an example of how to use the [HTRCXSetGlobalOutput](#) function.

```
HTRCXSetGlobalOutput(RCX_OUT_A, RCX_OUT_ON);
```

### 9.251 ex\_HTRCXSetIRLinkPort.nxc

This is an example of how to use the [HTRCXSetIRLinkPort](#) function.

```
SetSensorLowspeed(S1);
```

### 9.252 ex\_HTRCXSetMaxPower.nxc

This is an example of how to use the [HTRCXSetMaxPower](#) function.

```
HTRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

### 9.253 ex\_HTRCXSetMessage.nxc

This is an example of how to use the [HTRCXSetMessage](#) function.

```
HTRCXSetMessage(20);
```

### 9.254 ex\_HTRCXSetOutput.nxc

This is an example of how to use the [HTRCXSetOutput](#) function.

```
HTRCXSetOutput (RCX_OUT_A, RCX_OUT_ON);
```

### 9.255 ex\_HTRCXSetPower.nxc

This is an example of how to use the [HTRCXSetPower](#) function.

```
HTRCXSetPower (RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

### 9.256 ex\_HTRCXSetPriority.nxc

This is an example of how to use the [HTRCXSetPriority](#) function.

```
HTRCXSetPriority (2);
```

### 9.257 ex\_HTRCXSetSensorMode.nxc

This is an example of how to use the [HTRCXSetSensorMode](#) function.

```
HTRCXSetSensorMode (S1, SENSOR_MODE_BOOL);
```

### 9.258 ex\_HTRCXSetSensorType.nxc

This is an example of how to use the [HTRCXSetSensorType](#) function.

```
HTRCXSetSensorType (S1, SENSOR_TYPE_TOUCH);
```

### 9.259 ex\_HTRCXSetSleepTime.nxc

This is an example of how to use the [HTRCXSetSleepTime](#) function.

```
HTRCXSetSleepTime (4);
```

**9.260 ex\_HTRCXSetTxPower.nxc**

This is an example of how to use the [HTRCXSetTxPower](#) function.

```
HTRCXSetTxPower(0);
```

**9.261 ex\_HTRCXSetWatch.nxc**

This is an example of how to use the [HTRCXSetWatch](#) function.

```
HTRCXSetWatch(3, 30);
```

**9.262 ex\_HTRCXStartTask.nxc**

This is an example of how to use the [HTRCXStartTask](#) function.

```
HTRCXStartTask(2);
```

**9.263 ex\_HTRCXStopAllTasks.nxc**

This is an example of how to use the [HTRCXStopAllTasks](#) function.

```
HTRCXStopAllTasks();
```

**9.264 ex\_HTRCXStopTask.nxc**

This is an example of how to use the [HTRCXStopTask](#) function.

```
HTRCXStopTask(1);
```

**9.265 ex\_HTRCXToggle.nxc**

This is an example of how to use the [HTRCXToggle](#) function.

```
HTRCXToggle(RCX_OUT_A);
```

### 9.266 ex\_HTRCXUnmuteSound.nxc

This is an example of how to use the [HTRCXUnmuteSound](#) function.

```
HTRCXUnmuteSound();
```

### 9.267 ex-HTScoutCalibrateSensor.nxc

This is an example of how to use the [HTScoutCalibrateSensor](#) function.

```
HTScoutCalibrateSensor();
```

### 9.268 ex-HTScoutMuteSound.nxc

This is an example of how to use the [HTScoutMuteSound](#) function.

```
HTScoutMuteSound();
```

### 9.269 ex-HTScoutSelectSounds.nxc

This is an example of how to use the [HTScoutSelectSounds](#) function.

```
HTScoutSelectSounds(0);
```

### 9.270 ex-HTScoutSendVLL.nxc

This is an example of how to use the [HTScoutSendVLL](#) function.

```
HTScoutSendVLL(RCX_ConstantSrc, 0x30);
```

### 9.271 ex-HTScoutSetEventFeedback.nxc

This is an example of how to use the [HTScoutSetEventFeedback](#) function.

```
HTScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

### 9.272 ex\_HTScoutSetLight.nxc

This is an example of how to use the [HTScoutSetLight](#) function.

```
HTScoutSetLight (SCOUT_LIGHT_ON);
```

### 9.273 ex\_HTScoutSetScoutMode.nxc

This is an example of how to use the [HTScoutSetScoutMode](#) function.

```
HTScoutSetScoutMode (SCOUT_MODE_POWER);
```

### 9.274 ex\_HTScoutSetSensorClickTime.nxc

This is an example of how to use the [HTScoutSetSensorClickTime](#) function.

```
HTScoutSetSensorClickTime (RCX_ConstantSrc, 200);
```

### 9.275 ex\_HTScoutSetSensorHysteresis.nxc

This is an example of how to use the [HTScoutSetSensorHysteresis](#) function.

```
HTScoutSetSensorHysteresis (RCX_ConstantSrc, 50);
```

### 9.276 ex\_HTScoutSetSensorLowerLimit.nxc

This is an example of how to use the [HTScoutSetSensorLowerLimit](#) function.

```
HTScoutSetSensorLowerLimit (RCX_VariableSrc, 0);
```

### 9.277 ex\_HTScoutSetSensorUpperLimit.nxc

This is an example of how to use the [HTScoutSetSensorUpperLimit](#) function.

```
HTScoutSetSensorUpperLimit (RCX_VariableSrc, 0);
```

### 9.278 ex\_HTScoutUnmuteSound.nxc

This is an example of how to use the [HTScoutUnmuteSound](#) function.

```
HTScoutUnmuteSound();
```

### 9.279 ex\_I2CBytes.nxc

This is an example of how to use the [I2CBytes](#) function.

```
x = I2CBytes(S4, writebuf, cnt, readbuf);
```

### 9.280 ex\_I2CBytesReady.nxc

This is an example of how to use the [I2CBytesReady](#) function.

```
x = I2CBytesReady(S1);
```

### 9.281 ex\_I2CCheckStatus.nxc

This is an example of how to use the [I2CCheckStatus](#) function.

```
x = I2CCheckStatus(S1);
```

### 9.282 ex\_i2cdeviceid.nxc

This is an example of how to use the [I2CDeviceId](#) function.

```
task main()
{
  SetSensorLowspeed(S1);
  while (true) {
    TextOut(0, LCD_LINE1, I2CVendorId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
  }
}
```

### 9.283 ex\_i2cdeviceinfo.nxc

This is an example of how to use the [I2CDeviceInfo](#) function.

```
task main()
{
  SetSensorLowspeed(S1);
  TextOut(0, LCD_LINE1, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_DEVICE_ID));
  TextOut(0, LCD_LINE2, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_VENDOR_ID));
  TextOut(0, LCD_LINE3, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_VERSION));
  Wait(SEC_10);
}
```

### 9.284 ex\_I2CRead.nxc

This is an example of how to use the [I2CRead](#) function.

```
x = I2CRead(S1, 1, outbuffer);
```

### 9.285 ex\_I2CSendCommand.nxc

This is an example of how to use the [I2CSendCommand](#) function.

```
long result = I2CSendCommand(S1, I2C_ADDR_DEFAULT, HT_CMD_COLOR2_ACTIVE);
```

### 9.286 ex\_I2CStatus.nxc

This is an example of how to use the [I2CStatus](#) function.

```
x = I2CStatus(S1, nRead);
```

### 9.287 ex\_i2cvendorid.nxc

This is an example of how to use the [I2CVendorId](#) function.

```
task main()
{
  SetSensorLowspeed(S1);
  while (true) {
    TextOut(0, LCD_LINE1, I2CVendorId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
  }
}
```

### 9.288 ex\_i2cversion.nxc

This is an example of how to use the [I2CVersion](#) function.

```
task main()
{
  SetSensorLowspeed(S1);
  while (true) {
    TextOut(0, LCD_LINE1, I2CVendorId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
    TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
  }
}
```

### 9.289 ex\_I2CWrite.nxc

This is an example of how to use the [I2CWrite](#) function.

```
x = I2CWrite(S1, 1, inbuffer);
```

### 9.290 ex\_isalnum.nxc

This is an example of how to use the [isalnum](#) function.

```
i = isalnum(x);
```

### 9.291 ex\_isalpha.nxc

This is an example of how to use the [isalpha](#) function.

```
i = isalpha(x);
```

### 9.292 ex\_iscntrl.nxc

This is an example of how to use the [iscntrl](#) function.

```
i = iscntrl(x);
```

**9.293 ex\_isdigit.nxc**

This is an example of how to use the [isdigit](#) function.

```
i = isdigit(x);
```

**9.294 ex\_isgraph.nxc**

This is an example of how to use the [isgraph](#) function.

```
i = isgraph(x);
```

**9.295 ex\_islower.nxc**

This is an example of how to use the [islower](#) function.

```
i = islower(x);
```

**9.296 ex\_isnan.nxc**

This is an example of how to use the [isNAN](#) function.

```
task main()
{
  float j = -1;
  float f = sqrt(j);
  if (isNAN(f))
    TextOut(0, LCD_LINE1, "not a number");
  else
    NumOut(0, LCD_LINE1, f);
  NumOut(0, LCD_LINE2, f);
  Wait(SEC_5);
}
```

**9.297 ex\_isprint.nxc**

This is an example of how to use the [isprint](#) function.

```
i = isprint(x);
```

### 9.298 ex\_ispunct.nxc

This is an example of how to use the [ispunct](#) function.

```
i = ispunct(x);
```

### 9.299 ex\_isspace.nxc

This is an example of how to use the [isspace](#) function.

```
i = isspace(x);
```

### 9.300 ex\_isupper.nxc

This is an example of how to use the [isupper](#) function.

```
i = isupper(x);
```

### 9.301 ex\_isxdigit.nxc

This is an example of how to use the [isxdigit](#) function.

```
i = isxdigit(x);
```

### 9.302 ex\_joystickmsg.nxc

This is an example of how to use the [JoystickMessageRead](#) function along with the [JoystickMessageType](#) structure.

```
/*
struct JoystickMessageType {
    byte JoystickDir;
    byte LeftMotor;
    byte RightMotor;
    byte BothMotors;
    char LeftSpeed;
    char RightSpeed;
    unsigned long Buttons;
};
*/
task main()
```

```
{
  JoystickMessageType jmt;
  while (true)
  {
    char result = JoystickMessageRead(MAILBOX1, jmt);
    if (result == NO_ERR)
    {
      NumOut(0, LCD_LINE1, jmt.JoystickDir);
      NumOut(0, LCD_LINE2, jmt.LeftMotor);
      NumOut(0, LCD_LINE3, jmt.RightMotor);
      NumOut(0, LCD_LINE4, jmt.BothMotors);
      ClearLine(LCD_LINE5);
      NumOut(0, LCD_LINE5, jmt.LeftSpeed);
      ClearLine(LCD_LINE6);
      NumOut(0, LCD_LINE6, jmt.RightSpeed);
      ClearLine(LCD_LINE7);
      NumOut(0, LCD_LINE7, jmt.Buttons);
    }
    else
      NumOut(0, LCD_LINE8, result);
    Wait(MS_100);
  }
}
```

### 9.303 ex\_labs.nxc

This is an example of how to use the [labs](#) function.

```
task main()
{
  float j = -1;
  float f = sqrt(j);
  if (isnan(f))
    TextOut(0, LCD_LINE1, "not a number");
  else
    NumOut(0, LCD_LINE1, f);
  NumOut(0, LCD_LINE2, f);
  Wait(SEC_5);
}
```

### 9.304 ex\_ldiv.nxc

This is an example of how to use the [ldiv](#) function.

```
task main()
{
  long x, y;
  x = 314564;
  y = 33;
  ldiv_t r;
  r = ldiv(x, y);
  NumOut(0, LCD_LINE1, r.quot);
}
```

```
    NumOut(0, LCD_LINE2, r.rem);  
    Wait(SEC_3);  
}
```

### 9.305 ex\_leftstr.nxc

This is an example of how to use the [LeftStr](#) function.

```
task main()  
{  
    string s = "Now is the winter of our discontent";  
    TextOut(0, LCD_LINE1, LeftStr(s, 12));  
    Wait(SEC_4);  
}
```

### 9.306 ex\_LineOut.nxc

This is an example of how to use the [LineOut](#) function.

```
task main()  
{  
    repeat(10) {  
        LineOut(0, 0, DISPLAY_WIDTH, DISPLAY_HEIGHT, DRAW_OPT_LOGICAL_XOR);  
        Wait(SEC_2);  
    }  
}
```

### 9.307 ex\_log.nxc

This is an example of how to use the [log](#) function.

```
y = log(x);
```

### 9.308 ex\_log10.nxc

This is an example of how to use the [log10](#) function.

```
y = log10(x);
```

### 9.309 ex\_LongAbort.nxc

This is an example of how to use the [LongAbort](#) function.

```
x = LongAbort();
```

### 9.310 ex\_LowLevelModuleRoutines.nxc

This is an example of how to use the [SetIOMapBytes](#), [SetIOMapValue](#), [GetIOMapBytes](#), [GetIOMapValue](#), [GetLowSpeedModuleBytes](#), [GetDisplayModuleBytes](#), [GetCommModuleBytes](#), [GetCommandModuleBytes](#), [SetLowSpeedModuleBytes](#), [SetDisplayModuleBytes](#), [SetCommModuleBytes](#), [SetCommandModuleBytes](#), [ref SetIOMapBytesByID](#), [SetIOMapValueByID](#), [GetIOMapBytesByID](#), [GetIOMapValueByID](#), [SetCommandModuleValue](#), [SetIOCtrlModuleValue](#), [SetLoaderModuleValue](#), [SetUIModuleValue](#), [SetSoundModuleValue](#), [SetButtonModuleValue](#), [SetInputModuleValue](#), [SetOutputModuleValue](#), [SetLowSpeedModuleValue](#), [SetDisplayModuleValue](#), [SetCommModuleValue](#), [GetCommandModuleValue](#), [GetLoaderModuleValue](#), [GetUIModuleValue](#), [GetSoundModuleValue](#), [GetButtonModuleValue](#), [GetInputModuleValue](#), [GetOutputModuleValue](#), [GetLowSpeedModuleValue](#), [GetDisplayModuleValue](#), [GetCommModuleValue](#),

```
task main()
{
/*
 * \example ex_LowLevelModuleRoutines.nxc
 * This is an example of how to use the \ref SetIOMapBytes, \ref SetIOMapValue,
 * \ref GetIOMapBytes, \ref GetIOMapValue, \ref GetLowSpeedModuleBytes,
 * \ref GetDisplayModuleBytes, \ref GetCommModuleBytes, \ref GetCommandModuleBytes,
 * \ref SetLowSpeedModuleBytes, \ref SetDisplayModuleBytes, \ref SetCommModuleBytes,
 * \ref SetCommandModuleBytes, ref SetIOMapBytesByID, \ref SetIOMapValueByID,
 * \ref GetIOMapBytesByID, \ref GetIOMapValueByID, \ref SetCommandModuleValue,
 * \ref SetIOCtrlModuleValue, \ref SetLoaderModuleValue, \ref SetUIModuleValue,
 * \ref SetSoundModuleValue, \ref SetButtonModuleValue, \ref SetInputModuleValue,
 * \ref SetOutputModuleValue, \ref SetLowSpeedModuleValue, \ref SetDisplayModuleValue,
 * \ref SetCommModuleValue, \ref GetCommandModuleValue, \ref GetIOCtrlModuleValue,
 * \ref GetLoaderModuleValue, \ref GetUIModuleValue, \ref GetSoundModuleValue,
 * \ref GetButtonModuleValue, \ref GetInputModuleValue, \ref GetOutputModuleValue,
 * \ref GetLowSpeedModuleValue, \ref GetDisplayModuleValue, \ref GetCommModuleValue,
 */
}
```

### 9.311 ex\_LowspeedBytesReady.nxc

This is an example of how to use the [LowspeedBytesReady](#) function.

```
x = LowspeedBytesReady(S1);
```

### 9.312 ex\_LowspeedCheckStatus.nxc

This is an example of how to use the [LowspeedCheckStatus](#) function.

```
x = LowspeedCheckStatus(S1);
```

### 9.313 ex\_LowspeedRead.nxc

This is an example of how to use the [LowspeedRead](#) function.

```
x = LowspeedRead(S1, 1, outbuffer);
```

### 9.314 ex\_LowspeedStatus.nxc

This is an example of how to use the [LowspeedStatus](#) function.

```
x = LowspeedStatus(S1, nRead);
```

### 9.315 ex\_LowspeedWrite.nxc

This is an example of how to use the [LowspeedWrite](#) function.

```
x = LowspeedWrite(S1, 1, inbuffer);
```

### 9.316 ex\_LSChannelState.nxc

This is an example of how to use the [LSChannelState](#) function.

```
x = LSChannelState(S1);
```

### 9.317 ex\_LSErrorType.nxc

This is an example of how to use the [LSErrorType](#) function.

```
x = LSErrorType(S1);
```

### 9.318 ex\_LSInputBufferBytesToRx.nxc

This is an example of how to use the [LSInputBufferBytesToRx](#) function.

```
x = LSInputBufferBytesToRx(S1);
```

### 9.319 ex\_LSInputBufferInPtr.nxc

This is an example of how to use the [LSInputBufferInPtr](#) function.

```
x = LSInputBufferInPtr(S1);
```

### 9.320 ex\_LSInputBufferOutPtr.nxc

This is an example of how to use the [LSInputBufferOutPtr](#) function.

```
x = LSInputBufferOutPtr(S1);
```

### 9.321 ex\_LSMode.nxc

This is an example of how to use the [LSMode](#) function.

```
x = LSMode(S1);
```

### 9.322 ex\_LSNoRestartOnRead.nxc

This is an example of how to use the [LSNoRestartOnRead](#) function.

```
byte val = LSNoRestartOnRead();
```

### 9.323 ex\_LSOutputBufferBytesToRx.nxc

This is an example of how to use the [LSOutputBufferBytesToRx](#) function.

```
x = LSOutputBufferBytesToRx(S1);
```

### 9.324 ex\_LSOutputBufferInPtr.nxc

This is an example of how to use the [LSOutputBufferInPtr](#) function.

```
x = LSOutputBufferInPtr(S1);
```

### 9.325 ex\_LSOutputBufferOutPtr.nxc

This is an example of how to use the [LSOutputBufferOutPtr](#) function.

```
x = LSOutputBufferOutPtr(S1);
```

### 9.326 ex\_LSSpeed.nxc

This is an example of how to use the [LSSpeed](#) function.

```
x = LSSpeed();
```

### 9.327 ex\_LSState.nxc

This is an example of how to use the [LSState](#) function.

```
x = LSState();
```

### 9.328 ex\_memcmp.nxc

This is an example of how to use the [memcmp](#) function.

```
task main()
{
  byte myArray[] = {1, 2, 3, 4};
  byte x[] = {1, 2, 3, 5};
  int i = 5;
  int j;
  j = memcmp(myArray, x, 1); // returns -1, 0, or 1
  NumOut(0, LCD_LINE1, i);
  NumOut(0, LCD_LINE2, j);
  NumOut(0, LCD_LINE3, memcmp(i, j, 1));
  Wait(SEC_15);
}
```

### 9.329 ex\_memcpy.nxc

This is an example of how to use the [memcpy](#) function.

```
memcpy(myArray, anotherArray, 1);
```

### 9.330 ex\_memmove.nxc

This is an example of how to use the [memmove](#) function.

```
memmove(myArray, anotherArray, 1);
```

### 9.331 ex\_midstr.nxc

This is an example of how to use the [MidStr](#) function.

```
task main()
{
  string s = "Now is the winter of our discontent";
  TextOut(0, LCD_LINE1, MidStr(s, 12, 5));
  Wait(SEC_4);
}
```

### 9.332 ex\_motoractualspeed.nxc

This is an example of how to use the [MotorActualSpeed](#) function.

```
x = MotorActualSpeed(OUT_A);
```

### 9.333 ex\_motorblocktachocount.nxc

This is an example of how to use the [MotorBlockTachoCount](#) function.

```
x = MotorBlockTachoCount(OUT_A);
```

### 9.334 ex\_motormode.nxc

This is an example of how to use the [MotorMode](#) function.

```
x = MotorMode(OUT_A);
```

### 9.335 ex\_motoroutputoptions.nxc

This is an example of how to use the [MotorOutputOptions](#) function.

```
task main()
{
  NumOut(0, LCD_LINE1, MotorOutputOptions(OUT_A));
  while(true);
}
```

### 9.336 ex\_motoroverload.nxc

This is an example of how to use the [MotorOverload](#) function.

```
x = MotorOverload(OUT_A);
```

### 9.337 ex\_motorpower.nxc

This is an example of how to use the [MotorPower](#) function.

```
x = MotorPower(OUT_A);
```

### 9.338 ex\_motorpwnfreq.nxc

This is an example of how to use the [MotorPwnFreq](#) function.

```
x = MotorPwnFreq();
```

### 9.339 ex\_motorregdvalue.nxc

This is an example of how to use the [MotorRegDValue](#) function.

```
x = MotorRegDValue(OUT_A);
```

### 9.340 ex\_motorregivalue.nxc

This is an example of how to use the [MotorRegIValue](#) function.

```
x = MotorRegIValue(OUT_A);
```

**9.341 ex\_motorregpvalue.nxc**

This is an example of how to use the [MotorRegPValue](#) function.

```
x = MotorRegPValue (OUT_A);
```

**9.342 ex\_motorregulation.nxc**

This is an example of how to use the [MotorRegulation](#) function.

```
x = MotorRegulation (OUT_A);
```

**9.343 ex\_motorrotationcount.nxc**

This is an example of how to use the [MotorRotationCount](#) function.

```
x = MotorRotationCount (OUT_A);
```

**9.344 ex\_motorruntime.nxc**

This is an example of how to use the [MotorRunState](#) function.

```
x = MotorRunState (OUT_A);
```

**9.345 ex\_motortachocount.nxc**

This is an example of how to use the [MotorTachoCount](#) function.

```
x = MotorTachoCount (OUT_A);
```

**9.346 ex\_motortacholimit.nxc**

This is an example of how to use the [MotorTachoLimit](#) function.

```
x = MotorTachoLimit (OUT_A);
```

**9.347 ex\_motorturnratio.nxc**

This is an example of how to use the [MotorTurnRatio](#) function.

```
x = MotorTurnRatio (OUT_A);
```

**9.348 ex\_MSADPAOff.nxc**

This is an example of how to use the [MSADPAOff](#) function.

```
char result = MSADPAOff(S1, MS_ADDR_DISTNX);
```

**9.349 ex\_MSADPAOn.nxc**

This is an example of how to use the [MSADPAOn](#) function.

```
char result = MSADPAOn(S1, MS_ADDR_DISTNX);
```

**9.350 ex\_MSDeenergize.nxc**

This is an example of how to use the [MSDeenergize](#) function.

```
char result = MSDeenergize(S1, I2C_ADDR_DEFAULT);
```

**9.351 ex\_MSEnergize.nxc**

This is an example of how to use the [MSEnergize](#) function.

```
char result = MSEnergize(S1, I2C_ADDR_DEFAULT);
```

**9.352 ex\_MSIRTrain.nxc**

This is an example of how to use the [MSIRTrain](#) function.

```
char result = MSIRTrain(S1, I2C_ADDR_DEFAULT, TRAIN_CHANNEL_1,  
    TRAIN_FUNC_INCR_SPEED);
```

**9.353 ex\_MSPFComboDirect.nxc**

This is an example of how to use the [MSPFComboDirect](#) function.

```
char result = MSPFComboDirect(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_CMD_STOP,  
    PF_CMD_FWD);
```

### 9.354 ex\_MSPFComboPWM.nxc

This is an example of how to use the [MSPFComboPWM](#) function.

```
char result = MSPFComboPWM(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_PWM_REV4,  
    PF_PWM_FWD5);
```

### 9.355 ex\_MSPFRawOutput.nxc

This is an example of how to use the [MSPFRawOutput](#) function.

```
char result = MSPFRawOutput(S1, I2C_ADDR_DEFAULT, 0x0a, 0x01, 0x02);
```

### 9.356 ex\_MSPFRepeat.nxc

This is an example of how to use the [MSPFRepeat](#) function.

```
char result = MSPFRepeat(S1, I2C_ADDR_DEFAULT, 5, 100);
```

### 9.357 ex\_MSPFSingleOutputCST.nxc

This is an example of how to use the [MSPFSingleOutputCST](#) function.

```
char result = MSPFSingleOutputCST(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
    PF_CST_SET1_SET2);
```

### 9.358 ex\_MSPFSingleOutputPWM.nxc

This is an example of how to use the [MSPFSingleOutputPWM](#) function.

```
char result = MSPFSingleOutputPWM(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
    PF_PWM_FWD5);
```

### 9.359 ex\_MSPFSinglePin.nxc

This is an example of how to use the [MSPFSinglePin](#) function.

```
char result = MSPFSinglePin(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
    PF_PIN_C1, PF_FUNC_SET, true);
```

### 9.360 ex\_MSPFTrain.nxc

This is an example of how to use the [MSPFTrain](#) function.

```
char result = MSPFTrain(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1,  
    TRAIN_FUNC_INCR_SPEED);
```

### 9.361 ex\_MSRCXAbsVar.nxc

This is an example of how to use the [MSRCXAbsVar](#) function.

```
MSRCXAbsVar(0, RCX_VariableSrc, 0);
```

### 9.362 ex\_MSRCXAddToDatalog.nxc

This is an example of how to use the [MSRCXAddToDatalog](#) function.

```
MSRCXAddToDatalog(RCX_InputValueSrc, S1);
```

### 9.363 ex\_MSRCXAndVar.nxc

This is an example of how to use the [MSRCXAndVar](#) function.

```
MSRCXAndVar(0, RCX_ConstantSrc, 0x7f);
```

### 9.364 ex\_MSRCXBatteryLevel.nxc

This is an example of how to use the [MSRCXBatteryLevel](#) function.

```
x = MSRCXBatteryLevel();
```

### 9.365 ex\_MSRCXBoot.nxc

This is an example of how to use the [MSRCXBoot](#) function.

```
MSRCXBoot();
```

### 9.366 ex\_MSRCXCalibrateEvent.nxc

This is an example of how to use the [MSRCXCalibrateEvent](#) function.

```
MSRCXCalibrateEvent(0, 200, 500, 50);
```

### 9.367 ex\_MSRCXClearAllEvents.nxc

This is an example of how to use the [MSRCXClearAllEvents](#) function.

```
MSRCXClearAllEvents();
```

### 9.368 ex\_MSRCXClearCounter.nxc

This is an example of how to use the [MSRCXClearCounter](#) function.

```
MSRCXClearCounter(0);
```

### 9.369 ex\_MSRCXClearMsg.nxc

This is an example of how to use the [MSRCXClearMsg](#) function.

```
MSRCXClearMsg();
```

### 9.370 ex\_MSRCXClearSensor.nxc

This is an example of how to use the [MSRCXClearSensor](#) function.

```
MSRCXClearSensor(S1);
```

### 9.371 ex\_MSRCXClearSound.nxc

This is an example of how to use the [MSRCXClearSound](#) function.

```
MSRCXClearSound();
```

**9.372 ex\_MSRCXClearTimer.nxc**

This is an example of how to use the [MSRCXClearTimer](#) function.

```
MSRCXClearTimer(0);
```

**9.373 ex\_MSRCXCreateDatalog.nxc**

This is an example of how to use the [MSRCXCreateDatalog](#) function.

```
MSRCXCreateDatalog(50);
```

**9.374 ex\_MSRCXDecCounter.nxc**

This is an example of how to use the [MSRCXDecCounter](#) function.

```
MSRCXDecCounter(0);
```

**9.375 ex\_MSRCXDeleteSub.nxc**

This is an example of how to use the [MSRCXDeleteSub](#) function.

```
MSRCXDeleteSub(2);
```

**9.376 ex\_MSRCXDeleteSubs.nxc**

This is an example of how to use the [MSRCXDeleteSubs](#) function.

```
MSRCXDeleteSubs();
```

**9.377 ex\_MSRCXDeleteTask.nxc**

This is an example of how to use the [MSRCXDeleteTask](#) function.

```
MSRCXDeleteTask(3);
```

### 9.378 ex\_MSRCXDeleteTasks.nxc

This is an example of how to use the [MSRCXDeleteTasks](#) function.

```
MSRCXDeleteTasks();
```

### 9.379 ex\_MSRCXDisableOutput.nxc

This is an example of how to use the [MSRCXDisableOutput](#) function.

```
MSRCXDisableOutput(RCX_OUT_A);
```

### 9.380 ex\_MSRCXDivVar.nxc

This is an example of how to use the [MSRCXDivVar](#) function.

```
MSRCXDivVar(0, RCX_ConstantSrc, 2);
```

### 9.381 ex\_MSRCXEnableOutput.nxc

This is an example of how to use the [MSRCXEnableOutput](#) function.

```
MSRCXEnableOutput(RCX_OUT_A);
```

### 9.382 ex\_MSRCXEvent.nxc

This is an example of how to use the [MSRCXEvent](#) function.

```
MSRCXEvent(RCX_ConstantSrc, 2);
```

### 9.383 ex\_MSRCXFloat.nxc

This is an example of how to use the [MSRCXFloat](#) function.

```
MSRCXFloat(RCX_OUT_A);
```

**9.384 ex\_MSRCXFwd.nxc**

This is an example of how to use the [MSRCXFwd](#) function.

```
MSRCXFwd(RCX_OUT_A);
```

**9.385 ex\_MSRCXIncCounter.nxc**

This is an example of how to use the [MSRCXIncCounter](#) function.

```
MSRCXIncCounter(0);
```

**9.386 ex\_MSRCXInvertOutput.nxc**

This is an example of how to use the [MSRCXInvertOutput](#) function.

```
MSRCXInvertOutput(RCX_OUT_A);
```

**9.387 ex\_MSRCXMulVar.nxc**

This is an example of how to use the [MSRCXMulVar](#) function.

```
MSRCXMulVar(0, RCX_VariableSrc, 4);
```

**9.388 ex\_MSRCXMuteSound.nxc**

This is an example of how to use the [MSRCXMuteSound](#) function.

```
MSRCXMuteSound();
```

**9.389 ex\_MSRCXObvertOutput.nxc**

This is an example of how to use the [MSRCXObvertOutput](#) function.

```
MSRCXObvertOutput(RCX_OUT_A);
```

**9.390 ex\_MSRCXOff.nxc**

This is an example of how to use the [MSRCXOff](#) function.

```
MSRCXOff (RCX_OUT_A);
```

**9.391 ex\_MSRCXOn.nxc**

This is an example of how to use the [MSRCXOn](#) function.

```
MSRCXOn (RCX_OUT_A);
```

**9.392 ex\_MSRCXOnFor.nxc**

This is an example of how to use the [MSRCXOnFor](#) function.

```
MSRCXOnFor (RCX_OUT_A, 100);
```

**9.393 ex\_MSRCXOnFwd.nxc**

This is an example of how to use the [MSRCXOnFwd](#) function.

```
MSRCXOnFwd (RCX_OUT_A);
```

**9.394 ex\_MSRCXOnRev.nxc**

This is an example of how to use the [MSRCXOnRev](#) function.

```
MSRCXOnRev (RCX_OUT_A);
```

**9.395 ex\_MSRCXOrVar.nxc**

This is an example of how to use the [MSRCXOrVar](#) function.

```
MSRCXOrVar (0, RCX_ConstantSrc, 0xCC);
```

**9.396 ex\_MSRCXPBTurnOff.nxc**

This is an example of how to use the [MSRCXPBTurnOff](#) function.

```
MSRCXPBTurnOff ();
```

**9.397 ex\_MSRCXPing.nxc**

This is an example of how to use the [MSRCXPing](#) function.

```
MSRCXPing ();
```

**9.398 ex\_MSRCXPlaySound.nxc**

This is an example of how to use the [MSRCXPlaySound](#) function.

```
MSRCXPlaySound (RCX_SOUND_UP);
```

**9.399 ex\_MSRCXPlayTone.nxc**

This is an example of how to use the [MSRCXPlayTone](#) function.

```
MSRCXPlayTone (440, 100);
```

**9.400 ex\_MSRCXPlayToneVar.nxc**

This is an example of how to use the [MSRCXPlayToneVar](#) function.

```
MSRCXPlayToneVar (0, 50);
```

**9.401 ex\_MSRCXPoll.nxc**

This is an example of how to use the [MSRCXPoll](#) function.

```
x = MSRCXPoll (RCX_VariableSrc, 0);
```

### 9.402 ex\_MSRCXPollMemory.nxc

This is an example of how to use the [MSRCXPollMemory](#) function.

```
MSRCXPollMemory(0, 10);
```

### 9.403 ex\_MSRCXRemote.nxc

This is an example of how to use the [MSRCXRemote](#) function.

```
MSRCXRemote(RCX_RemotePlayASound);
```

### 9.404 ex\_MSRCXReset.nxc

This is an example of how to use the [MSRCXReset](#) function.

```
MSRCXReset();
```

### 9.405 ex\_MSRCXRev.nxc

This is an example of how to use the [MSRCXRev](#) function.

```
MSRCXRev(RCX_OUT_A);
```

### 9.406 ex\_MSRCXSelectDisplay.nxc

This is an example of how to use the [MSRCXSelectDisplay](#) function.

```
MSRCXSelectDisplay(RCX_VariableSrc, 2);
```

### 9.407 ex\_MSRCXSelectProgram.nxc

This is an example of how to use the [MSRCXSelectProgram](#) function.

```
MSRCXSelectProgram(3);
```

### 9.408 ex\_MSRCXSendSerial.nxc

This is an example of how to use the [MSRCXSendSerial](#) function.

```
MSRCXSendSerial(0, 10);
```

### 9.409 ex\_MSRCXSet.nxc

This is an example of how to use the [MSRCXSet](#) function.

```
MSRCXSet(RCX_VariableSrc, 0, RCX_RandomSrc, 10000);
```

### 9.410 ex\_MSRCXSetDirection.nxc

This is an example of how to use the [MSRCXSetDirection](#) function.

```
MSRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

### 9.411 ex\_MSRCXSetEvent.nxc

This is an example of how to use the [MSRCXSetEvent](#) function.

```
MSRCXSetEvent(0, RCX_ConstantSrc, 5);
```

### 9.412 ex\_MSRCXSetGlobalDirection.nxc

This is an example of how to use the [MSRCXSetGlobalDirection](#) function.

```
MSRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

### 9.413 ex\_MSRCXSetGlobalOutput.nxc

This is an example of how to use the [MSRCXSetGlobalOutput](#) function.

```
MSRCXSetGlobalOutput(RCX_OUT_A, RCX_OUT_ON);
```

**9.414 ex\_MSRCXSetMaxPower.nxc**

This is an example of how to use the [MSRCXSetMaxPower](#) function.

```
MSRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

**9.415 ex\_MSRCXSetMessage.nxc**

This is an example of how to use the [MSRCXSetMessage](#) function.

```
MSRCXSetMessage(20);
```

**9.416 ex\_MSRCXSetNRLinkPort.nxc**

This is an example of how to use the [MSRCXSetNRLinkPort](#) function.

```
MSRCXSetNRLinkPort(S1, MS_ADDR_NRLINK);
```

**9.417 ex\_MSRCXSetOutput.nxc**

This is an example of how to use the [MSRCXSetOutput](#) function.

```
MSRCXSetOutput(RCX_OUT_A, RCX_OUT_ON);
```

**9.418 ex\_MSRCXSetPower.nxc**

This is an example of how to use the [MSRCXSetPower](#) function.

```
MSRCXSetPower(RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

**9.419 ex\_MSRCXSetPriority.nxc**

This is an example of how to use the [MSRCXSetPriority](#) function.

```
MSRCXSetPriority(2);
```

### 9.420 ex\_MSRCXSetSensorMode.nxc

This is an example of how to use the [MSRCXSetSensorMode](#) function.

```
MSRCXSetSensorMode (S1, SENSOR_MODE_BOOL);
```

### 9.421 ex\_MSRCXSetSensorType.nxc

This is an example of how to use the [MSRCXSetSensorType](#) function.

```
MSRCXSetSensorType (S1, SENSOR_TYPE_TOUCH);
```

### 9.422 ex\_MSRCXSetSleepTime.nxc

This is an example of how to use the [MSRCXSetSleepTime](#) function.

```
MSRCXSetSleepTime (4);
```

### 9.423 ex\_MSRCXSetTxPower.nxc

This is an example of how to use the [MSRCXSetTxPower](#) function.

```
MSRCXSetTxPower (0);
```

### 9.424 ex\_MSRCXSetUserDisplay.nxc

This is an example of how to use the [MSRCXSetUserDisplay](#) function.

```
MSRCXSetUserDisplay (RCX_VariableSrc, 0, 2);
```

### 9.425 ex\_MSRCXSetVar.nxc

This is an example of how to use the [MSRCXSetVar](#) function.

```
MSRCXSetVar (0, RCX_VariableSrc, 1);
```

**9.426 ex\_MSRCXSetWatch.nxc**

This is an example of how to use the [MSRCXSetWatch](#) function.

```
MSRCXSetWatch(3, 30);
```

**9.427 ex\_MSRCXSgnVar.nxc**

This is an example of how to use the [MSRCXSgnVar](#) function.

```
MSRCXSgnVar(0, RCX_VariableSrc, 0);
```

**9.428 ex\_MSRCXStartTask.nxc**

This is an example of how to use the [MSRCXStartTask](#) function.

```
MSRCXStartTask(2);
```

**9.429 ex\_MSRCXStopAllTasks.nxc**

This is an example of how to use the [MSRCXStopAllTasks](#) function.

```
MSRCXStopAllTasks();
```

**9.430 ex\_MSRCXStopTask.nxc**

This is an example of how to use the [MSRCXStopTask](#) function.

```
MSRCXStopTask(1);
```

**9.431 ex\_MSRCXSubVar.nxc**

This is an example of how to use the [MSRCXSubVar](#) function.

```
MSRCXSubVar(0, RCX_RandomSrc, 10);
```

### 9.432 ex\_MSRCXSumVar.nxc

This is an example of how to use the [MSRCXSumVar](#) function.

```
MSRCXSumVar(0, RCX_InputValueSrc, S1);
```

### 9.433 ex\_MSRCXToggle.nxc

This is an example of how to use the [MSRCXToggle](#) function.

```
MSRCXToggle(RCX_OUT_A);
```

### 9.434 ex\_MSRCXUnlock.nxc

This is an example of how to use the [MSRCXUnlock](#) function.

```
MSRCXUnlock();
```

### 9.435 ex\_MSRCXUnmuteSound.nxc

This is an example of how to use the [MSRCXUnmuteSound](#) function.

```
MSRCXUnmuteSound();
```

### 9.436 ex\_MSReadValue.nxc

This is an example of how to use the [MSReadValue](#) function.

```
byte value = MSReadValue(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, 1);
```

### 9.437 ex\_MSScoutCalibrateSensor.nxc

This is an example of how to use the [MSScoutCalibrateSensor](#) function.

```
MSScoutCalibrateSensor();
```

**9.438 ex\_MSScoutMuteSound.nxc**

This is an example of how to use the [MSScoutMuteSound](#) function.

```
MSScoutMuteSound();
```

**9.439 ex\_MSScoutSelectSounds.nxc**

This is an example of how to use the [MSScoutSelectSounds](#) function.

```
MSScoutSelectSounds(0);
```

**9.440 ex\_MSScoutSendVLL.nxc**

This is an example of how to use the [MSScoutSendVLL](#) function.

```
MSScoutSendVLL(RCX_ConstantSrc, 0x30);
```

**9.441 ex\_MSScoutSetCounterLimit.nxc**

This is an example of how to use the [MSScoutSetCounterLimit](#) function.

```
MSScoutSetCounterLimit(0, RCX_ConstantSrc, 2000);
```

**9.442 ex\_MSScoutSetEventFeedback.nxc**

This is an example of how to use the [MSScoutSetEventFeedback](#) function.

```
MSScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

**9.443 ex\_MSScoutSetLight.nxc**

This is an example of how to use the [MSScoutSetLight](#) function.

```
MSScoutSetLight(SCOUT_LIGMS_ON);
```

#### 9.444 ex\_MSScoutSetScoutMode.nxc

This is an example of how to use the [MSScoutSetScoutMode](#) function.

```
MSScoutSetScoutMode (SCOUT_MODE_POWER);
```

#### 9.445 ex\_MSScoutSetScoutRules.nxc

This is an example of how to use the [MSScoutSetScoutRules](#) function.

```
MSScoutSetScoutRules (SCOUT_MR_FORWARD, SCOUT_TR_REVERSE, SCOUT_LR_IGNORE,  
SCOUT_TGS_SHORT, SCOUT_FXR_BUG);
```

#### 9.446 ex\_MSScoutSetSensorClickTime.nxc

This is an example of how to use the [MSScoutSetSensorClickTime](#) function.

```
MSScoutSetSensorClickTime (RCX_ConstantSrc, 200);
```

#### 9.447 ex\_MSScoutSetSensorHysteresis.nxc

This is an example of how to use the [MSScoutSetSensorHysteresis](#) function.

```
MSScoutSetSensorHysteresis (RCX_ConstantSrc, 50);
```

#### 9.448 ex\_MSScoutSetSensorLowerLimit.nxc

This is an example of how to use the [MSScoutSetSensorLowerLimit](#) function.

```
MSScoutSetSensorLowerLimit (RCX_VariableSrc, 0);
```

#### 9.449 ex\_MSScoutSetSensorUpperLimit.nxc

This is an example of how to use the [MSScoutSetSensorUpperLimit](#) function.

```
MSScoutSetSensorUpperLimit (RCX_VariableSrc, 0);
```

### 9.450 ex\_MSScoutSetTimerLimit.nxc

This is an example of how to use the [MSScoutSetTimerLimit](#) function.

```
MSScoutSetTimerLimit(0, RCX_ConstantSrc, 10000);
```

### 9.451 ex\_MSScoutUnmuteSound.nxc

This is an example of how to use the [MSScoutUnmuteSound](#) function.

```
MSScoutUnmuteSound();
```

### 9.452 ex\_muldiv32.nxc

This is an example of how to use the [muldiv32](#) function.

```
y = muldiv32(a, b, c);
```

### 9.453 ex\_nbcopt.nxc

This is an example of how to use the [ArrayIndex](#), [ArrayReplace](#), [BranchComp](#), and [BranchTest](#) functions.

```
task main()
{
    float A[3][3];
    float C[][];
    int R, S;
    float tmp[], arr_temp[], val_temp;
    int s, r;
    ArrayInit(tmp, 0, R);
    ArrayInit(C, tmp, S);
    s = S;
    lbl_Trans_start_s:
    {
        s--;
        r = R;
        lbl_Trans_start_r:
        {
            r--;
            ArrayIndex(arr_temp, A, r);
            ArrayIndex(val_temp, arr_temp, s);
            ArrayReplace(tmp, r, val_temp);
        }
        BranchComp(GT, lbl_Trans_start_r, r, 0);
    }
}
```

```
        ArrayReplace(C, s, tmp);
    }
    BranchTest(GT, lbl_Trans_start_s, s);
}
```

#### 9.454 ex\_NRLink2400.nxc

This is an example of how to use the [NRLink2400](#) function.

```
char result = NRLink2400(S1, MS_ADDR_NRLINK);
```

#### 9.455 ex\_NRLink4800.nxc

This is an example of how to use the [NRLink4800](#) function.

```
char result = NRLink4800(S1, MS_ADDR_NRLINK);
```

#### 9.456 ex\_NRLinkFlush.nxc

This is an example of how to use the [NRLinkFlush](#) function.

```
char result = NRLinkFlush(S1, MS_ADDR_NRLINK);
```

#### 9.457 ex\_NRLinkIRLong.nxc

This is an example of how to use the [NRLinkIRLong](#) function.

```
char result = NRLinkIRLong(S1, MS_ADDR_NRLINK);
```

#### 9.458 ex\_NRLinkIRShort.nxc

This is an example of how to use the [NRLinkIRShort](#) function.

```
char result = NRLinkIRShort(S1, MS_ADDR_NRLINK);
```

#### 9.459 ex\_NRLinkSetPF.nxc

This is an example of how to use the [NRLinkSetPF](#) function.

```
char result = NRLinkSetPF(S1, MS_ADDR_NRLINK);
```

**9.460 ex\_NRLinkSetRCX.nxc**

This is an example of how to use the [NRLinkSetRCX](#) function.

```
char result = NRLinkSetRCX(S1, MS_ADDR_NRLINK);
```

**9.461 ex\_NRLinkSetTrain.nxc**

This is an example of how to use the [NRLinkSetTrain](#) function.

```
char result = NRLinkSetTrain(S1, MS_ADDR_NRLINK);
```

**9.462 ex\_NRLinkStatus.nxc**

This is an example of how to use the [NRLinkStatus](#) function.

```
byte result = NRLinkStatus(S1, MS_ADDR_NRLINK);
```

**9.463 ex\_NRLinkTxRaw.nxc**

This is an example of how to use the [NRLinkTxRaw](#) function.

```
byte result = NRLinkTxRaw(S1, MS_ADDR_NRLINK);
```

**9.464 ex\_NumOut.nxc**

This is an example of how to use the [NumOut](#) function.

```
NumOut(0, LCD_LINE1, x);
```

**9.465 ex\_NumToStr.nxc**

This is an example of how to use the [NumToStr](#) function.

```
msg = NumToStr(-2); // returns "-2" in a string
```

### 9.466 ex\_NXTHID.nxc

This is an example of how to use the [NXTHIDAsciiMode](#), [NXTHIDDirectMode](#), [NXTHIDTransmit](#), [NXTHIDLoadCharacter](#), [SetSensorLowspeed](#), and [Wait](#) functions.

```
task main()
{
  SetSensorLowspeed(S1); // NXTHID is an i2c device

  char result;

  // configure device in ASCII mode
  result = NXTHIDAsciiMode(S1, MS_ADDR_NXTHID);

  // load a character
  result = NXTHIDLoadCharacter(S1, MS_ADDR_NXTHID, NXTHID_MOD_NONE, 'A');

  // transmit the character
  result = NXTHIDTransmit(S1, MS_ADDR_NXTSERVO);

  Wait(SEC_5);

  // configure device in Direct mode
  result = NXTHIDDirectMode(S1, MS_ADDR_NXTHID);

  // load a character
  result = NXTHIDLoadCharacter(S1, MS_ADDR_NXTHID, NXTHID_MOD_LEFT_CTRL, 'd'); //
    ctrl+d

  // transmit the character
  result = NXTHIDTransmit(S1, MS_ADDR_NXTSERVO);

  Wait(SEC_5);
}
```

### 9.467 ex\_NXTLineLeader.nxc

This is an example of how to use the [NXTLineLeaderSteering](#), [NXTLineLeaderAverage](#), [NXTLineLeaderResult](#), [NXTLineLeaderPowerDown](#), [NXTLineLeaderPowerUp](#), [NXTLineLeaderInvert](#), [NXTLineLeaderReset](#), [NXTLineLeaderSnapshot](#), [NXTLineLeaderCalibrateWhite](#), [NXTLineLeaderCalibrateBlack](#), [SetNXTLineLeaderSetpoint](#), [SetNXTLineLeaderKpValue](#), [SetNXTLineLeaderKiValue](#), [SetNXTLineLeaderKpValue](#), [SetNXTLineLeaderKpFactor](#), [SetNXTLineLeaderKiFactor](#), [SetNXTLineLeaderKdFactor](#), [SetSensorLowspeed](#), [NumOut](#), and [Wait](#) functions.

```
task main()
{
  SetSensorLowspeed(S1); // NXTLineLeader is an i2c device

  char val;
  // position sensor over white surface for 1 second
  val = NXTLineLeaderCalibrateWhite(S1, MS_ADDR_LINELDR);
}
```

```
Wait(SEC_1);

// position sensor over black surface for 1 second
val = NXTLineLeaderCalibrateBlack(S1, MS_ADDR_LINELDR);

Wait(SEC_1);

// position sensor over line
byte steering, average, result;
steering = NXTLineLeaderSteering(S1, MS_ADDR_LINELDR);
average = NXTLineLeaderAverage(S1, MS_ADDR_LINELDR);
result = NXTLineLeaderResult(S1, MS_ADDR_LINELDR);

NumOut(0, LCD_LINE1, steering);
NumOut(0, LCD_LINE2, average);
NumOut(0, LCD_LINE3, result);

Wait(SEC_5);

// put the device to sleep
val = NXTLineLeaderPowerDown(S1, MS_ADDR_LINELDR);

Wait(SEC_5);

// wake up the device
val = NXTLineLeaderPowerUp(S1, MS_ADDR_LINELDR);

// invert colors (white line on black surface)
val = NXTLineLeaderInvert(S1, MS_ADDR_LINELDR);

Wait(SEC_5);

// reset back to default colors
val = NXTLineLeaderReset(S1, MS_ADDR_LINELDR);

Wait(SEC_5);

// take a snapshot of the surface below the device
val = NXTLineLeaderSnapshot(S1, MS_ADDR_LINELDR);

// set sensor configuration values to non-defaults
val = SetNXTLineLeaderSetpoint(S1, MS_ADDR_LINELDR, 10); // default is 45
// set PID values
val = SetNXTLineLeaderKpValue(S1, MS_ADDR_LINELDR, 100); // default is 25
val = SetNXTLineLeaderKiValue(S1, MS_ADDR_LINELDR, 10); // default is 0
val = SetNXTLineLeaderKdValue(S1, MS_ADDR_LINELDR, 50); // default is 8
// set PID factors
val = SetNXTLineLeaderKpFactor(S1, MS_ADDR_LINELDR, 40); // default is 32
val = SetNXTLineLeaderKiFactor(S1, MS_ADDR_LINELDR, 40); // default is 32
val = SetNXTLineLeaderKdFactor(S1, MS_ADDR_LINELDR, 40); // default is 32

Wait(SEC_5);
}
```

### 9.468 ex\_NXTPowerMeter.nxc

This is an example of how to use the [NXTPowerMeterResetCounters](#), [NXTPowerMeterPresentCurrent](#), [NXTPowerMeterPresentVoltage](#), [NXTPowerMeterCapacityUsed](#), [NXTPowerMeterPresentPower](#), [NXTPowerMeterTotalPowerConsumed](#), [NXTPowerMeterMaxCurrent](#), [NXTPowerMeterMinCurrent](#), [NXTPowerMeterMaxVoltage](#), [NXTPowerMeterMinVoltage](#), [NXTPowerMeterElapsedTime](#), [NXTPowerMeterErrorCount](#), [SetSensorLowspeed](#), [NumOut](#), and [Wait](#) functions.

```
task main()
{
    SetSensorLowspeed(S1); // NXTPowerMeter is an i2c device

    char result;

    // reset the counters
    result = NXTPowerMeterResetCounters(S1, MS_ADDR_IVSENS);

    // wait 10 seconds
    Wait(SEC_10);

    // output values

    NumOut(0, LCD_LINE1, NXTPowerMeterPresentCurrent(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE2, NXTPowerMeterPresentVoltage(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE3, NXTPowerMeterCapacityUsed(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE4, NXTPowerMeterPresentPower(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE5, NXTPowerMeterTotalPowerConsumed(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE6, NXTPowerMeterMaxCurrent(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE7, NXTPowerMeterMinCurrent(S1, MS_ADDR_IVSENS));
    Wait(SEC_5);
    NumOut(0, LCD_LINE1, NXTPowerMeterMaxVoltage(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE2, NXTPowerMeterMinVoltage(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE3, NXTPowerMeterElapsedTime(S1, MS_ADDR_IVSENS));
    NumOut(0, LCD_LINE4, NXTPowerMeterErrorCount(S1, MS_ADDR_IVSENS));
    Wait(SEC_5);
}
```

### 9.469 ex\_NXTServo.nxc

This is an example of how to use the [NXTServoPosition](#), [NXTServoSpeed](#), [NXTServoBatteryVoltage](#), [SetNXTServoSpeed](#), [SetNXTServoQuickPosition](#), [SetNXTServoPosition](#), [NXTServoReset](#), [NXTServoHaltMacro](#), [NXTServoResumeMacro](#), [NXTServoPauseMacro](#), [NXTServoInit](#), [NXTServoGotoMacroAddress](#), [NXTServoEditMacro](#), [NXTServoQuitEdit](#), [SetSensorLowspeed](#), [NumOut](#), and [Wait](#) functions.

```
task main()
{
    SetSensorLowspeed(S1); // NXTServo is an i2c device

    // edit a macro
```

```
char result;
result = NXTServoEditMacro(S1, MS_ADDR_NXTSERVO);

// TODO: write bytes of macro data to the device

result = NXTServoQuitEdit(S1);

// run a macro at address 0x30
result = NXTServoGotoMacroAddress(S1, MS_ADDR_NXTSERVO, 0x30);
Wait(SEC_1);

// pause the macro
result = NXTServoPauseMacro(S1, MS_ADDR_NXTSERVO);
Wait(SEC_1);

// resume the macro
result = NXTServoResumeMacro(S1, MS_ADDR_NXTSERVO);
Wait(SEC_1);

// halt the macro
result = NXTServoHaltMacro(S1, MS_ADDR_NXTSERVO);

// set a non-default speed value for a servo (0 = full speed)
result = SetNXTServoSpeed(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1, 10);

// set a non-default quick position value for a servo
result = SetNXTServoQuickPosition(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1,
    NXTSERVO_QPOS_MIN);

// Wait a bit for the servo to reach its new position
Wait(SEC_5);

// set a non-default position value for a servo
result = SetNXTServoPosition(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1,
    NXTSERVO_POS_CENTER);

// store these non-default values as the initial position for this servo
result = NXTServoInit(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1);

// output the battery voltage
NumOut(0, LCD_LINE1, NXTServoBatteryVoltage(S1, MS_ADDR_NXTSERVO));

// output the current position
NumOut(0, LCD_LINE2, NXTServoPosition(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1));

// output the current speed
NumOut(0, LCD_LINE3, NXTServoSpeed(S1, MS_ADDR_NXTSERVO, NXTSERVO_SERVO_1));

Wait(SEC_5);

// reset the device back to default speed/position (0/1500) settings for all se
    rvos
result = NXTServoReset(S1, MS_ADDR_NXTSERVO);

Wait(SEC_5);
}
```

### 9.470 ex\_NXTSumoEyes.nxc

This is an example of how to use the [SetSensorNXTSumoEyes](#), [SensorNXTSumoEyes](#), [SensorNXTSumoEyesRaw](#), [NumOut](#), and [Wait](#) functions.

```
inline void TurnLeft() { }
inline void TurnRight() { }
inline void GoStraight() { }
inline void SearchForObstacle() { }

task main()
{
  SetSensorNXTSumoEyes(S1, true); // long range
  while(true)
  {
    char zone = SensorNXTSumoEyes(S1);
    switch (zone) {
      case NXTSE_ZONE_LEFT:
        TurnLeft();
        break;
      case NXTSE_ZONE_RIGHT:
        TurnRight();
        break;
      case NXTSE_ZONE_FRONT:
        GoStraight();
        break;
      default:
        SearchForObstacle();
        break;
    }
    NumOut(0, LCD_LINE1, SensorNXTSumoEyesRaw(S1));
    Wait(MS_500);
  }
}
```

### 9.471 ex\_off.nxc

This is an example of how to use the [Off](#) function.

```
Off(OUT_A); // turn off output A
```

### 9.472 ex\_offex.nxc

This is an example of how to use the [OffEx](#) function.

```
OffEx(OUT_A, RESET_NONE); // turn off output A
```

### 9.473 ex\_OnBrickProgramPointer.nxc

This is an example of how to use the [OnBrickProgramPointer](#) function.

```
x = OnBrickProgramPointer();
```

### 9.474 ex\_onfwd.nxc

This is an example of how to use the [OnFwd](#) function.

```
OnFwd(OUT_A, 75);
```

### 9.475 ex\_onfwdex.nxc

This is an example of how to use the [OnFwdEx](#) function.

```
OnFwdEx(OUT_A, 75, RESET_NONE);
```

### 9.476 ex\_onfwdreg.nxc

This is an example of how to use the [OnFwdReg](#) function.

```
OnFwdReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

### 9.477 ex\_onfwdregex.nxc

This is an example of how to use the [OnFwdRegEx](#) function.

```
OnFwdRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

### 9.478 ex\_onfwdregexpid.nxc

This is an example of how to use the [OnFwdRegExPID](#) function.

```
OnFwdRegExPID(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE, 30, 50, 90);
```

### 9.479 ex\_onfwdregpid.nxc

This is an example of how to use the [OnFwdRegPID](#) function.

```
OnFwdRegPID(OUT_A, 75, OUT_REGMODE_SPEED, 30, 50, 90); // regulate speed
```

### 9.480 ex\_onfwdsync.nxc

This is an example of how to use the [OnFwdSync](#) function.

```
OnFwdSync(OUT_AB, 75, -100); // spin right
```

### 9.481 ex\_onfwdsyncex.nxc

This is an example of how to use the [OnFwdSyncEx](#) function.

```
OnFwdSyncEx(OUT_AB, 75, 0, RESET_NONE);
```

### 9.482 ex\_onfwdsyncexpid.nxc

This is an example of how to use the [OnFwdSyncExPID](#) function.

```
OnFwdSyncExPID(OUT_AB, 75, 0, RESET_NONE, 30, 50, 90);
```

### 9.483 ex\_onfwdsyncpid.nxc

This is an example of how to use the [OnFwdSyncPID](#) function.

```
task main()
{
  OnFwdSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin right
  Wait(SEC_5);
}
```

### 9.484 ex\_onrev.nxc

This is an example of how to use the [OnRev](#) function.

```
OnRev(OUT_A, 75);
```

### 9.485 ex\_onrevex.nxc

This is an example of how to use the [OnRevEx](#) function.

```
OnRevEx(OUT_A, 75, RESET_NONE);
```

**9.486 ex\_onrevreg.nxc**

This is an example of how to use the [OnRevReg](#) function.

```
OnRevReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

**9.487 ex\_onrevregex.nxc**

This is an example of how to use the [OnRevRegEx](#) function.

```
OnRevRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

**9.488 ex\_onrevregexpid.nxc**

This is an example of how to use the [OnRevRegExPID](#) function.

```
OnRevRegExPID(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE, 30, 50, 90);
```

**9.489 ex\_onrevregpid.nxc**

This is an example of how to use the [OnRevRegPID](#) function.

```
OnRevRegPID(OUT_A, 75, OUT_REGMODE_SPEED, 30, 50, 90); // regulate speed
```

**9.490 ex\_onrevsync.nxc**

This is an example of how to use the [OnRevSync](#) function.

```
OnRevSync(OUT_AB, 75, -100); // spin left
```

**9.491 ex\_onrevsyncex.nxc**

This is an example of how to use the [OnRevSyncEx](#) function.

```
OnRevSyncEx(OUT_AB, 75, -100, RESET_NONE); // spin left
```

**9.492 ex\_onrevsyncexpid.nxc**

This is an example of how to use the [OnRevSyncExPID](#) function.

```
OnRevSyncExPID(OUT_AB, 75, -100, RESET_NONE, 30, 50, 90); // spin left
```

### 9.493 ex\_onrevsyncpid.nxc

This is an example of how to use the `OnRevSyncPID` function.

```
task main()
{
  OnRevSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin left
  Wait(SEC_5);
}
```

### 9.494 ex\_OpenFileAppend.nxc

This is an example of how to use the `OpenFileAppend` function.

```
result = OpenFileAppend("data.txt", fsize, handle);
```

### 9.495 ex\_OpenFileRead.nxc

This is an example of how to use the `OpenFileRead` function.

```
result = OpenFileRead("data.txt", fsize, handle);
```

### 9.496 ex\_OpenFileReadLinear.nxc

This is an example of how to use the `OpenFileReadLinear` function.

```
result = OpenFileReadLinear("data.txt", fsize, handle);
```

### 9.497 ex\_PFMate.nxc

This is an example of how to use the `PFMateSend`, `PFMateSendRaw`, `SetSensor-Lowspeed`, and `Wait` functions.

```
task main()
{
  SetSensorLowspeed(S1); // PFMate is an i2c device
  // motor a forward full speed, motor b reverse full speed
  bool result = PFMateSend(S1, MS_ADDR_PFMATE, PFMATE_CHANNEL_1,
    PFMATE_MOTORS_BOTH, PF_CMD_FWD, 7, PF_CMD_REV, 7);
  Wait(SEC_5);
  byte b1, b2;
  b1 = 0xFF;
}
```

```
b2 = 0x00;
result = PFMateSendRaw(S1, MS_ADDR_PFMATE, PFMATE_CHANNEL_1, b1, b2);
Wait(SEC_5);
}
```

### 9.498 ex\_PlayFile.nxc

This is an example of how to use the [PlayFile](#) function.

```
PlayFile("startup.rso");
```

### 9.499 ex\_PlayFileEx.nxc

This is an example of how to use the [PlayFileEx](#) function.

```
PlayFileEx("startup.rso", 3, true);
```

### 9.500 ex\_playsound.nxc

This is an example of how to use the [PlaySound](#) function.

```
task main()
{
    PlaySound(SOUND_UP);
    PlaySound(SOUND_DOWN);
    Wait(SEC_1);
    PlaySound(SOUND_LOW_BEEP);
    Wait(MS_500);
    PlaySound(SOUND_FAST_UP);
}
```

### 9.501 ex\_PlayTone.nxc

This is an example of how to use the [PlayTone](#) function.

```
PlayTone(440, 500);    // Play 'A' for one half second
```

### 9.502 ex\_PlayToneEx.nxc

This is an example of how to use the [PlayToneEx](#) function.

```
PlayToneEx(440, 500, 2, false);
```

### 9.503 ex\_playtones.nxc

This is an example of how to use the [PlayTones](#) function along with the [Tone](#) structure.

```
Tone sweepUp[] = {
  TONE_C4, MS_50,
  TONE_E4, MS_50,
  TONE_G4, MS_50,
  TONE_C5, MS_50,
  TONE_E5, MS_50,
  TONE_G5, MS_50,
  TONE_C6, MS_200
};

task main()
{
  PlayTones(sweepUp);
  Wait(SEC_1);
}
```

### 9.504 ex\_PointOut.nxc

This is an example of how to use the [PointOut](#) function.

```
PointOut(40, 40);
```

### 9.505 ex\_PolyOut.nxc

This is an example of how to use the [PolyOut](#) function.

```
LocationType myPoints[] = {16,16, 8,40, 32,52, 20,36, 52,36, 56,52, 64,32, 44,20,
  24,20};

task main()
{
  PolyOut(myPoints, false);
  Wait(SEC_2);
  ClearScreen();
  for(int i=0;i<10;i++) {
    PolyOut(myPoints, DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE);
    Wait(SEC_1);
  }
  PolyOut(myPoints, true|DRAW_OPT_FILL_SHAPE);
  Wait(SEC_2);
  ClearScreen();
  for (int i=0;i<100;i++) {
    PolyOut(myPoints, DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE);
    Wait(MS_100);
  }
  Wait(SEC_1);
}
```

```
}
```

### 9.506 ex\_Pos.nxc

This is an example of how to use the [Pos](#) and [NumOut](#) functions.

```
task main()
{
  string s1 = "hi there";
  string s2 = "the";
  NumOut(0, LCD_LINE1, Pos(s2, s1));
  while(true);
}
```

### 9.507 ex\_PosReg.nxc

This is an example of how to use the [PosRegEnable](#), [PosRegSetAngle](#), [PosRegAddAngle](#), [PosRegSetMax](#), [SetMotorRegulationTime](#), [SetMotorRegulationOptions](#), [MotorRegulationTime](#), [MotorRegulationOptions](#), [MotorMaxSpeed](#), and [MotorMaxAcceleration](#) functions.

```
task main()
{
  byte rt = MotorRegulationTime();
  SetMotorRegulationTime(MS_10);
  byte ro = MotorRegulationOptions();
  SetMotorRegulationOptions(OUT_REGOPTION_NO_SATURATION);
  PosRegSetMax(OUT_A, 75, 15);
  byte ms = MotorMaxSpeed(OUT_A);
  byte ma = MotorMaxAcceleration(OUT_A);
  PosRegEnable(OUT_A);
  PosRegSetAngle(OUT_A, 360);
  Wait(5000);
  PosRegAddAngle(OUT_A, 360);
  Wait(5000);
}
```

### 9.508 ex\_pow.nxc

This is an example of how to use the [pow](#) function.

```
y = pow(x, 3);
```

### 9.509 ex\_PowerDown.nxc

This is an example of how to use the [PowerDown](#) functions.

```
PowerDown();
```

### 9.510 ex\_Precedes.nxc

This is an example of how to use the [Precedes](#) statement.

```
Precedes(moving, drawing, playing);
```

### 9.511 ex\_printf.nxc

This is an example of how to use the [printf](#) function.

```
printf("value = %d", value);
```

### 9.512 ex\_proto.nxc

This is an example of how to use the [SensorHTProtoAnalog](#), [ReadSensorHTProtoAllAnalog](#), [SetSensorHTProtoDigitalControl](#), [SetSensorHTProtoDigital](#), [SensorHTProtoDigital](#), and [SensorHTProtoDigitalControl](#) functions.

```
task main()
{
  SetSensorLowSpeed(S1);
  SetHTProtoDigitalControl(S1, 0xFF); // all outputs
  SetHTProtoDigital(S1, DIGI_PIN0|DIGI_PIN1|DIGI_PIN2);
  NumOut(0, LCD_LINE1, SensorHTProtoDigitalControl(S1));
  NumOut(0, LCD_LINE2, SensorHTProtoDigital(S1));
  NumOut(0, LCD_LINE3, SensorHTProtoAnalog(S1, HTPROTO_A0));
  int a0, a1, a2, a3, a4;
  ReadSensorHTProtoAllAnalog(S1, a0, a1, a2, a3, a4);
  NumOut(0, LCD_LINE4, a0);
  NumOut(0, LCD_LINE5, a1);
  NumOut(0, LCD_LINE6, a2);
  NumOut(0, LCD_LINE7, a3);
  NumOut(0, LCD_LINE8, a4);
  Wait(SEC_5);
}
```

### 9.513 ex\_PSPNXAnalog.nxc

This is an example of how to use the [PSPNXAnalog](#) function.

```
char result = PSPNXAnalog(S1, MS_ADDR_PSPNX);
```

### 9.514 ex\_PSPNxDigital.nxc

This is an example of how to use the [PSPNxDigital](#) function.

```
char result = PSPNxDigital(S1, MS_ADDR_PSPNX);
```

### 9.515 ex\_putc.nxc

This is an example of how to use the [putc](#) function.

```
putc(ch, handle);
```

### 9.516 ex\_rand.nxc

This is an example of how to use the [rand](#) function.

```
unsigned long x = rand(); // 0..RAND_MAX
```

### 9.517 ex\_Random.nxc

This is an example of how to use the [Random](#) function.

```
int x = Random(); // signed int between -32767..32767
unsigned i = Random(100); // 0..99

int ending = 4000, starting = 1000;
unsigned int j = Random(ending-starting)+starting; // 1000..3999
```

### 9.518 ex\_Read.nxc

This is an example of how to use the [Read](#) function.

```
result = Read(handle, value);
```

### 9.519 ex\_ReadButtonEx.nxc

This is an example of how to use the [ReadButtonEx](#) function.

```
ReadButtonEx(BTN1, true, pressed, count);
```

### 9.520 ex\_ReadBytes.nxc

This is an example of how to use the [ReadBytes](#) function.

```
result = ReadBytes(handle, len, buffer);
```

### 9.521 ex\_readi2cregister.nxc

This is an example of how to use the [ReadI2CRegister](#) function.

```
char result = ReadI2CRegister(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, out);
```

### 9.522 ex\_ReadLn.nxc

This is an example of how to use the [ReadLn](#) function.

```
result = ReadLn(handle, value);
```

### 9.523 ex\_ReadNRLinkBytes.nxc

This is an example of how to use the [ReadNRLinkBytes](#) function.

```
bool result = ReadNRLinkBytes(S1, MS_ADDR_NRLINK, data);
```

### 9.524 ex\_ReadSensorColorEx.nxc

This is an example of how to use the [ReadSensorColorEx](#) function.

```
unsigned int rawData[], normData[];
int scaledData[];
int cval;
int result = ReadSensorColorEx(S1, cval, rawData, normData, scaledData);
```

### 9.525 ex\_ReadSensorColorRaw.nxc

This is an example of how to use the [ReadSensorColorRaw](#) function.

```
unsigned int rawData[];
int result = ReadSensorColorRaw(S1, rawData);
```

### 9.526 ex\_ReadSensorEMeter.nxc

This is an example of how to use the [ReadSensorEMeter](#) function.

```
float vIn, aIn, vOut, aOut, wIn, wOut;
int joules;

char result = ReadSensorEMeter(S1, vIn, aIn, vOut, aOut, joules, wIn, wOut);
```

### 9.527 ex\_ReadSensorHTAccel.nxc

This is an example of how to use the [ReadSensorHTAccel](#) function.

```
bVal = ReadSensorHTAccel(S1, x, y, z);
```

### 9.528 ex\_ReadSensorHTAngle.nxc

This is an example of how to use the [ReadSensorHTAngle](#) function.

```
task main()
{
  int angle, rpm;
  long accangle;
  SetSensorLowspeed(S4);
  while (true) {
    ClearScreen();
    ReadSensorHTAngle(S4, angle, accangle, rpm);
    NumOut(0, LCD_LINE1, angle);
    NumOut(0, LCD_LINE2, accangle);
    NumOut(0, LCD_LINE3, rpm);
    Wait(MS_500);
  }
}
```

### 9.529 ex\_ReadSensorHTBarometric.nxc

This is an example of how to use the [ReadSensorHTBarometric](#) function.

```
task main()
{
  SetSensorLowspeed(S3);
  int temp;
  unsigned int press;
  while (true)
  {
    ReadSensorHTBarometric(S3, temp, press);
    NumOut(0, LCD_LINE1, temp);
  }
}
```

```
TextOut(40, LCD_LINE1, " 1/10ths C");
NumOut(0, LCD_LINE2, press);
float tc = temp / 10.0;
TextOut(0, LCD_LINE3, FormatNum("%5.2f C", tc));
TextOut(0, LCD_LINE4, FormatNum("%5.2f F", tc*9/5+32));
TextOut(0, LCD_LINE5, FormatNum("%3.3f inHg", press/1000.0));
Wait(MS_20);
}
}
```

### 9.530 ex\_ReadSensorHTColor.nxc

This is an example of how to use the [ReadSensorHTColor](#) function.

```
bVal = ReadSensorHTColor(S1, c, r, g, b);
```

### 9.531 ex\_ReadSensorHTColor2Active.nxc

This is an example of how to use the [ReadSensorHTColor2Active](#) function.

```
byte cnum, red, green, blue, white;
bool result = ReadSensorHTColor2Active(S1, cnum, red, green, blue, white);
```

### 9.532 ex\_ReadSensorHTIRReceiver.nxc

This is an example of how to use the [ReadSensorHTIRReceiver](#) function.

```
char pfdata[];
bool result = ReadSensorHTIRReceiver(S1, pfdata);
```

### 9.533 ex\_ReadSensorHTIRReceiverEx.nxc

This is an example of how to use the [ReadSensorHTIRReceiverEx](#) function.

```
char pfchar;
bool result = ReadSensorHTIRReceiverEx(S1, HT_CH1_A, pfchar);
```

### 9.534 ex\_ReadSensorHTIRSeeker.nxc

This is an example of how to use the [ReadSensorHTIRSeeker](#) function.

```
bVal = ReadSensorHTIRSeeker(port, dir, s1, s3, s5, s7, s9);
```

### 9.535 ex\_ReadSensorHTIRSeeker2AC.nxc

This is an example of how to use the [ReadSensorHTIRSeeker2AC](#) function.

```
byte s1, s3, s5, s7, s9;
bool result = ReadSensorHTIRSeeker2AC(S1, dir, s1, s3, s5, s7, s9);
```

### 9.536 ex\_ReadSensorHTIRSeeker2DC.nxc

This is an example of how to use the [ReadSensorHTIRSeeker2DC](#) function.

```
byte s1, s3, s5, s7, s9, avg;
bool result = ReadSensorHTIRSeeker2DC(S1, dir, s1, s3, s5, s7, s9, avg);
```

### 9.537 ex\_ReadSensorHTNormalizedColor.nxc

This is an example of how to use the [ReadSensorHTNormalizedColor](#) function.

```
bVal = ReadSensorHTNormalizedColor(S1, c, r, g, b);
```

### 9.538 ex\_ReadSensorHTNormalizedColor2Active.nxc

This is an example of how to use the [ReadSensorHTNormalizedColor2Active](#) function.

```
byte cidx, red, green, blue;
bool result = ReadSensorHTNormalizedColor2Active(S1, cidx, red, green, blue);
```

### 9.539 ex\_ReadSensorHTRawColor.nxc

This is an example of how to use the [ReadSensorHTRawColor](#) function.

```
bVal = ReadSensorHTRawColor(S1, r, g, b);
```

### 9.540 ex\_ReadSensorHTRawColor2.nxc

This is an example of how to use the [ReadSensorHTRawColor2](#) function.

```
unsigned int red, green, blue, white;
bool result = ReadSensorHTRawColor2(S1, red, green, blue, white);
```

### 9.541 ex\_ReadSensorHTTouchMultiplexer.nxc

This is an example of how to use the [ReadSensorHTTouchMultiplexer](#) function.

```
task main()
{
  byte t1, t2, t3, t4;
  SetSensorTouch(S1);
  while (true) {
    ReadSensorHTTouchMultiplexer(S1, t1, t2, t3, t4);
    if (t1)
      TextOut(0, LCD_LINE1, "1 pressed" );
    else
      TextOut(0, LCD_LINE1, "      " );
    if (t2)
      TextOut(0, LCD_LINE2, "2 pressed" );
    else
      TextOut(0, LCD_LINE2, "      " );
    if (t3)
      TextOut(0, LCD_LINE3, "3 pressed" );
    else
      TextOut(0, LCD_LINE3, "      " );
    if (t4)
      TextOut(0, LCD_LINE4, "4 pressed" );
    else
      TextOut(0, LCD_LINE4, "      " );
  }
}
```

### 9.542 ex\_ReadSensorMSAccel.nxc

This is an example of how to use the [ReadSensorMSAccel](#) function.

```
int x, y, z;
bool result = ReadSensorMSAccel(S1, MS_ADDR_ACCLNX, x, y, z);
```

### 9.543 ex\_ReadSensorMSPlayStation.nxc

This is an example of how to use the [ReadSensorMSPlayStation](#) function.

```
task main()
{
  SetSensorLowspeed(S1);
  PSPNxAnalog(S1, MS_ADDR_PSPNX);
  byte btnset1, btnset2, xleft, yleft, xright, yright;
  while (true)
  {
    ClearScreen();
    bool result = ReadSensorMSPlayStation(S1, MS_ADDR_PSPNX,
      btnset1, btnset2, xleft, yleft, xright, yright);
    if (result)
```

```

{
  NumOut( 0, LCD_LINE1, xleft);
  NumOut(40, LCD_LINE1, yleft);
  NumOut( 0, LCD_LINE2, xright);
  NumOut(40, LCD_LINE2, yright);
  // button set 1
  if (!(btnset1 & PSP_BTNSET1_DOWN))
    TextOut( 0, LCD_LINE3, "D");
  if (!(btnset1 & PSP_BTNSET1_UP))
    TextOut( 8, LCD_LINE3, "U");
  if (!(btnset1 & PSP_BTNSET1_LEFT))
    TextOut(16, LCD_LINE3, "L");
  if (!(btnset1 & PSP_BTNSET1_RIGHT))
    TextOut(24, LCD_LINE3, "R");
  if (!(btnset1 & PSP_BTNSET1_L3))
    TextOut(32, LCD_LINE3, "l");
  if (!(btnset1 & PSP_BTNSET1_R3))
    TextOut(40, LCD_LINE3, "r");
  // button set 2
  if (!(btnset2 & PSP_BTNSET2_SQUARE))
    TextOut( 0, LCD_LINE4, "S");
  if (!(btnset2 & PSP_BTNSET2_CROSS))
    TextOut( 8, LCD_LINE4, "X");
  if (!(btnset2 & PSP_BTNSET2_CIRCLE))
    TextOut(16, LCD_LINE4, "C");
  if (!(btnset2 & PSP_BTNSET2_TRIANGLE))
    TextOut(24, LCD_LINE4, "T");
  if (!(btnset2 & PSP_BTNSET2_R1))
    TextOut(32, LCD_LINE4, "r");
  if (!(btnset2 & PSP_BTNSET2_L1))
    TextOut(40, LCD_LINE4, "l");
  if (!(btnset2 & PSP_BTNSET2_R2))
    TextOut(48, LCD_LINE4, "R");
  if (!(btnset2 & PSP_BTNSET2_L2))
    TextOut(56, LCD_LINE4, "L");
  Wait(MS_500);
}
}
}

```

### 9.544 ex\_ReadSensorMSRTClock.nxc

This is an example of how to use the [ReadSensorMSRTClock](#) function.

```
ReadSensorMSRTClock(S1, ss, mm, hh, dow, dd, mon, yy);
```

### 9.545 ex\_ReadSensorMSTilt.nxc

This is an example of how to use the [ReadSensorMSTilt](#) function.

```
byte x, y, z;
bool result = ReadSensorMSTilt(S1, MS_ADDR_ACCLNX, x, y, z);
```

### 9.546 ex\_ReadSensorUSEx.nxc

This is an example of how to use the [ReadSensorUSEx](#) function.

```
byte values[];  
char result = ReadSensorUSEx(S1, values);
```

### 9.547 ex\_RebootInFirmwareMode.nxc

This is an example of how to use the [RebootInFirmwareMode](#) functions.

```
RebootInFirmwareMode();
```

### 9.548 ex\_ReceiveMessage.nxc

This is an example of how to use the [ReceiveMessage](#) function.

```
x = RecieveMessage(MAILBOX1, true, buffer);
```

### 9.549 ex\_ReceiveRemoteBool.nxc

This is an example of how to use the [ReceiveRemoteBool](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, bvalue);
```

### 9.550 ex\_ReceiveRemoteMessageEx.nxc

This is an example of how to use the [ReceiveRemoteMessageEx](#) function.

```
x = ReceiveRemoteMessageEx(MAILBOX1, true, strval, val, bval);
```

### 9.551 ex\_ReceiveRemoteNumber.nxc

This is an example of how to use the [ReceiveRemoteNumber](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, value);
```

### 9.552 ex\_ReceiveRemoteString.nxc

This is an example of how to use the [ReceiveRemoteString](#) function.

```
x = ReceiveRemoteString(queue, true, strval);
```

### 9.553 ex\_RechargeableBattery.nxc

This is an example of how to use the [RechargeableBattery](#) function.

```
x = RechargeableBattery();
```

### 9.554 ex\_RectOut.nxc

This is an example of how to use the [RectOut](#) function.

```
RectOut(40, 40, 30, 10);
```

### 9.555 ex\_reladdressof.nxc

This is an example of how to use the [reladdressOf](#) function.

```
task main()
{
    char x[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    unsigned long ptr = reladdressOf(x);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    IOMapReadByIDType args;
    args.ModuleID = CommandModuleID;
    args.Offset = CommandOffsetMemoryPool+ptr;
    args.Count = 10;
    SysIOMapReadByID(args);
    NumOut(0, LCD_LINE2, x[0]);
    NumOut(20, LCD_LINE2, x[1]);
    NumOut(40, LCD_LINE2, x[2]);
    NumOut(60, LCD_LINE2, x[3]);
    NumOut(80, LCD_LINE2, x[4]);
    NumOut(0, LCD_LINE3, args.Buffer[0]);
    NumOut(20, LCD_LINE3, args.Buffer[1]);
    NumOut(40, LCD_LINE3, args.Buffer[2]);
    NumOut(60, LCD_LINE3, args.Buffer[3]);
    NumOut(80, LCD_LINE3, args.Buffer[4]);
    args.Buffer++;
    args.Buffer *= 3;
    IOMapWriteByIDType a2;
```

```
a2.ModuleID = CommandModuleID;
a2.Offset   = CommandOffsetMemoryPool+ptr;
a2.Buffer   = args.Buffer;
SysIOMapWriteByID(a2);
NumOut( 0, LCD_LINE4, x[0]);
NumOut(20, LCD_LINE4, x[1]);
NumOut(40, LCD_LINE4, x[2]);
NumOut(60, LCD_LINE4, x[3]);
NumOut(80, LCD_LINE4, x[4]);
Wait(SEC_10);
}
```

### 9.556 ex\_Release.nxc

This is an example of how to use the [Release](#) function.

```
Acquire(motorMutex); // make sure we have exclusive access
// use the motors
Release(motorMutex); // release mutex for other tasks
```

### 9.557 ex\_RemoteBluetoothFactoryReset.nxc

This is an example of how to use the [RemoteBluetoothFactoryReset](#) function.

```
char result = RemoteBluetoothFactoryReset(CONN_HS1); // cannot be used over a blu
eooth connection
```

### 9.558 ex\_RemoteCloseFile.nxc

This is an example of how to use the [RemoteCloseFile](#) function.

```
char result = RemoteCloseFile(CONN_BT1, handle);
```

### 9.559 ex\_RemoteConnectionIdle.nxc

This is an example of how to use the [RemoteConnectionIdle](#) function.

```
bool result = RemoteConnectionIdle(CONN_BT1);
```

### 9.560 ex\_RemoteConnectionWrite.nxc

This is an example of how to use the [RemoteConnectionWrite](#) function.

```
char result = RemoteConnectionWrite(CONN_BT1, dataBuf);
```

### 9.561 ex\_RemoteDatalogRead.nxc

This is an example of how to use the [RemoteDatalogRead](#) function.

```
byte count;
byte data[];

char result = RemoteDatalogRead(CONN_BT1, true, count, data);
```

### 9.562 ex\_RemoteDatalogSetTimes.nxc

This is an example of how to use the [RemoteDatalogSetTimes](#) function.

```
char result = RemoteDatalogSetTimes(CONN_BT1, 1000);
```

### 9.563 ex\_RemoteDeleteFile.nxc

This is an example of how to use the [RemoteDeleteFile](#) function.

```
char result = RemoteDeleteFile(CONN_BT1, "test.dat");
```

### 9.564 ex\_RemoteDeleteUserFlash.nxc

This is an example of how to use the [RemoteDeleteUserFlash](#) function.

```
char result = RemoteDeleteUserFlash(CONN_BT1);
```

### 9.565 ex\_RemoteFindFirstFile.nxc

This is an example of how to use the [RemoteFindFirstFile](#) function.

```
long size;
string name;
byte handle;

char result = RemoteFindFirstFile(CONN_BT1, "*.rxn", handle, name, size);
```

### 9.566 ex\_RemoteFindNextFile.nxc

This is an example of how to use the [RemoteFindNextFile](#) function.

```
byte handle;
string name;
long size;

char result = RemoteFindNextFile(CONN_BT1, handle, name, size);
```

### 9.567 ex\_RemoteGetBatteryLevel.nxc

This is an example of how to use the [RemoteGetBatteryLevel](#) function.

```
int blevel;

char result = RemoteGetBatteryLevel(CONN_BT1, blevel);
```

### 9.568 ex\_RemoteGetBluetoothAddress.nxc

This is an example of how to use the [RemoteGetBluetoothAddress](#) function.

```
byte btaddr[];

char result = RemoteGetBluetoothAddress(CONN_BT1, btaddr);
```

### 9.569 ex\_RemoteGetConnectionCount.nxc

This is an example of how to use the [RemoteGetConnectionCount](#) function.

```
byte cnt;

char result = RemoteGetConnectionCount(CONN_BT1, cnt);
```

### 9.570 ex\_RemoteGetConnectionName.nxc

This is an example of how to use the [RemoteGetConnectionName](#) function.

```
string name;
byte idx = 1;

char result = RemoteGetConnectionName(CONN_BT1, idx, name);
```

### 9.571 ex\_RemoteGetContactCount.nxc

This is an example of how to use the [RemoteGetContactCount](#) function.

```
byte cnt;

char result = RemoteGetContactCount(CONN_BT1, cnt);
```

### 9.572 ex\_RemoteGetContactName.nxc

This is an example of how to use the [RemoteGetContactName](#) function.

```
string name;
byte idx = 1;

char result = RemoteGetContactName(CONN_BT1, idx, name);
```

### 9.573 ex\_RemoteGetCurrentProgramName.nxc

This is an example of how to use the [RemoteGetCurrentProgramName](#) function.

```
string name;

char result = RemoteGetCurrentProgramName(CONN_BT1, name);
```

### 9.574 ex\_RemoteGetDeviceInfo.nxc

This is an example of how to use the [RemoteGetDeviceInfo](#) function.

```
string name;
byte btaddr[], btsignal[];
long freemem;

char result = RemoteGetDeviceInfo(CONN_BT1, name, btaddr, btsignal, freemem);
```

### 9.575 ex\_RemoteGetFirmwareVersion.nxc

This is an example of how to use the [RemoteGetFirmwareVersion](#) function.

```
byte pmin, pmaj, fmin, fmaj;

char result = RemoteGetFirmwareVersion(CONN_BT1, pmin, pmaj, fmin, fmaj);
```

### 9.576 ex\_RemoteGetInputValues.nxc

This is an example of how to use the [RemoteGetInputValues](#) function.

```
InputValuesType params;

char result = RemoteGetInputValues(CONN_BT1, params);
```

### 9.577 ex\_RemoteGetOutputState.nxc

This is an example of how to use the [RemoteGetOutputState](#) function.

```
OutputStateType params;  
char result = RemoteGetOutputState(CONN_BT1, params);
```

### 9.578 ex\_RemoteGetProperty.nxc

This is an example of how to use the [RemoteGetProperty](#) function.

```
byte value;  
char result = RemoteGetProperty(CONN_BT1, RC_PROP_SOUND_LEVEL, value);
```

### 9.579 ex\_RemoteIOMapRead.nxc

This is an example of how to use the [RemoteIOMapRead](#) function.

```
int numbytes = 1;  
byte data[];  
char result = RemoteIOMapRead(CONN_BT1, CommandModuleID, CommandOffsetProgStatus,  
    numbytes, data);
```

### 9.580 ex\_RemoteIOMapWriteBytes.nxc

This is an example of how to use the [RemoteIOMapWriteBytes](#) function.

```
byte data[] = {1};  
char result = RemoteIOMapWriteBytes(CONN_BT1, CommandModuleID,  
    CommandOffsetProgStatus, data);
```

### 9.581 ex\_RemoteIOMapWriteValue.nxc

This is an example of how to use the [RemoteIOMapWriteValue](#) function.

```
byte value;  
char result = RemoteIOMapWriteValue(CONN_BT1, CommandModuleID,  
    CommandOffsetProgStatus, value);
```

### 9.582 ex\_RemoteKeepAlive.nxc

This is an example of how to use the [RemoteKeepAlive](#) function.

```
x = RemoteKeepAlive(1);
```

### 9.583 ex\_RemoteLowspeedGetStatus.nxc

This is an example of how to use the [RemoteLowspeedGetStatus](#) function.

```
byte value;  
char result = RemoteLowspeedGetStatus(CONN_BT1, value);
```

### 9.584 ex\_RemoteLowspeedRead.nxc

This is an example of how to use the [RemoteLowspeedRead](#) function.

```
byte port = S1;  
byte bread;  
byte data[];  
char result = RemoteLowspeedRead(CONN_BT1, port, bread, data);
```

### 9.585 ex\_RemoteLowspeedWrite.nxc

This is an example of how to use the [RemoteLowspeedWrite](#) function.

```
byte port = S1;  
byte txlen = 2;  
byte rxlen = 8;  
byte data[] = {0x02, 0x00};  
char result = RemoteLowspeedWrite(CONN_BT1, port, txlen, rxlen, data);
```

### 9.586 ex\_RemoteMessageRead.nxc

This is an example of how to use the [RemoteMessageRead](#) function.

```
x = RemoteMessageRead(1, 5);
```

### 9.587 ex\_RemoteMessageWrite.nxc

This is an example of how to use the [RemoteMessageWrite](#) function.

```
x = RemoteMessageWrite(1, 5, "test");
```

### 9.588 ex\_RemoteOpenAppendData.nxc

This is an example of how to use the [RemoteOpenAppendData](#) function.

```
byte handle;  
long size;  
  
char result = RemoteOpenAppendData(CONN_BT1, "test.dat", handle, size);
```

### 9.589 ex\_RemoteOpenRead.nxc

This is an example of how to use the [RemoteOpenRead](#) function.

```
byte handle;  
long size;  
  
char result = RemoteOpenRead(CONN_BT1, "test.dat", handle, size);
```

### 9.590 ex\_RemoteOpenWrite.nxc

This is an example of how to use the [RemoteOpenWrite](#) function.

```
byte handle;  
long size = 1024;  
  
char result = RemoteOpenWrite(CONN_BT1, "test.dat", size, handle);
```

### 9.591 ex\_RemoteOpenWriteData.nxc

This is an example of how to use the [RemoteOpenWriteData](#) function.

```
byte handle;  
long size = 1024;  
  
char result = RemoteOpenWriteData(CONN_BT1, "test.dat", size, handle);
```

### 9.592 ex\_RemoteOpenWriteLinear.nxc

This is an example of how to use the [RemoteOpenWriteLinear](#) function.

```
byte handle;
long size = 1024;

char result = RemoteOpenWriteLinear(CONN_BT1, "test.rxe", size, handle);
```

### 9.593 ex\_RemotePlaySoundFile.nxc

This is an example of how to use the [RemotePlaySoundFile](#) function.

```
x = RemotePlaySoundFile(1, "click.rso", false);
```

### 9.594 ex\_RemotePlayTone.nxc

This is an example of how to use the [RemotePlayTone](#) function.

```
x = RemotePlayTone(1, 440, 1000);
```

### 9.595 ex\_RemotePollCommand.nxc

This is an example of how to use the [RemotePollCommand](#) function.

```
byte len;
byte data[];

char result = RemotePollCommand(CONN_BT1, 0, len, data);
```

### 9.596 ex\_RemotePollCommandLength.nxc

This is an example of how to use the [RemotePollCommandLength](#) function.

```
byte len;

char result = RemotePollCommandLength(CONN_BT1, 0, len);
```

### 9.597 ex\_RemoteRead.nxc

This is an example of how to use the [RemoteRead](#) function.

```
byte handle;
int numbytes = 10;
byte data[];

char result = RemoteRead(CONN_BT1, handle, numbytes, data);
```

### 9.598 ex\_RemoteRenameFile.nxc

This is an example of how to use the [RemoteRenameFile](#) function.

```
char result = RemoteRenameFile(CONN_BT1, "test.dat", "test2.dat");
```

### 9.599 ex\_RemoteResetMotorPosition.nxc

This is an example of how to use the [RemoteResetMotorPosition](#) function.

```
x = RemoteResetMotorPosition(1, OUT_A, true);
```

### 9.600 ex\_RemoteResetScaledValue.nxc

This is an example of how to use the [RemoteResetScaledValue](#) function.

```
x = RemoteResetScaledValue(1, S1);
```

### 9.601 ex\_RemoteResetTachoCount.nxc

This is an example of how to use the [RemoteResetTachoCount](#) function.

```
char result = RemoteResetTachoCount(CONN_BT1, OUT_A);
```

### 9.602 ex\_RemoteSetBrickName.nxc

This is an example of how to use the [RemoteSetBrickName](#) function.

```
char result = RemoteSetBrickName(CONN_HS1, "NEWNAME");
```

### 9.603 ex\_RemoteSetInputMode.nxc

This is an example of how to use the [RemoteSetInputMode](#) function.

```
x = RemoteSetInputMode(1, S1, SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW);
```

### 9.604 ex\_RemoteSetOutputState.nxc

This is an example of how to use the [RemoteSetOutputState](#) function.

```
x = RemoteSetOutputState(1, OUT_A, 75, OUT_MODE_MOTORON, OUT_REGMODE_IDLE, 0,
    OUT_RUNSTATE_RUNNING, 0);
```

### 9.605 ex\_RemoteSetProperty.nxc

This is an example of how to use the [RemoteSetProperty](#) function.

```
byte value = 3;
char result = RemoteSetProperty(CONN_BT1, RC_PROP_SOUND_LEVEL, value);
```

### 9.606 ex\_RemoteStartProgram.nxc

This is an example of how to use the [RemoteStartProgram](#) function.

```
x = RemoteStartProgram(1, "myprog.rxe");
```

### 9.607 ex\_RemoteStopProgram.nxc

This is an example of how to use the [RemoteStopProgram](#) function.

```
x = RemoteStopProgram(1);
```

### 9.608 ex\_RemoteStopSound.nxc

This is an example of how to use the [RemoteStopSound](#) function.

```
x = RemoteStopSound(1);
```

### 9.609 ex\_RemoteWrite.nxc

This is an example of how to use the [RemoteWrite](#) function.

```
byte handle;
int numbytes = 10;
byte data[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

char result = RemoteWrite(CONN_BT1, handle, numbytes, data);
```

### 9.610 ex\_remove.nxc

This is an example of how to use the [remove](#) function.

```
result = remove("data.txt");
```

### 9.611 ex\_rename.nxc

This is an example of how to use the [rename](#) function.

```
result = rename("data.txt", "mydata.txt");
```

### 9.612 ex\_RenameFile.nxc

This is an example of how to use the [RenameFile](#) function.

```
result = RenameFile("data.txt", "mydata.txt");
```

### 9.613 ex\_resetalltachocounts.nxc

This is an example of how to use the [ResetAllTachoCounts](#) function.

```
ResetAllTachoCounts(OUT_AB);
```

### 9.614 ex\_resetblocktachocount.nxc

This is an example of how to use the [ResetBlockTachoCount](#) function.

```
ResetBlockTachoCount(OUT_AB);
```

### 9.615 ex\_resetrotationcount.nxc

This is an example of how to use the [ResetRotationCount](#) function.

```
ResetRotationCount(OUT_AB);
```

### 9.616 ex\_ResetScreen.nxc

This is an example of how to use the [ResetScreen](#) function.

```
ResetScreen();
```

### 9.617 ex\_ResetSensor.nxc

This is an example of how to use the [ResetSensor](#) function.

```
ResetSensor(x); // x = S1
```

### 9.618 ex\_ResetSensorHTAngle.nxc

This is an example of how to use the [ResetSensorHTAngle](#) function.

```
task main ()
{
  SetSensorLowspeed(S4);
  ResetSensorHTAngle(S4, HTANGLE_MODE_RESET);
  Wait(50);
}
```

### 9.619 ex\_ResetSleepTimer.nxc

This is an example of how to use the [ResetSleepTimer](#) function.

```
ResetSleepTimer();
```

### 9.620 ex\_resettachocount.nxc

This is an example of how to use the [ResetTachoCount](#) function.

```
ResetTachoCount(OUT_AB);
```

### 9.621 ex\_resizefile.nxc

This is an example of how to use the [ResizeFile](#) function.

```
result = ResizeFile("data.txt", 2048);
```

### 9.622 ex\_ResolveHandle.nxc

This is an example of how to use the [ResolveHandle](#) function.

```
result = ResolveHandle("data.txt", handle, bCanWrite);
```

### 9.623 ex\_rewind.nxc

This is an example of how to use the [rewind](#) function.

```
rewind(handle);
```

### 9.624 ex\_RFIDInit.nxc

This is an example of how to use the [RFIDInit](#) function.

```
bool result = RFIDInit(S1);
```

### 9.625 ex\_RFIDMode.nxc

This is an example of how to use the [RFIDMode](#) function.

```
bool result = RFIDMode(S1, RFID_MODE_CONTINUOUS);
```

### 9.626 ex\_RFIDRead.nxc

This is an example of how to use the [RFIDRead](#) function.

```
byte output[];  
bool result = RFIDRead(S1, output);
```

### 9.627 ex\_RFIDReadContinuous.nxc

This is an example of how to use the [RFIDReadContinuous](#) function.

```
byte output[];  
bool result = RFIDReadContinuous(S1, output);
```

### 9.628 ex\_RFIDReadSingle.nxc

This is an example of how to use the [RFIDReadSingle](#) function.

```
byte output[];
bool result = RFIDReadSingle(S1, output);
```

### 9.629 ex\_RFIDStatus.nxc

This is an example of how to use the [RFIDStatus](#) function.

```
byte result = RFIDStatus(S1);
```

### 9.630 ex\_RFIDStop.nxc

This is an example of how to use the [RFIDStop](#) function.

```
bool result = RFIDStop(S1);
```

### 9.631 ex\_rightstr.nxc

This is an example of how to use the [RightStr](#) function.

```
task main()
{
  string s = "Now is the winter of our discontent";
  TextOut(0, LCD_LINE1, RightStr(s, 12));
  Wait(SEC_4);
}
```

### 9.632 ex\_rotatemotor.nxc

This is an example of how to use the [RotateMotor](#) function.

```
RotateMotor(OUT_A, 75, 45); // forward 45 degrees
RotateMotor(OUT_A, -75, 45); // reverse 45 degrees
```

### 9.633 ex\_rotatemotorex.nxc

This is an example of how to use the [RotateMotorEx](#) function.

```
RotateMotorEx(OUT_AB, 75, 360, 50, true, true);
```

### 9.634 ex\_rotatemotorexpid.nxc

This is an example of how to use the [RotateMotorExPID](#) function.

```
RotateMotorExPID(OUT_AB, 75, 360, 50, true, true, 30, 50, 90);
```

### 9.635 ex\_rotatemotorpid.nxc

This is an example of how to use the [RotateMotorPID](#) function.

```
RotateMotorPID(OUT_A, 75, 45, 20, 40, 100);
```

### 9.636 ex\_RS485Receive.nxc

This is an example of how to use the [RS485Control](#), [RS485DataAvailable](#), [RS485Disable](#), [RS485Initialize](#), [RS485Enable](#), [UseRS485](#), [RS485Uart](#), [RS485Status](#), [RS485Read](#), [RS485ReadEx](#), [TextOut](#), and [Wait](#) functions.

```
// RS-485 receiver program
inline void WaitForMessageToBeSent()
{
    while(RS485SendingData())
        Wait(MS_1);
}

task main()
{
    byte mlen;
    string buffer;
    // configure the S4 port as RS485
    UseRS485();
    // make sure the RS485 system is turned on
    RS485Enable();
    // // initialize the UART to default values
    // RS485Initialize();
    // configure the UART (this is equivalent to RS485Initialize)
    RS485Uart(HS_BAUD_DEFAULT, HS_MODE_DEFAULT);

    Wait(MS_1); // make sure everything is turned on
    byte ACK[] = {1};
    while (true) {
        // wait for a message to arrive.

        // read the number of bytes message
        until(RS485DataAvailable() >= 5);

        // read the number of bytes
        RS485Read(buffer);
        long cnt = 0;
        UnflattenVar(buffer, cnt);
    }
}
```

```

    // send out ACK
    RS485Write(ACK);
    WaitForMessageToBeSent();

    // now wait for the real message
    until(RS485DataAvailable() >= cnt);

    // now read the actual message
    RS485ReadEx(buffer, cnt);
//    RS485Read(buffer);

    // send out ACK
    RS485Write(ACK);
    WaitForMessageToBeSent();

    // display message
    TextOut(0, LCD_LINE1, buffer);
}
}

```

### 9.637 ex\_RS485Send.nxc

This is an example of how to use the [RS485Control](#), [RS485Disable](#), [RS485Initialize](#), [RS485Enable](#), [UseRS485](#), [RS485Uart](#), [RS485Status](#), [RS485Write](#), [RS485SendingData](#), [SendRS485String](#), [SendRS485Bool](#), [SendRS485Number](#), [TextOut](#), and [Wait](#) functions.

```

// RS-485 sender program

inline void WaitForMessageToBeSent()
{
    while(RS485SendingData())
        Wait(MS_1);
}

task main()
{
    // configure the S4 port as RS485
    UseRS485();
    // make sure the RS485 system is turned on
    RS485Enable();
    // initialize the UART to default values
    // low level API function call (allows changing UART settings)
    RS485Uart(HS_BAUD_DEFAULT, HS_MODE_DEFAULT);
//    // hi level API function call
//    RS485Initialize();
    Wait(MS_1); // make sure everything gets turned on okay
    int i;
    byte buffer[];
    while (true) {
        string msg;
        msg = "goofy " + NumToStr(i);
    }
}

```

```
TextOut(0, LCD_LINE1, msg);

// send the # of bytes (5 bytes)
byte cnt = ArrayLen(msg);
SendRS485Number(cnt);
WaitForMessageToBeSent();

// wait for ACK from recipient
until(RS485DataAvailable());
RS485Read(buffer);

// now send the message
SendRS485String(msg);
WaitForMessageToBeSent();

// wait for ACK from recipient
until(RS485DataAvailable());
RS485Read(buffer);

i++;
}
// disable RS485 (not usually needed)
RS485Disable();
}
```

### 9.638 ex\_RunNRLinkMacro.nxc

This is an example of how to use the [RunNRLinkMacro](#) function.

```
char result = RunNRLinkMacro(S1, MS_ADDR_NRLINK, macro);
```

### 9.639 ex\_SendMessage.nxc

This is an example of how to use the [SendMessage](#) function.

```
x = SendMessage(MAILBOX1, data);
```

### 9.640 ex\_SendRemoteBool.nxc

This is an example of how to use the [SendRemoteBool](#) function.

```
x = SendRemoteBool(1, MAILBOX1, false);
```

### 9.641 ex\_SendRemoteNumber.nxc

This is an example of how to use the [SendRemoteNumber](#) function.

```
x = SendRemoteNumber(1, MAILBOX1, 123);
```

### 9.642 `ex_SendRemoteString.nxc`

This is an example of how to use the [SendRemoteString](#) function.

```
x = SendRemoteString(1, MAILBOX1, "hello world");
```

### 9.643 `ex_SendResponseBool.nxc`

This is an example of how to use the [SendResponseBool](#) function.

```
x = SendResponseBool(MAILBOX1, false);
```

### 9.644 `ex_SendResponseNumber.nxc`

This is an example of how to use the [SendResponseNumber](#) function.

```
x = SendResponseNumber(MAILBOX1, 123);
```

### 9.645 `ex_SendResponseString.nxc`

This is an example of how to use the [SendResponseString](#) function.

```
x = SendResponseString(MAILBOX1, "hello world");
```

### 9.646 `ex_Sensor.nxc`

This is an example of how to use the [Sensor](#) function.

```
x = Sensor(S1); // read sensor 1
```

### 9.647 `ex_SensorBoolean.nxc`

This is an example of how to use the [SensorBoolean](#) function.

```
x = SensorBoolean(S1);
```

**9.648 ex\_SensorDigiPinsDirection.nxc**

This is an example of how to use the [SensorDigiPinsDirection](#) function.

```
x = SensorDigiPinsDirection(S1);
```

**9.649 ex\_SensorDigiPinsOutputLevel.nxc**

This is an example of how to use the [SensorDigiPinsOutputLevel](#) function.

```
x = SensorDigiPinsOutputLevel(S1);
```

**9.650 ex\_SensorDigiPinsStatus.nxc**

This is an example of how to use the [SensorDigiPinsStatus](#) function.

```
x = SensorDigiPinsStatus(S1);
```

**9.651 ex\_SensorHTColorNum.nxc**

This is an example of how to use the [SensorHTColorNum](#) function.

```
x = SensorHTColorNum(S1);
```

**9.652 ex\_SensorHTCompass.nxc**

This is an example of how to use the [SensorHTCompass](#) function.

```
x = SensorHTCompass(S1);
```

**9.653 ex\_SensorHTEOPD.nxc**

This is an example of how to use the [SensorHTEOPD](#) function.

```
int val = SensorHTEOPD(S1);
```

### 9.654 ex\_SensorHTGyro.nxc

This is an example of how to use the [SensorHTGyro](#) function.

```
task main()
{
    int offset = 400;
    SetSensorHTGyro(S1);
    NumOut(0, LCD_LINE1, SensorHTGyro(S1, offset+5));
    Wait(SEC_9);
}
```

### 9.655 ex\_SensorHTIRSeeker2ACDir.nxc

This is an example of how to use the [SensorHTIRSeeker2ACDir](#) function.

```
int val = SensorHTIRSeeker2ACDir(S1);
```

### 9.656 ex\_SensorHTIRSeeker2Addr.nxc

This is an example of how to use the [SensorHTIRSeeker2Addr](#) function.

```
int val = SensorHTIRSeeker2Addr(S1, HTIR2_REG_DCAVG);
```

### 9.657 ex\_SensorHTIRSeeker2DCDir.nxc

This is an example of how to use the [SensorHTIRSeeker2DCDir](#) function.

```
int val = SensorHTIRSeeker2DCDir(S1);
```

### 9.658 ex\_SensorHTIRSeekerDir.nxc

This is an example of how to use the [SensorHTIRSeekerDir](#) function.

```
x = SensorHTIRSeekerDir(S1);
```

### 9.659 ex\_SensorHTMagnet.nxc

This is an example of how to use the [SensorHTMagnet](#) function.

```
int value = SensorHTMagnet(S1);
```

### 9.660 ex\_SensorInvalid.nxc

This is an example of how to use the [SensorInvalid](#) function.

```
x = SensorInvalid(S1);
```

### 9.661 ex\_SensorMode.nxc

This is an example of how to use the [SensorMode](#) function.

```
x = SensorMode(S1);
```

### 9.662 ex\_SensorMSCompass.nxc

This is an example of how to use the [SensorMSCompass](#) function.

```
x = SensorMSCompass(S1, MS_ADDR_CMP5NX);
```

### 9.663 ex\_SensorMSDROD.nxc

This is an example of how to use the [SensorMSDROD](#) function.

```
x = SensorMSDROD(S1);
```

### 9.664 ex\_SensorMSPressure.nxc

This is an example of how to use the [SensorMSPressure](#) function.

```
int val = SensorMSPressure(S1);
```

### 9.665 ex\_SensorMSPressureRaw.nxc

This is an example of how to use the [SensorMSPressureRaw](#) function.

```
int val = SensorMSPressureRaw(S1);
```

**9.666 ex\_SensorNormalized.nxc**

This is an example of how to use the [SensorNormalized](#) function.

```
x = SensorNormalized(S1);
```

**9.667 ex\_SensorRaw.nxc**

This is an example of how to use the [SensorRaw](#) function.

```
x = SensorRaw(S1);
```

**9.668 ex\_SensorScaled.nxc**

This is an example of how to use the [SensorScaled](#) function.

```
x = SensorScaled(S1);
```

**9.669 ex\_SensorTemperature.nxc**

This is an example of how to use the [SensorTemperature](#) function.

```
float temp = SensorTemperature(S1);
```

**9.670 ex\_SensorType.nxc**

This is an example of how to use the [SensorType](#) function.

```
x = SensorType(S1);
```

**9.671 ex\_SensorUS.nxc**

This is an example of how to use the [SensorUS](#) function.

```
x = SensorUS(S4); // read sensor 4
```

### 9.672 ex\_SensorValue.nxc

This is an example of how to use the [SensorValue](#) function.

```
unsigned int val = SensorValue(S1);
```

### 9.673 ex\_SensorValueBool.nxc

This is an example of how to use the [SensorValueBool](#) function.

```
bool val = SensorValueBool(S1);
```

### 9.674 ex\_SensorValueRaw.nxc

This is an example of how to use the [SensorValueRaw](#) function.

```
unsigned int val = SensorValueRaw(S1);
```

### 9.675 ex\_SetAbortFlag.nxc

This is an example of how to use the [SetAbortFlag](#) function.

```
task main()
{
    // Set exit button to end program only if it is pressed for longer than 2 seconds
    #ifndef __ENHANCED_FIRMWARE
        SetLongAbort(true);
        // is equivalent to
        SetAbortFlag(BTNSTATE_LONG_PRESSED_EV);
    #endif

    while(true)
    {
        ClearScreen();
        // Display on NXT Screen: "Press the exit button longer (for 2 seconds) to exit"
        TextOut(0, LCD_LINE1, "Press the exit", 0);
        TextOut(0, LCD_LINE2, "button longer", 0);
        TextOut(0, LCD_LINE3, "(for 2 seconds)", 0);
        TextOut(0, LCD_LINE4, "to exit.", 0);

        // Display number of times the user has pressed the exit button (for less than 2 seconds)
        NumOut(0, LCD_LINE8, ButtonPressCount(BTNEXIT), 0);
    }
}
```

```
        // Wait until user presses and releases exit button before continuing loop
        while(!(ButtonPressed(BTNEXIT, 0)));
        while(ButtonPressed(BTNEXIT, 0));
    }
}
```

### 9.676 ex\_SetACCLNxSensitivity.nxc

This is an example of how to use the [SetACCLNxSensitivity](#) function.

```
result = SetACCLNxSensitivity(S1, MS_ADDR_ACCLNX, ACCL_SENSITIVITY_LEVEL_1);
```

### 9.677 ex\_SetBatteryState.nxc

This is an example of how to use the [SetBatteryState](#) function.

```
SetBatteryState(4);
```

### 9.678 ex\_SetBluetoothState.nxc

This is an example of how to use the [SetBluetoothState](#) function.

```
SetBluetoothState(UI_BT_STATE_OFF);
```

### 9.679 ex\_SetBTInputBuffer.nxc

This is an example of how to use the [SetBTInputBuffer](#) function.

```
SetBTInputBuffer(0, 10, buffer);
```

### 9.680 ex\_SetBTInputBufferInPtr.nxc

This is an example of how to use the [SetBTInputBufferInPtr](#) function.

```
SetBTInputBufferInPtr(0);
```

### 9.681 ex\_SetBTInputBufferOutPtr.nxc

This is an example of how to use the [SetBTInputBufferOutPtr](#) function.

```
SetBTInputBufferOutPtr(0);
```

### 9.682 ex\_SetBTOutputBuffer.nxc

This is an example of how to use the [SetBTOutputBuffer](#) function.

```
SetBTOutputBuffer(0, 10, buffer);
```

### 9.683 ex\_SetBTOutputBufferInPtr.nxc

This is an example of how to use the [SetBTOutputBufferInPtr](#) function.

```
SetBTOutputBufferInPtr(0);
```

### 9.684 ex\_SetBTOutputBufferOutPtr.nxc

This is an example of how to use the [SetBTOutputBufferOutPtr](#) function.

```
SetBTOutputBufferOutPtr(0);
```

### 9.685 ex\_SetButtonLongPressCount.nxc

This is an example of how to use the [SetButtonLongPressCount](#) function.

```
SetButtonLongPressCount(BTN1, value);
```

### 9.686 ex\_SetButtonLongReleaseCount.nxc

This is an example of how to use the [SetButtonLongReleaseCount](#) function.

```
SetButtonLongReleaseCount(BTN1, value);
```

### 9.687 ex\_SetButtonPressCount.nxc

This is an example of how to use the [SetButtonPressCount](#) function.

```
SetButtonPressCount (BTN1, value);
```

### 9.688 ex\_SetButtonReleaseCount.nxc

This is an example of how to use the [SetButtonReleaseCount](#) function.

```
SetButtonReleaseCount (BTN1, value);
```

### 9.689 ex\_SetButtonShortReleaseCount.nxc

This is an example of how to use the [SetButtonShortReleaseCount](#) function.

```
SetButtonShortReleaseCount (BTN1, value);
```

### 9.690 ex\_SetButtonState.nxc

This is an example of how to use the [SetButtonState](#) function.

```
SetButtonState (BTN1, BTNSTATE_PRESSED_EV);
```

### 9.691 ex\_SetCommandFlags.nxc

This is an example of how to use the [SetCommandFlags](#) function.

```
SetCommandFlags (UI_FLAGS_REDRAW_STATUS);
```

### 9.692 ex\_SetCustomSensorActiveStatus.nxc

This is an example of how to use the [SetCustomSensorActiveStatus](#) function.

```
SetCustomSensorActiveStatus (S1, true);
```

**9.693 ex\_SetCustomSensorPercentFullScale.nxc**

This is an example of how to use the [SetCustomSensorPercentFullScale](#) function.

```
SetCustomSensorPercentFullScale(S1, 100);
```

**9.694 ex\_SetCustomSensorZeroOffset.nxc**

This is an example of how to use the [SetCustomSensorZeroOffset](#) function.

```
SetCustomSensorZeroOffset(S1, 12);
```

**9.695 ex\_setdisplaycontrast.nxc**

This is an example of how to use the [SetDisplayContrast](#) function.

```
SetDisplayContrast(DISPLAY_CONTRAST_DEFAULT);
```

**9.696 ex\_SetDisplayDisplay.nxc**

This is an example of how to use the [SetDisplayDisplay](#) function.

```
SetDisplayDisplay(x);
```

**9.697 ex\_SetDisplayEraseMask.nxc**

This is an example of how to use the [SetDisplayEraseMask](#) function.

```
SetDisplayEraseMask(x);
```

**9.698 ex\_SetDisplayFlags.nxc**

This is an example of how to use the [SetDisplayFlags](#) function.

```
SetDisplayFlags(x);
```

## 9.699 ex\_setdisplayfont.nxc

This is an example of how to use the [SetDisplayFont](#) function.

```

const byte NewFont[] =
{
    0x04,0x00, // Graphics Format
    0x02,0x40, // Graphics DataSize
    0x10,      // Graphics Count X
    0x06,      // Graphics Count Y
    0x06,      // Graphics Width
    0x08,      // Graphics Height
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,0x06,0x00,0x00,0x00,0x07,0x03,0x00,0x07
    ,0x03,0x00,0x24,0x7E,0x24,0x7E,0x24,0x00,0x24,0x2B,0x6A,0x12,0x00,0x00,0x63,0x13,
    0x08,0x64,0x63,0x00,0x30,0x4C,0x52,0x22,0x50,0x00,0x00,0x07,0x03,0x00,0x00,0x00,0
    x00,0x3E,0x41,0x00,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x
    08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x08
    ,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,
    0x3E,0x51,0x49,0x45,0x3E,0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x62,0x51,0x49,0x49
    ,0x46,0x00,0x22,0x49,0x49,0x49,0x36,0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x2F,0x49,
    0x49,0x49,0x31,0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
    x36,0x49,0x49,0x49,0x36,0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00,0x6C,0x6C,0x00,0x
    00,0x00,0x00,0xEC,0x6C,0x00,0x00,0x00,0x08,0x14,0x22,0x41,0x00,0x00,0x24,0x24,0x2
    4,0x24,0x24,0x00,0x00,0x41,0x22,0x14,0x08,0x00,0x02,0x01,0x59,0x09,0x06,0x00,
    0x3E,0x41,0x5D,0x55,0x1E,0x00,0x7E,0x11,0x11,0x11,0x7E,0x00,0x7F,0x49,0x49,0x49
    ,0x36,0x00,0x3E,0x41,0x41,0x41,0x22,0x00,0x7F,0x41,0x41,0x41,0x3E,0x00,0x7F,0x49,
    0x49,0x49,0x41,0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
    x7F,0x08,0x08,0x08,0x7F,0x00,0x00,0x41,0x7F,0x41,0x00,0x00,0x30,0x40,0x40,0x40,0x
    3F,0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x7F,0x40,0x40,0x40,0x00,0x00,0x7F,0x02,0x0
    4,0x02,0x7F,0x00,0x7F,0x02,0x04,0x08,0x7F,0x00,0x3E,0x41,0x41,0x41,0x3E,0x00,
    0x7F,0x09,0x09,0x09,0x06,0x00,0x3E,0x41,0x51,0x21,0x5E,0x00,0x7F,0x09,0x09,0x19
    ,0x66,0x00,0x26,0x49,0x49,0x49,0x32,0x00,0x01,0x01,0x7F,0x01,0x01,0x00,0x3F,0x40,
    0x40,0x40,0x3F,0x00,0x1F,0x20,0x40,0x20,0x1F,0x00,0x3F,0x40,0x3C,0x40,0x3F,0x00,0
    x63,0x14,0x08,0x14,0x63,0x00,0x07,0x08,0x70,0x08,0x07,0x00,0x71,0x49,0x45,0x43,0x
    00,0x00,0x00,0x7F,0x41,0x41,0x00,0x00,0x02,0x04,0x08,0x10,0x20,0x00,0x00,0x41,0x4
    1,0x7F,0x00,0x00,0x04,0x02,0x01,0x02,0x04,0x00,0x80,0x80,0x80,0x80,0x80,0x00,
    0x00,0x02,0x05,0x02,0x00,0x00,0x20,0x54,0x54,0x54,0x78,0x00,0x7F,0x44,0x44,0x44
    ,0x38,0x00,0x38,0x44,0x44,0x44,0x28,0x00,0x38,0x44,0x44,0x44,0x7F,0x00,0x38,0x54,
    0x54,0x54,0x08,0x00,0x08,0x7E,0x09,0x09,0x00,0x00,0x18,0x24,0xA4,0xA4,0xFC,0x00,0
    x7F,0x04,0x04,0x78,0x00,0x00,0x00,0x00,0x7D,0x40,0x00,0x00,0x40,0x80,0x84,0x7D,0x
    00,0x00,0x7F,0x10,0x28,0x44,0x00,0x00,0x00,0x00,0x7F,0x40,0x00,0x00,0x7C,0x04,0x1
    8,0x04,0x78,0x00,0x7C,0x04,0x04,0x78,0x00,0x00,0x38,0x44,0x44,0x44,0x38,0x00,
    0xFC,0x44,0x44,0x44,0x38,0x00,0x38,0x44,0x44,0x44,0x44,0xFC,0x00,0x44,0x78,0x44,0x04
    ,0x08,0x00,0x08,0x54,0x54,0x54,0x20,0x00,0x04,0x3E,0x44,0x24,0x00,0x00,0x3C,0x40,
    0x20,0x7C,0x00,0x00,0x1C,0x20,0x40,0x20,0x1C,0x00,0x3C,0x60,0x30,0x60,0x3C,0x00,0
    x6C,0x10,0x10,0x6C,0x00,0x00,0x9C,0xA0,0x60,0x3C,0x00,0x00,0x64,0x54,0x54,0x4C,0x
    00,0x00,0x08,0x3E,0x41,0x41,0x00,0x00,0x00,0x00,0x77,0x00,0x00,0x00,0x00,0x41,0x4
    1,0x3E,0x08,0x00,0x02,0x01,0x02,0x01,0x00,0x00,0x10,0x20,0x40,0x38,0x07,0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    byte myData[800];
    ptr = addr(NewFont);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
}

```

```
SetDisplayFont(ptr);
TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
SetDisplayFont(pOldFont);
TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
Wait(SEC_10);
}
```

### 9.700 ex\_SetDisplayNormal.nxc

This is an example of how to use the [SetDisplayNormal](#) function.

```
SetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

### 9.701 ex\_SetDisplayPopup.nxc

This is an example of how to use the [SetDisplayPopup](#) function.

```
SetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

### 9.702 ex\_SetDisplayTextLinesCenterFlags.nxc

This is an example of how to use the [SetDisplayTextLinesCenterFlags](#) function.

```
SetDisplayTextLinesCenterFlags(x);
```

### 9.703 ex\_SetDisplayUpdateMask.nxc

This is an example of how to use the [SetDisplayUpdateMask](#) function.

```
SetDisplayUpdateMask(x);
```

### 9.704 ex\_SetHSFlags.nxc

This is an example of how to use the [SetHSFlags](#) function.

```
SetHSFlags(0);
```

**9.705 ex\_SetHSInputBuffer.nxc**

This is an example of how to use the [SetHSInputBuffer](#) function.

```
SetHSInputBuffer(0, 10, buffer);
```

**9.706 ex\_SetHSInputBufferInPtr.nxc**

This is an example of how to use the [SetHSInputBufferInPtr](#) function.

```
SetHSInputBufferInPtr(0);
```

**9.707 ex\_SetHSInputBufferOutPtr.nxc**

This is an example of how to use the [SetHSInputBufferOutPtr](#) function.

```
SetHSInputBufferOutPtr(0);
```

**9.708 ex\_sethsmode.nxc**

This is an example of how to use the [SetHSMode](#) function.

```
SetHSMode(HS_MODE_8N1);
```

**9.709 ex\_SetHSOutputBuffer.nxc**

This is an example of how to use the [SetHSOutputBuffer](#) function.

```
SetHSOutputBuffer(0, 10, buffer);
```

**9.710 ex\_SetHSOutputBufferInPtr.nxc**

This is an example of how to use the [SetHSOutputBufferInPtr](#) function.

```
SetHSOutputBufferInPtr(0);
```

**9.711 ex\_SetHSOutputBufferOutPtr.nxc**

This is an example of how to use the [SetHSOutputBufferOutPtr](#) function.

```
SetHSOutputBufferOutPtr (0);
```

**9.712 ex\_SetHSSpeed.nxc**

This is an example of how to use the [SetHSSpeed](#) function.

```
SetHSSpeed (1);
```

**9.713 ex\_SetHSState.nxc**

This is an example of how to use the [SetHSState](#) function.

```
SetHSState (1);
```

**9.714 ex\_sethtcolor2mode.nxc**

This is an example of how to use the [SetHTColor2Mode](#) function.

```
SetHTColor2Mode (S1, HT_CMD_COLOR2_ACTIVE);
```

**9.715 ex\_sethtirseeker2mode.nxc**

This is an example of how to use the [SetHTIRSeeker2Mode](#) function.

```
SetHTIRSeeker2Mode (S1, HTIR2_MODE_1200);
```

**9.716 ex\_SetInput.nxc**

This is an example of how to use the [SetInput](#) function.

```
SetInput (S1, Type, SENSOR_TYPE_SOUND_DB);
```

### 9.717 ex\_SetLongAbort.nxc

This is an example of how to use the [SetLongAbort](#) function.

```
task main()
{
    // Set exit button to end program only if it is pressed for longer than 2 seconds
    #ifndef __ENHANCED_FIRMWARE
        SetLongAbort(true);
        // is equivalent to
        SetAbortFlag(BTNSTATE_LONG_PRESSED_EV);
    #endif

    while(true)
    {
        ClearScreen();
        // Display on NXT Screen: "Press the exit button longer (for 2 seconds) to exit"
        TextOut(0, LCD_LINE1, "Press the exit", 0);
        TextOut(0, LCD_LINE2, "button longer", 0);
        TextOut(0, LCD_LINE3, "(for 2 seconds)", 0);
        TextOut(0, LCD_LINE4, "to exit.", 0);

        // Display number of times the user has pressed the exit button (for less than 2 seconds)
        NumOut(0, LCD_LINE8, ButtonPressCount(BTNEXIT), 0);

        // Wait until user presses and releases exit button before continuing loop
        while(!(ButtonPressed(BTNEXIT, 0)));
        while(ButtonPressed(BTNEXIT, 0));
    }
}
```

### 9.718 ex\_SetMotorPwnFreq.nxc

This is an example of how to use the [SetMotorPwnFreq](#) function.

```
SetMotorPwnFreq(x);
```

### 9.719 ex\_SetOnBrickProgramPointer.nxc

This is an example of how to use the [SetOnBrickProgramPointer](#) function.

```
SetOnBrickProgramPointer(2);
```

### 9.720 ex\_setoutput.nxc

This is an example of how to use the [SetOutput](#) function.

```
SetOutput(OUT_AB, TachoLimit, 720); // set tacho limit
```

### 9.721 ex\_SetSensor.nxc

This is an example of how to use the [SetSensor](#) function.

```
SetSensor(S1, SENSOR_TOUCH);
```

### 9.722 ex\_setsensorboolean.nxc

This is an example of how to use the [SetSensorBoolean](#) function.

```
SetHTIRSeeker2Mode(S1, HTIR2_MODE_1200);
```

### 9.723 ex\_setsensorcolorblue.nxc

This is an example of how to use the [SetSensorColorBlue](#) function.

```
SetSensorColorBlue(S1);
```

### 9.724 ex\_setsensorcolorfull.nxc

This is an example of how to use the [SetSensorColorFull](#) function.

```
SetSensorColorFull(S1);
```

### 9.725 ex\_setsensorcolorgreen.nxc

This is an example of how to use the [SetSensorColorGreen](#) function.

```
SetSensorColorGreen(S1);
```

### 9.726 ex\_setsensorcolornone.nxc

This is an example of how to use the [SetSensorColorNone](#) function.

```
SetSensorColorNone(S1);
```

**9.727 ex\_setsensorcolorred.nxc**

This is an example of how to use the [SetSensorColorRed](#) function.

```
SetSensorColorRed(S1);
```

**9.728 ex\_SetSensorDigiPinsDirection.nxc**

This is an example of how to use the [SetSensorDigiPinsDirection](#) function.

```
SetSensorDigiPinsDirection(S1, 1);
```

**9.729 ex\_SetSensorDigiPinsOutputLevel.nxc**

This is an example of how to use the [SetSensorDigiPinsOutputLevel](#) function.

```
SetSensorDigiPinsOutputLevel(S1, 100);
```

**9.730 ex\_SetSensorDigiPinsStatus.nxc**

This is an example of how to use the [SetSensorDigiPinsStatus](#) function.

```
SetSensorDigiPinsStatus(S1, false);
```

**9.731 ex\_SetSensorEMeter.nxc**

This is an example of how to use the [SetSensorEMeter](#) function.

```
SetSensorEMeter(S1);
```

**9.732 ex\_setsensorhteopd.nxc**

This is an example of how to use the [SetSensorHTEOPD](#) function.

```
SetSensorHTEOPD(S1);
```

**9.733 ex\_SetSensorHTGyro.nxc**

This is an example of how to use the [SetSensorHTGyro](#) function.

```
SetSensorHTGyro (S1);
```

**9.734 ex\_SetSensorHTMagnet.nxc**

This is an example of how to use the [SetSensorHTMagnet](#) function.

```
SetSensorHTMagnet (S1);
```

**9.735 ex\_SetSensorLight.nxc**

This is an example of how to use the [SetSensorLight](#) function.

```
SetSensorLight (S1);
```

**9.736 ex\_SetSensorLowspeed.nxc**

This is an example of how to use the [SetSensorLowspeed](#) function.

```
SetSensorLowspeed (S1);
```

**9.737 ex\_SetSensorMode.nxc**

This is an example of how to use the [SetSensorMode](#) function.

```
SetSensorMode (S1, SENSOR_MODE_RAW); // raw mode
```

**9.738 ex\_setsensormsdrod.nxc**

This is an example of how to use the [SetSensorMSDROD](#) function.

```
SetSensorMSDROD (S1);
```

**9.739 ex\_setsensormspressure.nxc**

This is an example of how to use the [SetSensorMSPressure](#) function.

```
SetSensorMSPressure (S1);
```

**9.740 ex\_SetSensorSound.nxc**

This is an example of how to use the [SetSensorSound](#) function.

```
SetSensorSound (S1);
```

**9.741 ex\_SetSensorTemperature.nxc**

This is an example of how to use the [SetSensorTemperature](#) function.

```
SetSensorTemperature (S1);
```

**9.742 ex\_SetSensorTouch.nxc**

This is an example of how to use the [SetSensorTouch](#) function.

```
SetSensorTouch (S1);
```

**9.743 ex\_SetSensorType.nxc**

This is an example of how to use the [SetSensorType](#) function.

```
SetSensorType (S1, SENSOR_TYPE_TOUCH);
```

**9.744 ex\_SetSensorUltrasonic.nxc**

This is an example of how to use the [SetSensorUltrasonic](#) function.

```
SetSensorUltrasonic (S1);
```

**9.745 ex\_setsleeptime.nxc**

This is an example of how to use the [SetSleepTime](#) function.

```
SetSleepTime(5); // sleep in 5 minutes
```

**9.746 ex\_SetSleepTimeout.nxc**

This is an example of how to use the [SetSleepTimeout](#) function.

```
SetSleepTimeout(8);
```

**9.747 ex\_SetSleepTimer.nxc**

This is an example of how to use the [SetSleepTimer](#) function.

```
SetSleepTimer(3);
```

**9.748 ex\_SetSoundDuration.nxc**

This is an example of how to use the [SetSoundDuration](#) function.

```
SetSoundDuration(500);
```

**9.749 ex\_SetSoundFlags.nxc**

This is an example of how to use the [SetSoundFlags](#) function.

```
SetSoundFlags(SOUND_FLAGS_UPDATE);
```

**9.750 ex\_SetSoundFrequency.nxc**

This is an example of how to use the [SetSoundFrequency](#) function.

```
SetSoundFrequency(440);
```

**9.751 ex\_SetSoundMode.nxc**

This is an example of how to use the [SetSoundMode](#) function.

```
SetSoundMode (SOUND_MODE_ONCE);
```

**9.752 ex\_SetSoundModuleState.nxc**

This is an example of how to use the [SetSoundModuleState](#) function.

```
SetSoundModuleState (SOUND_STATE_STOP);
```

**9.753 ex\_SetSoundSampleRate.nxc**

This is an example of how to use the [SetSoundSampleRate](#) function.

```
SetSoundSampleRate (4000);
```

**9.754 ex\_SetSoundVolume.nxc**

This is an example of how to use the [SetSoundVolume](#) function.

```
SetSoundVolume (3);
```

**9.755 ex\_SetUIButton.nxc**

This is an example of how to use the [SetUIButton](#) function.

```
SetUIButton (UI_BUTTON_ENTER);
```

**9.756 ex\_SetUIState.nxc**

This is an example of how to use the [SetUIState](#) function.

```
SetUIState (UI_STATE_LOW_BATTERY);
```

**9.757 ex\_SetUSBInputBuffer.nxc**

This is an example of how to use the [SetUSBInputBuffer](#) function.

```
SetUSBInputBuffer(0, 10, buffer);
```

**9.758 ex\_SetUSBInputBufferInPtr.nxc**

This is an example of how to use the [SetUSBInputBufferInPtr](#) function.

```
SetUSBInputBufferInPtr(0);
```

**9.759 ex\_SetUSBInputBufferOutPtr.nxc**

This is an example of how to use the [SetUSBInputBufferOutPtr](#) function.

```
SetUSBInputBufferOutPtr(0);
```

**9.760 ex\_SetUSBOutputBuffer.nxc**

This is an example of how to use the [SetUSBOutputBuffer](#) function.

```
SetUSBOutputBuffer(0, 10, buffer);
```

**9.761 ex\_SetUSBOutputBufferInPtr.nxc**

This is an example of how to use the [SetUSBOutputBufferInPtr](#) function.

```
SetUSBOutputBufferInPtr(0);
```

**9.762 ex\_SetUSBOutputBufferOutPtr.nxc**

This is an example of how to use the [SetUSBOutputBufferOutPtr](#) function.

```
SetUSBOutputBufferOutPtr(0);
```

**9.763 ex\_SetUSBPollBuffer.nxc**

This is an example of how to use the [SetUSBPollBuffer](#) function.

```
SetUSBPollBuffer(0, 10, buffer);
```

**9.764 ex\_SetUSBPollBufferInPtr.nxc**

This is an example of how to use the [SetUSBPollBufferInPtr](#) function.

```
SetUSBPollBufferInPtr(0);
```

**9.765 ex\_SetUSBPollBufferOutPtr.nxc**

This is an example of how to use the [SetUSBPollBufferOutPtr](#) function.

```
SetUSBPollBufferOutPtr(0);
```

**9.766 ex\_SetUsbState.nxc**

This is an example of how to use the [SetUSBState](#) function.

```
SetUSBState(0);
```

**9.767 ex\_SetVMRunState.nxc**

This is an example of how to use the [SetVMRunState](#) function.

```
SetVMRunState(VM_RUN_PAUSE); // pause the virtual machine. This could be used li  
ke a breakpoint
```

**9.768 ex\_SetVolume.nxc**

This is an example of how to use the [SetVolume](#) function.

```
SetVolume(3);
```

### 9.769 ex\_sign.nxc

This is an example of how to use the [sign](#) function.

```
char val = sign(x); // return -1, 0, or 1
```

### 9.770 ex\_sin\_cos.nxc

This is an example of how to use the [cos](#) and the [sin](#) functions.

```
// ex_sin_cos.nxc
// Run this program and you will see a circle appear on the NXT screen in a
// strange random way. No two runs will produce the circle in exactly the same
// way.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define SCREEN_WIDTH 100
#define SCREEN_HEIGHT 64
#define X_ZERO (SCREEN_WIDTH / 2)
#define Y_ZERO (SCREEN_HEIGHT / 2)
#define R (Y_ZERO - 2)
#define MAX_DEG 360

// Convert a float to its nearest integer value.
inline int integer(float x)
{
    return trunc(x + 0.5);
}

task main()
{
    while(true)
    {
        float angle = RADIANS_PER_DEGREE * Random(MAX_DEG);
        float x = X_ZERO + R * cos(angle);
        float y = Y_ZERO + R * sin(angle);
        PointOut(integer(x), integer(y));
        // Without the Wait, the program runs too fast!
        Wait(MS_20);
    }
}
```

### 9.771 ex\_sind\_cosd.nxc

This is an example of how to use the [cosd](#) and [sind](#) functions.

```
// ex_sind_cosd.nxc
// Run this program and you will see a circle appear on the NXT screen in a
```

```
// strange random way. No two runs will produce the circle in exactly the same
// way.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define SCREEN_WIDTH 100
#define SCREEN_HEIGHT 64
#define X_ZERO (SCREEN_WIDTH / 2)
#define Y_ZERO (SCREEN_HEIGHT / 2)
#define R (Y_ZERO - 2)
#define MAX_DEG 360

// Convert a float to its nearest integer value.
inline int integer(float x)
{
    return trunc(x + 0.5);
}

task main()
{
    while(true)
    {
        float angle = Random(MAX_DEG);
        float x = X_ZERO + R * cosd(angle);
        float y = Y_ZERO + R * sind(angle);
        PointOut(integer(x), integer(y));
        // Without the Wait, the program runs too fast!
        Wait(MS_20);
    }
}
```

### 9.772 ex\_sinh.nxc

This is an example of how to use the [sinh](#) function.

```
x = sinh(y);
```

### 9.773 ex\_SizeOf.nxc

This is an example of how to use the [SizeOf](#) and [NumOut](#) functions.

```
task main()
{
    int x;
    float f;
    byte b;
    byte data[] = {1, 2, 3, 4, 5, 6};

    NumOut(0, LCD_LINE1, SizeOf(x));
    NumOut(0, LCD_LINE2, SizeOf(f));
    NumOut(0, LCD_LINE3, SizeOf(b));
}
```

```
NumOut(0, LCD_LINE4, SizeOf(data));  
while(true);  
}
```

### 9.774 ex\_SleepNow.nxc

This is an example of how to use the [SleepNow](#) functions.

```
SleepNow();
```

### 9.775 ex\_sleeptime.nxc

This is an example of how to use the [SleepTime](#) function.

```
x = SleepTime(); // read sleep time
```

### 9.776 ex\_SleepTimeout.nxc

This is an example of how to use the [SleepTimeout](#) function.

```
byte x = SleepTimeout();
```

### 9.777 ex\_SleepTimer.nxc

This is an example of how to use the [SleepTimer](#) function.

```
byte x = SleepTimer();
```

### 9.778 ex\_SoundDuration.nxc

This is an example of how to use the [SoundDuration](#) function.

```
x = SoundDuration();
```

### 9.779 ex\_SoundFlags.nxc

This is an example of how to use the [SoundFlags](#) function.

```
x = SoundFlags();
```

**9.780 ex\_SoundFrequency.nxc**

This is an example of how to use the [SoundFrequency](#) function.

```
x = SoundFrequency ();
```

**9.781 ex\_SoundMode.nxc**

This is an example of how to use the [SoundMode](#) function.

```
x = SoundMode ();
```

**9.782 ex\_SoundSampleRate.nxc**

This is an example of how to use the [SoundSampleRate](#) function.

```
x = SoundSampleRate ();
```

**9.783 ex\_SoundState.nxc**

This is an example of how to use the [SoundState](#) function.

```
x = SoundState ();
```

**9.784 ex\_SoundVolume.nxc**

This is an example of how to use the [SoundVolume](#) function.

```
x = SoundVolume ();
```

**9.785 ex\_sprintf.nxc**

This is an example of how to use the [sprintf](#) function.

```
sprintf(msg, "value = %d", value);
```

**9.786 ex\_sqrt.nxc**

This is an example of how to use the [sqrt](#) function.

```
x = sqrt(x);
```

**9.787 ex\_srand.nxc**

This is an example of how to use the [srand](#) function.

```
unsigned long newseed = srand(0);
```

**9.788 ex\_StartTask.nxc**

This is an example of how to use the [StartTask](#) function.

```
StartTask(sound); // start the sound task
```

**9.789 ex\_Stop.nxc**

This is an example of how to use the [Stop](#) function.

```
Stop(x == 24); // stop the program if x==24
```

**9.790 ex\_StopAllTasks.nxc**

This is an example of how to use the [StopAllTasks](#) function.

```
StopAllTasks(); // stop the program
```

**9.791 ex\_StopSound.nxc**

This is an example of how to use the [StopSound](#) function.

```
StopSound();
```

### 9.792 ex\_StopTask.nxc

This is an example of how to use the [StopTask](#) function.

```
StopTask(sound); // stop the sound task
```

### 9.793 ex\_StrCat.nxc

This is an example of how to use the [strcat](#) function.

```
strcat(msg, "foo"); // msg = msg+"foo"
```

### 9.794 ex\_StrCatOld.nxc

This is an example of how to use the [StrCat](#) function.

```
task main()
{
  string msgs[] = {"please work", "testing, 1, 2, 3"};
  string tmp = "123456";
  string a = "AA", b = "BB", c = "CC";

  TextOut(0, LCD_LINE3, StrCat(a, SubStr(tmp, 2, 3), msgs[0]));
  Wait(SEC_5);
}
```

### 9.795 ex\_strcmp.nxc

This is an example of how to use the [strcmp](#) function.

```
int i = strcmp(msg, "foo"); // returns -1, 0, or 1
```

### 9.796 ex\_strcpy.nxc

This is an example of how to use the [strcpy](#) function.

```
strcpy(msg, "foo"); // msg = "foo"
```

**9.797 ex\_StrIndex.nxc**

This is an example of how to use the [StrIndex](#) function.

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    NumOut(0, LCD_LINE5, StrIndex(msgs[0], 0));
    string msg = "hi there";
    byte x = StrIndex(msg, 2); // return the value of msg[2]
    Wait(SEC_5);
}
```

**9.798 ex\_string.nxc**

This is an example of how to use the string API functions: [StrToNum](#), [StrLen](#), [StrIndex](#), [NumToStr](#), [StrCat](#), [SubStr](#), [Flatten](#), [StrReplace](#), [FormatNum](#), [FlattenVar](#), [UnflattenVar](#), [ByteArrayToStr](#), [ByteArrayToStrEx](#), and [StrToByteArray](#).

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    string fmts[] = {"x = %4.4d", "0x%x"};
    string tmp = "123456";
    string s = SubStr(tmp, 2, 3);
    string a = "AA", b = "BB", c = "CC";

    TextOut(0, LCD_LINE1, s);
    TextOut(0, LCD_LINE2, SubStr(msgs[0], 2, 3));
    TextOut(0, LCD_LINE3, StrCat(a, SubStr(tmp, 2, 3), msgs[0]));
    TextOut(0, LCD_LINE4, StrReplace(msgs[0], 2, StrCat(a, b)));
    NumOut(0, LCD_LINE5, StrIndex(msgs[0], 0));
    NumOut(0, LCD_LINE6, StrLen(msgs[0]));
    TextOut(0, LCD_LINE7, FormatNum(fmts[0], Random(34)));
    float val = StrToNum("10.5abc123");
    NumOut(0, LCD_LINE8, val);
    Wait(SEC_5);
    ClearScreen();
    TextOut(0, LCD_LINE1, NumToStr(PI));
    int x = 0x7172;
    string foo = FlattenVar(x);
    TextOut(0, LCD_LINE2, foo);
    TextOut(0, LCD_LINE3, Flatten(0x7374));
    NumOut(0, LCD_LINE4, strlen(foo));
    NumOut(40, LCD_LINE4, UnflattenVar(foo, x));
    TextOut(0, LCD_LINE5, FormatNum(fmts[1], x));
    string bats = tmp; // "123456"
    TextOut(0, LCD_LINE6, bats);
    byte data[];
    StrToByteArray(bats, data);
    TextOut(0, LCD_LINE7, ByteArrayToStr(data));
    ByteArrayToStrEx(data, tmp);
    TextOut(0, LCD_LINE8, tmp);
}
```

```
    Wait(SEC_10);  
}
```

### 9.799 ex\_StrLen.nxc

This is an example of how to use the [strlen](#) function.

```
task main()  
{  
    string msg = "hi there";  
    byte x = strlen(msg); // return the length of msg  
}
```

### 9.800 ex\_StrLenOld.nxc

This is an example of how to use the [StrLen](#) function.

```
task main()  
{  
    string msgs[] = {"please work", "testing, 1, 2, 3"};  
    string msg = "hi there";  
    byte x = StrLen(msg); // return the length of msg  
  
    NumOut(0, LCD_LINE6, StrLen(msgs[0]));  
    Wait(SEC_5);  
}
```

### 9.801 ex\_strncat.nxc

This is an example of how to use the [strncat](#) function.

```
strncat(msg, "foo", 2); // msg = msg+"fo"
```

### 9.802 ex\_strncmp.nxc

This is an example of how to use the [strncmp](#) function.

```
int i = strncmp(msg, "foo", 2); // returns -1, 0, or 1
```

### 9.803 ex\_strncpy.nxc

This is an example of how to use the [strncpy](#) function.

```
strncpy(msg, "foo", 2); // msg = "fo"
```

### 9.804 ex\_StrReplace.nxc

This is an example of how to use the [StrReplace](#) function.

```
task main()
{
  string msgs[] = {"please work", "testing", 1, 2, 3};
  string a = "AA", b = "BB", c = "CC";
  TextOut(0, LCD_LINE4, StrReplace(msgs[0], 2, StrCat(a, b)));
  string msg = StrReplace("testing", 3, "xx"); // returns "tesxxng"

  Wait(SEC_5);
}
```

### 9.805 ex\_StrToByteArray.nxc

This is an example of how to use the [StrToByteArray](#) function.

```
StrToByteArray(myStr, myArray);
```

### 9.806 ex\_strtod.nxc

This is an example of how to use the [strtod](#) function.

```
task main()
{
  string str, endptr;
  str = "3.1415926e2abcdefg";
  float f = strtod(str, endptr);
  NumOut(0, LCD_LINE1, f);
  TextOut(0, LCD_LINE2, str);
  TextOut(0, LCD_LINE3, endptr);
  Wait(SEC_6);
}
```

### 9.807 ex\_strtol.nxc

This is an example of how to use the [strtol](#) function.

```
task main()
{
  string str, endptr;
```

```
    str = "3.1415926e2abcdefg";
    long l = strtol(str, endptr);
    NumOut(0, LCD_LINE1, l);
    TextOut(0, LCD_LINE2, str);
    TextOut(0, LCD_LINE3, endptr);
    Wait(SEC_6);
}
```

### 9.808 ex\_StrToNum.nxc

This is an example of how to use the [StrToNum](#) function.

```
x = StrToNum(strVal);
```

### 9.809 ex\_strtoul.nxc

This is an example of how to use the [strtoul](#) function.

```
task main()
{
    string str, endptr;
    str = "3.1415926e2abcdefg";
    unsigned long l = strtoul(str, endptr);
    NumOut(0, LCD_LINE1, l);
    TextOut(0, LCD_LINE2, str);
    TextOut(0, LCD_LINE3, endptr);
    Wait(SEC_6);
}
```

### 9.810 ex\_SubStr.nxc

This is an example of how to use the [SubStr](#) function.

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    TextOut(0, LCD_LINE2, SubStr(msgs[0], 2, 3));
    string msg = SubStr("test", 1, 2); // returns "es"
    Wait(SEC_5);
}
```

### 9.811 ex\_superpro.nxc

This is an example of how to use the [SensorHTSuperProAnalog](#), [ReadSensorHTSuperProAllAnalog](#), [SetSensorHTSuperProDigitalControl](#), [SetSensorHTSuperProDigital](#),

[SensorHTSuperProDigital](#), [SetSensorHTSuperProAnalogOut](#), [SensorHTSuperProLED](#), [SensorHTSuperProStrobe](#), [SensorHTSuperProProgramControl](#), [SetSensorHTSuperProLED](#), [SetSensorHTSuperProStrobe](#), [SetSensorHTSuperProProgramControl](#), [ReadSensorHTSuperProAnalogOut](#), and [SensorHTSuperProDigitalControl](#) functions.

```
task main()
{
  SetSensorLowspeed(S1);
  SetHTSuperProDigitalControl(S1, 0xFF); // all outputs
  SetHTSuperProDigital(S1, DIGI_PIN0|DIGI_PIN1|DIGI_PIN2);
  SetHTSuperProLED(S1, LED_BLUE);
  SetHTSuperProStrobe(S1, STROBE_S0);
  SetHTSuperProProgramControl(S1, 0x01);
  NumOut(0, LCD_LINE1, SensorHTSuperProDigitalControl(S1));
  NumOut(40, LCD_LINE1, SensorHTSuperProDigital(S1));
  NumOut(0, LCD_LINE2, SensorHTSuperProLED(S1));
  NumOut(0, LCD_LINE3, SensorHTSuperProStrobe(S1));
  NumOut(0, LCD_LINE4, SensorHTSuperProProgramControl(S1));
  NumOut(0, LCD_LINE5, SensorHTSuperProAnalog(S1, HTSPRO_A0));
  SetHTSuperProAnalogOut(S1, HTSPRO_DAC1, DAC_MODE_SINEWAVE, 1000, 512);
  byte m;
  int f, v;
  ReadSensorHTSuperProAnalogOut(S1, HTSPRO_DAC0, m, f, v);
  NumOut(0, LCD_LINE6, m);
  NumOut(0, LCD_LINE7, f);
  NumOut(0, LCD_LINE8, v);
  int a0, a1, a2, a3;
  while (true) {
    ReadSensorHTSuperProAllAnalog(S1, a0, a1, a2, a3);
    NumOut(40, LCD_LINE5, a0);
    NumOut(40, LCD_LINE6, a1);
    NumOut(40, LCD_LINE7, a2);
    NumOut(40, LCD_LINE8, a3);
  }
}
```

## 9.812 ex\_syscall.nxc

This is an example of how to use the [SysCall](#) function.

```
task main()
{
  DrawTextType dtArgs;
  dtArgs.Location.X = 0;
  dtArgs.Location.Y = LCD_LINE1;
  dtArgs.Text = "Please Work";
  SysCall(DrawText, dtArgs);
}
```

## 9.813 ex\_SysColorSensorRead.nxc

This is an example of how to use the [SysColorSensorRead](#) function.

```

task main()
{
  SetSensorColorFull(S1);
  ColorSensorReadType csr;
  csr.Port = S1;
  SysColorSensorRead(csr);
  if (csr.Result == NO_ERR) {
    NumOut(0, LCD_LINE1, csr.ColorValue);
  }
}

```

### 9.814 ex\_syscommbtcheckstatus.nxc

This is an example of how to use the [SysCommBTCheckStatus](#) function along with the [CommBTCheckStatusType](#) structure.

```

task main()
{
  CommBTCheckStatusType args;
  args.Connection = 1;
  SysCommBTCheckStatus(args);
  if (args.Result == LDR_SUCCESS) { /* do something */ }
}

```

### 9.815 ex\_syscommbtconnection.nxc

This is an example of how to use the [SysCommBTConnection](#) function along with the [CommBTConnectionType](#) structure.

```

#define CONNECTION 1
task main()
{
  CommBTConnectionType args;
  args.Name = "NXT2"; // whatever the slave NXT's name is
  args.ConnectionSlot = CONNECTION; // this is the desired connection slot (the a
    bove code uses 1)
  args.Action = TRUE; // could use some #define with a non-zero value to connect.
    0 == disconnect
  if(!BluetoothStatus(CONNECTION)==NO_ERR)
  {
    SysCommBTConnection(args); // try to connect.
    for (int i = 0; i < 2000; i++) {
      NumOut(0, LCD_LINE3, args.Result);
      Wait(1);
    }
  }
  // Wait(5000); // let the connection get created
  if (args.Result == LDR_SUCCESS)
  {
    // we are connected
    TextOut(0, LCD_LINE1, "success");
  }
  else {

```

```
        TextOut(0, LCD_LINE1, "failure");
        NumOut(0, LCD_LINE2, args.Result);
    }
}
Wait(SEC_10);
}
```

### 9.816 ex\_SysCommBTOff.nxc

This is an example of how to use the [SysCommBTOff](#) function along with the [CommBTOffType](#) structure.

```
task main()
{
    CommBTOffType bt;
    bt.PowerState = false;
    SysCommBTOff(bt);
    if (bt.Result == NO_ERR)
        TextOut(0, LCD_LINE1, "BT is off");
}
```

### 9.817 ex\_syscommbtwrite.nxc

This is an example of how to use the [SysCommBTWrite](#) function along with the [CommBTWriteType](#) structure.

```
task main()
{
    CommBTWriteType args;
    args.Connection = 1;
    args.Buffer = myData;
    SysCommBTWrite(args);
}
```

### 9.818 ex\_syscommexecutefunction.nxc

This is an example of how to use the [SysCommExecuteFunction](#) function along with the [CommExecuteFunctionType](#) structure.

```
task main()
{
    CommExecuteFunctionType args;
    args.Cmd = INTF_BTOFF;
    SysCommExecuteFunction(args);
}
```

### 9.819 ex\_SysCommHSCheckStatus.nxc

This is an example of how to use the [SysCommHSCheckStatus](#) function along with the [CommHSCheckStatusType](#) structure.

```
task main()
{
    CommHSCheckStatusType hsc;
    SysCommHSCheckStatus(hsc);
    if (hsc.SendingData)
        TextOut(0, LCD_LINE1, "sending data");
    else if (hsc.DataAvailable)
        TextOut(0, LCD_LINE1, "data available");
}
```

### 9.820 ex\_SysCommHSControl.nxc

This is an example of how to use the [SysCommHSControl](#) function along with the [CommHSControlType](#) structure.

```
task main()
{
    CommHSControlType hsc;
    hsc.Command = HS_CTRL_INIT;
    SysCommHSControl(hsc);
    if (hsc.Result)
        TextOut(0, LCD_LINE1, "hi-speed initialized");
    Wait(SEC_10);
}
```

### 9.821 ex\_SysCommHSRead.nxc

This is an example of how to use the [SysCommHSRead](#) function along with the [CommHSReadWriteType](#) structure.

```
task main()
{
    CommHSReadWriteType hsr;
    SysCommHSRead(hsr);
    if (hsr.Status == NO_ERR)
        TextOut(0, LCD_LINE1, hsr.Buffer);
    Wait(SEC_1);
}
```

### 9.822 ex\_SysCommHSWrite.nxc

This is an example of how to use the [SysCommHSWrite](#) function along with the [CommHSReadWriteType](#) structure.

```
task main()
{
  // configure the hi-speed port and turn it on
  // ...
  // no write to the port
  CommHSReadWriteType rwt;
  ArrayBuild(rwt.Buffer, 0x01, 0x02, 0x03, 0x04); // four bytes
  SysCommHSWrite(rwt);
  if (rwt.Status = NO_ERR) {
    // do something
  }
}
```

### 9.823 ex\_syscommlscheckstatus.nxc

This is an example of how to use the [SysCommLSCheckStatus](#) function along with the [CommLSCheckStatusType](#) structure.

```
task main()
{
  CommLSCheckStatusType args;
  args.Port = S1;
  SysCommLSCheckStatus(args);
  // is the status (Result) IDLE?
  if (args.Result == LOWSPEED_IDLE) { /* proceed */ }
}
```

### 9.824 ex\_syscommlsread.nxc

This is an example of how to use the [SysCommLSRead](#) function along with the [CommLSReadType](#) structure.

```
task main()
{
  CommLSReadType args;
  args.Port = S1;
  args.Buffer = myBuf;
  args.BufferLen = 8;
  SysCommLSRead(args);
  // check Result for error status & use Buffer contents
}
```

### 9.825 ex\_syscommlswrite.nxc

This is an example of how to use the [SysCommLSWrite](#) function along with the [CommLSWriteType](#) structure.

```
task main()
{
```

```
CommLSWriteType args;
args.Port = S1;
args.Buffer = myBuf;
args.ReturnLen = 8;
SysCommLSWrite(args);
// check Result for error status
}
```

### 9.826 ex\_syscommLSwriteex.nxc

This is an example of how to use the [SysCommLSWriteEx](#) function along with the [CommLSWriteExType](#) structure.

```
task main()
{
  CommLSWriteExType args;
  args.Port = S1;
  args.Buffer = myBuf;
  args.ReturnLen = 8;
  args.NoRestartOnRead = true;
  SysCommLSWriteEx(args);
  if (args.Result == NO_ERR)
  {
    // do something
  }
}
```

### 9.827 ex\_SysComputeCalibValue.nxc

This is an example of how to use the [SysComputeCalibValue](#) function along with the [ComputeCalibValueType](#) structure.

```
task main()
{
  ComputeCalibValueType args;
  args.Name = "light";
  args.RawVal = Sensor(S1);
  SysComputeCalibValue(args);
  if (args.Result == NO_ERR)
  {
    TextOut(0, LCD_LINE1, "calib computed");
  }
}
```

### 9.828 ex\_sysdataloggettimes.nxc

This is an example of how to use the [SysDatalogGetTimes](#) function along with the [DatalogGetTimesType](#) structure.

```
task main()
{
```

```
DatalogGetTimesType args;
SysDatalogGetTimes(args);
NumOut(0, LCD_LINE4, args.SyncTime);
NumOut(0, LCD_LINE5, args.SyncTick);
Wait(SEC_5);
}
```

### 9.829 ex\_SysDatalogWrite.nxc

This is an example of how to use the [SysDatalogWrite](#) function along with the [DatalogWriteType](#) structure.

```
task main()
{
    DatalogWriteType args;
    ArrayBuild(args.Message, 0x01, 0x02);
    SysDatalogWrite(args);
    if (args.Result == NO_ERR)
        TextOut(0, LCD_LINE1, "success");
}
```

### 9.830 ex\_sysdisplayexecutefunction.nxc

This is an example of how to use the [SysDisplayExecuteFunction](#) function along with the [DisplayExecuteFunctionType](#) structure.

```
task main()
{
    DisplayExecuteFunctionType args;
    args.Cmd = DISPLAY_ERASE_ALL;
    SysDisplayExecuteFunction(args);
}
```

### 9.831 ex\_sysdrawcircle.nxc

This is an example of how to use the [SysDrawCircle](#) function along with the [DrawCircleType](#) structure.

```
task main()
{
    DrawCircleType dcArgs;
    dcArgs.Center.X = 20;
    dcArgs.Center.Y = 20;
    dcArgs.Size = 10; // radius
    dcArgs.Options = 0x01; // clear before drawing
    SysDrawCircle(dcArgs);
}
```

### 9.832 ex\_SysDrawEllipse.nxc

This is an example of how to use the [SysDrawEllipse](#) function along with the [DrawEllipseType](#) structure.

```
task main()
{
  DrawEllipseType args;
  args.Center.X = 50;
  args.Center.Y = 32;
  repeat (10) {
    args.SizeX = 20+Random(15);
    args.SizeY = 20+Random(10);
    args.Options = DRAW_OPT_FILL_SHAPE|DRAW_OPT_LOGICAL_XOR;
    SysDrawEllipse(args);
  }
  while(true);
}
```

### 9.833 ex\_sysdrawfont.nxc

This is an example of how to use the [SysDrawFont](#) function along with the [DrawFont-  
Type](#) structure.

```
#download "PropTiny.ric"

task main()
{
  DrawFontType dfArgs;
  dfArgs.Location.X = 10;
  dfArgs.Location.Y = 59;
  dfArgs.Filename = "PropTiny.ric" ;
  dfArgs.Text = "Hello" ;
  dfArgs.Options = DRAW_OPT_NORMAL|DRAW_OPT_FONT_DIR_L2RT;
  SysDrawFont(dfArgs);
  Wait(SEC_4);
}
```

### 9.834 ex\_sysdrawgraphic.nxc

This is an example of how to use the [SysDrawGraphic](#) function along with the [Draw-  
GraphicType](#) structure.

```
task main()
{
  DrawGraphicType dgArgs;
  dgArgs.Location.X = 20;
  dgArgs.Location.Y = 20;
  dgArgs.Filename = "image.ric";
  ArrayInit(dgArgs.Variables, 0, 10); // 10 zeros
}
```

```

dgArgs.Variables[0] = 12;
dgArgs.Variables[1] = 14; // etc...
dgArgs.Options = 0x00; // do not clear before drawing
SysDrawGraphic(dgArgs);
}

```

### 9.835 ex\_sysdrawgraphicarray.nxc

This is an example of how to use the [SysDrawGraphicArray](#) function along with the [DrawGraphicArrayType](#) structure.

```

byte ric_data[] = {
    RICOpsprite(1, 64, 2,
        RICSpriteData(0xFF, 0xFF, 0x80, 0x01, 0x80, 0x41,
            0x80, 0x21, 0x80, 0x11, 0x80, 0x09,
            0x80, 0x05, 0x80, 0x09, 0x80, 0x11,
            0x80, 0x21, 0x80, 0x41, 0x80, 0x81,
            0x81, 0x01, 0x82, 0x01, 0x84, 0x01,
            0x88, 0x01, 0x90, 0x01, 0xA0, 0x01,
            0x90, 0x01, 0x88, 0x01, 0x84, 0x01,
            0x82, 0x01, 0x81, 0x01, 0x80, 0x81,
            0x80, 0x41, 0x80, 0x21, 0x80, 0x11,
            0x80, 0x09, 0x80, 0x05, 0x80, 0x09,
            0x80, 0x11, 0x80, 0x21, 0x80, 0x41,
            0x80, 0x81, 0x81, 0x01, 0x82, 0x01,
            0x84, 0x01, 0x88, 0x01, 0x90, 0x01,
            0xA0, 0x01, 0x90, 0x01, 0x88, 0x01,
            0x84, 0x01, 0x82, 0x01, 0x81, 0x01,
            0x80, 0x81, 0x80, 0x41, 0x80, 0x21,
            0x80, 0x11, 0x80, 0x09, 0x80, 0x05,
            0x80, 0x09, 0x80, 0x11, 0x80, 0x21,
            0x80, 0x41, 0x80, 0x81, 0x81, 0x01,
            0x82, 0x01, 0x84, 0x01, 0x88, 0x01,
            0x90, 0x01, 0xA0, 0x01, 0x80, 0x01,
            0xFF, 0xFF)),
    RICOpcopybits(0, 1,
        RICimgrect(
            RICimgpoint(0, 0), 16, 64),
        RICimgpoint(0, 0))
};

task main()
{
    DrawGraphicArrayType args;
    args.Location.X = 0;
    args.Location.Y = 0;
    args.Data = ric_data;
    SysDrawGraphicArray(args);
    Wait(SEC_5);
}

```

### 9.836 ex\_sysdrawline.nxc

This is an example of how to use the [SysDrawLine](#) function along with the [DrawLineType](#) structure.

```
task main()
{
    DrawLineType dlArgs;
    dlArgs.StartLoc.X = 20;
    dlArgs.StartLoc.Y = 20;
    dlArgs.EndLoc.X = 60;
    dlArgs.EndLoc.Y = 60;
    dlArgs.Options = 0x01; // clear before drawing
    SysDrawLine(dlArgs);
}
```

### 9.837 ex\_sysdrawpoint.nxc

This is an example of how to use the [SysDrawPoint](#) function along with the [DrawPointType](#) structure.

```
task main()
{
    DrawPointType dpArgs;
    dpArgs.Location.X = 20;
    dpArgs.Location.Y = 20;
    dpArgs.Options = 0x04; // clear this pixel
    SysDrawPoint(dpArgs);
}
```

### 9.838 ex\_sysdrawpolygon.nxc

This is an example of how to use the [SysDrawPolygon](#) function along with the [DrawPolygonType](#) structure.

```
LocationType myPoints[] = {16,16, 8,40, 32,52, 20,36, 52,36, 56,52, 64,32, 44,20,
                          24,20};

task main()
{
    DrawPolygonType args;
    args.Points = myPoints;
    args.Options = 0x00;
    SysDrawPolygon(args);
    Wait(SEC_2);
    ClearScreen();
    args.Options = DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE;
    for(int i=0;i<10;i++) {
        SysDrawPolygon(args);
        Wait(SEC_1);
    }
}
```

```
}
args.Options = true|DRAW_OPT_FILL_SHAPE;
SysDrawPolygon(args);
Wait(SEC_2);
ClearScreen();
args.Options = DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE;
for (int i=0;i<100;i++) {
    SysDrawPolygon(args);
    Wait(MS_100);
}
Wait(SEC_1);
}
```

### 9.839 ex\_sysdrawrect.nxc

This is an example of how to use the [SysDrawRect](#) function along with the [DrawRect-Type](#) structure.

```
task main()
{
    DrawRectType drArgs;
    drArgs.Location.X = 20;
    drArgs.Location.Y = 20;
    drArgs.Size.Width = 20;
    drArgs.Size.Height = 10;
    drArgs.Options = 0x00; // do not clear before drawing
    SysDrawRect(drArgs);
}
```

### 9.840 ex\_sysdrawtext.nxc

This is an example of how to use the [SysDrawText](#) function along with the [DrawText-Type](#) structure.

```
task main()
{
    DrawTextType dtArgs;
    dtArgs.Location.X = 0;
    dtArgs.Location.Y = LCD_LINE1;
    dtArgs.Text = "Please Work";
    dtArgs.Options = 0x01; // clear before drawing
    SysDrawText(dtArgs);
}
```

### 9.841 ex\_sysfileclose.nxc

This is an example of how to use the [SysFileClose](#) function along with the [FileClose-Type](#) structure.

```
task main()
{
  FileCloseType fcArgs;
  fcArgs.FileHandle = foArgs.FileHandle;
  SysFileClose(fcArgs);
}
```

### 9.842 ex\_sysfiledelete.nxc

This is an example of how to use the [SysFileDelete](#) function along with the [FileDeleteType](#) structure.

```
task main()
{
  FileDeleteType fdArgs;
  fdArgs.Filename = "myfile.txt";
  SysFileDelete(fdArgs); // delete the file
}
```

### 9.843 ex\_sysfilefindfirst.nxc

This is an example of how to use the [SysFileFindFirst](#) function along with the [FileFindType](#) structure.

```
task main()
{
  FileFindType args;
  args.Filename = "*.*";
  SysFileFindFirst(args);
  TextOut(0, LCD_LINE1, args.Filename);
}
```

### 9.844 ex\_sysfilefindnext.nxc

This is an example of how to use the [SysFileFindNext](#) function along with the [FileFindType](#) structure.

```
task main()
{
  FileFindType args;
  args.FileHandle = prev.FileHandle;
  SysFileFindNext(args);
  TextOut(0, LCD_LINE1, args.Filename);
}
```

### 9.845 ex\_sysfileopenappend.nxc

This is an example of how to use the [SysFileOpenAppend](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    SysFileOpenAppend(foArgs); // open the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
        // up to the remaining available length in Length
    }
}
```

### 9.846 ex\_sysfileopenread.nxc

This is an example of how to use the [SysFileOpenRead](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    SysFileOpenRead(foArgs); // open the file for reading
    if (foArgs.Result == NO_ERR) {
        // read data from the file using FileHandle
    }
}
```

### 9.847 ex\_sysfileopenreadlinear.nxc

This is an example of how to use the [SysFileOpenReadLinear](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.rxe";
    SysFileOpenReadLinear(foArgs); // open the file for
    reading
    if (foArgs.Result == NO_ERR) {
        // read data from the file using FileHandle
    }
}
```

### 9.848 ex\_sysfileopenwrite.nxc

This is an example of how to use the [SysFileOpenWrite](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    foArgs.Length = 256; // create with capacity for 256 bytes
    SysFileOpenWrite(foArgs); // create the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
    }
}
```

### 9.849 ex\_sysfileopenwritelinear.nxc

This is an example of how to use the [SysFileOpenWriteLinear](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    foArgs.Length = 256; // create with capacity for 256 bytes
    SysFileOpenWriteLinear(foArgs); // create the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
    }
}
```

### 9.850 ex\_sysfileopenwritenonlinear.nxc

This is an example of how to use the [SysFileOpenWriteNonLinear](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    foArgs.Length = 256; // create with capacity for 256 bytes
    SysFileOpenWriteNonLinear(foArgs); // create the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
    }
}
```

### 9.851 ex\_sysfileread.nxc

This is an example of how to use the [SysFileRead](#) function along with the [FileReadWriteType](#) structure.

```
task main()
{
    FileReadWriteType frArgs;
    frArgs.FileHandle = foArgs.FileHandle;
    frArgs.Length = 12; // number of bytes to read
    SysFileRead(frArgs);
    if (frArgs.Result == NO_ERR) {
        TextOut(0, LCD_LINE1, frArgs.Buffer);
        // show how many bytes were actually read
        NumOut(0, LCD_LINE2, frArgs.Length);
    }
}
```

### 9.852 ex\_sysfilerename.nxc

This is an example of how to use the [SysFileRename](#) function along with the [FileRenameType](#) structure.

```
task main()
{
    FileRenameType frArgs;
    frArgs.OldFilename = "myfile.txt";
    frArgs.NewFilename = "myfile2.txt";
    SysFileRename(frArgs);
    if (frArgs.Result == LDR_SUCCESS) { /* do something */ }
}
```

### 9.853 ex\_sysfileresize.nxc

This is an example of how to use the [SysFileResize](#) function along with the [FileResizeType](#) structure.

```
task main()
{
    byte handle;
    // get a file handle
    // ...
    // resize the file
    FileResizeType args;
    args.FileHandle = handle;
    args.NewSize = 2048;
    SysFileResize(args);
    if (args.Result == NO_ERR)
    {
        // do something
    }
}
```

```
}  
}
```

### 9.854 ex\_sysfileresolvehandle.nxc

This is an example of how to use the [SysFileResolveHandle](#) function along with the [FileResolveHandleType](#) structure.

```
task main()  
{  
    FileResolveHandleType frhArgs;  
    frhArgs.Filename = "myfile.txt";  
    SysFileResolveHandle(frhArgs);  
    if (frhArgs.Result == LDR_SUCCESS) {  
        // use the FileHandle as needed  
        if (frhArgs.WriteHandle) {  
            // file is open for writing  
        }  
        else {  
            // file is open for reading  
        }  
    }  
}
```

### 9.855 ex\_sysfileseek.nxc

This is an example of how to use the [SysFileSeek](#) function along with the [FileSeekType](#) structure.

```
task main()  
{  
    byte handle;  
    // get a file handle  
    // ...  
    FileSeekType args;  
    args.FileHandle = handle;  
    args.Origin = SEEK_SET;  
    args.Length = 65;  
    SysFileSeek(args);  
    if (args.Result == NO_ERR)  
    {  
        // do something  
    }  
}
```

### 9.856 ex\_sysfilewrite.nxc

This is an example of how to use the [SysFileWrite](#) function along with the [FileRead-WriteType](#) structure.

```

task main()
{
  FileReadWriteType fwArgs;
  fwArgs.FileHandle = foArgs.FileHandle;
  fwArgs.Buffer = "data to write";
  SysFileWrite(fwArgs);
  if (fwArgs.Result == NO_ERR) {
    // display number of bytes written
    NumOut(0, LCD_LINE1, fwArgs.Length);
  }
}

```

### 9.857 ex\_sysgetstarttick.nxc

This is an example of how to use the [SysGetStartTick](#) function along with the [GetStartTickType](#) structure.

```

task main()
{
  GetStartTickType gstArgs;
  SysGetStartTick(gstArgs);
  unsigned long myStart = gstArgs.Result;
}

```

### 9.858 ex\_sysinputpinfunction.nxc

This is an example of how to use the [SysInputPinFunction](#) function along with the [InputPinFunctionType](#) structure.

```

#ifdef OLD_COMPILER

struct InputPinFunctionType {
  unsigned int Result; // The function call result. Possible return values are ER
    R_INVALID_PORT or NO_ERR.
  byte Cmd;           // The command to execute. See \ref InputPinFuncConstants.
    You can add a microsecond wait after the command by ORing INPUT_PINCMD_WAIT(use
    c) with the command Value. Wait times can range from 1 to 63 microseconds.
  byte Port;          // The input port. See \ref InPorts.
  byte Pin;           // The digital pin(s). See \ref InputDigiPinConstants. Whe
    n setting pin direction you must OR the desired direction constant into this fiel
    d. See INPUT_PINDIR_INPUT and INPUT_PINDIR_OUTPUT from the \r
    ef InputPinFuncConstants group. You can OR together the digital pin constants to
    operate on both in a single call.
  byte Data;          // The pin value(s). This field is only used by the INPUT_
    PINCMD_READ command.
};

#define INPUT_PINCMD_DIR    0x00 // Set digital pin(s) direction
#define INPUT_PINCMD_SET    0x01 // Set digital pin(s)
#define INPUT_PINCMD_CLEAR  0x02 // Clear digital pin(s)
#define INPUT_PINCMD_READ   0x03 // Read digital pin(s)

```

```

#define INPUT_PINCMD_MASK    0x03 // Mask for the two bits used by pin function co
                                mmands
#define INPUT_PINCMD_WAIT(_usec) ((_usec)<<2) // A wait value in microseconds tha
                                t can be added after one of the above commands by ORing with the command
#define INPUT_PINDIR_OUTPUT 0x00 // Use with the direction command to set directi
                                on to input.  OR this with the pin value.
#define INPUT_PINDIR_INPUT  0x04 // Use with the direction command to set directi
                                on to output.  OR this with the pin value.

#define InputPinFunction 77

#endif

task main()
{
    InputPinFunctionType pftSet, pftClear, pftDir;

    // use these parameters to set the pin direction
    pftDir.Port = S1;
    pftDir.Pin = INPUT_DIGI0|INPUT_PINDIR_OUTPUT;
    pftDir.Cmd = INPUT_PINCMD_DIR;

    // use these parameters to SET the pin
    pftSet.Port = S1;
    pftSet.Pin = INPUT_DIGI0;
    pftSet.Cmd = INPUT_PINCMD_SET|INPUT_PINCMD_WAIT(2);

    // use these parameters to CLEAR the pin
    pftClear.Port = S1;
    pftClear.Pin = INPUT_DIGI0;
    pftClear.Cmd = INPUT_PINCMD_CLEAR|INPUT_PINCMD_WAIT(30);

    SysInputPinFunction(pftDir); // set the direction to output

    while(true)
    {
        SysInputPinFunction(pftSet);
        SysInputPinFunction(pftClear);
    }
}

```

## 9.859 ex\_sysiomapread.nxc

This is an example of how to use the [SysIOMapRead](#) function along with the [IOMapReadType](#) structure.

```

task main()
{
    IOMapReadType args;
    args.ModuleName = CommandModuleName;
    args.Offset = CommandOffsetTick;
    args.Count = 4; // this value happens to be 4 bytes long
    SysIOMapRead(args);
    if (args.Result == NO_ERR) { /* do something with data */ }
}

```

```
}
```

### 9.860 ex\_sysiomapreadbyid.nxc

This is an example of how to use the [SysIOMapReadByID](#) function along with the [IOMapReadByIDType](#) structure.

```
task main()
{
    IOMapReadByIDType args;
    args.ModuleID = CommandModuleID;
    args.Offset = CommandOffsetTick;
    args.Count = 4; // this value happens to be 4 bytes long
    SysIOMapReadByID(args);
    if (args.Result == NO_ERR) { /* do something with data */ }
}
```

### 9.861 ex\_sysiomapwrite.nxc

This is an example of how to use the [SysIOMapWrite](#) function along with the [IOMapWriteType](#) structure.

```
task main()
{
    IOMapWriteType args;
    args.ModuleName = SoundModuleName;
    args.Offset = SoundOffsetSampleRate;
    args.Buffer = theData;
    SysIOMapWrite(args);
}
```

### 9.862 ex\_sysiomapwritebyid.nxc

This is an example of how to use the [SysIOMapWriteByID](#) function along with the [IOMapWriteByIDType](#) structure.

```
task main()
{
    IOMapWriteByIDType args;
    args.ModuleID = SoundModuleID;
    args.Offset = SoundOffsetSampleRate;
    args.Buffer = theData;
    SysIOMapWriteByID(args);
}
```

### 9.863 ex\_syskeepalive.nxc

This is an example of how to use the [SysKeepAlive](#) function along with the [KeepAliveType](#) structure.

```
task main()
{
    KeepAliveType kaArgs;
    SysKeepAlive(kaArgs); // reset sleep timer
}
```

### 9.864 ex\_syslistfiles.nxc

This is an example of how to use the [SysListFiles](#) function along with the [ListFilesType](#) structure.

```
task main()
{
    ListFilesType args;
    args.Pattern = "*.rxex";
    SysListFiles(args);
    if (args.Result == NO_ERR && ArrayLen(args.FileList) > 0)
    {
        TextOut(0, LCD_LINE6, args.FileList[0]);
    }
    Wait(SEC_4);
}
```

### 9.865 ex\_sysloaderexecutefunction.nxc

This is an example of how to use the [SysLoaderExecuteFunction](#) function along with the [LoaderExecuteFunctionType](#) structure.

```
task main()
{
    LoaderExecuteFunctionType args;
    args.Cmd = 0xA0; // delete user flash
    SysLoaderExecuteFunction(args);
}
```

### 9.866 ex\_sysmemorymanager.nxc

This is an example of how to use the [SysMemoryManager](#) function along with the [MemoryManagerType](#) structure.

```
task main() {
    byte data[];
```

```

byte data2[];
int data3[];
int ps, ds;
MemoryManagerType args;
args.Compact = false;
SysMemoryManager(args);
NumOut(0, LCD_LINE1, args.PoolSize);
NumOut(0, LCD_LINE2, args.DataspaceSize);
Wait(SEC_5);
ClearScreen();
Wait(SEC_1);
ArrayInit(data, 10, 3000);
data[10]++;
ps = data[10];
data2 = data;
ArrayBuild(data3, ps, ds, ps, ds, ps, ds, ps, ds);
SysMemoryManager(args);
NumOut(0, LCD_LINE1, args.PoolSize);
NumOut(0, LCD_LINE2, args.DataspaceSize);
NumOut(0, LCD_LINE8, data[10]);
Wait(SEC_5);
ClearScreen();
Wait(SEC_1);
SysMemoryManager(args);
NumOut(0, LCD_LINE1, args.PoolSize);
NumOut(0, LCD_LINE2, args.DataspaceSize);
NumOut(0, LCD_LINE8, data2[10]);
Wait(SEC_5);
ClearScreen();
Wait(SEC_1);
// while(true);
}

```

### 9.867 ex\_sysmessageread.nxc

This is an example of how to use the [SysMessageRead](#) function along with the [MessageReadType](#) structure.

```

task main()
{
    MessageReadType args;
    args.QueueID = MAILBOX1; // 0
    args.Remove = true;
    SysMessageRead(args);
    if (args.Result == NO_ERR) {
        TextOut(0, LCD_LINE1, args.Message);
    }
}

```

### 9.868 ex\_sysmessagewrite.nxc

This is an example of how to use the [SysMessageWrite](#) function along with the [MessageWriteType](#) structure.

```
task main()
{
    MessageWriteType args;
    args.QueueID = MAILBOX1; // 0
    args.Message = "testing";
    SysMessageWrite(args);
    // check Result for error status
}
```

### 9.869 ex\_sysrandomex.nxc

This is an example of how to use the [SysRandomEx](#) function along with the [RandomExType](#) structure.

```
task main()
{
    RandomExType rnArgs;
    SysRandomEx(rnArgs);
    unsigned long myRandomValue = rnArgs.Seed;
}
```

### 9.870 ex\_sysrandomnumber.nxc

This is an example of how to use the [SysRandomNumber](#) function along with the [RandomNumberType](#) structure.

```
task main()
{
    RandomNumberType rnArgs;
    SysRandomNumber(rnArgs);
    int myRandomValue = rnArgs.Result;
}
```

### 9.871 ex\_sysreadbutton.nxc

This is an example of how to use the [SysReadButton](#) function along with the [ReadButtonType](#) structure.

```
task main()
{
    ReadButtonType rbArgs;
    rbArgs.Index = BTNRIGHT;
    SysReadButton(rbArgs);
    if (rbArgs.Pressed) { /* do something */ }
}
```

### 9.872 ex\_SysReadLastResponse.nxc

This is an example of how to use the [SysReadLastResponse](#) function.

```
ReadLastResponseType args;
args.Clear = true;
SysReadLastResponse(args);
if (args.Result == NO_ERR) {
    NumOut(0, LCD_LINE1, args.Length);
    NumOut(0, LCD_LINE2, args.Command);
    // also could output args.Buffer[i]
}
```

### 9.873 ex\_SysReadSemData.nxc

This is an example of how to use the [SysReadSemData](#) function along with the [ReadSemDataType](#) structure.

```
task main()
{
    ReadSemDataType args;
    args.Request = true;
    SysReadSemData(args);
    NumOut(0, LCD_LINE1, args.SemData);
}
```

### 9.874 ex\_syssetscreenmode.nxc

This is an example of how to use the [SysSetScreenMode](#) function along with the [SetScreenModeType](#) structure.

```
task main()
{
    SetScreenModeType ssmArgs;
    ssmArgs.ScreenMode = 0x00; // restore default NXT screen
    SysSetScreenMode(ssmArgs);
}
```

### 9.875 ex\_SysSetSleepTimeout.nxc

This is an example of how to use the [SysSetSleepTimeout](#) function.

```
task main()
{
    SetSleepTimeoutType args;
    args.TheSleepTimeoutMS = MIN_1*5;
    SysSetSleepTimeout(args);
}
```

### 9.876 ex\_syssoundgetstate.nxc

This is an example of how to use the [SysSoundGetState](#) function along with the [SoundGetStateType](#) structure.

```
task main()
{
    SoundGetStateType sgsArgs;
    SysSoundGetState(sgsArgs);
    if (sgsArgs.State == SOUND_STATE_IDLE) { /* do stuff */}
}
```

### 9.877 ex\_syssoundplayfile.nxc

This is an example of how to use the [SysSoundPlayFile](#) function along with the [SoundPlayFileType](#) structure.

```
task main()
{
    SoundPlayFileType spfArgs;
    spfArgs.Filename = "hello.rso";
    spfArgs.Loop = false;
    spfArgs.SoundLevel = 3;
    SysSoundPlayFile(spfArgs);
}
```

### 9.878 ex\_syssoundplaytone.nxc

This is an example of how to use the [SysSoundPlayTone](#) function along with the [SoundPlayToneType](#) structure.

```
task main()
{
    SoundPlayToneType sptArgs;
    sptArgs.Frequency = 440;
    sptArgs.Duration = 1000; // 1 second
    sptArgs.Loop = false;
    sptArgs.SoundLevel = 3;
    SysSoundPlayTone(sptArgs);
}
```

### 9.879 ex\_syssoundsetstate.nxc

This is an example of how to use the [SysSoundSetState](#) function along with the [SoundSetStateType](#) structure.

```
task main()
```

```
{
    SoundSetStateType sssArgs;
    sssArgs.State = SOUND_STATE_STOP;
    SysSoundSetState(sssArgs);
}
```

### 9.880 ex\_SysUpdateCalibCacheInfo.nxc

This is an example of how to use the [SysUpdateCalibCacheInfo](#) function along with the [UpdateCalibCacheInfoType](#) structure.

```
task main()
{
    UpdateCalibCacheInfoType args;
    args.Name = "light";
    args.MinVal = 0;
    args.MaxVal = 1023;
    SysUpdateCalibCacheInfo(args);
    NumOut(0, LCD_LINE1, args.Result);
}
```

### 9.881 ex\_SysWriteSemData.nxc

This is an example of how to use the [SysWriteSemData](#) function along with the [WriteSemDataType](#) structure.

```
task main()
{
    WriteSemDataType args;
    args.NewVal = 0x4;
    args.Request = true;
    args.ClearBits = false;
    SysWriteSemData(args);
    NumOut(0, LCD_LINE1, args.SemData);
}
```

### 9.882 ex\_tan.nxc

This is an example of how to use the [tan](#) function.

```
// ex_tan.nxc
// Display values generated by the tan API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define DELTA PI / 8

// Angles from -3/8 PI radians to almost P1/2 radians stepped by PI/8 radians.
const float data[] =
```

```

{
  -3 * DELTA,
  -2 * DELTA,
  -DELTA,
  0.0,
  DELTA,
  2 * DELTA,
  3 * DELTA,
  4 * DELTA - 0.01
};

// Display a table of angles and their tangents. The angles are the ones
// specified above.
task main()
{
  const int items = ArrayLen(data);
  for (int i = 0; i < items; ++i)
  {
    int screen_y = 56 - 8 * i;
    float angle = data[i];
    TextOut(0, screen_y, FormatNum("%7.4f", angle));
    TextOut(45, screen_y, FormatNum("%8.4f", tan(angle)));
  }
  while (true);
}

```

### 9.883 ex\_tand.nxc

This is an example of how to use the [tand](#) function.

```

// ex_tand.nxc
// Display values generated by the tand API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define DELTA 22.5

// Angles from -67.5 degrees to almost 90.0 degrees stepped by 22.5 degrees.
const float data[] =
{
  -3 * DELTA,
  -2 * DELTA,
  -DELTA,
  0.0,
  DELTA,
  2 * DELTA,
  3 * DELTA,
  4 * DELTA - 0.01
};

// Display a table of angles and their tangents. The angles are the ones
// specified above.
task main()
{
  const int items = ArrayLen(data);

```

```
for (int i = 0; i < items; ++i)
{
    int screen_y = 56 - 8 * i;
    float angle = data[i];
    TextOut(0, screen_y, FormatNum("%5.1f", angle));
    TextOut(40, screen_y, FormatNum("%8.4f", tand(angle)));
}
while (true);
}
```

### 9.884 ex\_tanh.nxc

This is an example of how to use the [tanh](#) function.

```
x = tanh(y);
```

### 9.885 ex\_TextOut.nxc

This is an example of how to use the [TextOut](#) function.

```
TextOut(0, LCD_LINE3, "Hello World!");
```

### 9.886 ex\_tolower.nxc

This is an example of how to use the [tolower](#) function.

```
i = tolower(x);
```

### 9.887 ex\_toupper.nxc

This is an example of how to use the [toupper](#) function.

```
i = toupper(x);
```

### 9.888 ex\_trunc.nxc

This is an example of how to use the [trunc](#) function.

```
y = trunc(x);
```

### 9.889 ex\_UIButton.nxc

This is an example of how to use the [UIButton](#) function.

```
x = UIButton();
```

### 9.890 ex\_UIState.nxc

This is an example of how to use the [UIState](#) function.

```
x = UIState();
```

### 9.891 ex\_UiUsbState.nxc

This is an example of how to use the [UsbState](#) function.

```
value = UsbState();
```

### 9.892 ex\_UnflattenVar.nxc

This is an example of how to use the [UnflattenVar](#) function.

```
task main()
{
  long data[] = {-50123, 68142, 128176, -45123};
  long data2[4];
  float fdata[] = {12.123, 3.14159, 2.68};
  float fdata2[3];
  NumOut(0, LCD_LINE1, data[0]);
  NumOut(0, LCD_LINE2, fdata[1]);
  string sdata = FlattenVar(data);
  string tmp;
  // transfer the string to another NXT
  tmp = sdata;
  UnflattenVar(tmp, data2);
  NumOut(0, LCD_LINE3, data2[0]);
  sdata = FlattenVar(fdata);
  // transfer the string to another NXT
  tmp = sdata;
  UnflattenVar(tmp, fdata2);
  NumOut(0, LCD_LINE4, fdata2[1]);
  Wait(SEC_5);
}
```

**9.893 ex\_USBInputBufferInPtr.nxc**

This is an example of how to use the [USBInputBufferInPtr](#) function.

```
byte x = USBInputBufferInPtr();
```

**9.894 ex\_USBInputBufferOutPtr.nxc**

This is an example of how to use the [USBInputBufferOutPtr](#) function.

```
byte x = USBInputBufferOutPtr();
```

**9.895 ex\_USBOutputBufferInPtr.nxc**

This is an example of how to use the [USBOutputBufferInPtr](#) function.

```
byte x = USBOutputBufferInPtr();
```

**9.896 ex\_USBOutputBufferOutPtr.nxc**

This is an example of how to use the [USBOutputBufferOutPtr](#) function.

```
byte x = USBOutputBufferOutPtr();
```

**9.897 ex\_USBPollBufferInPtr.nxc**

This is an example of how to use the [USBPollBufferInPtr](#) function.

```
byte x = USBPollBufferInPtr();
```

**9.898 ex\_USBPollBufferOutPtr.nxc**

This is an example of how to use the [USBPollBufferOutPtr](#) function.

```
byte x = USBPollBufferOutPtr();
```

### 9.899 ex\_UsbState.nxc

This is an example of how to use the [USBState](#) function.

```
byte x = USBPollBufferOutPtr();
```

### 9.900 ex\_VMRunState.nxc

This is an example of how to use the [VMRunState](#) function.

```
x = VMRunState();
```

### 9.901 ex\_Volume.nxc

This is an example of how to use the [Volume](#) function.

```
x = Volume();
```

### 9.902 ex\_wait.nxc

This is an example of how to use the [Wait](#) function.

```
task main()
{
    Wait(SEC_5); // wait 5 seconds
    Wait(Random(SEC_1)); // wait random time up to 1 second
}
```

### 9.903 ex\_Write.nxc

This is an example of how to use the [Write](#) function.

```
result = Write(handle, value);
```

### 9.904 ex\_WriteBytes.nxc

This is an example of how to use the [WriteBytes](#) function.

```
result = WriteBytes(handle, buffer, count);
```

### 9.905 ex\_WriteBytesEx.nxc

This is an example of how to use the [WriteBytesEx](#) function.

```
result = WriteBytesEx(handle, len, buffer);
```

### 9.906 ex\_writei2cregister.nxc

This is an example of how to use the [WriteI2CRegister](#) function.

```
char result = WriteI2CRegister(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, US_CMD_OFF);
```

### 9.907 ex\_WriteLn.nxc

This is an example of how to use the [WriteLn](#) function.

```
result = WriteLn(handle, value);
```

### 9.908 ex\_WriteLnString.nxc

This is an example of how to use the [WriteLnString](#) function.

```
result = WriteLnString(handle, "testing", count);
```

### 9.909 ex\_writenrlinkbytes.nxc

This is an example of how to use the [WriteNRLinkBytes](#) function.

```
byte data[] = {0x01, 0x02, 0x03};  
char result = WriteNRLinkBytes(S1, MS_ADDR_NRLINK, data);
```

### 9.910 ex\_WriteString.nxc

This is an example of how to use the [WriteString](#) function.

```
result = WriteString(handle, "testing", count);
```

## 9.911 ex\_xg1300.nxc

This is an example of how to use the [ResetMIXG1300L](#), [SetSensorMIXG1300LScale](#), [SensorMIXG1300LScale](#), and [ReadSensorMIXG1300L](#) functions.

```
/*
struct XGPacketType {
    int AccAngle;
    int TurnRate;
    int XAxis;
    int YAxis;
    int ZAxis;
};
*/
task main()
{
    XGPacketType data;
    string msg;
    ReadButtonType rbArgs;
    int i = 0;
    //Initialize system
    SetSensorLowspeed(S1);
    //Resets sensor and waits for hardware to settle
    ResetMIXG1300L(S1);
    Wait(MS_500);
    //Main loop
    while (true)
    {
        ClearScreen();
        if (ReadSensorMIXG1300L(S1, data))
        {
            TextOut(0, LCD_LINE1, "<<RESET SENSOR", false);
            //Print sensor data
            TextOut(0, LCD_LINE2, "ANGLE:");
            NumOut(40, LCD_LINE2, data.AccAngle);
            TextOut(0, LCD_LINE3, "RATE:");
            NumOut(40, LCD_LINE3, data.TurnRate);
            TextOut(0, LCD_LINE6, "ACC_X:");
            NumOut(40, LCD_LINE6, data.XAxis);
            TextOut(0, LCD_LINE7, "ACC_Y:");
            NumOut(40, LCD_LINE7, data.YAxis);
            TextOut(0, LCD_LINE8, "ACC_Z:");
            NumOut(40, LCD_LINE8, data.ZAxis);
            //Reset sensor if user presses the left key
            rbArgs.Index = BTNLEFT;
            SysReadButton(rbArgs);
            if (rbArgs.Pressed)
            {
                ResetMIXG1300L(S1);
                i = 0;
                TextOut(0, LCD_LINE1, "Resetting Device ", false);
                Wait(MS_500);
            }
            //Change accelerometer range if user presses the right key
            rbArgs.Index = BTNRIGHT;
            SysReadButton(rbArgs);
        }
    }
}
```

```
    if (rbArgs.Pressed)
    {
        i++;
        i%=3;
        TextOut(0, LCD_LINE5, "Range == " , false);
        if (i == 0) {
            SetSensorMIXG1300LScale(S1, XG1300L_SCALE_2G);
        }
        else if (i == 1) {
            SetSensorMIXG1300LScale(S1, XG1300L_SCALE_4G);
        }
        else
            SetSensorMIXG1300LScale(S1, XG1300L_SCALE_8G);
        NumOut(60, LCD_LINE5, SensorMIXG1300LScale(S1));
        Wait(MS_500);
    }
    Wait(MS_100);
}
}
```

## 9.912 ex\_yield.nxc

This is an example of how to use the [Yield](#) function.

```
task play() {
    while (true) {
        PlayTone(TONE_A4, MS_500);
        Wait(SEC_1);
    }
}

task drive()
{
    while (true) {
        OnFwd(OUT_A, 50);
        Yield();
    }
}

task main()
{
    Precedes(drive, play);
}
```

## 9.913 glBoxDemo.nxc

This is an example of how to use the [glInit](#), [glBeginObject](#), [glBegin](#), [glAddVertex](#), [glEnd](#), [glEndObject](#), [glSetAngleX](#), [glBeginRender](#), [glAddToAngleY](#), [glCallObject](#), and [glFinishRender](#) functions.

```

/*-----*/
; File      : glBoxDemo.nbc
; Description : A program demonstrating a 3D box...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----*/

task main()
{
    // Initialize the 3D engine...
    glInit();
    // Create a cube, this is the first object which will be object id 0...
    glCube(GL_POLYGON, 20);
    glBox(GL_POLYGON, 20, 30, 40);
    // Set the main view x-angle...
    glSetAngleX(45);
    while (true)
    {
        // Rotate the main view....
        glAddToAngleY(8);
        glAddToAngleX(4);
        // Setup for rendering...
        glBeginRender();
        // translate object 1
        glObjectAction(0, GL_TRANSLATE_X, 20);
        glObjectAction(1, GL_TRANSLATE_X, -20);
        // Call the object with id 0...
        glCallObject(0);
        glCallObject(1);
        // Finish, clear the screen, rotate and render the called objects...
        glFinishRender();
        Wait(MS_20);
    }
}

```

## 9.914 glCircleDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glBeginRender](#), [glAddToAngleY](#), [glAddToAngleX](#), [glCallObject](#), [glSet](#), and [glFinishRender](#) functions.

```

/*-----*/
; File      : glBoxDemo.nbc
; Description : A program demonstrating a 3D box with circles on the edges...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----*/

task main()
{
    // Initialize the 3D engine...
    glInit();
    // Create a box, this is the first object which will be object id 0...
    glBox(GL_POLYGON, 30, 30, 30);
    // Create a box, this is the second object which will be object id 1...
    glBox(GL_CIRCLE, 30, 30, 30);
    // Set the main view x-angle...
    glSetAngleX(45);
}

```

```

while (true)
{
    // Rotate the main view....
    glAddToAngleY(3);
    glAddToAngleX(5);
    // Setup for rendering...
    glBeginRender();
    glSet(GL_CULL_MODE, GL_CULL_NONE);
    // Call the object with id 0...
    glCallObject(0);
    // Call the object with id 1...
    glCallObject(1);
    // Finish, clear the screen, rotate and render the called objects...
    glFinishRender();
}
}

```

### 9.915 glRotateDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glBeginRender](#), [glCallObject](#), [glObjectAction](#), and [glFinishRender](#) functions.

```

/*-----
; File      : glRotateDemo.nbc
; Description : A program demonstrating two 3D boxes with rotate actions...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----*/

int angleX, angleY;

task main()
{
    // Initialize the 3D engine...
    glInit();
    // Create a box, this is the first object which will be object id 0...
    glBox(GL_POLYGON, 20, 20, 20);
    // Create a box, this is the second object which will be object id 1...
    glBox(GL_POLYGON, 40, 40, 40);
    glSetAngleX(30);
    angleX = 0;
    angleY = 0;
    while (true) {
        // Setup for rendering...
        glBeginRender();
        // Call the object with id 0...
        glCallObject(0);
        glObjectAction(0, GL_ROTATE_X, angleX);
        // Call the object with id 1...
        glCallObject(1);
        glObjectAction(1, GL_ROTATE_Y, angleY);
        // Finish, clear the screen, rotate and render the called objects...
        glFinishRender();
        angleX += 3;
        angleX %= 360;
        angleY += 5;
    }
}

```

```

    angleY %= 360;
  }
}

```

## 9.916 glScaleDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glAddToAngleY](#), [glBeginRender](#), [glCallObject](#), [glObjectAction](#), and [glFinishRender](#) functions.

```

/*-----
; File          : glScaleDemo.nxc
; Description   : A program demonstrating a scaling action...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----*/

int scaleX, scaleXStep;

task main()
{
  // Initialize the 3D engine...
  glInit();
  // Create a box, this is the first object which will be object id 0...
  glBox(GL_POLYGON, 20, 20, 20);
  // Set the main view x-angle...
  glSetAngleX(45);
  // Initialize the scaling vars...
  scaleX    = 256;
  scaleXStep = 16;
  while (true)
  {
    // Rotate the main view...
    glAddToAngleY(8);
    // Setup for rendering...
    glBeginRender();
    // Call the object with id 0...
    glObjectAction(0, GL_SCALE_X, scaleX);
    glCallObject(0);
    // Finish, clear the screen, rotate and render the called objects...
    glFinishRender();
    // Scale between 256..512...
    scaleX += scaleXStep;
    if (scaleX >= 512)
      scaleXStep = -16;
    else if (scaleX <= 256)
      scaleXStep = 16;
    Wait(MS_20);
  }
}

```

## 9.917 glTranslateDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glAddToAngleY](#), [glBeginRender](#), [glSet](#), [glCallObject](#), [glObjectAction](#), and [glFinishRender](#) functions.

```
/*-----*/
; File      : glTranslateDemo.nbc
; Description : A program demonstrating two 3D boxes with translate actions...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----*/

int translateX, translateXStep,
    translateY, translateYStep,
    translateZ, translateZStep;

task main()
{
    // Initialize the 3D engine...
    glInit();

    // Create a box, this is the first object which will be object id 0...
    glBox(GL_POLYGON, 20, 20, 20);

    // Create a box, this is the second object which will be object id 1...
    glBox(GL_POLYGON, 40, 40, 40);

    glSetAngleX(30);

    translateX = 0;
    translateXStep = 1;
    translateY = 0;
    translateYStep = 1;
    translateZ = 0;
    translateZStep = 2;
    while (true)
    {
        // Rotate the main view....
        glAddToAngleY(2);
        // Setup for rendering...
        glBeginRender();
        glSet(GL_CULL_MODE, GL_CULL_NONE);

        // Call the object with id 0...
        glCallObject(0);
        glObjectAction(0, GL_TRANSLATE_X, translateX);
        glObjectAction(0, GL_TRANSLATE_Z, translateZ);
        glObjectAction(0, GL_TRANSLATE_Y, translateY);

        // Call the object with id 1...
        glCallObject(1);

        // Finish, clear the screen, rotate and render the called objects...
        glFinishRender();

        translateX += translateXStep;
        if (translateX >= 10)
            translateXStep = -1;
        else if (translateX <= -10)
            translateXStep = 1;

        translateZ += translateZStep;
        if (translateZ >= 10)

```

```
    translateZStep = -2;
    else if (translateZ <= -10)
        translateZStep = 2;

    translateY += translateYStep;
    if (translateY >= 10)
        translateYStep = -1;
    else if (translateY <= -10)
        translateYStep = 1;
}
}
```

### 9.918 util\_battery\_1.nxc

This is an example of how to use the [BatteryLevel](#) function.

```
// Display battery voltage for four seconds.
// This version does not use floats and will work on NXTs with firmware 1.07.

// Note: the BatteryLevel API call returns an unsigned integer giving the
// battery level in millivolts.
task main()
{
    unsigned int millivolts = BatteryLevel();
    unsigned int int_volts = millivolts / 1000;
    millivolts %= 1000;
    string left = FormatNum("Battery: %d", int_volts);
    string right = FormatNum(":%03d", millivolts);
    TextOut(0, LCD_LINE2, StrCat(left, right), true);
    Wait(SEC_4);
}
```

### 9.919 util\_battery\_2.nxc

This is an example of how to use the [BatteryLevel](#) function.

```
// Display battery voltage for four seconds.
// This version uses floats and requires NXTs with firmware 1.28 or later.

// Note: the BatteryLevel API call returns an unsigned integer giving the
// battery level in millivolts.
task main()
{
    float volts = BatteryLevel() / 1000.0;
    TextOut(0, LCD_LINE2, FormatNum("Battery: %5.3f", volts), true);
    Wait(SEC_4);
}
```

## 9.920 util\_rpm.nxc

This is an example of how to use the [CurrentTick](#) and [MotorRotationCount](#) functions.

```
// Display RPM of motor attached to the port MOTOR while running at full speed.
// The program runs continuously until stopped by pressing the gray NXT button.
// Requires NXT firmware 1.28 or later (uses floating point arithmetic).

// CurrentTick returns milliseconds in a long integer.
// MotorRotationCount returns degrees in a long integer.

#define MOTOR OUT_A
#define FULL_SPEED 100
#define DEG_TO_RPM 166.6667 // converts degrees per millisecond to RPM

long prev_tick;
long prev_deg = 0;

string rpm_msg()
{
    long dt = CurrentTick() - prev_tick;
    long deg = MotorRotationCount(MOTOR) - prev_deg;
    float rpm = deg * DEG_TO_RPM / dt;
    prev_deg = MotorRotationCount(MOTOR);
    prev_tick = CurrentTick();
    return FormatNum("RPM: %5.1f", rpm);
}

task main()
{
    prev_tick = CurrentTick();
    OnFwd(MOTOR, FULL_SPEED);
    while (true)
    {
        Wait(MS_500); // update display every 0.5 seconds
        TextOut(0, LCD_LINE2, rpm_msg(), true);
    }
}
```

## Index

- `_SENSOR_CFG`
  - `NXCDefs.h`, 1480
  - `SensorTypeModes`, 282
- A simple 3D graphics library, 264
- abort
  - `cstdlibAPI`, 646
  - `NXCDefs.h`, 1503
- AbortFlag
  - `NXCDefs.h`, 1504
  - `UiModuleFunctions`, 561
- abs
  - `cstdlibAPI`, 646
  - `NXCDefs.h`, 1504
- AccAngle
  - `XGPacketType`, 1114
- `ACCL_CMD_RESET_CAL`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_X_CAL`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_X_CAL_END`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_Y_CAL`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_Y_CAL_END`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_Z_CAL`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_CMD_Z_CAL_END`
  - `MSACCLNx`, 949
  - `NBCCCommon.h`, 1163
- `ACCL_REG_SENS_LVL`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_X_ACCEL`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_X_OFFSET`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_X_RANGE`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_X_TILT`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Y_ACCEL`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Y_OFFSET`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Y_RANGE`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Y_TILT`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Z_ACCEL`
  - `MSACCLNx`, 950
  - `NBCCCommon.h`, 1164
- `ACCL_REG_Z_OFFSET`
  - `MSACCLNx`, 951
  - `NBCCCommon.h`, 1165
- `ACCL_REG_Z_RANGE`
  - `MSACCLNx`, 951
  - `NBCCCommon.h`, 1165
- `ACCL_REG_Z_TILT`
  - `MSACCLNx`, 951
  - `NBCCCommon.h`, 1165
- `ACCL_SENSITIVITY_LEVEL_1`
  - `MSACCLNxSLevel`, 951
  - `NBCCCommon.h`, 1165
- `ACCL_SENSITIVITY_LEVEL_2`
  - `MSACCLNxSLevel`, 951
  - `NBCCCommon.h`, 1165
- `ACCL_SENSITIVITY_LEVEL_3`
  - `MSACCLNxSLevel`, 952
  - `NBCCCommon.h`, 1165

- ACCL\_SENSITIVITY\_LEVEL\_4
  - MSACCLNxSLevel, 952
  - NBCCCommon.h, 1165
- ACCLNxCalibrateX
  - MindSensorsAPI, 143
  - NXCDefs.h, 1504
- ACCLNxCalibrateXEnd
  - MindSensorsAPI, 144
  - NXCDefs.h, 1505
- ACCLNxCalibrateY
  - MindSensorsAPI, 144
  - NXCDefs.h, 1505
- ACCLNxCalibrateYEnd
  - MindSensorsAPI, 145
  - NXCDefs.h, 1506
- ACCLNxCalibrateZ
  - MindSensorsAPI, 145
  - NXCDefs.h, 1506
- ACCLNxCalibrateZEnd
  - MindSensorsAPI, 146
  - NXCDefs.h, 1507
- ACCLNxResetCalibration
  - MindSensorsAPI, 146
  - NXCDefs.h, 1507
- ACCLNxSensitivity
  - MindSensorsAPI, 146
  - NXCDefs.h, 1507
- ACCLNxXOffset
  - MindSensorsAPI, 147
  - NXCDefs.h, 1508
- ACCLNxXRange
  - MindSensorsAPI, 147
  - NXCDefs.h, 1508
- ACCLNxYOffset
  - MindSensorsAPI, 148
  - NXCDefs.h, 1509
- ACCLNxYRange
  - MindSensorsAPI, 148
  - NXCDefs.h, 1509
- ACCLNxZOffset
  - MindSensorsAPI, 149
  - NXCDefs.h, 1510
- ACCLNxZRange
  - MindSensorsAPI, 149
  - NXCDefs.h, 1510
- Acos
  - cmathAPI, 604
  - NXCDefs.h, 1480
- acos
  - cmathAPI, 617
  - NXCDefs.h, 1510
- AcosD
  - cmathAPI, 605
  - NXCDefs.h, 1480
- acosd
  - cmathAPI, 617
  - NXCDefs.h, 1511
- Acquire
  - CommandModuleFunctions, 420
  - NXCDefs.h, 1511
- Action
  - CommBTConnectionType, 1021
- ActualSpeedField
  - NBCCCommon.h, 1165
  - OutputFieldConstants, 785
- addressOf
  - cstringAPI, 658
  - NXCDefs.h, 1512
- addressOfEx
  - cstringAPI, 658
  - NXCDefs.h, 1512
- Array API functions, 449
- Array operation constants, 680
- ArrayBuild
  - ArrayFunctions, 451
  - NXCDefs.h, 1513
- ArrayFunctions
  - ArrayBuild, 451
  - ArrayIndex, 451
  - ArrayInit, 452
  - ArrayLen, 452
  - ArrayMax, 453
  - ArrayMean, 453
  - ArrayMin, 454
  - ArrayOp, 455
  - ArrayReplace, 455
  - ArraySort, 456
  - ArrayStd, 457
  - ArraySubset, 457
  - ArraySum, 458
  - ArraySumSqr, 458
- ArrayIndex

- ArrayFunctions, [451](#)
  - NXCDefs.h, [1513](#)
- ArrayInit
  - ArrayFunctions, [452](#)
  - NXCDefs.h, [1514](#)
- ArrayLen
  - ArrayFunctions, [452](#)
  - NXCDefs.h, [1514](#)
- ArrayMax
  - ArrayFunctions, [453](#)
  - NXCDefs.h, [1515](#)
- ArrayMean
  - ArrayFunctions, [453](#)
  - NXCDefs.h, [1516](#)
- ArrayMin
  - ArrayFunctions, [454](#)
  - NXCDefs.h, [1516](#)
- ArrayOp
  - ArrayFunctions, [455](#)
  - NXCDefs.h, [1517](#)
- ArrayOpConstants
  - OPARR\_MAX, [681](#)
  - OPARR\_MEAN, [681](#)
  - OPARR\_MIN, [681](#)
  - OPARR\_SORT, [681](#)
  - OPARR\_STD, [681](#)
  - OPARR\_SUM, [681](#)
  - OPARR\_SUMSQR, [681](#)
- ArrayReplace
  - ArrayFunctions, [455](#)
  - NXCDefs.h, [1518](#)
- ArraySort
  - ArrayFunctions, [456](#)
  - NXCDefs.h, [1518](#)
- ArrayStd
  - ArrayFunctions, [457](#)
  - NXCDefs.h, [1519](#)
- ArraySubset
  - ArrayFunctions, [457](#)
  - NXCDefs.h, [1520](#)
- ArraySum
  - ArrayFunctions, [458](#)
  - NXCDefs.h, [1520](#)
- ArraySumSqr
  - ArrayFunctions, [458](#)
  - NXCDefs.h, [1521](#)
- Asin
  - cmathAPI, [605](#)
  - NXCDefs.h, [1481](#)
- asin
  - cmathAPI, [617](#)
  - NXCDefs.h, [1521](#)
- AsinD
  - cmathAPI, [605](#)
  - NXCDefs.h, [1481](#)
- asind
  - cmathAPI, [618](#)
  - NXCDefs.h, [1522](#)
- Atan
  - cmathAPI, [606](#)
  - NXCDefs.h, [1481](#)
- atan
  - cmathAPI, [618](#)
  - NXCDefs.h, [1522](#)
- Atan2
  - cmathAPI, [606](#)
  - NXCDefs.h, [1482](#)
- atan2
  - cmathAPI, [619](#)
  - NXCDefs.h, [1523](#)
- Atan2D
  - cmathAPI, [607](#)
  - NXCDefs.h, [1482](#)
- atan2d
  - cmathAPI, [620](#)
  - NXCDefs.h, [1524](#)
- AtanD
  - cmathAPI, [607](#)
  - NXCDefs.h, [1483](#)
- atand
  - cmathAPI, [620](#)
  - NXCDefs.h, [1524](#)
- atof
  - cstdlibAPI, [647](#)
  - NXCDefs.h, [1525](#)
- atoi
  - cstdlibAPI, [647](#)
  - NXCDefs.h, [1526](#)
- atol
  - cstdlibAPI, [648](#)
  - NXCDefs.h, [1526](#)

- Basic analog sensor value names, 312
- BasicSensorValues
  - SENSOR\_1, 312
  - SENSOR\_2, 312
  - SENSOR\_3, 312
  - SENSOR\_4, 312
- BatteryLevel
  - NXCDefs.h, 1527
  - UiModuleFunctions, 562
- BatteryState
  - NXCDefs.h, 1527
  - UiModuleFunctions, 562
- BaudRate
  - CommHSControlType, 1028
- bcd2dec
  - cmathAPI, 621
  - NXCDefs.h, 1527
- BITMAP\_1
  - DisplayModuleConstants, 812
  - NBCCCommon.h, 1166
- BITMAP\_2
  - DisplayModuleConstants, 812
  - NBCCCommon.h, 1166
- BITMAP\_3
  - DisplayModuleConstants, 812
  - NBCCCommon.h, 1166
- BITMAP\_4
  - DisplayModuleConstants, 812
  - NBCCCommon.h, 1166
- BITMAPS
  - DisplayModuleConstants, 813
  - NBCCCommon.h, 1166
- BlockTachoCount
  - OutputStateType, 1094
- BlockTachoCountField
  - NBCCCommon.h, 1166
  - OutputFieldConstants, 785
- Bluetooth hardware status constants, 839
- Bluetooth State constants, 833
- Bluetooth state status constants, 835
- BluetoothState
  - NXCDefs.h, 1528
  - UiModuleFunctions, 562
- BluetoothState constants, 751
- BluetoothStatus
  - CommModuleFunctions, 471
  - NXCDefs.h, 1528
- BluetoothWrite
  - CommModuleFunctions, 471
  - NXCDefs.h, 1529
- BothMotors
  - JoystickMessageType, 1081
- BranchComp
  - CommandModuleFunctions, 420
  - NXCDefs.h, 1529
- BranchTest
  - CommandModuleFunctions, 420
  - NXCDefs.h, 1530
- BREAKOUT\_REQ
  - CommandVMState, 705
  - NBCCCommon.h, 1166
- BrickDataBluecoreVersion
  - CommModuleFunctions, 472
  - NXCDefs.h, 1530
- BrickDataBtHardwareStatus
  - CommModuleFunctions, 472
  - NXCDefs.h, 1530
- BrickDataBtStateStatus
  - CommModuleFunctions, 472
  - NXCDefs.h, 1531
- BrickDataName
  - CommModuleFunctions, 473
  - NXCDefs.h, 1531
- BrickDataTimeoutValue
  - CommModuleFunctions, 473
  - NXCDefs.h, 1531
- BT\_ARM\_CMD\_MODE
  - CommBtStateConstants, 834
  - NBCCCommon.h, 1166
- BT\_ARM\_DATA\_MODE
  - CommBtStateConstants, 834
  - NBCCCommon.h, 1167
- BT\_ARM\_OFF
  - CommBtStateConstants, 834
  - NBCCCommon.h, 1167
- BT\_BRICK\_PORT\_OPEN
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_BRICK\_VISIBILITY
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_CMD\_BYTE

- CommMiscConstants, 832
- NBCCCommon.h, 1167
- BT\_CMD\_READY
  - CommStatusCodesConstants, 855
  - NBCCCommon.h, 1167
- BT\_CONNECTION\_0\_ENABLE
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_CONNECTION\_1\_ENABLE
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_CONNECTION\_2\_ENABLE
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_CONNECTION\_3\_ENABLE
  - CommBtStateStatusConstants, 836
  - NBCCCommon.h, 1167
- BT\_DEFAULT\_INQUIRY\_MAX
  - CommMiscConstants, 832
  - NBCCCommon.h, 1168
- BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO
  - CommMiscConstants, 832
  - NBCCCommon.h, 1168
- BT\_DEVICE\_AWAY
  - CommDeviceStatusConstants, 852
  - NBCCCommon.h, 1168
- BT\_DEVICE\_EMPTY
  - CommDeviceStatusConstants, 852
  - NBCCCommon.h, 1168
- BT\_DEVICE\_KNOWN
  - CommDeviceStatusConstants, 852
  - NBCCCommon.h, 1168
- BT\_DEVICE\_NAME
  - CommDeviceStatusConstants, 852
  - NBCCCommon.h, 1168
- BT\_DEVICE\_UNKNOWN
  - CommDeviceStatusConstants, 852
  - NBCCCommon.h, 1168
- BT\_DISABLE
  - CommBtHwStatusConstants, 839
  - NBCCCommon.h, 1168
- BT\_ENABLE
  - CommBtHwStatusConstants, 839
  - NBCCCommon.h, 1168
- BTConnectionClass
  - CommModuleFunctions, 473
  - NXCDefs.h, 1532
- BTConnectionHandleNum
  - CommModuleFunctions, 474
  - NXCDefs.h, 1532
- BTConnectionLinkQuality
  - CommModuleFunctions, 474
  - NXCDefs.h, 1532
- BTConnectionName
  - CommModuleFunctions, 475
  - NXCDefs.h, 1533
- BTConnectionPinCode
  - CommModuleFunctions, 475
  - NXCDefs.h, 1533
- BTConnectionStreamStatus
  - CommModuleFunctions, 475
  - NXCDefs.h, 1534
- BTDataMode
  - CommModuleFunctions, 476
  - NXCDefs.h, 1534
- BTDeviceClass
  - CommModuleFunctions, 476
  - NXCDefs.h, 1534
- BTDeviceCount
  - CommModuleFunctions, 477
  - NXCDefs.h, 1535
- BTDeviceName
  - CommModuleFunctions, 477
  - NXCDefs.h, 1535
- BTDeviceNameCount
  - CommModuleFunctions, 477
  - NXCDefs.h, 1535
- BTDeviceStatus
  - CommModuleFunctions, 478
  - NXCDefs.h, 1536
- BTInputBufferInPtr
  - CommModuleFunctions, 478
  - NXCDefs.h, 1536
- BTInputBufferOutPtr
  - CommModuleFunctions, 478
  - NXCDefs.h, 1536
- BTN1
  - ButtonNameConstants, 741
  - NBCCCommon.h, 1168
- BTN2
  - ButtonNameConstants, 741

- NBCCCommon.h, 1169
- BTN3
  - ButtonNameConstants, 741
  - NBCCCommon.h, 1169
- BTN4
  - ButtonNameConstants, 741
  - NBCCCommon.h, 1169
- BTNCENTER
  - ButtonNameConstants, 741
  - NBCCCommon.h, 1169
- BTNEXIT
  - ButtonNameConstants, 741
  - NBCCCommon.h, 1169
- BTNLEFT
  - ButtonNameConstants, 742
  - NBCCCommon.h, 1169
- BTNRIGHT
  - ButtonNameConstants, 742
  - NBCCCommon.h, 1170
- BTNSTATE\_LONG\_PRESSED\_EV
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1170
- BTNSTATE\_LONG\_RELEASED\_EV
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1170
- BTNSTATE\_NONE
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1170
- BTNSTATE\_PRESSED\_EV
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1170
- BTNSTATE\_PRESSED\_STATE
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1170
- BTNSTATE\_SHORT\_RELEASED\_EV
  - ButtonStateConstants, 743
  - NBCCCommon.h, 1171
- BTOutputBufferInPtr
  - CommModuleFunctions, 479
  - NXCDefs.h, 1537
- BTOutputBufferOutPtr
  - CommModuleFunctions, 479
  - NXCDefs.h, 1537
- Buffer
  - CommBTWriteType, 1023
  - CommHSReadWriteType, 1029
  - CommLSReadType, 1032
  - CommLSWriteExType, 1033
  - CommLSWriteType, 1035
  - FileReadWriteType, 1062
  - IOMapReadByIDType, 1075
  - IOMapReadType, 1076
  - IOMapWriteByIDType, 1078
  - IOMapWriteType, 1079
  - LoaderExecuteFunctionType, 1088
  - ReadLastResponseType, 1099
- BufferLen
  - CommHSReadWriteType, 1029
  - CommLSReadType, 1032
- Button module, 53
- Button module constants, 740
- Button module functions, 551
- Button module IOMAP offsets, 744
- Button module types, 551
- Button name constants, 740
- ButtonCount
  - ButtonModuleFunctions, 553
  - NXCDefs.h, 1537
- ButtonIOMAP
  - ButtonOffsetLongPressCnt, 744
  - ButtonOffsetLongRelCnt, 744
  - ButtonOffsetPressedCnt, 744
  - ButtonOffsetRelCnt, 744
  - ButtonOffsetShortRelCnt, 744
  - ButtonOffsetState, 744
- ButtonLongPressCount
  - ButtonModuleFunctions, 553
  - NXCDefs.h, 1538
- ButtonLongReleaseCount
  - ButtonModuleFunctions, 553
  - NXCDefs.h, 1538
- ButtonModuleFunctions
  - ButtonCount, 553
  - ButtonLongPressCount, 553
  - ButtonLongReleaseCount, 553
  - ButtonPressCount, 554
  - ButtonPressed, 554
  - ButtonReleaseCount, 554
  - ButtonShortReleaseCount, 555
  - ButtonState, 555
  - ReadButtonEx, 556
  - SetButtonLongPressCount, 556

- SetButtonLongReleaseCount, [556](#)
- SetButtonPressCount, [557](#)
- SetButtonReleaseCount, [557](#)
- SetButtonShortReleaseCount, [557](#)
- SetButtonState, [558](#)
- SysReadButton, [558](#)
- ButtonModuleID
  - ModuleIDConstants, [259](#)
  - NBCCCommon.h, [1171](#)
- ButtonModuleName
  - ModuleNameConstants, [257](#)
  - NBCCCommon.h, [1171](#)
- ButtonNameConstants
  - BTN1, [741](#)
  - BTN2, [741](#)
  - BTN3, [741](#)
  - BTN4, [741](#)
  - BTNCENTER, [741](#)
  - BTNEXIT, [741](#)
  - BTNLEFT, [742](#)
  - BTNRIGHT, [742](#)
  - NO\_OF\_BTNS, [742](#)
- ButtonOffsetLongPressCnt
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonOffsetLongRelCnt
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonOffsetPressedCnt
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonOffsetRelCnt
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonOffsetShortRelCnt
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonOffsetState
  - ButtonIOMAP, [744](#)
  - NBCCCommon.h, [1171](#)
- ButtonPressCount
  - ButtonModuleFunctions, [554](#)
  - NXCDefs.h, [1539](#)
- ButtonPressed
  - ButtonModuleFunctions, [554](#)
  - NXCDefs.h, [1539](#)
- ButtonReleaseCount
  - ButtonModuleFunctions, [554](#)
  - NXCDefs.h, [1539](#)
- Buttons
  - JoystickMessageType, [1081](#)
- ButtonShortReleaseCount
  - ButtonModuleFunctions, [555](#)
  - NXCDefs.h, [1540](#)
- ButtonState
  - ButtonModuleFunctions, [555](#)
  - NXCDefs.h, [1540](#)
- ButtonState constants, [742](#)
- ButtonStateConstants
  - BTNSTATE\_LONG\_PRESSED\_-EV, [743](#)
  - BTNSTATE\_LONG\_RELEASED\_-EV, [743](#)
  - BTNSTATE\_NONE, [743](#)
  - BTNSTATE\_PRESSED\_EV, [743](#)
  - BTNSTATE\_PRESSED\_STATE, [743](#)
  - BTNSTATE\_SHORT\_-RELEASED\_EV, [743](#)
- ByteArrayToStr
  - cstringAPI, [659](#)
  - NXCDefs.h, [1541](#)
- ByteArrayToStrEx
  - cstringAPI, [659](#)
  - NXCDefs.h, [1541](#)
- BytesReady
  - CommLSCheckStatusType, [1031](#)
- Calibrated
  - InputValueType, [1073](#)
- CalibratedValue
  - InputValueType, [1073](#)
- Ceil
  - cmathAPI, [608](#)
  - NXCDefs.h, [1483](#)
- ceil
  - cmathAPI, [621](#)
  - NXCDefs.h, [1541](#)
- Center
  - DrawCircleType, [1043](#)
  - DrawEllipseType, [1045](#)
- CHAR\_BIT

- NBCCCommon.h, 1171
- NXTLimits, 1011
- CHAR\_MAX
  - NBCCCommon.h, 1172
  - NXTLimits, 1011
- CHAR\_MIN
  - NBCCCommon.h, 1172
  - NXTLimits, 1011
- CircleOut
  - DisplayModuleFunctions, 352
  - NXCDefs.h, 1542
- Clear
  - ReadLastResponseType, 1099
- ClearBits
  - WriteSemDataType, 1113
- ClearLine
  - DisplayModuleFunctions, 353
  - NXCDefs.h, 1542
- ClearScreen
  - DisplayModuleFunctions, 353
  - NXCDefs.h, 1543
- ClearSensor
  - InputModuleFunctions, 289
  - NXCDefs.h, 1543
- CloseFile
  - LoaderModuleFunctions, 576
  - NXCDefs.h, 1543
- CLUMP\_DONE
  - CommandVMState, 705
  - NBCCCommon.h, 1172
- CLUMP\_SUSPEND
  - CommandVMState, 705
  - NBCCCommon.h, 1172
- cmath API, 599
- cmathAPI
  - Acos, 604
  - acos, 617
  - AcosD, 605
  - acosd, 617
  - Asin, 605
  - asin, 617
  - AsinD, 605
  - asind, 618
  - Atan, 606
  - atan, 618
  - Atan2, 606
  - atan2, 619
  - Atan2D, 607
  - atan2d, 620
  - AtanD, 607
  - atand, 620
  - bcd2dec, 621
  - Ceil, 608
  - ceil, 621
  - Cos, 608
  - cos, 622
  - CosD, 608
  - cosd, 622
  - Cosh, 609
  - cosh, 622
  - CoshD, 609
  - coshd, 623
  - Exp, 609
  - exp, 623
  - Floor, 610
  - floor, 624
  - Frac, 610
  - frac, 624
  - isNAN, 625
  - Log, 611
  - log, 625
  - Log10, 611
  - log10, 626
  - MulDiv32, 612
  - muldiv32, 626
  - Pow, 612
  - pow, 627
  - sign, 627
  - Sin, 612
  - sin, 628
  - SinD, 613
  - sind, 628
  - Sinh, 613
  - sinh, 629
  - SinhD, 614
  - sinhd, 629
  - Sqrt, 614
  - sqrt, 629
  - Tan, 614
  - tan, 630
  - TanD, 615
  - tand, 630

- Tanh, [615](#)
- tanh, [631](#)
- TanhD, [616](#)
- tanhd, [631](#)
- Trunc, [616](#)
- trunc, [632](#)
- VectorCross, [632](#)
- VectorDot, [632](#)
- VectorNormalize, [633](#)
- Cmd
  - CommExecuteFunctionType, [1025](#)
  - DisplayExecuteFunctionType, [1040](#)
  - InputPinFunctionType, [1071](#)
  - LoaderExecuteFunctionType, [1088](#)
- cmpconst
  - EQ, [448](#)
  - GT, [448](#)
  - GTEQ, [449](#)
  - LT, [449](#)
  - LTEQ, [449](#)
  - NEQ, [449](#)
- Coast
  - NXCDefs.h, [1544](#)
  - OutputModuleFunctions, [318](#)
- CoastEx
  - NXCDefs.h, [1544](#)
  - OutputModuleFunctions, [318](#)
- Codatex API Functions, [229](#)
- Codatex device constants, [977](#)
- Codatex RFID sensor constants, [977](#)
- Codatex RFID sensor modes, [978](#)
- CodatexAPI
  - RFIDInit, [230](#)
  - RFIDMode, [231](#)
  - RFIDRead, [231](#)
  - RFIDReadContinuous, [232](#)
  - RFIDReadSingle, [232](#)
  - RFIDStatus, [232](#)
  - RFIDStop, [233](#)
- Color calibration constants, [766](#)
- Color calibration state constants, [765](#)
- Color sensor array indices, [763](#)
- Color values, [764](#)
- ColorADRaw
  - InputModuleFunctions, [289](#)
  - NXCDefs.h, [1545](#)
- ColorBoolean
  - InputModuleFunctions, [290](#)
  - NXCDefs.h, [1545](#)
- ColorCalibration
  - InputModuleFunctions, [290](#)
  - NXCDefs.h, [1546](#)
- ColorCalibrationState
  - InputModuleFunctions, [291](#)
  - NXCDefs.h, [1546](#)
- ColorCalLimits
  - InputModuleFunctions, [291](#)
  - NXCDefs.h, [1547](#)
- ColorSensorRaw
  - InputModuleFunctions, [292](#)
  - NXCDefs.h, [1547](#)
- ColorSensorRead
  - NBCCommon.h, [1172](#)
  - SysCallConstants, [684](#)
- ColorSensorReadType, [1018](#)
  - ColorValue, [1018](#)
  - Invalid, [1018](#)
  - NormalizedArray, [1019](#)
  - Port, [1019](#)
  - RawArray, [1019](#)
  - Result, [1019](#)
  - ScaledArray, [1019](#)
- ColorSensorValue
  - InputModuleFunctions, [292](#)
  - NXCDefs.h, [1548](#)
- ColorValue
  - ColorSensorReadType, [1018](#)
- COM\_CHANNEL\_FOUR\_ACTIVE
  - LowSpeedStateConstants, [794](#)
  - NBCCommon.h, [1172](#)
- COM\_CHANNEL\_NONE\_ACTIVE
  - LowSpeedStateConstants, [794](#)
  - NBCCommon.h, [1172](#)
- COM\_CHANNEL\_ONE\_ACTIVE
  - LowSpeedStateConstants, [794](#)
  - NBCCommon.h, [1172](#)
- COM\_CHANNEL\_THREE\_ACTIVE
  - LowSpeedStateConstants, [795](#)
  - NBCCommon.h, [1172](#)
- COM\_CHANNEL\_TWO\_ACTIVE
  - LowSpeedStateConstants, [795](#)
  - NBCCommon.h, [1172](#)

- Combined sensor type and mode constants, 281
- Comm module, 52
- Comm module constants, 830
- Comm module functions, 462
- Comm module interface function constants, 852
- Comm module IOMAP offsets, 856
- Comm module status code constants, 855
- Comm module types, 461
- Command
  - CommHSControlType, 1028
  - ReadLastResponseType, 1099
- Command module, 50
- Command module constants, 50
- Command module functions, 414
- Command module IOMAP offsets, 712
- Command module types, 412
- CommandCommErrors
  - ERR\_COMM\_BUFFER\_FULL, 710
  - ERR\_COMM\_BUS\_ERR, 710
  - ERR\_COMM\_CHAN\_INVALID, 710
  - ERR\_COMM\_CHAN\_NOT\_READY, 710
- CommandFatalErrors
  - ERR\_ARG, 707
  - ERR\_BAD\_POOL\_SIZE, 707
  - ERR\_BAD\_PTR, 707
  - ERR\_CLUMP\_COUNT, 707
  - ERR\_DEFAULT\_OFFSETS, 707
  - ERR\_FILE, 707
  - ERR\_INSANE\_OFFSET, 707
  - ERR\_INSTR, 707
  - ERR\_LOADER\_ERR, 707
  - ERR\_MEM, 708
  - ERR\_MEMMGR\_FAIL, 708
  - ERR\_NO\_ACTIVE\_CLUMP, 708
  - ERR\_NO\_CODE, 708
  - ERR\_NON\_FATAL, 708
  - ERR\_SPOTCHECK\_FAIL, 708
  - ERR\_VER, 708
- CommandFlags
  - NXCDefs.h, 1548
  - UiModuleFunctions, 563
- CommandFlags constants, 745
- CommandGenErrors
  - ERR\_INVALID\_FIELD, 709
  - ERR\_INVALID\_PORT, 709
  - ERR\_INVALID\_QUEUE, 709
  - ERR\_INVALID\_SIZE, 709
  - ERR\_NO\_PROG, 709
- CommandIOMAP
  - CommandOffsetActivateFlag, 713
  - CommandOffsetAwake, 713
  - CommandOffsetDeactivateFlag, 713
  - CommandOffsetFileName, 713
  - CommandOffsetFormatString, 713
  - CommandOffsetMemoryPool, 713
  - CommandOffsetOffsetDS, 713
  - CommandOffsetOffsetDVA, 714
  - CommandOffsetPRCHandler, 714
  - CommandOffsetProgStatus, 714
  - CommandOffsetSyncTick, 714
  - CommandOffsetSyncTime, 714
  - CommandOffsetTick, 714
- CommandModuleConstants
  - NO\_ERR, 51
  - POOL\_MAX\_SIZE, 52
  - STAT\_COMM\_PENDING, 52
  - STAT\_MSG\_EMPTY\_MAILBOX, 52
- CommandModuleFunctions
  - Acquire, 420
  - BranchComp, 420
  - BranchTest, 420
  - CurrentTick, 421
  - ExitTo, 421
  - FirstTick, 422
  - Follows, 422
  - GetButtonModuleValue, 422
  - GetCommandModuleBytes, 423
  - GetCommandModuleValue, 423
  - GetCommModuleBytes, 423
  - GetCommModuleValue, 424
  - GetDisplayModuleBytes, 424
  - GetDisplayModuleValue, 425
  - GetInputModuleValue, 425
  - GetIOMapBytes, 425
  - GetIOMapBytesByID, 426
  - GetIOMapValue, 426

- GetIOMapValueByID, [427](#)
- GetLastResponseInfo, [427](#)
- GetLoaderModuleValue, [428](#)
- GetLowSpeedModuleBytes, [428](#)
- GetLowSpeedModuleValue, [429](#)
- GetMemoryInfo, [429](#)
- GetOutputModuleValue, [429](#)
- GetSoundModuleValue, [430](#)
- GetUIModuleValue, [430](#)
- Precedes, [430](#)
- Release, [431](#)
- ResetSleepTimer, [431](#)
- SetButtonModuleValue, [432](#)
- SetCommandModuleBytes, [432](#)
- SetCommandModuleValue, [432](#)
- SetCommModuleBytes, [433](#)
- SetCommModuleValue, [433](#)
- SetDisplayModuleBytes, [433](#)
- SetDisplayModuleValue, [434](#)
- SetInputModuleValue, [434](#)
- SetIOCtrlModuleValue, [435](#)
- SetIOMapBytes, [435](#)
- SetIOMapBytesByID, [435](#)
- SetIOMapValue, [436](#)
- SetIOMapValueByID, [436](#)
- SetLoaderModuleValue, [437](#)
- SetLowSpeedModuleBytes, [437](#)
- SetLowSpeedModuleValue, [438](#)
- SetOutputModuleValue, [438](#)
- SetSoundModuleValue, [438](#)
- SetUIModuleValue, [439](#)
- StartTask, [439](#)
- Stop, [439](#)
- StopAllTasks, [440](#)
- StopTask, [440](#)
- SysCall, [440](#)
- SysComputeCalibValue, [441](#)
- SysDatalogGetTimes, [441](#)
- SysDatalogWrite, [442](#)
- SysGetStartTick, [442](#)
- SysIOMapRead, [443](#)
- SysIOMapReadByID, [443](#)
- SysIOMapWrite, [444](#)
- SysIOMapWriteByID, [444](#)
- SysKeepAlive, [444](#)
- SysMemoryManager, [445](#)
- SysReadLastResponse, [445](#)
- SysReadSemData, [445](#)
- SysUpdateCalibCacheInfo, [446](#)
- SysWriteSemData, [446](#)
- Wait, [447](#)
- Yield, [448](#)
- CommandModuleID
  - ModuleIDConstants, [259](#)
  - NBCCCommon.h, [1173](#)
- CommandModuleName
  - ModuleNameConstants, [257](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetActivateFlag
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetAwake
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetDeactivateFlag
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetFileName
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetFormatString
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetMemoryPool
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1173](#)
- CommandOffsetOffsetDS
  - CommandIOMAP, [713](#)
  - NBCCCommon.h, [1174](#)
- CommandOffsetOffsetDVA
  - CommandIOMAP, [714](#)
  - NBCCCommon.h, [1174](#)
- CommandOffsetPRCHandler
  - CommandIOMAP, [714](#)
  - NBCCCommon.h, [1174](#)
- CommandOffsetProgStatus
  - CommandIOMAP, [714](#)
  - NBCCCommon.h, [1174](#)
- CommandOffsetSyncTick
  - CommandIOMAP, [714](#)
  - NBCCCommon.h, [1174](#)
- CommandOffsetSyncTime

- CommandIOMAP, 714
- NBCCCommon.h, 1174
- CommandOffsetTick
  - CommandIOMAP, 714
  - NBCCCommon.h, 1174
- CommandProgStatus
  - PROG\_ABORT, 712
  - PROG\_ERROR, 712
  - PROG\_IDLE, 712
  - PROG\_OK, 712
  - PROG\_RESET, 712
  - PROG\_RUNNING, 712
- CommandRCErrors
  - ERR\_RC\_BAD\_PACKET, 711
  - ERR\_RC\_FAILED, 711
  - ERR\_RC\_ILLEGAL\_VAL, 711
  - ERR\_RC\_UNKNOWN\_CMD, 711
- CommandVMState
  - BREAKOUT\_REQ, 705
  - CLUMP\_DONE, 705
  - CLUMP\_SUSPEND, 705
  - ROTATE\_QUEUE, 706
  - STOP\_REQ, 706
  - TIMES\_UP, 706
- CommBTCheckStatus
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommBTCheckStatusType, 1019
  - Connection, 1020
  - Result, 1020
- CommBTConnection
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommBTConnectionType, 1020
  - Action, 1021
  - ConnectionSlot, 1021
  - Name, 1021
  - Result, 1021
- CommBtHwStatusConstants
  - BT\_DISABLE, 839
  - BT\_ENABLE, 839
- CommBTONOff
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommBTONOffType, 1022
  - PowerState, 1022
  - Result, 1022
- CommBTRead
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommBtStateConstants
  - BT\_ARM\_CMD\_MODE, 834
  - BT\_ARM\_DATA\_MODE, 834
  - BT\_ARM\_OFF, 834
- CommBtStateStatusConstants
  - BT\_BRICK\_PORT\_OPEN, 836
  - BT\_BRICK\_VISIBILITY, 836
  - BT\_CONNECTION\_0\_ENABLE, 836
  - BT\_CONNECTION\_1\_ENABLE, 836
  - BT\_CONNECTION\_2\_ENABLE, 836
  - BT\_CONNECTION\_3\_ENABLE, 836
- CommBTWrite
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommBTWriteType, 1023
  - Buffer, 1023
  - Connection, 1023
  - Result, 1024
- CommConnectionConstants
  - CONN\_BT0, 837
  - CONN\_BT1, 837
  - CONN\_BT2, 837
  - CONN\_BT3, 838
  - CONN\_HS4, 838
  - CONN\_HS\_1, 838
  - CONN\_HS\_2, 838
  - CONN\_HS\_3, 838
  - CONN\_HS\_4, 838
  - CONN\_HS\_5, 838
  - CONN\_HS\_6, 838
  - CONN\_HS\_7, 838
  - CONN\_HS\_8, 838
  - CONN\_HS\_ALL, 839
- CommDataModeConstants
  - DATA\_MODE\_GPS, 834
  - DATA\_MODE\_MASK, 835
  - DATA\_MODE\_NXT, 835
  - DATA\_MODE\_RAW, 835

- DATA\_MODE\_UPDATE, 835
- CommDeviceStatusConstants
  - BT\_DEVICE\_AWAY, 852
  - BT\_DEVICE\_EMPTY, 852
  - BT\_DEVICE\_KNOWN, 852
  - BT\_DEVICE\_NAME, 852
  - BT\_DEVICE\_UNKNOWN, 852
- CommExecuteFunction
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommExecuteFunctionType, 1024
  - Cmd, 1025
  - Name, 1026
  - Param1, 1026
  - Param2, 1026
  - Param3, 1026
  - Result, 1026
  - RetVal, 1026
- CommHiSpeedAddressConstants
  - HS\_ADDRESS\_1, 850
  - HS\_ADDRESS\_2, 850
  - HS\_ADDRESS\_3, 850
  - HS\_ADDRESS\_4, 850
  - HS\_ADDRESS\_5, 851
  - HS\_ADDRESS\_6, 851
  - HS\_ADDRESS\_7, 851
  - HS\_ADDRESS\_8, 851
  - HS\_ADDRESS\_ALL, 851
- CommHiSpeedBaudConstants
  - HS\_BAUD\_115200, 843
  - HS\_BAUD\_1200, 843
  - HS\_BAUD\_14400, 843
  - HS\_BAUD\_19200, 844
  - HS\_BAUD\_230400, 844
  - HS\_BAUD\_2400, 844
  - HS\_BAUD\_28800, 844
  - HS\_BAUD\_3600, 844
  - HS\_BAUD\_38400, 844
  - HS\_BAUD\_460800, 844
  - HS\_BAUD\_4800, 844
  - HS\_BAUD\_57600, 844
  - HS\_BAUD\_7200, 844
  - HS\_BAUD\_76800, 845
  - HS\_BAUD\_921600, 845
  - HS\_BAUD\_9600, 845
  - HS\_BAUD\_DEFAULT, 845
- CommHiSpeedCombinedConstants
  - HS\_MODE\_7E1, 849
  - HS\_MODE\_8N1, 849
- CommHiSpeedCtrlConstants
  - HS\_CTRL\_EXIT, 842
  - HS\_CTRL\_INIT, 842
  - HS\_CTRL\_UART, 842
- CommHiSpeedDataBitsConstants
  - HS\_MODE\_5\_DATA, 847
  - HS\_MODE\_6\_DATA, 847
  - HS\_MODE\_7\_DATA, 847
  - HS\_MODE\_8\_DATA, 847
- CommHiSpeedFlagsConstants
  - HS\_UPDATE, 840
- CommHiSpeedModeConstants
  - HS\_MODE\_DEFAULT, 846
  - HS\_MODE\_MASK, 846
  - HS\_MODE\_UART\_RS232, 846
  - HS\_MODE\_UART\_RS485, 846
  - HS\_UART\_MASK, 846
- CommHiSpeedParityConstants
  - HS\_MODE\_E\_PARITY, 848
  - HS\_MODE\_M\_PARITY, 848
  - HS\_MODE\_N\_PARITY, 849
  - HS\_MODE\_O\_PARITY, 849
  - HS\_MODE\_S\_PARITY, 849
- CommHiSpeedStateConstants
  - HS\_BYTES\_REMAINING, 841
  - HS\_DEFAULT, 841
  - HS\_DISABLE, 841
  - HS\_ENABLE, 841
  - HS\_INIT\_RECEIVER, 841
  - HS\_INITIALISE, 841
  - HS\_SEND\_DATA, 842
- CommHiSpeedStopBitsConstants
  - HS\_MODE\_10\_STOP, 848
  - HS\_MODE\_15\_STOP, 848
  - HS\_MODE\_20\_STOP, 848
- CommHSCheckStatus
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommHSCheckStatusType, 1026
  - DataAvailable, 1027
  - SendingData, 1027
- CommHSControl
  - NBCCCommon.h, 1175

- SysCallConstants, 684
- CommHSControlType, 1027
  - BaudRate, 1028
  - Command, 1028
  - Mode, 1028
  - Result, 1028
- CommHSRead
  - NBCCCommon.h, 1175
  - SysCallConstants, 684
- CommHSReadWriteType, 1029
  - Buffer, 1029
  - BufferLen, 1029
  - Status, 1030
- CommHSWrite
  - NBCCCommon.h, 1175
  - SysCallConstants, 685
- CommInterfaceConstants
  - INTF\_BTOFF, 853
  - INTF\_BTON, 853
  - INTF\_CONNECT, 853
  - INTF\_CONNECTBYNAME, 853
  - INTF\_CONNECTREQ, 853
  - INTF\_DISCONNECT, 853
  - INTF\_DISCONNECTALL, 854
  - INTF\_EXTREAD, 854
  - INTF\_FACTORYRESET, 854
  - INTF\_OPENSTREAM, 854
  - INTF\_PINREQ, 854
  - INTF\_REMOVEDEVICE, 854
  - INTF\_SEARCH, 854
  - INTF\_SENDDATA, 854
  - INTF\_SENDFILE, 854
  - INTF\_SETBTNAME, 854
  - INTF\_SETCMDMODE, 855
  - INTF\_STOPSEARCH, 855
  - INTF\_VISIBILITY, 855
- CommIOMAP
  - CommOffsetBrickDataBdAddr, 858
  - CommOffsetBrickDataBluecoreVersion, 858
  - CommOffsetBrickDataBtHwStatus, 858
  - CommOffsetBrickDataBtStateStatus, 858
  - CommOffsetBrickDataName, 858
  - CommOffsetBrickDataTimeOutValue, 858
  - CommOffsetBtConnectTableBdAddr, 858
  - CommOffsetBtConnectTableClassOfDevice, 858
  - CommOffsetBtConnectTableHandleNr, 858
  - CommOffsetBtConnectTableLinkQuality, 859
  - CommOffsetBtConnectTableName, 859
  - CommOffsetBtConnectTablePinCode, 859
  - CommOffsetBtConnectTableStreamStatus, 859
  - CommOffsetBtDataMode, 859
  - CommOffsetBtDeviceCnt, 859
  - CommOffsetBtDeviceNameCnt, 859
  - CommOffsetBtDeviceTableBdAddr, 859
  - CommOffsetBtDeviceTableClassOfDevice, 859
  - CommOffsetBtDeviceTableDeviceStatus, 859
  - CommOffsetBtDeviceTableName, 860
  - CommOffsetBtInBufBuf, 860
  - CommOffsetBtInBufInPtr, 860
  - CommOffsetBtInBufOutPtr, 860
  - CommOffsetBtOutBufBuf, 860
  - CommOffsetBtOutBufInPtr, 860
  - CommOffsetBtOutBufOutPtr, 860
  - CommOffsetHsAddress, 860
  - CommOffsetHsDataMode, 860
  - CommOffsetHsFlags, 860
  - CommOffsetHsInBufBuf, 861
  - CommOffsetHsInBufInPtr, 861
  - CommOffsetHsInBufOutPtr, 861
  - CommOffsetHsMode, 861
  - CommOffsetHsOutBufBuf, 861
  - CommOffsetHsOutBufInPtr, 861
  - CommOffsetHsOutBufOutPtr, 861
  - CommOffsetHsSpeed, 861
  - CommOffsetHsState, 861
  - CommOffsetPFunc, 861

- CommOffsetPFuncTwo, 862
- CommOffsetUsbInBufBuf, 862
- CommOffsetUsbInBufInPtr, 862
- CommOffsetUsbInBufOutPtr, 862
- CommOffsetUsbOutBufBuf, 862
- CommOffsetUsbOutBufInPtr, 862
- CommOffsetUsbOutBufOutPtr, 862
- CommOffsetUsbPollBufBuf, 862
- CommOffsetUsbPollBufInPtr, 862
- CommOffsetUsbPollBufOutPtr, 862
- CommOffsetUsbState, 863
- CommLSCheckStatus
  - NBCCCommon.h, 1176
  - SysCallConstants, 685
- CommLSCheckStatusType, 1030
  - BytesReady, 1031
  - Port, 1031
  - Result, 1031
- CommLSRead
  - NBCCCommon.h, 1176
  - SysCallConstants, 685
- CommLSReadType, 1031
  - Buffer, 1032
  - BufferLen, 1032
  - Port, 1032
  - Result, 1032
- CommLSWrite
  - NBCCCommon.h, 1176
  - SysCallConstants, 685
- CommLSWriteEx
  - NBCCCommon.h, 1176
  - SysCallConstants, 685
- CommLSWriteExType, 1033
  - Buffer, 1033
  - NoRestartOnRead, 1033
  - Port, 1034
  - Result, 1034
  - ReturnLen, 1034
- CommLSWriteType, 1034
  - Buffer, 1035
  - Port, 1035
  - Result, 1035
  - ReturnLen, 1035
- CommMiscConstants
  - BT\_CMD\_BYTE, 832
  - BT\_DEFAULT\_INQUIRY\_MAX, 832
  - BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO, 832
  - MAX\_BT\_MSG\_SIZE, 832
  - SIZE\_OF\_BDADDR, 832
  - SIZE\_OF\_BRICK\_NAME, 832
  - SIZE\_OF\_BT\_CONNECT\_TABLE, 832
  - SIZE\_OF\_BT\_DEVICE\_TABLE, 832
  - SIZE\_OF\_BT\_NAME, 832
  - SIZE\_OF\_BT\_PINCODE, 832
  - SIZE\_OF\_BTBUF, 833
  - SIZE\_OF\_CLASS\_OF\_DEVICE, 833
  - SIZE\_OF\_HSBUF, 833
  - SIZE\_OF\_USBBUF, 833
  - SIZE\_OF\_USBDATA, 833
  - USB\_PROTOCOL\_OVERHEAD, 833
- CommModuleDCFunctions
  - RemoteDatalogRead, 519
  - RemoteDatalogSetTimes, 520
  - RemoteGetBatteryLevel, 520
  - RemoteGetConnectionCount, 521
  - RemoteGetConnectionName, 522
  - RemoteGetContactCount, 522
  - RemoteGetContactName, 523
  - RemoteGetCurrentProgramName, 523
  - RemoteGetInputValues, 524
  - RemoteGetOutputState, 524
  - RemoteGetProperty, 525
  - RemoteKeepAlive, 526
  - RemoteLowspeedGetStatus, 526
  - RemoteLowspeedRead, 527
  - RemoteLowspeedWrite, 527
  - RemoteMessageRead, 528
  - RemoteMessageWrite, 528
  - RemotePlaySoundFile, 529
  - RemotePlayTone, 529
  - RemoteResetMotorPosition, 530
  - RemoteResetScaledValue, 530
  - RemoteResetTachoCount, 531
  - RemoteSetInputMode, 531

- RemoteSetOutputState, 532
- RemoteSetProperty, 533
- RemoteStartProgram, 533
- RemoteStopProgram, 534
- RemoteStopSound, 534
- CommModuleFunctions
  - BluetoothStatus, 471
  - BluetoothWrite, 471
  - BrickDataBluecoreVersion, 472
  - BrickDataBtHardwareStatus, 472
  - BrickDataBtStateStatus, 472
  - BrickDataName, 473
  - BrickDataTimeoutValue, 473
  - BTConnectionClass, 473
  - BTConnectionHandleNum, 474
  - BTConnectionLinkQuality, 474
  - BTConnectionName, 475
  - BTConnectionPinCode, 475
  - BTConnectionStreamStatus, 475
  - BTDataMode, 476
  - BTDeviceClass, 476
  - BTDeviceCount, 477
  - BTDeviceName, 477
  - BTDeviceNameCount, 477
  - BTDeviceStatus, 478
  - BTInputBufferInPtr, 478
  - BTInputBufferOutPtr, 478
  - BTOutputBufferInPtr, 479
  - BTOutputBufferOutPtr, 479
  - GetBrickDataAddress, 479
  - GetBTConnectionAddress, 480
  - GetBTDeviceAddress, 480
  - GetBTInputBuffer, 480
  - GetBTOutputBuffer, 481
  - GetHSInputBuffer, 481
  - GetHSOutputBuffer, 482
  - GetUSBInputBuffer, 482
  - GetUSBOutputBuffer, 483
  - GetUSBPollBuffer, 483
  - HSAddress, 483
  - HSDDataMode, 484
  - HSFlags, 484
  - HSInputBufferInPtr, 484
  - HSInputBufferOutPtr, 485
  - HSMMode, 485
  - HSOutputBufferInPtr, 485
  - HSOutputBufferOutPtr, 486
  - HSSpeed, 486
  - HSState, 486
  - JoystickMessageRead, 487
  - ReceiveMessage, 487
  - ReceiveRemoteBool, 488
  - ReceiveRemoteMessageEx, 488
  - ReceiveRemoteNumber, 489
  - ReceiveRemoteString, 489
  - RemoteConnectionIdle, 490
  - RemoteConnectionWrite, 490
  - RS485Control, 491
  - RS485DataAvailable, 491
  - RS485Disable, 492
  - RS485Enable, 492
  - RS485Initialize, 493
  - RS485Read, 493
  - RS485ReadEx, 493
  - RS485SendingData, 494
  - RS485Status, 494
  - RS485Uart, 495
  - RS485Write, 495
  - SendMessage, 496
  - SendRemoteBool, 496
  - SendRemoteNumber, 497
  - SendRemoteString, 497
  - SendResponseBool, 498
  - SendResponseNumber, 498
  - SendResponseString, 499
  - SendRS485Bool, 499
  - SendRS485Number, 499
  - SendRS485String, 500
  - SetBTDataMode, 500
  - SetBTInputBuffer, 501
  - SetBTInputBufferInPtr, 501
  - SetBTInputBufferOutPtr, 502
  - SetBTOutputBuffer, 502
  - SetBTOutputBufferInPtr, 502
  - SetBTOutputBufferOutPtr, 503
  - SetHSAddress, 503
  - SetHSDDataMode, 503
  - SetHSFlags, 503
  - SetHSInputBuffer, 504
  - SetHSInputBufferInPtr, 504
  - SetHSInputBufferOutPtr, 504
  - SetHSMMode, 505

- SetHSOutputBuffer, 505
- SetHSOutputBufferInPtr, 506
- SetHSOutputBufferOutPtr, 506
- SetHSSpeed, 506
- SetHSSState, 507
- SetUSBInputBuffer, 507
- SetUSBInputBufferInPtr, 507
- SetUSBInputBufferOutPtr, 507
- SetUSBOutputBuffer, 508
- SetUSBOutputBufferInPtr, 508
- SetUSBOutputBufferOutPtr, 508
- SetUSBPollBuffer, 509
- SetUSBPollBufferInPtr, 509
- SetUSBPollBufferOutPtr, 509
- SetUSBState, 510
- SysCommBTCheckStatus, 510
- SysCommBTConnection, 510
- SysCommBTOff, 511
- SysCommBTWrite, 511
- SysCommExecuteFunction, 512
- SysCommHSCheckStatus, 512
- SysCommHSControl, 512
- SysCommHSRead, 513
- SysCommHSWrite, 513
- SysMessageRead, 514
- SysMessageWrite, 514
- USBInputBufferInPtr, 514
- USBInputBufferOutPtr, 515
- USBOutputBufferInPtr, 515
- USBOutputBufferOutPtr, 515
- USBPollBufferInPtr, 516
- USBPollBufferOutPtr, 516
- USBState, 516
- UseRS485, 517
- CommModuleID
  - ModuleIDConstants, 259
  - NBCCCommon.h, 1176
- CommModuleName
  - ModuleNameConstants, 257
  - NBCCCommon.h, 1176
- CommModuleSCFunctions
  - RemoteBluetoothFactoryReset, 537
  - RemoteCloseFile, 537
  - RemoteDeleteFile, 538
  - RemoteDeleteUserFlash, 538
  - RemoteFindFirstFile, 539
  - RemoteFindNextFile, 540
  - RemoteGetBluetoothAddress, 540
  - RemoteGetDeviceInfo, 541
  - RemoteGetFirmwareVersion, 541
  - RemoteIOMapRead, 542
  - RemoteIOMapWriteBytes, 543
  - RemoteIOMapWriteValue, 543
  - RemoteOpenAppendData, 544
  - RemoteOpenRead, 545
  - RemoteOpenWrite, 545
  - RemoteOpenWriteData, 546
  - RemoteOpenWriteLinear, 546
  - RemotePollCommand, 547
  - RemotePollCommandLength, 548
  - RemoteRead, 548
  - RemoteRenameFile, 549
  - RemoteSetBrickName, 550
  - RemoteWrite, 550
- CommOffsetBrickDataBdAddr
  - CommIOMAP, 858
  - NBCCCommon.h, 1176
- CommOffsetBrickDataBluecoreVersion
  - CommIOMAP, 858
  - NBCCCommon.h, 1176
- CommOffsetBrickDataBtHwStatus
  - CommIOMAP, 858
  - NBCCCommon.h, 1176
- CommOffsetBrickDataBtStateStatus
  - CommIOMAP, 858
  - NBCCCommon.h, 1176
- CommOffsetBrickDataName
  - CommIOMAP, 858
  - NBCCCommon.h, 1177
- CommOffsetBrickDataTimeOutValue
  - CommIOMAP, 858
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTableBdAddr
  - CommIOMAP, 858
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTableClassOfDevice
  - CommIOMAP, 858
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTableHandleNr
  - CommIOMAP, 858
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTableLinkQuality

- CommIOMAP, 859
- NBCCCommon.h, 1177
- CommOffsetBtConnectTableName
  - CommIOMAP, 859
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTablePinCode
  - CommIOMAP, 859
  - NBCCCommon.h, 1177
- CommOffsetBtConnectTableStreamStatus
  - CommIOMAP, 859
  - NBCCCommon.h, 1177
- CommOffsetBtDataMode
  - CommIOMAP, 859
  - NBCCCommon.h, 1177
- CommOffsetBtDeviceCnt
  - CommIOMAP, 859
  - NBCCCommon.h, 1178
- CommOffsetBtDeviceNameCnt
  - CommIOMAP, 859
  - NBCCCommon.h, 1178
- CommOffsetBtDeviceTableBdAddr
  - CommIOMAP, 859
  - NBCCCommon.h, 1178
- CommOffsetBtDeviceTableClassOfDevice
  - CommIOMAP, 859
  - NBCCCommon.h, 1178
- CommOffsetBtDeviceTableDeviceStatus
  - CommIOMAP, 859
  - NBCCCommon.h, 1178
- CommOffsetBtDeviceTableName
  - CommIOMAP, 860
  - NBCCCommon.h, 1178
- CommOffsetBtInBufBuf
  - CommIOMAP, 860
  - NBCCCommon.h, 1178
- CommOffsetBtInBufInPtr
  - CommIOMAP, 860
  - NBCCCommon.h, 1178
- CommOffsetBtInBufOutPtr
  - CommIOMAP, 860
  - NBCCCommon.h, 1178
- CommOffsetBtOutBufBuf
  - CommIOMAP, 860
  - NBCCCommon.h, 1178
- CommOffsetBtOutBufInPtr
  - CommIOMAP, 860
- NBCCCommon.h, 1179
- CommOffsetBtOutBufOutPtr
  - CommIOMAP, 860
  - NBCCCommon.h, 1179
- CommOffsetHsAddress
  - CommIOMAP, 860
  - NBCCCommon.h, 1179
- CommOffsetHsDataMode
  - CommIOMAP, 860
  - NBCCCommon.h, 1179
- CommOffsetHsFlags
  - CommIOMAP, 860
  - NBCCCommon.h, 1179
- CommOffsetHsInBufBuf
  - CommIOMAP, 861
  - NBCCCommon.h, 1179
- CommOffsetHsInBufInPtr
  - CommIOMAP, 861
  - NBCCCommon.h, 1179
- CommOffsetHsInBufOutPtr
  - CommIOMAP, 861
  - NBCCCommon.h, 1179
- CommOffsetHsMode
  - CommIOMAP, 861
  - NBCCCommon.h, 1179
- CommOffsetHsOutBufBuf
  - CommIOMAP, 861
  - NBCCCommon.h, 1179
- CommOffsetHsOutBufInPtr
  - CommIOMAP, 861
  - NBCCCommon.h, 1180
- CommOffsetHsOutBufOutPtr
  - CommIOMAP, 861
  - NBCCCommon.h, 1180
- CommOffsetHsSpeed
  - CommIOMAP, 861
  - NBCCCommon.h, 1180
- CommOffsetHsState
  - CommIOMAP, 861
  - NBCCCommon.h, 1180
- CommOffsetPFunc
  - CommIOMAP, 861
  - NBCCCommon.h, 1180
- CommOffsetPFuncTwo
  - CommIOMAP, 862
  - NBCCCommon.h, 1180

- CommOffsetUsbInBufBuf
  - CommIOMAP, 862
  - NBCCCommon.h, 1180
- CommOffsetUsbInBufInPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1180
- CommOffsetUsbInBufOutPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1180
- CommOffsetUsbOutBufBuf
  - CommIOMAP, 862
  - NBCCCommon.h, 1180
- CommOffsetUsbOutBufInPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1181
- CommOffsetUsbOutBufOutPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1181
- CommOffsetUsbPollBufBuf
  - CommIOMAP, 862
  - NBCCCommon.h, 1181
- CommOffsetUsbPollBufInPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1181
- CommOffsetUsbPollBufOutPtr
  - CommIOMAP, 862
  - NBCCCommon.h, 1181
- CommOffsetUsbState
  - CommIOMAP, 863
  - NBCCCommon.h, 1181
- CommStatusCodesConstants
  - BT\_CMD\_READY, 855
  - HS\_CMD\_READY, 855
  - LR\_COULD\_NOT\_SAVE, 856
  - LR\_ENTRY\_REMOVED, 856
  - LR\_STORE\_IS\_FULL, 856
  - LR\_SUCCESS, 856
  - LR\_UNKNOWN\_ADDR, 856
  - USB\_CMD\_READY, 856
- Communications specific errors, 709
- Compact
  - MemoryManagerType, 1090
- Comparison Constants, 448
- ComputeCalibValue
  - NBCCCommon.h, 1181
  - SysCallConstants, 685
- ComputeCalibValueType, 1036
  - Name, 1036
  - RawVal, 1037
  - Result, 1037
- ConfigureTemperatureSensor
  - LowSpeedModuleFunctions, 389
  - NXCDefs.h, 1549
- CONN\_BT0
  - CommConnectionConstants, 837
  - NBCCCommon.h, 1181
- CONN\_BT1
  - CommConnectionConstants, 837
  - NBCCCommon.h, 1181
- CONN\_BT2
  - CommConnectionConstants, 837
  - NBCCCommon.h, 1182
- CONN\_BT3
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1182
- CONN\_HS4
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1182
- CONN\_HS\_1
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1182
- CONN\_HS\_2
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1182
- CONN\_HS\_3
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1182
- CONN\_HS\_4
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1183
- CONN\_HS\_5
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1183
- CONN\_HS\_6
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1183
- CONN\_HS\_7
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1183
- CONN\_HS\_8
  - CommConnectionConstants, 838
  - NBCCCommon.h, 1183

- CONN\_HS\_ALL
  - CommConnectionConstants, 839
  - NBCCCommon.h, 1183
- Connection
  - CommBTCheckStatusType, 1020
  - CommBTWriteType, 1023
- ConnectionSlot
  - CommBTConnectionType, 1021
- Constants to use with the Input module's Pin function, 770
- Copy
  - cstringAPI, 660
  - NXCDefs.h, 1549
- Cos
  - cmathAPI, 608
  - NXCDefs.h, 1483
- cos
  - cmathAPI, 622
  - NXCDefs.h, 1550
- CosD
  - cmathAPI, 608
  - NXCDefs.h, 1484
- cosd
  - cmathAPI, 622
  - NXCDefs.h, 1550
- Cosh
  - cmathAPI, 609
  - NXCDefs.h, 1484
- cosh
  - cmathAPI, 622
  - NXCDefs.h, 1551
- CoshD
  - cmathAPI, 609
  - NXCDefs.h, 1485
- coshd
  - cmathAPI, 623
  - NXCDefs.h, 1551
- Count
  - IOMapReadByIDType, 1075
  - IOMapReadType, 1076
  - ReadButtonType, 1098
- CreateFile
  - LoaderModuleFunctions, 577
  - NXCDefs.h, 1552
- CreateFileLinear
  - LoaderModuleFunctions, 577
  - NXCDefs.h, 1552
- CreateFileNonLinear
  - LoaderModuleFunctions, 578
  - NXCDefs.h, 1553
- cstdio API, 633
- cstdioAPI
  - fclose, 636
  - feof, 636
  - fflush, 637
  - fgetc, 637
  - fgets, 637
  - fopen, 638
  - fprintf, 638
  - fputc, 639
  - fputs, 639
  - fseek, 640
  - ftell, 640
  - getc, 635
  - getchar, 641
  - printf, 641
  - putc, 635
  - remove, 642
  - rename, 642
  - rewind, 642
  - set\_fopen\_size, 643
  - sprintf, 643
- cstdlib API, 645
- cstdlib API types, 655
- cstdlibAPI
  - abort, 646
  - abs, 646
  - atof, 647
  - atoi, 647
  - atol, 648
  - div, 649
  - labs, 649
  - ldiv, 649
  - rand, 650
  - Random, 650
  - srand, 651
  - strtod, 651
  - strtol, 652
  - strtoul, 653
  - SysRandomEx, 654
  - SysRandomNumber, 654
- cstring API, 655

- cstringAPI
  - addressOf, [658](#)
  - addressOfEx, [658](#)
  - ByteArrayToStr, [659](#)
  - ByteArrayToStrEx, [659](#)
  - Copy, [660](#)
  - Flatten, [660](#)
  - FlattenVar, [660](#)
  - FormatNum, [661](#)
  - LeftStr, [662](#)
  - memcmp, [662](#)
  - memcpy, [662](#)
  - memmove, [663](#)
  - MidStr, [663](#)
  - NumToStr, [664](#)
  - Pos, [664](#)
  - reladdressOf, [665](#)
  - RightStr, [665](#)
  - StrCat, [666](#)
  - strcat, [665](#)
  - strcmp, [666](#)
  - strcpy, [667](#)
  - StrIndex, [667](#)
  - StrLen, [668](#)
  - strlen, [668](#)
  - strncat, [669](#)
  - strncmp, [669](#)
  - strncpy, [670](#)
  - StrReplace, [670](#)
  - StrToByteArray, [671](#)
  - StrToNum, [671](#)
  - SubStr, [672](#)
  - UnflattenVar, [672](#)
- CT\_ADDR\_RFID
  - CTRFIDConstants, [978](#)
  - NBCCCommon.h, [1183](#)
- CT\_REG\_DATA
  - CTRFIDConstants, [978](#)
  - NBCCCommon.h, [1183](#)
- CT\_REG\_MODE
  - CTRFIDConstants, [978](#)
  - NBCCCommon.h, [1183](#)
- CT\_REG\_STATUS
  - CTRFIDConstants, [978](#)
  - NBCCCommon.h, [1183](#)
- CTRFIDConstants
  - CT\_ADDR\_RFID, [978](#)
  - CT\_REG\_DATA, [978](#)
  - CT\_REG\_MODE, [978](#)
  - CT\_REG\_STATUS, [978](#)
- CTRFIDModeConstants
  - RFID\_MODE\_CONTINUOUS, [978](#)
  - RFID\_MODE\_SINGLE, [978](#)
  - RFID\_MODE\_STOP, [979](#)
- ctype API, [673](#)
- ctypeAPI
  - isalnum, [674](#)
  - isalpha, [674](#)
  - isctrl, [675](#)
  - isdigit, [675](#)
  - isgraph, [675](#)
  - islower, [676](#)
  - isprint, [676](#)
  - ispunct, [677](#)
  - isspace, [677](#)
  - isupper, [677](#)
  - isxdigit, [678](#)
  - tolower, [678](#)
  - toupper, [679](#)
- CurrentTick
  - CommandModuleFunctions, [421](#)
  - NXCDefs.h, [1553](#)
- CustomSensorActiveStatus
  - InputModuleFunctions, [293](#)
  - NXCDefs.h, [1554](#)
- CustomSensorPercentFullScale
  - InputModuleFunctions, [293](#)
  - NXCDefs.h, [1554](#)
- CustomSensorZeroOffset
  - InputModuleFunctions, [294](#)
  - NXCDefs.h, [1554](#)
- DAC\_MODE\_DCOUT
  - DacModeConstants, [123](#)
  - NBCCCommon.h, [1184](#)
- DAC\_MODE\_PWMVOLTAGE
  - DacModeConstants, [123](#)
  - NBCCCommon.h, [1184](#)
- DAC\_MODE\_SAWNEGWAVE
  - DacModeConstants, [123](#)
  - NBCCCommon.h, [1184](#)
- DAC\_MODE\_SAWPOSWAVE

- DacModeConstants, 123
- NBCCCommon.h, 1184
- DAC\_MODE\_SINEWAVE
  - DacModeConstants, 123
  - NBCCCommon.h, 1184
- DAC\_MODE\_SQUAREWAVE
  - DacModeConstants, 123
  - NBCCCommon.h, 1184
- DAC\_MODE\_TRIANGLEWAVE
  - DacModeConstants, 123
  - NBCCCommon.h, 1184
- DacModeConstants
  - DAC\_MODE\_DCOUT, 123
  - DAC\_MODE\_PWMVOLTAGE, 123
  - DAC\_MODE\_SAWNEGWAVE, 123
  - DAC\_MODE\_SAWPOSWAVE, 123
  - DAC\_MODE\_SINEWAVE, 123
  - DAC\_MODE\_SQUAREWAVE, 123
  - DAC\_MODE\_TRIANGLEWAVE, 123
- Data
  - DrawGraphicArrayType, 1048
  - InputPinFunctionType, 1071
- Data mode constants, 834
- Data type limits, 1010
- DATA\_MODE\_GPS
  - CommDataModeConstants, 834
  - NBCCCommon.h, 1184
- DATA\_MODE\_MASK
  - CommDataModeConstants, 835
  - NBCCCommon.h, 1185
- DATA\_MODE\_NXT
  - CommDataModeConstants, 835
  - NBCCCommon.h, 1185
- DATA\_MODE\_RAW
  - CommDataModeConstants, 835
  - NBCCCommon.h, 1185
- DATA\_MODE\_UPDATE
  - CommDataModeConstants, 835
  - NBCCCommon.h, 1185
- DataAvailable
  - CommHSCheckStatusType, 1027
- DatalogGetTimes
  - NBCCCommon.h, 1185
- SysCallConstants, 685
- DatalogGetTimesType, 1037
  - SyncTick, 1038
  - SyncTime, 1038
- DatalogWrite
  - NBCCCommon.h, 1185
  - SysCallConstants, 685
- DatalogWriteType, 1038
  - Message, 1039
  - Result, 1039
- DataspaceSize
  - MemoryManagerType, 1090
- DEGREES\_PER\_RADIAN
  - MiscConstants, 261
  - NBCCCommon.h, 1185
- DeleteFile
  - LoaderModuleFunctions, 579
  - NXCDefs.h, 1555
- Device status constants, 851
- Dexter Industries API Functions, 233
- Dexter Industries device constants, 979
- Dexter Industries GPS sensor constants, 979
- Dexter Industries IMU Accelerometer control register 1 constants, 1005
- Dexter Industries IMU Accelerometer control register 2 constants, 1006
- Dexter Industries IMU Accelerometer interrupt latch reset register constants, 1004
- Dexter Industries IMU Accelerometer mode control register constants, 1003
- Dexter Industries IMU Accelerometer register constants, 998
- Dexter Industries IMU Accelerometer status register constants, 1002
- Dexter Industries IMU Gyro control register 1 constants, 987
- Dexter Industries IMU Gyro control register 2 constants, 989
- Dexter Industries IMU Gyro control register 3 constants, 991

- Dexter Industries IMU Gyro control register 4 constants, [993](#)
- Dexter Industries IMU Gyro control register 5 constants, [994](#)
- Dexter Industries IMU Gyro FIFO control register onstants, [995](#)
- Dexter Industries IMU Gyro register constants, [983](#)
- Dexter Industries IMU Gyro status register constants, [997](#)
- Dexter Industries IMU sensor constants, [981](#)
- DexterIndustriesAPI
  - ReadSensorDIAccl, [236](#)
  - ReadSensorDIAccl8, [236](#)
  - ReadSensorDIAccl8Raw, [236](#)
  - ReadSensorDIAcclDrift, [237](#)
  - ReadSensorDIAcclRaw, [237](#)
  - ReadSensorDIGyro, [238](#)
  - ReadSensorDIGyroRaw, [238](#)
  - SensorDIAcclStatus, [239](#)
  - SensorDIGPSDistanceToWaypoint, [239](#)
  - SensorDIGPSHeading, [240](#)
  - SensorDIGPSHeadingToWaypoint, [240](#)
  - SensorDIGPSLatitude, [241](#)
  - SensorDIGPSLongitude, [241](#)
  - SensorDIGPSRelativeHeading, [241](#)
  - SensorDIGPSStatus, [242](#)
  - SensorDIGPSTime, [242](#)
  - SensorDIGPSVelocity, [243](#)
  - SensorDIGyroStatus, [243](#)
  - SensorDIGyroTemperature, [243](#)
  - SetSensorDIAccl, [244](#)
  - SetSensorDIAcclDrift, [244](#)
  - SetSensorDIAcclEx, [245](#)
  - SetSensorDIGPSWaypoint, [245](#)
  - SetSensorDIGyro, [246](#)
  - SetSensorDIGyroEx, [246](#)
- DGPS\_REG\_DISTANCE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1185](#)
- DGPS\_REG\_HEADING
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1185](#)
- DGPS\_REG\_LASTANGLE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_LATITUDE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_LONGITUDE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_SETLATITUDE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_SETLONGITUDE
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_STATUS
  - DIGPSConstants, [980](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_TIME
  - DIGPSConstants, [981](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_VELOCITY
  - DIGPSConstants, [981](#)
  - NBCCCommon.h, [1186](#)
- DGPS\_REG\_WAYANGLE
  - DIGPSConstants, [981](#)
  - NBCCCommon.h, [1186](#)
- DI\_ADDR\_ACCL
  - DIIMUConstants, [983](#)
  - NBCCCommon.h, [1186](#)
- DI\_ADDR\_DGPS
  - DIGPSConstants, [981](#)
  - NBCCCommon.h, [1187](#)
- DI\_ADDR\_GYRO
  - DIIMUConstants, [983](#)
  - NBCCCommon.h, [1187](#)
- DIACCL\_CTRL1\_FILT\_BW125
  - DIIMUAccelCtrl1Constants, [1005](#)
  - NBCCCommon.h, [1187](#)
- DIACCL\_CTRL1\_INT2TOINT1
  - DIIMUAccelCtrl1Constants, [1005](#)
  - NBCCCommon.h, [1187](#)
- DIACCL\_CTRL1\_LEVELPULSE
  - DIIMUAccelCtrl1Constants, [1006](#)
  - NBCCCommon.h, [1187](#)
- DIACCL\_CTRL1\_NO\_XDETECT

- DIIMUAccelCtrl1Constants, 1006
- NBCCCommon.h, 1187
- DIACCL\_CTRL1\_NO\_YDETECT
  - DIIMUAccelCtrl1Constants, 1006
  - NBCCCommon.h, 1187
- DIACCL\_CTRL1\_NO\_ZDETECT
  - DIIMUAccelCtrl1Constants, 1006
  - NBCCCommon.h, 1187
- DIACCL\_CTRL1\_PULSELEVEL
  - DIIMUAccelCtrl1Constants, 1006
  - NBCCCommon.h, 1187
- DIACCL\_CTRL1\_PULSEPULSE
  - DIIMUAccelCtrl1Constants, 1006
  - NBCCCommon.h, 1187
- DIACCL\_CTRL1\_THRESH\_INT
  - DIIMUAccelCtrl1Constants, 1006
  - NBCCCommon.h, 1188
- DIACCL\_CTRL2\_DETPOL\_NEGAND
  - DIIMUAccelCtrl2Constants, 1007
  - NBCCCommon.h, 1188
- DIACCL\_CTRL2\_DRIVE\_STRONG
  - DIIMUAccelCtrl2Constants, 1007
  - NBCCCommon.h, 1188
- DIACCL\_CTRL2\_LVLPOL\_NEGAND
  - DIIMUAccelCtrl2Constants, 1007
  - NBCCCommon.h, 1188
- DIACCL\_INTERRUPT\_LATCH\_ -  
CLEAR1
  - DIIMUAccelInterruptLatchCon-  
stants, 1005
  - NBCCCommon.h, 1188
- DIACCL\_INTERRUPT\_LATCH\_ -  
CLEAR2
  - DIIMUAccelInterruptLatchCon-  
stants, 1005
  - NBCCCommon.h, 1188
- DIACCL\_MODE\_GLVL2
  - DIIMUAccelModeConstants, 1003
  - NBCCCommon.h, 1188
- DIACCL\_MODE\_GLVL4
  - DIIMUAccelModeConstants, 1003
  - NBCCCommon.h, 1188
- DIACCL\_MODE\_GLVL8
  - DIIMUAccelModeConstants, 1003
  - NBCCCommon.h, 1188
- DIACCL\_MODE\_LVLDETECT
  - DIIMUAccelModeConstants, 1004
  - NBCCCommon.h, 1189
- DIACCL\_MODE\_MEASURE
  - DIIMUAccelModeConstants, 1004
  - NBCCCommon.h, 1189
- DIACCL\_MODE\_PLSDetect
  - DIIMUAccelModeConstants, 1004
  - NBCCCommon.h, 1189
- DIACCL\_MODE\_STANDBY
  - DIIMUAccelModeConstants, 1004
  - NBCCCommon.h, 1189
- DIACCL\_REG\_CTRL1
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1189
- DIACCL\_REG\_CTRL2
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1189
- DIACCL\_REG\_DETECTSRC
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1189
- DIACCL\_REG\_I2CADDR
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1189
- DIACCL\_REG\_INTLATCH
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1189
- DIACCL\_REG\_LATENCYTM
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1190
- DIACCL\_REG\_LVLDETTTHR
  - DIIMUAccelRegisterConstants, 999
  - NBCCCommon.h, 1190
- DIACCL\_REG\_MODECTRL
  - DIIMUAccelRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_OUTTEMP
  - DIIMUAccelRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_PLSDETTHR
  - DIIMUAccelRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_PLSDURVAL
  - DIIMUAccelRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_STATUS
  - DIIMUAccelRegisterConstants, 1000

- NBCCCommon.h, 1190
- DIACCL\_REG\_TIMEWINDOW
  - DIIMUAcclRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_USERINFO
  - DIIMUAcclRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_WHOAMI
  - DIIMUAcclRegisterConstants, 1000
  - NBCCCommon.h, 1190
- DIACCL\_REG\_X8
  - DIIMUAcclRegisterConstants, 1000
  - NBCCCommon.h, 1191
- DIACCL\_REG\_XHIGH
  - DIIMUAcclRegisterConstants, 1000
  - NBCCCommon.h, 1191
- DIACCL\_REG\_XHIGHDRIFT
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_XLOW
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_XLOWDRIFT
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_Y8
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_YHIGH
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_YHIGHDRIFT
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_YLOW
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_YLOWDRIFT
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1191
- DIACCL\_REG\_Z8
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1192
- DIACCL\_REG\_ZHIGH
  - DIIMUAcclRegisterConstants, 1001
  - NBCCCommon.h, 1192
- DIACCL\_REG\_ZHIGHDRIFT
  - DIIMUAcclRegisterConstants, 1002
  - NBCCCommon.h, 1192
- DIACCL\_REG\_ZLOW
  - DIIMUAcclRegisterConstants, 1002
  - NBCCCommon.h, 1192
- DIACCL\_REG\_ZLOWDRIFT
  - DIIMUAcclRegisterConstants, 1002
  - NBCCCommon.h, 1192
- DIACCL\_STATUS\_DATAOVER
  - DIIMUAcclStatusConstants, 1002
  - NBCCCommon.h, 1192
- DIACCL\_STATUS\_DATAREADY
  - DIIMUAcclStatusConstants, 1002
  - NBCCCommon.h, 1192
- DIACCL\_STATUS\_PARITYERR
  - DIIMUAcclStatusConstants, 1003
  - NBCCCommon.h, 1192
- DIGI\_PIN0
  - DigitalPinConstants, 125
  - NBCCCommon.h, 1192
- DIGI\_PIN1
  - DigitalPinConstants, 125
  - NBCCCommon.h, 1193
- DIGI\_PIN2
  - DigitalPinConstants, 125
  - NBCCCommon.h, 1193
- DIGI\_PIN3
  - DigitalPinConstants, 125
  - NBCCCommon.h, 1193
- DIGI\_PIN4
  - DigitalPinConstants, 126
  - NBCCCommon.h, 1193
- DIGI\_PIN5
  - DigitalPinConstants, 126
  - NBCCCommon.h, 1193
- DIGI\_PIN6
  - DigitalPinConstants, 126
  - NBCCCommon.h, 1193
- DIGI\_PIN7
  - DigitalPinConstants, 126
  - NBCCCommon.h, 1193
- DigitalPinConstants
  - DIGI\_PIN0, 125
  - DIGI\_PIN1, 125
  - DIGI\_PIN2, 125

- DIGI\_PIN3, [125](#)
- DIGI\_PIN4, [126](#)
- DIGI\_PIN5, [126](#)
- DIGI\_PIN6, [126](#)
- DIGI\_PIN7, [126](#)
- DIGPSConstants
  - DGPS\_REG\_DISTANCE, [980](#)
  - DGPS\_REG\_HEADING, [980](#)
  - DGPS\_REG\_LASTANGLE, [980](#)
  - DGPS\_REG\_LATITUDE, [980](#)
  - DGPS\_REG\_LONGITUDE, [980](#)
  - DGPS\_REG\_SETLATITUDE, [980](#)
  - DGPS\_REG\_SETLONGITUDE, [980](#)
  - DGPS\_REG\_STATUS, [980](#)
  - DGPS\_REG\_TIME, [981](#)
  - DGPS\_REG\_VELOCITY, [981](#)
  - DGPS\_REG\_WAYANGLE, [981](#)
  - DI\_ADDR\_DGPS, [981](#)
- DIGYRO\_CTRL1\_BANDWIDTH\_1
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1193](#)
- DIGYRO\_CTRL1\_BANDWIDTH\_2
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_BANDWIDTH\_3
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_BANDWIDTH\_4
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_100
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_200
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_400
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_800
  - DIIMUGyroCtrl1Constants, [988](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_NORMAL
  - DIIMUGyroCtrl1Constants, [989](#)
  - NBCCCommon.h, [1194](#)
- DIGYRO\_CTRL1\_POWERDOWN
  - DIIMUGyroCtrl1Constants, [989](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL1\_XENABLE
  - DIIMUGyroCtrl1Constants, [989](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL1\_YENABLE
  - DIIMUGyroCtrl1Constants, [989](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL1\_ZENABLE
  - DIIMUGyroCtrl1Constants, [989](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_001
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_002
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_005
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_01
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_02
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_05
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_1
  - DIIMUGyroCtrl2Constants, [990](#)
  - NBCCCommon.h, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_2
  - DIIMUGyroCtrl2Constants, [991](#)
  - NBCCCommon.h, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_4
  - DIIMUGyroCtrl2Constants, [991](#)
  - NBCCCommon.h, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_FREQ\_8
  - DIIMUGyroCtrl2Constants, [991](#)
  - NBCCCommon.h, [1196](#)

- DIGYRO\_CTRL2\_HPMODE\_-  
  AUTOINT  
    DIIMUGyroCtrl2Constants, 991  
    NBCCCommon.h, 1196
- DIGYRO\_CTRL2\_HPMODE\_-  
  NORMAL  
    DIIMUGyroCtrl2Constants, 991  
    NBCCCommon.h, 1196
- DIGYRO\_CTRL2\_HPMODE\_REFSIG  
  DIIMUGyroCtrl2Constants, 991  
  NBCCCommon.h, 1196
- DIGYRO\_CTRL2\_HPMODE\_RESET  
  DIIMUGyroCtrl2Constants, 991  
  NBCCCommon.h, 1196
- DIGYRO\_CTRL3\_INT1\_BOOT  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1196
- DIGYRO\_CTRL3\_INT1\_ENABLE  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1196
- DIGYRO\_CTRL3\_INT1\_LOWACTIVE  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL3\_INT2\_DATAREADY  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL3\_INT2\_EMPTY  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL3\_INT2\_OVERRUN  
  DIIMUGyroCtrl3Constants, 992  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL3\_INT2\_-  
  WATERMARK  
    DIIMUGyroCtrl3Constants, 992  
    NBCCCommon.h, 1197
- DIGYRO\_CTRL3\_OPENDRAIN  
  DIIMUGyroCtrl3Constants, 993  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL4\_BIGENDIAN  
  DIIMUGyroCtrl4Constants, 993  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL4\_BLOCKDATA  
  DIIMUGyroCtrl4Constants, 993  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL4\_SCALE\_2000  
  DIIMUGyroCtrl4Constants, 993  
  NBCCCommon.h, 1197
- DIGYRO\_CTRL4\_SCALE\_250  
  DIIMUGyroCtrl4Constants, 994  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL4\_SCALE\_500  
  DIIMUGyroCtrl4Constants, 994  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_FIFOENABLE  
  DIIMUGyroCtrl5Constants, 994  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_HPENABLE  
  DIIMUGyroCtrl5Constants, 994  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_1  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_2  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_3  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_1  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_2  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_3  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1199
- DIGYRO\_CTRL5\_REBOOTMEM  
  DIIMUGyroCtrl5Constants, 995  
  NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_BYPASS  
  DIIMUGyroFifoCtrlConstants, 996  
  NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_-  
  BYPASS2STREAM  
    DIIMUGyroFifoCtrlConstants, 996  
    NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_FIFO  
  DIIMUGyroFifoCtrlConstants, 996  
  NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_STREAM

- DIIMUGyroFifoCtrlConstants, 996
- NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_STREAM2FIFO
  - DIIMUGyroFifoCtrlConstants, 996
  - NBCCCommon.h, 1199
- DIGYRO\_FIFOCTRL\_WATERMARK\_MASK
  - DIIMUGyroFifoCtrlConstants, 996
  - NBCCCommon.h, 1199
- DIGYRO\_REG\_CTRL1
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1199
- DIGYRO\_REG\_CTRL1AUTO
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1199
- DIGYRO\_REG\_CTRL2
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_CTRL3
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_CTRL4
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_CTRL5
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_FIFOCTRL
  - DIIMUGyroRegisterConstants, 984
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_FIFOSRC
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_INT1\_CFG
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_INT1\_DUR
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_INT1\_SRC
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_INT1\_XHI
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1200
- DIGYRO\_REG\_INT1\_XLO
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_INT1\_YHI
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_INT1\_YLO
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_INT1\_ZHI
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_INT1\_ZLO
  - DIIMUGyroRegisterConstants, 985
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_OUTTEMP
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_REFERENCE
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_STATUS
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_TEMPAUTO
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_WHOAMI
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1201
- DIGYRO\_REG\_XHIGH
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_XLOW
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_XLOWBURST
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_YHIGH
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_YLOW
  - DIIMUGyroRegisterConstants, 986
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_YLOWBURST
  - DIIMUGyroRegisterConstants, 987

- NBCCCommon.h, 1202
- DIGYRO\_REG\_ZHIGH
  - DIIMUGyroRegisterConstants, 987
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_ZLOW
  - DIIMUGyroRegisterConstants, 987
  - NBCCCommon.h, 1202
- DIGYRO\_REG\_ZLOWBURST
  - DIIMUGyroRegisterConstants, 987
  - NBCCCommon.h, 1202
- DIGYRO\_STATUS\_XDATA
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1202
- DIGYRO\_STATUS\_XOVER
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_XYZDATA
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_XYZOVER
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_YDATA
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_YOVER
  - DIIMUGyroStatusConstants, 997
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_ZDATA
  - DIIMUGyroStatusConstants, 998
  - NBCCCommon.h, 1203
- DIGYRO\_STATUS\_ZOVER
  - DIIMUGyroStatusConstants, 998
  - NBCCCommon.h, 1203
- DIIMUAccelCtrl1Constants
  - DIACCL\_CTRL1\_FILT\_BW125, 1005
  - DIACCL\_CTRL1\_INT2TOINT1, 1005
  - DIACCL\_CTRL1\_LEVELPULSE, 1006
  - DIACCL\_CTRL1\_NO\_XDETECT, 1006
  - DIACCL\_CTRL1\_NO\_YDETECT, 1006
  - DIACCL\_CTRL1\_NO\_ZDETECT, 1006
  - DIACCL\_CTRL1\_PULSELEVEL, 1006
  - DIACCL\_CTRL1\_PULSEPULSE, 1006
  - DIACCL\_CTRL1\_THRESH\_INT, 1006
- DIIMUAccelCtrl2Constants
  - DIACCL\_CTRL2\_DETPOL\_-NEGAND, 1007
  - DIACCL\_CTRL2\_DRIVE\_-STRONG, 1007
  - DIACCL\_CTRL2\_LVLPOL\_-NEGAND, 1007
- DIIMUAccelInterruptLatchConstants
  - DIACCL\_INTERRUPT\_LATCH\_-CLEAR1, 1005
  - DIACCL\_INTERRUPT\_LATCH\_-CLEAR2, 1005
- DIIMUAccelModeConstants
  - DIACCL\_MODE\_GLVL2, 1003
  - DIACCL\_MODE\_GLVL4, 1003
  - DIACCL\_MODE\_GLVL8, 1003
  - DIACCL\_MODE\_LVLDETECT, 1004
  - DIACCL\_MODE\_MEASURE, 1004
  - DIACCL\_MODE\_PLSDETECT, 1004
  - DIACCL\_MODE\_STANDBY, 1004
- DIIMUAccelStatusConstants
  - DIACCL\_STATUS\_DATAOVER, 1002
  - DIACCL\_STATUS\_DATAREADY, 1002
  - DIACCL\_STATUS\_PARITYERR, 1003
- DIIMUAccelRegisterConstants
  - DIACCL\_REG\_CTRL1, 999
  - DIACCL\_REG\_CTRL2, 999
  - DIACCL\_REG\_DETECTSRC, 999
  - DIACCL\_REG\_I2CADDR, 999
  - DIACCL\_REG\_INTLATCH, 999
  - DIACCL\_REG\_LATENCYTM, 999
  - DIACCL\_REG\_LVLDETTTHR, 999

- DIACCL\_REG\_MODECTRL, 1000
- DIACCL\_REG\_OUTTEMP, 1000
- DIACCL\_REG\_PLSDETTHR, 1000
- DIACCL\_REG\_PLSDURVAL, 1000
- DIACCL\_REG\_STATUS, 1000
- DIACCL\_REG\_TIMEWINDOW, 1000
- DIACCL\_REG\_USERINFO, 1000
- DIACCL\_REG\_WHOAMI, 1000
- DIACCL\_REG\_X8, 1000
- DIACCL\_REG\_XHIGH, 1000
- DIACCL\_REG\_XHIGHDRIFT, 1001
- DIACCL\_REG\_XLOW, 1001
- DIACCL\_REG\_XLOWDRIFT, 1001
- DIACCL\_REG\_Y8, 1001
- DIACCL\_REG\_YHIGH, 1001
- DIACCL\_REG\_YHIGHDRIFT, 1001
- DIACCL\_REG\_YLOW, 1001
- DIACCL\_REG\_YLOWDRIFT, 1001
- DIACCL\_REG\_Z8, 1001
- DIACCL\_REG\_ZHIGH, 1001
- DIACCL\_REG\_ZHIGHDRIFT, 1002
- DIACCL\_REG\_ZLOW, 1002
- DIACCL\_REG\_ZLOWDRIFT, 1002
- DIIMUConstants
  - DI\_ADDR\_ACCL, 983
  - DI\_ADDR\_GYRO, 983
- DIIMUGyroCtrl1Constants
  - DIGYRO\_CTRL1\_-BANDWIDTH\_1, 988
  - DIGYRO\_CTRL1\_-BANDWIDTH\_2, 988
  - DIGYRO\_CTRL1\_-BANDWIDTH\_3, 988
  - DIGYRO\_CTRL1\_-BANDWIDTH\_4, 988
  - DIGYRO\_CTRL1\_DATARATE\_-100, 988
  - DIGYRO\_CTRL1\_DATARATE\_-200, 988
  - DIGYRO\_CTRL1\_DATARATE\_-400, 988
  - DIGYRO\_CTRL1\_DATARATE\_-800, 988
  - DIGYRO\_CTRL1\_NORMAL, 989
  - DIGYRO\_CTRL1\_POWERDOWN, 989
  - DIGYRO\_CTRL1\_XENABLE, 989
  - DIGYRO\_CTRL1\_YENABLE, 989
  - DIGYRO\_CTRL1\_ZENABLE, 989
- DIIMUGyroCtrl2Constants
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_001, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_002, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_005, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_01, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_02, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_05, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_1, 990
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_2, 991
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_4, 991
  - DIGYRO\_CTRL2\_CUTOFF\_-FREQ\_8, 991
  - DIGYRO\_CTRL2\_HPMODE\_-AUTOINT, 991
  - DIGYRO\_CTRL2\_HPMODE\_-NORMAL, 991
  - DIGYRO\_CTRL2\_HPMODE\_-REFSIG, 991
  - DIGYRO\_CTRL2\_HPMODE\_-RESET, 991
- DIIMUGyroCtrl3Constants
  - DIGYRO\_CTRL3\_INT1\_BOOT, 992
  - DIGYRO\_CTRL3\_INT1\_ENABLE, 992

- DIGYRO\_CTRL3\_INT1\_-  
LOWACTIVE, 992
- DIGYRO\_CTRL3\_INT2\_-  
DATAREADY, 992
- DIGYRO\_CTRL3\_INT2\_EMPTY,  
992
- DIGYRO\_CTRL3\_INT2\_-  
OVERRUN, 992
- DIGYRO\_CTRL3\_INT2\_-  
WATERMARK, 992
- DIGYRO\_CTRL3\_OPENDRAIN,  
993
- DIIMUGyroCtrl4Constants
  - DIGYRO\_CTRL4\_BIGENDIAN,  
993
  - DIGYRO\_CTRL4\_BLOCKDATA,  
993
  - DIGYRO\_CTRL4\_SCALE\_2000,  
993
  - DIGYRO\_CTRL4\_SCALE\_250,  
994
  - DIGYRO\_CTRL4\_SCALE\_500,  
994
- DIIMUGyroCtrl5Constants
  - DIGYRO\_CTRL5\_FIFOENABLE,  
994
  - DIGYRO\_CTRL5\_HPENABLE,  
994
  - DIGYRO\_CTRL5\_INT1\_SEL\_1,  
995
  - DIGYRO\_CTRL5\_INT1\_SEL\_2,  
995
  - DIGYRO\_CTRL5\_INT1\_SEL\_3,  
995
  - DIGYRO\_CTRL5\_OUT\_SEL\_1,  
995
  - DIGYRO\_CTRL5\_OUT\_SEL\_2,  
995
  - DIGYRO\_CTRL5\_OUT\_SEL\_3,  
995
  - DIGYRO\_CTRL5\_REBOOTMEM,  
995
- DIIMUGyroFifoCtrlConstants
  - DIGYRO\_FIFOCTRL\_BYPASS,  
996
  - DIGYRO\_FIFOCTRL\_-  
BYPASS2STREAM, 996
  - DIGYRO\_FIFOCTRL\_FIFO, 996
  - DIGYRO\_FIFOCTRL\_STREAM,  
996
  - DIGYRO\_FIFOCTRL\_-  
STREAM2FIFO, 996
  - DIGYRO\_FIFOCTRL\_-  
WATERMARK\_MASK,  
996
- DIIMUGyroRegisterConstants
  - DIGYRO\_REG\_CTRL1, 984
  - DIGYRO\_REG\_CTRL1AUTO, 984
  - DIGYRO\_REG\_CTRL2, 984
  - DIGYRO\_REG\_CTRL3, 984
  - DIGYRO\_REG\_CTRL4, 984
  - DIGYRO\_REG\_CTRL5, 984
  - DIGYRO\_REG\_FIFOCTRL, 984
  - DIGYRO\_REG\_FIFOSRC, 985
  - DIGYRO\_REG\_INT1\_CFG, 985
  - DIGYRO\_REG\_INT1\_DUR, 985
  - DIGYRO\_REG\_INT1\_SRC, 985
  - DIGYRO\_REG\_INT1\_XHI, 985
  - DIGYRO\_REG\_INT1\_XLO, 985
  - DIGYRO\_REG\_INT1\_YHI, 985
  - DIGYRO\_REG\_INT1\_YLO, 985
  - DIGYRO\_REG\_INT1\_ZHI, 985
  - DIGYRO\_REG\_INT1\_ZLO, 985
  - DIGYRO\_REG\_OUTTEMP, 986
  - DIGYRO\_REG\_REFERENCE, 986
  - DIGYRO\_REG\_STATUS, 986
  - DIGYRO\_REG\_TEMPAUTO, 986
  - DIGYRO\_REG\_WHOAMI, 986
  - DIGYRO\_REG\_XHIGH, 986
  - DIGYRO\_REG\_XLOW, 986
  - DIGYRO\_REG\_XLOWBURST,  
986
  - DIGYRO\_REG\_YHIGH, 986
  - DIGYRO\_REG\_YLOW, 986
  - DIGYRO\_REG\_YLOWBURST,  
987
  - DIGYRO\_REG\_ZHIGH, 987
  - DIGYRO\_REG\_ZLOW, 987
  - DIGYRO\_REG\_ZLOWBURST,  
987
- DIIMUGyroStatusConstants

- DIGYRO\_STATUS\_XDATA, [997](#)
- DIGYRO\_STATUS\_XOVER, [997](#)
- DIGYRO\_STATUS\_XYZDATA, [997](#)
- DIGYRO\_STATUS\_XYZOVER, [997](#)
- DIGYRO\_STATUS\_YDATA, [997](#)
- DIGYRO\_STATUS\_YOVER, [997](#)
- DIGYRO\_STATUS\_ZDATA, [998](#)
- DIGYRO\_STATUS\_ZOVER, [998](#)
- Direct Command functions, [517](#)
- Display contrast constants, [824](#)
- Display flags, [823](#)
- Display module, [58](#)
- Display module constants, [810](#)
- Display module functions, [348](#)
- Display module IOMAP offsets, [827](#)
- Display module types, [347](#)
- DISPLAY\_BUSY
  - DisplayFlagsGroup, [824](#)
  - NBCCCommon.h, [1203](#)
- DISPLAY\_CHAR
  - DisplayExecuteFunctionConstants, [817](#)
  - NBCCCommon.h, [1203](#)
- DISPLAY\_CONTRAST\_DEFAULT
  - DisplayContrastConstants, [825](#)
  - NBCCCommon.h, [1203](#)
- DISPLAY\_CONTRAST\_MAX
  - DisplayContrastConstants, [825](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_ERASE\_ALL
  - DisplayExecuteFunctionConstants, [817](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_ERASE\_LINE
  - DisplayExecuteFunctionConstants, [817](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_FILL\_REGION
  - DisplayExecuteFunctionConstants, [817](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_FRAME
  - DisplayExecuteFunctionConstants, [817](#)
- NBCCCommon.h, [1204](#)
- DISPLAY\_HEIGHT
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_HORIZONTAL\_LINE
  - DisplayExecuteFunctionConstants, [817](#)
  - NBCCCommon.h, [1204](#)
- DISPLAY\_MENUICONS\_X\_DIFF
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_MENUICONS\_X\_OFFS
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_MENUICONS\_Y
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_ON
  - DisplayFlagsGroup, [824](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_PIXEL
  - DisplayExecuteFunctionConstants, [818](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_POPUP
  - DisplayFlagsGroup, [824](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_REFRESH
  - DisplayFlagsGroup, [824](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_REFRESH\_DISABLED
  - DisplayFlagsGroup, [824](#)
  - NBCCCommon.h, [1205](#)
- DISPLAY\_VERTICAL\_LINE
  - DisplayExecuteFunctionConstants, [818](#)
  - NBCCCommon.h, [1206](#)
- DISPLAY\_WIDTH
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1206](#)
- DisplayContrast
  - DisplayModuleFunctions, [354](#)
  - NXCDefs.h, [1555](#)
- DisplayContrastConstants
  - DISPLAY\_CONTRAST\_DEFAULT, [825](#)

- DISPLAY\_CONTRAST\_MAX, 825
- DisplayDisplay
  - DisplayModuleFunctions, 354
  - NXCDefs.h, 1556
- DisplayDrawOptionConstants
  - DRAW\_OPT\_CLEAR, 819
  - DRAW\_OPT\_CLEAR\_EXCEPT\_-STATUS\_SCREEN, 819
  - DRAW\_OPT\_CLEAR\_PIXELS, 819
  - DRAW\_OPT\_CLEAR\_SCREEN\_-MODES, 819
  - DRAW\_OPT\_CLEAR\_WHOLE\_-SCREEN, 819
  - DRAW\_OPT\_FILL\_SHAPE, 820
  - DRAW\_OPT\_INVERT, 820
  - DRAW\_OPT\_LOGICAL\_AND, 820
  - DRAW\_OPT\_LOGICAL\_COPY, 820
  - DRAW\_OPT\_LOGICAL\_-OPERATIONS, 820
  - DRAW\_OPT\_LOGICAL\_OR, 820
  - DRAW\_OPT\_LOGICAL\_XOR, 821
  - DRAW\_OPT\_NORMAL, 821
  - DRAW\_OPT\_POLYGON\_-POLYLINE, 821
- DisplayEraseMask
  - DisplayModuleFunctions, 354
  - NXCDefs.h, 1556
- DisplayExecuteFunction
  - NBCCCommon.h, 1206
  - SysCallConstants, 685
- DisplayExecuteFunction constants, 816
- DisplayExecuteFunctionConstants
  - DISPLAY\_CHAR, 817
  - DISPLAY\_ERASE\_ALL, 817
  - DISPLAY\_ERASE\_LINE, 817
  - DISPLAY\_FILL\_REGION, 817
  - DISPLAY\_FRAME, 817
  - DISPLAY\_HORIZONTAL\_LINE, 817
  - DISPLAY\_PIXEL, 818
  - DISPLAY\_VERTICAL\_LINE, 818
- DisplayExecuteFunctionType, 1039
- Cmd, 1040
- On, 1040
- Status, 1041
- X1, 1041
- X2, 1041
- Y1, 1041
- Y2, 1041
- DisplayFlags
  - DisplayModuleFunctions, 355
  - NXCDefs.h, 1556
- DisplayFlagsGroup
  - DISPLAY\_BUSY, 824
  - DISPLAY\_ON, 824
  - DISPLAY\_POPUP, 824
  - DISPLAY\_REFRESH, 824
  - DISPLAY\_REFRESH\_DISABLED, 824
- DisplayFont
  - DisplayModuleFunctions, 355
  - NXCDefs.h, 1557
- DisplayFontDrawOptionConstants
  - DRAW\_OPT\_FONT\_DIR\_B2TL, 822
  - DRAW\_OPT\_FONT\_DIR\_B2TR, 822
  - DRAW\_OPT\_FONT\_DIR\_L2RB, 822
  - DRAW\_OPT\_FONT\_DIR\_L2RT, 822
  - DRAW\_OPT\_FONT\_DIR\_R2LB, 822
  - DRAW\_OPT\_FONT\_DIR\_R2LT, 823
  - DRAW\_OPT\_FONT\_DIR\_T2BL, 823
  - DRAW\_OPT\_FONT\_DIR\_T2BR, 823
  - DRAW\_OPT\_FONT\_-DIRECTIONS, 823
  - DRAW\_OPT\_FONT\_WRAP, 823
- DisplayIOMAP
  - DisplayOffsetContrast, 828
  - DisplayOffsetDisplay, 828
  - DisplayOffsetEraseMask, 828
  - DisplayOffsetFlags, 828
  - DisplayOffsetNormal, 828

- DisplayOffsetPBitmaps, 828
- DisplayOffsetPFont, 828
- DisplayOffsetPFunc, 828
- DisplayOffsetPMenuIcons, 828
- DisplayOffsetPMenuText, 829
- DisplayOffsetPopup, 829
- DisplayOffsetPScreens, 829
- DisplayOffsetPStatusIcons, 829
- DisplayOffsetPStatusText, 829
- DisplayOffsetPStepIcons, 829
- DisplayOffsetPTextLines, 829
- DisplayOffsetStatusIcons, 829
- DisplayOffsetStepIcons, 829
- DisplayOffsetTextLinesCenterFlags, 829
- DisplayOffsetUpdateMask, 830
- DisplayModuleConstants
  - BITMAP\_1, 812
  - BITMAP\_2, 812
  - BITMAP\_3, 812
  - BITMAP\_4, 812
  - BITMAPS, 813
  - DISPLAY\_HEIGHT, 813
  - DISPLAY\_MENUICONS\_X\_DIFF, 813
  - DISPLAY\_MENUICONS\_X\_OFFS, 813
  - DISPLAY\_MENUICONS\_Y, 813
  - DISPLAY\_WIDTH, 813
  - FRAME\_SELECT, 813
  - MENUICON\_CENTER, 813
  - MENUICON\_LEFT, 814
  - MENUICON\_RIGHT, 814
  - MENUICONS, 814
  - MENUTEXT, 814
  - SCREEN\_BACKGROUND, 814
  - SCREEN\_LARGE, 814
  - SCREEN\_MODE\_CLEAR, 814
  - SCREEN\_MODE\_RESTORE, 814
  - SCREEN\_SMALL, 815
  - SCREENS, 815
  - SPECIALS, 815
  - STATUSICON\_BATTERY, 815
  - STATUSICON\_BLUETOOTH, 815
  - STATUSICON\_USB, 815
  - STATUSICON\_VM, 815
- STATUSICONS, 815
- STATUSTEXT, 815
- STEPICON\_1, 815
- STEPICON\_2, 816
- STEPICON\_3, 816
- STEPICON\_4, 816
- STEPICON\_5, 816
- STEPICONS, 816
- STEPLINE, 816
- TOPLINE, 816
- DisplayModuleFunctions
  - CircleOut, 352
  - ClearLine, 353
  - ClearScreen, 353
  - DisplayContrast, 354
  - DisplayDisplay, 354
  - DisplayEraseMask, 354
  - DisplayFlags, 355
  - DisplayFont, 355
  - DisplayTextLinesCenterFlags, 355
  - DisplayUpdateMask, 356
  - EllipseOut, 356
  - FontNumOut, 357
  - FontTextOut, 357
  - GetDisplayNormal, 358
  - GetDisplayPopup, 359
  - GraphicArrayOut, 359
  - GraphicArrayOutEx, 360
  - GraphicOut, 361
  - GraphicOutEx, 361
  - LineOut, 362
  - NumOut, 362
  - PointOut, 363
  - PolyOut, 364
  - RectOut, 365
  - ResetScreen, 365
  - SetDisplayContrast, 366
  - SetDisplayDisplay, 366
  - SetDisplayEraseMask, 366
  - SetDisplayFlags, 367
  - SetDisplayFont, 367
  - SetDisplayNormal, 367
  - SetDisplayPopup, 368
  - SetDisplayTextLinesCenterFlags, 368
  - SetDisplayUpdateMask, 368

- SysDisplayExecuteFunction, 369
- SysDrawCircle, 369
- SysDrawEllipse, 369
- SysDrawFont, 370
- SysDrawGraphic, 370
- SysDrawGraphicArray, 371
- SysDrawLine, 371
- SysDrawPoint, 371
- SysDrawPolygon, 372
- SysDrawRect, 372
- SysDrawText, 372
- SysSetScreenMode, 373
- TextOut, 373
- DisplayModuleID
  - ModuleIDConstants, 259
  - NBCCCommon.h, 1206
- DisplayModuleName
  - ModuleNameConstants, 257
  - NBCCCommon.h, 1206
- DisplayOffsetContrast
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1206
- DisplayOffsetDisplay
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1206
- DisplayOffsetEraseMask
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1206
- DisplayOffsetFlags
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1206
- DisplayOffsetNormal
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1207
- DisplayOffsetPBitmaps
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1207
- DisplayOffsetPFont
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1207
- DisplayOffsetPFunc
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1207
- DisplayOffsetPMenuIcons
  - DisplayIOMAP, 828
  - NBCCCommon.h, 1207
- DisplayOffsetPMenuText
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1207
- DisplayOffsetPopup
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1207
- DisplayOffsetPScreens
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1207
- DisplayOffsetPStatusIcons
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1207
- DisplayOffsetPStatusText
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1207
- DisplayOffsetPStepIcons
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1208
- DisplayOffsetPTextLines
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1208
- DisplayOffsetStatusIcons
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1208
- DisplayOffsetStepIcons
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1208
- DisplayOffsetTextLinesCenterFlags
  - DisplayIOMAP, 829
  - NBCCCommon.h, 1208
- DisplayOffsetUpdateMask
  - DisplayIOMAP, 830
  - NBCCCommon.h, 1208
- DisplayTextLineConstants
  - TEXTLINE\_1, 826
  - TEXTLINE\_2, 826
  - TEXTLINE\_3, 826
  - TEXTLINE\_4, 826
  - TEXTLINE\_5, 826
  - TEXTLINE\_6, 826
  - TEXTLINE\_7, 827
  - TEXTLINE\_8, 827
  - TEXTLINES, 827
- DisplayTextLinesCenterFlags
  - DisplayModuleFunctions, 355
  - NXCDefs.h, 1557

- DisplayUpdateMask
  - DisplayModuleFunctions, 356
  - NXCDefs.h, 1557
- DIST\_CMD\_CUSTOM
  - MSDistNX, 939
  - NBCCCommon.h, 1208
- DIST\_CMD\_GP2D12
  - MSDistNX, 939
  - NBCCCommon.h, 1208
- DIST\_CMD\_GP2D120
  - MSDistNX, 939
  - NBCCCommon.h, 1208
- DIST\_CMD\_GP2YA02
  - MSDistNX, 939
  - NBCCCommon.h, 1208
- DIST\_CMD\_GP2YA21
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_DIST
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_DIST1
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_DIST\_MAX
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_DIST\_MIN
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_MODULE\_TYPE
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_NUM\_POINTS
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_VOLT
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DIST\_REG\_VOLT1
  - MSDistNX, 940
  - NBCCCommon.h, 1209
- DISTNxDistance
  - MindSensorsAPI, 149
  - NXCDefs.h, 1558
- DISTNxGP2D12
  - MindSensorsAPI, 150
  - NXCDefs.h, 1558
- DISTNxGP2D120
  - MindSensorsAPI, 150
  - NXCDefs.h, 1558
- DISTNxGP2YA02
  - MindSensorsAPI, 151
  - NXCDefs.h, 1559
- DISTNxGP2YA21
  - MindSensorsAPI, 151
  - NXCDefs.h, 1559
- DISTNxMaxDistance
  - MindSensorsAPI, 152
  - NXCDefs.h, 1560
- DISTNxMinDistance
  - MindSensorsAPI, 152
  - NXCDefs.h, 1560
- DISTNxModuleType
  - MindSensorsAPI, 153
  - NXCDefs.h, 1561
- DISTNxNumPoints
  - MindSensorsAPI, 153
  - NXCDefs.h, 1561
- DISTNxVoltage
  - MindSensorsAPI, 153
  - NXCDefs.h, 1562
- div
  - cstdlibAPI, 649
  - NXCDefs.h, 1562
- div\_t, 1042
  - quot, 1042
  - rem, 1042
- DRAW\_OPT\_CLEAR
  - DisplayDrawOptionConstants, 819
  - NBCCCommon.h, 1209
- DRAW\_OPT\_CLEAR\_EXCEPT\_  
STATUS\_SCREEN
  - DisplayDrawOptionConstants, 819
  - NBCCCommon.h, 1210
- DRAW\_OPT\_CLEAR\_PIXELS
  - DisplayDrawOptionConstants, 819
  - NBCCCommon.h, 1210
- DRAW\_OPT\_CLEAR\_SCREEN\_  
MODES
  - DisplayDrawOptionConstants, 819
  - NBCCCommon.h, 1210

- DRAW\_OPT\_CLEAR\_WHOLE\_-  
  SCREEN  
    DisplayDrawOptionConstants, [819](#)  
    NBCCCommon.h, [1210](#)
- DRAW\_OPT\_FILL\_SHAPE  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1210](#)
- DRAW\_OPT\_FONT\_DIR\_B2TL  
    DisplayFontDrawOptionConstants,  
      [822](#)  
    NBCCCommon.h, [1210](#)
- DRAW\_OPT\_FONT\_DIR\_B2TR  
    DisplayFontDrawOptionConstants,  
      [822](#)  
    NBCCCommon.h, [1210](#)
- DRAW\_OPT\_FONT\_DIR\_L2RB  
    DisplayFontDrawOptionConstants,  
      [822](#)  
    NBCCCommon.h, [1210](#)
- DRAW\_OPT\_FONT\_DIR\_L2RT  
    DisplayFontDrawOptionConstants,  
      [822](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT\_DIR\_R2LB  
    DisplayFontDrawOptionConstants,  
      [822](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT\_DIR\_R2LT  
    DisplayFontDrawOptionConstants,  
      [823](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT\_DIR\_T2BL  
    DisplayFontDrawOptionConstants,  
      [823](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT\_DIR\_T2BR  
    DisplayFontDrawOptionConstants,  
      [823](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT DIRECTIONS  
    DisplayFontDrawOptionConstants,  
      [823](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_FONT\_WRAP  
    DisplayFontDrawOptionConstants,  
      [823](#)  
    NBCCCommon.h, [1211](#)
- DRAW\_OPT\_INVERT  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_LOGICAL\_AND  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_LOGICAL\_COPY  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_LOGICAL\_-  
  OPERATIONS  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_LOGICAL\_OR  
    DisplayDrawOptionConstants, [820](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_LOGICAL\_XOR  
    DisplayDrawOptionConstants, [821](#)  
    NBCCCommon.h, [1212](#)
- DRAW\_OPT\_NORMAL  
    DisplayDrawOptionConstants, [821](#)  
    NBCCCommon.h, [1213](#)
- DRAW\_OPT\_POLYGON\_POLYLINE  
    DisplayDrawOptionConstants, [821](#)  
    NBCCCommon.h, [1213](#)
- DrawCircle  
    NBCCCommon.h, [1213](#)  
    SysCallConstants, [685](#)
- DrawCircleType, [1043](#)  
  Center, [1043](#)  
  Options, [1043](#)  
  Result, [1044](#)  
  Size, [1044](#)
- DrawEllipse  
    NBCCCommon.h, [1213](#)  
    SysCallConstants, [686](#)
- DrawEllipseType, [1044](#)  
  Center, [1045](#)  
  Options, [1045](#)  
  Result, [1045](#)  
  SizeX, [1045](#)  
  SizeY, [1045](#)
- DrawFont  
    NBCCCommon.h, [1213](#)  
    SysCallConstants, [686](#)

- DrawFontType, 1046
  - Filename, 1046
  - Location, 1046
  - Options, 1047
  - Result, 1047
  - Text, 1047
- DrawGraphic
  - NBCCCommon.h, 1213
  - SysCallConstants, 686
- DrawGraphicArray
  - NBCCCommon.h, 1213
  - SysCallConstants, 686
- DrawGraphicArrayType, 1047
  - Data, 1048
  - Location, 1048
  - Options, 1048
  - Result, 1048
  - Variables, 1048
- DrawGraphicType, 1049
  - Filename, 1049
  - Location, 1050
  - Options, 1050
  - Result, 1050
  - Variables, 1050
- Drawing option constants, 818
- DrawLine
  - NBCCCommon.h, 1214
  - SysCallConstants, 686
- DrawLineType, 1050
  - EndLoc, 1051
  - Options, 1051
  - Result, 1051
  - StartLoc, 1051
- DrawPoint
  - NBCCCommon.h, 1214
  - SysCallConstants, 686
- DrawPointType, 1052
  - Location, 1052
  - Options, 1053
  - Result, 1053
- DrawPolygon
  - NBCCCommon.h, 1214
  - SysCallConstants, 686
- DrawPolygonType, 1053
  - Options, 1054
  - Points, 1054
  - Result, 1054
- DrawRect
  - NBCCCommon.h, 1214
  - SysCallConstants, 686
- DrawRectType, 1054
  - Location, 1055
  - Options, 1055
  - Result, 1055
  - Size, 1055
- DrawText
  - NBCCCommon.h, 1214
  - SysCallConstants, 686
- DrawTextType, 1056
  - Location, 1056
  - Options, 1056
  - Result, 1057
  - Text, 1057
- Duration
  - SoundPlayToneType, 1107
  - Tone, 1109
- E-Meter sensor constants, 808
- EllipseOut
  - DisplayModuleFunctions, 356
  - NXCDefs.h, 1563
- EMETER\_REG\_AIN
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1214
- EMETER\_REG\_AOUT
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1214
- EMETER\_REG\_JOULES
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1214
- EMETER\_REG\_VIN
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1214
- EMETER\_REG\_VOUT
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1215
- EMETER\_REG\_WIN
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1215
- EMETER\_REG\_WOUT
  - EMeterI2CConstants, 809
  - NBCCCommon.h, 1215

- EMeterI2CConstants
  - EMETER\_REG\_AIN, 809
  - EMETER\_REG\_AOUT, 809
  - EMETER\_REG\_JOULES, 809
  - EMETER\_REG\_VIN, 809
  - EMETER\_REG\_VOUT, 809
  - EMETER\_REG\_WIN, 809
  - EMETER\_REG\_WOUT, 809
- EndLoc
  - DrawLineType, 1051
- EOF
  - LoaderModuleConstants, 717
  - NBCCCommon.h, 1215
- EQ
  - cmpconst, 448
  - NXCDefs.h, 1485
- ERR\_ARG
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1215
- ERR\_BAD\_POOL\_SIZE
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1215
- ERR\_BAD\_PTR
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1215
- ERR\_CLUMP\_COUNT
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1215
- ERR\_COMM\_BUFFER\_FULL
  - CommandCommErrors, 710
  - NBCCCommon.h, 1215
- ERR\_COMM\_BUS\_ERR
  - CommandCommErrors, 710
  - NBCCCommon.h, 1215
- ERR\_COMM\_CHAN\_INVALID
  - CommandCommErrors, 710
  - NBCCCommon.h, 1216
- ERR\_COMM\_CHAN\_NOT\_READY
  - CommandCommErrors, 710
  - NBCCCommon.h, 1216
- ERR\_DEFAULT\_OFFSETS
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1216
- ERR\_FILE
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1216
- ERR\_INSANE\_OFFSET
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1216
- ERR\_INSTR
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1216
- ERR\_INVALID\_FIELD
  - CommandGenErrors, 709
  - NBCCCommon.h, 1216
- ERR\_INVALID\_PORT
  - CommandGenErrors, 709
  - NBCCCommon.h, 1216
- ERR\_INVALID\_QUEUE
  - CommandGenErrors, 709
  - NBCCCommon.h, 1216
- ERR\_INVALID\_SIZE
  - CommandGenErrors, 709
  - NBCCCommon.h, 1216
- ERR\_LOADER\_ERR
  - CommandFatalErrors, 707
  - NBCCCommon.h, 1217
- ERR\_MEM
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1217
- ERR\_MEMMGR\_FAIL
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1217
- ERR\_NO\_ACTIVE\_CLUMP
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1217
- ERR\_NO\_CODE
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1217
- ERR\_NO\_PROG
  - CommandGenErrors, 709
  - NBCCCommon.h, 1217
- ERR\_NON\_FATAL
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1217
- ERR\_RC\_BAD\_PACKET
  - CommandRCErrors, 711
  - NBCCCommon.h, 1217
- ERR\_RC\_FAILED
  - CommandRCErrors, 711
  - NBCCCommon.h, 1217
- ERR\_RC\_ILLEGAL\_VAL

- CommandRCErrors, 711
  - NBCCCommon.h, 1217
- ERR\_RC\_UNKNOWN\_CMD
  - CommandRCErrors, 711
  - NBCCCommon.h, 1218
- ERR\_SPOTCHECK\_FAIL
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1218
- ERR\_VER
  - CommandFatalErrors, 708
  - NBCCCommon.h, 1218
- ExitTo
  - CommandModuleFunctions, 421
  - NXCDefs.h, 1563
- Exp
  - cmathAPI, 609
  - NXCDefs.h, 1485
- exp
  - cmathAPI, 623
  - NXCDefs.h, 1564
- FALSE
  - MiscConstants, 261
  - NBCCCommon.h, 1218
- Fatal errors, 706
- fclose
  - cstdioAPI, 636
  - NXCDefs.h, 1564
- feof
  - cstdioAPI, 636
  - NXCDefs.h, 1565
- fflush
  - cstdioAPI, 637
  - NXCDefs.h, 1565
- fgetc
  - cstdioAPI, 637
  - NXCDefs.h, 1565
- fgets
  - cstdioAPI, 637
  - NXCDefs.h, 1566
- FileClose
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileCloseType, 1057
  - FileHandle, 1058
  - Result, 1058
- FileDelete
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileDeleteType, 1058
  - Filename, 1059
  - Result, 1059
- FileFindFirst
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileFindNext
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileFindType, 1059
  - FileHandle, 1060
  - Filename, 1060
  - Length, 1060
  - Result, 1060
- FileHandle
  - FileCloseType, 1058
  - FileFindType, 1060
  - FileOpenType, 1061
  - FileReadWriteType, 1063
  - FileResizeType, 1065
  - FileResolveHandleType, 1066
  - FileSeekType, 1068
  - FileTellType, 1069
- FileList
  - ListFilesType, 1085
- Filename
  - DrawFontType, 1046
  - DrawGraphicType, 1049
  - FileDeleteType, 1059
  - FileFindType, 1060
  - FileOpenType, 1061
  - FileResolveHandleType, 1066
  - LoaderExecuteFunctionType, 1088
  - SoundPlayFileType, 1105
- FileOpenAppend
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileOpenRead
  - NBCCCommon.h, 1218
  - SysCallConstants, 687
- FileOpenReadLinear
  - NBCCCommon.h, 1219
  - SysCallConstants, 687

- FileOpenType, 1060
  - FileHandle, 1061
  - Filename, 1061
  - Length, 1061
  - Result, 1061
- FileOpenWrite
  - NBCCCommon.h, 1219
  - SysCallConstants, 687
- FileOpenWriteLinear
  - NBCCCommon.h, 1219
  - SysCallConstants, 687
- FileOpenWriteNonLinear
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileRead
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileReadWriteType, 1062
  - Buffer, 1062
  - FileHandle, 1063
  - Length, 1063
  - Result, 1063
- FileRename
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileRenameType, 1063
  - NewFilename, 1064
  - OldFilename, 1064
  - Result, 1064
- FileResize
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileResizeType, 1065
  - FileHandle, 1065
  - NewSize, 1065
  - Result, 1065
- FileResolveHandle
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileResolveHandleType, 1066
  - FileHandle, 1066
  - Filename, 1066
  - Result, 1067
  - WriteHandle, 1067
- FileSeek
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileSeekType, 1067
  - FileHandle, 1068
  - Length, 1068
  - Origin, 1068
  - Result, 1068
- FileTell
  - NBCCCommon.h, 1219
  - SysCallConstants, 688
- FileTellType, 1069
  - FileHandle, 1069
  - Position, 1069
  - Result, 1069
- FileWrite
  - NBCCCommon.h, 1220
  - SysCallConstants, 688
- FindFirstFile
  - LoaderModuleFunctions, 579
  - NXCDefs.h, 1566
- FindNextFile
  - LoaderModuleFunctions, 579
  - NXCDefs.h, 1567
- FirstTick
  - CommandModuleFunctions, 422
  - NXCDefs.h, 1567
- Flags
  - SoundGetStateType, 1104
  - SoundSetStateType, 1108
- Flatten
  - cstringAPI, 660
  - NXCDefs.h, 1568
- FlattenVar
  - cstringAPI, 660
  - NXCDefs.h, 1568
- Float
  - NXCDefs.h, 1569
  - OutputModuleFunctions, 319
- Floor
  - cmathAPI, 610
  - NXCDefs.h, 1486
- floor
  - cmathAPI, 624
  - NXCDefs.h, 1569
- Follows
  - CommandModuleFunctions, 422
  - NXCDefs.h, 1569

- Font drawing option constants, [821](#)
- FontNumOut
  - DisplayModuleFunctions, [357](#)
  - NXCDefs.h, [1570](#)
- FontTextOut
  - DisplayModuleFunctions, [357](#)
  - NXCDefs.h, [1571](#)
- fopen
  - cstdioAPI, [638](#)
  - NXCDefs.h, [1571](#)
- ForceOff
  - NXCDefs.h, [1572](#)
  - UiModuleFunctions, [563](#)
- FormatNum
  - cstringAPI, [661](#)
  - NXCDefs.h, [1572](#)
- fprintf
  - cstdioAPI, [638](#)
  - NXCDefs.h, [1573](#)
- fputc
  - cstdioAPI, [639](#)
  - NXCDefs.h, [1573](#)
- fputs
  - cstdioAPI, [639](#)
  - NXCDefs.h, [1574](#)
- Frac
  - cmathAPI, [610](#)
  - NXCDefs.h, [1486](#)
- frac
  - cmathAPI, [624](#)
  - NXCDefs.h, [1574](#)
- FRAME\_SELECT
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1220](#)
- FreeMemory
  - LoaderModuleFunctions, [580](#)
  - NXCDefs.h, [1575](#)
- Frequency
  - SoundPlayToneType, [1107](#)
  - Tone, [1109](#)
- FREQUENCY\_MAX
  - NBCCCommon.h, [1220](#)
  - SoundMisc, [732](#)
- FREQUENCY\_MIN
  - NBCCCommon.h, [1220](#)
  - SoundMisc, [732](#)
- fseek
  - cstdioAPI, [640](#)
  - NXCDefs.h, [1575](#)
- fseek origin constants, [644](#)
- fseekConstants
  - SEEK\_CUR, [644](#)
  - SEEK\_END, [644](#)
  - SEEK\_SET, [644](#)
- ftell
  - cstdioAPI, [640](#)
  - NXCDefs.h, [1576](#)
- General errors, [708](#)
- GenericI2CConstants
  - I2C\_ADDR\_DEFAULT, [802](#)
  - I2C\_REG\_CMD, [802](#)
  - I2C\_REG\_DEVICE\_ID, [802](#)
  - I2C\_REG\_VENDOR\_ID, [802](#)
  - I2C\_REG\_VERSION, [802](#)
- GetBrickDataAddress
  - CommModuleFunctions, [479](#)
  - NXCDefs.h, [1576](#)
- GetBTConnectionAddress
  - CommModuleFunctions, [480](#)
  - NXCDefs.h, [1576](#)
- GetBTDeviceAddress
  - CommModuleFunctions, [480](#)
  - NXCDefs.h, [1577](#)
- GetBTInputBuffer
  - CommModuleFunctions, [480](#)
  - NXCDefs.h, [1577](#)
- GetBTOutputBuffer
  - CommModuleFunctions, [481](#)
  - NXCDefs.h, [1578](#)
- GetButtonModuleValue
  - CommandModuleFunctions, [422](#)
  - NXCDefs.h, [1578](#)
- getc
  - cstdioAPI, [635](#)
  - NXCDefs.h, [1486](#)
- getchar
  - cstdioAPI, [641](#)
  - NXCDefs.h, [1578](#)
- GetCommandModuleBytes
  - CommandModuleFunctions, [423](#)
  - NXCDefs.h, [1579](#)

- GetCommandModuleValue
  - CommandModuleFunctions, 423
  - NXCDefs.h, 1579
- GetCommModuleBytes
  - CommandModuleFunctions, 423
  - NXCDefs.h, 1580
- GetCommModuleValue
  - CommandModuleFunctions, 424
  - NXCDefs.h, 1580
- GetDisplayModuleBytes
  - CommandModuleFunctions, 424
  - NXCDefs.h, 1580
- GetDisplayModuleValue
  - CommandModuleFunctions, 425
  - NXCDefs.h, 1581
- GetDisplayNormal
  - DisplayModuleFunctions, 358
  - NXCDefs.h, 1581
- GetDisplayPopup
  - DisplayModuleFunctions, 359
  - NXCDefs.h, 1582
- GetHSInputBuffer
  - CommModuleFunctions, 481
  - NXCDefs.h, 1582
- GetHSOutputBuffer
  - CommModuleFunctions, 482
  - NXCDefs.h, 1583
- GetInput
  - InputModuleFunctions, 294
  - NXCDefs.h, 1583
- GetInputModuleValue
  - CommandModuleFunctions, 425
  - NXCDefs.h, 1583
- GetIOMapBytes
  - CommandModuleFunctions, 425
  - NXCDefs.h, 1584
- GetIOMapBytesByID
  - CommandModuleFunctions, 426
  - NXCDefs.h, 1584
- GetIOMapValue
  - CommandModuleFunctions, 426
  - NXCDefs.h, 1585
- GetIOMapValueByID
  - CommandModuleFunctions, 427
  - NXCDefs.h, 1585
- GetLastResponseInfo
  - CommandModuleFunctions, 427
  - NXCDefs.h, 1586
- GetLoaderModuleValue
  - CommandModuleFunctions, 428
  - NXCDefs.h, 1586
- GetLowSpeedModuleBytes
  - CommandModuleFunctions, 428
  - NXCDefs.h, 1587
- GetLowSpeedModuleValue
  - CommandModuleFunctions, 429
  - NXCDefs.h, 1587
- GetLSInputBuffer
  - LowLevelLowSpeedModuleFunctions, 405
  - NXCDefs.h, 1587
- GetLSOutputBuffer
  - LowLevelLowSpeedModuleFunctions, 405
  - NXCDefs.h, 1588
- GetMemoryInfo
  - CommandModuleFunctions, 429
  - NXCDefs.h, 1588
- GetOutput
  - NXCDefs.h, 1589
  - OutputModuleFunctions, 319
- GetOutputModuleValue
  - CommandModuleFunctions, 429
  - NXCDefs.h, 1589
- GetSoundModuleValue
  - CommandModuleFunctions, 430
  - NXCDefs.h, 1590
- GetStartTick
  - NBCCommon.h, 1220
  - SysCallConstants, 688
- GetStartTickType, 1070
  - Result, 1070
- GetUIModuleValue
  - CommandModuleFunctions, 430
  - NXCDefs.h, 1590
- GetUSBInputBuffer
  - CommModuleFunctions, 482
  - NXCDefs.h, 1590
- GetUSBOutputBuffer
  - CommModuleFunctions, 483
  - NXCDefs.h, 1591
- GetUSBPollBuffer

- CommModuleFunctions, 483
- NXCDefs.h, 1591
- GL\_CAMERA\_DEPTH
  - GLConstantsSettings, 1016
  - NBCCCommon.h, 1220
- GL\_CIRCLE
  - GLConstantsBeginModes, 1013
  - NBCCCommon.h, 1220
- GL\_CIRCLE\_SIZE
  - GLConstantsSettings, 1016
  - NBCCCommon.h, 1220
- GL\_CULL\_BACK
  - GLConstantsCullMode, 1017
  - NBCCCommon.h, 1220
- GL\_CULL\_FRONT
  - GLConstantsCullMode, 1017
  - NBCCCommon.h, 1221
- GL\_CULL\_MODE
  - GLConstantsSettings, 1016
  - NBCCCommon.h, 1221
- GL\_CULL\_NONE
  - GLConstantsCullMode, 1017
  - NBCCCommon.h, 1221
- GL\_LINE
  - GLConstantsBeginModes, 1013
  - NBCCCommon.h, 1221
- GL\_POINT
  - GLConstantsBeginModes, 1013
  - NBCCCommon.h, 1221
- GL\_POLYGON
  - GLConstantsBeginModes, 1013
  - NBCCCommon.h, 1221
- GL\_ROTATE\_X
  - GLConstantsActions, 1014
  - NBCCCommon.h, 1221
- GL\_ROTATE\_Y
  - GLConstantsActions, 1014
  - NBCCCommon.h, 1222
- GL\_ROTATE\_Z
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_SCALE\_X
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_SCALE\_Y
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_SCALE\_Z
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_TRANSLATE\_X
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_TRANSLATE\_Y
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1222
- GL\_TRANSLATE\_Z
  - GLConstantsActions, 1015
  - NBCCCommon.h, 1223
- GL\_ZOOM\_FACTOR
  - GLConstantsSettings, 1016
  - NBCCCommon.h, 1223
- glAddToAngleX
  - GraphicsLibrary, 266
  - NXCDefs.h, 1592
- glAddToAngleY
  - GraphicsLibrary, 266
  - NXCDefs.h, 1592
- glAddToAngleZ
  - GraphicsLibrary, 266
  - NXCDefs.h, 1592
- glAddVertex
  - GraphicsLibrary, 267
  - NXCDefs.h, 1593
- glBegin
  - GraphicsLibrary, 267
  - NXCDefs.h, 1593
- glBeginObject
  - GraphicsLibrary, 267
  - NXCDefs.h, 1593
- glBeginRender
  - GraphicsLibrary, 267
  - NXCDefs.h, 1593
- glBox
  - GraphicsLibrary, 268
  - NXCDefs.h, 1594
- glCallObject
  - GraphicsLibrary, 268
  - NXCDefs.h, 1594
- GLConstantsActions
  - GL\_ROTATE\_X, 1014
  - GL\_ROTATE\_Y, 1014

- GL\_ROTATE\_Z, [1015](#)
- GL\_SCALE\_X, [1015](#)
- GL\_SCALE\_Y, [1015](#)
- GL\_SCALE\_Z, [1015](#)
- GL\_TRANSLATE\_X, [1015](#)
- GL\_TRANSLATE\_Y, [1015](#)
- GL\_TRANSLATE\_Z, [1015](#)
- GLConstantsBeginModes
  - GL\_CIRCLE, [1013](#)
  - GL\_LINE, [1013](#)
  - GL\_POINT, [1013](#)
  - GL\_POLYGON, [1013](#)
- GLConstantsCullMode
  - GL\_CULL\_BACK, [1017](#)
  - GL\_CULL\_FRONT, [1017](#)
  - GL\_CULL\_NONE, [1017](#)
- GLConstantsSettings
  - GL\_CAMERA\_DEPTH, [1016](#)
  - GL\_CIRCLE\_SIZE, [1016](#)
  - GL\_CULL\_MODE, [1016](#)
  - GL\_ZOOM\_FACTOR, [1016](#)
- glCos32768
  - GraphicsLibrary, [268](#)
  - NXCDefs.h, [1594](#)
- glCube
  - GraphicsLibrary, [269](#)
  - NXCDefs.h, [1595](#)
- glEnd
  - GraphicsLibrary, [269](#)
  - NXCDefs.h, [1595](#)
- glEndObject
  - GraphicsLibrary, [269](#)
  - NXCDefs.h, [1595](#)
- glFinishRender
  - GraphicsLibrary, [269](#)
  - NXCDefs.h, [1595](#)
- glInit
  - GraphicsLibrary, [270](#)
  - NXCDefs.h, [1596](#)
- glObjectAction
  - GraphicsLibrary, [270](#)
  - NXCDefs.h, [1596](#)
- glPyramid
  - GraphicsLibrary, [270](#)
  - NXCDefs.h, [1596](#)
- glSet
  - GraphicsLibrary, [271](#)
  - NXCDefs.h, [1597](#)
- glSetAngleX
  - GraphicsLibrary, [271](#)
  - NXCDefs.h, [1597](#)
- glSetAngleY
  - GraphicsLibrary, [271](#)
  - NXCDefs.h, [1597](#)
- glSetAngleZ
  - GraphicsLibrary, [272](#)
  - NXCDefs.h, [1598](#)
- glSin32768
  - GraphicsLibrary, [272](#)
  - NXCDefs.h, [1598](#)
- GraphicArrayOut
  - DisplayModuleFunctions, [359](#)
  - NXCDefs.h, [1598](#)
- GraphicArrayOutEx
  - DisplayModuleFunctions, [360](#)
  - NXCDefs.h, [1599](#)
- GraphicOut
  - DisplayModuleFunctions, [361](#)
  - NXCDefs.h, [1600](#)
- GraphicOutEx
  - DisplayModuleFunctions, [361](#)
  - NXCDefs.h, [1600](#)
- Graphics library actions, [1014](#)
- Graphics library begin modes, [1013](#)
- Graphics library cull mode, [1017](#)
- Graphics library settings, [1016](#)
- GraphicsLibrary
  - glAddToAngleX, [266](#)
  - glAddToAngleY, [266](#)
  - glAddToAngleZ, [266](#)
  - glAddVertex, [267](#)
  - glBegin, [267](#)
  - glBeginObject, [267](#)
  - glBeginRender, [267](#)
  - glBox, [268](#)
  - glCallObject, [268](#)
  - glCos32768, [268](#)
  - glCube, [269](#)
  - glEnd, [269](#)
  - glEndObject, [269](#)
  - glFinishRender, [269](#)
  - glInit, [270](#)

- glObjectAction, [270](#)
- glPyramid, [270](#)
- glSet, [271](#)
- glSetAngleX, [271](#)
- glSetAngleY, [271](#)
- glSetAngleZ, [272](#)
- glSin32768, [272](#)
- GT
  - cmpconst, [448](#)
  - NXCDefs.h, [1487](#)
- GTEQ
  - cmpconst, [449](#)
  - NXCDefs.h, [1487](#)
- Height
  - SizeType, [1103](#)
- Hi-speed port address constants, [850](#)
- Hi-speed port baud rate constants, [843](#)
- Hi-speed port combined UART constants, [849](#)
- Hi-speed port constants, [839](#)
- Hi-speed port data bits constants, [846](#)
- Hi-speed port flags constants, [840](#)
- Hi-speed port parity constants, [848](#)
- Hi-speed port state constants, [841](#)
- Hi-speed port stop bits constants, [847](#)
- Hi-speed port SysCommHSCControl constants, [842](#)
- Hi-speed port UART mode constants, [845](#)
- HiTechnic Angle sensor constants, [920](#)
- HiTechnic API Functions, [59](#)
- HiTechnic Barometric sensor constants, [921](#)
- HiTechnic Color2 constants, [918](#)
- HiTechnic device constants, [911](#)
- HiTechnic IRReceiver constants, [916](#)
- HiTechnic IRSeeker2 constants, [914](#)
- HiTechnic Prototype board analog input constants, [924](#)
- HiTechnic Prototype board constants, [922](#)
- HiTechnic SuperPro analog input index constants, [932](#)
- HiTechnic SuperPro analog output index constants, [933](#)
- HiTechnic SuperPro constants, [925](#)
- HiTechnic/mindsensors Power Function/IR Train constants, [899](#)
- HiTechnicAPI
  - HTIRTrain, [69](#)
  - HTPFComboDirect, [69](#)
  - HTPFComboPWM, [70](#)
  - HTPFRawOutput, [71](#)
  - HTPFRepeat, [71](#)
  - HTPFSingleOutputCST, [72](#)
  - HTPFSingleOutputPWM, [72](#)
  - HTPFSinglePin, [73](#)
  - HTPFTrain, [74](#)
  - HTRCXAddToDatalog, [74](#)
  - HTRCXBatteryLevel, [75](#)
  - HTRCXCLEARAllEvents, [75](#)
  - HTRCXCLEARCounter, [75](#)
  - HTRCXCLEARMsg, [76](#)
  - HTRCXCLEARSensor, [76](#)
  - HTRCXCLEARSound, [76](#)
  - HTRCXCLEARTimer, [76](#)
  - HTRCXCreateDatalog, [77](#)
  - HTRCXDecCounter, [77](#)
  - HTRCXDeleteSub, [77](#)
  - HTRCXDeleteSubs, [78](#)
  - HTRCXDeleteTask, [78](#)
  - HTRCXDeleteTasks, [78](#)
  - HTRCXDisableOutput, [78](#)
  - HTRCXEnableOutput, [79](#)
  - HTRCXEvent, [79](#)
  - HTRCXFloat, [79](#)
  - HTRCXFwd, [80](#)
  - HTRCXIncCounter, [80](#)
  - HTRCXInvertOutput, [80](#)
  - HTRCXMuteSound, [81](#)
  - HTRCXObvertOutput, [81](#)
  - HTRCXOff, [81](#)
  - HTRCXOn, [82](#)
  - HTRCXOnFor, [82](#)
  - HTRCXOnFwd, [82](#)
  - HTRCXOnRev, [83](#)
  - HTRCXPBTurnOff, [83](#)
  - HTRCXPing, [83](#)
  - HTRCXPlaySound, [83](#)
  - HTRCXPlayTone, [84](#)
  - HTRCXPlayToneVar, [84](#)
  - HTRCXPoll, [84](#)

- HTRCXPollMemory, 85
- HTRCXRemote, 85
- HTRCXRev, 85
- HTRCXSelectDisplay, 86
- HTRCXSelectProgram, 86
- HTRCXSendSerial, 86
- HTRCXSetDirection, 87
- HTRCXSetEvent, 87
- HTRCXSetGlobalDirection, 88
- HTRCXSetGlobalOutput, 88
- HTRCXSetIRLinkPort, 88
- HTRCXSetMaxPower, 89
- HTRCXSetMessage, 89
- HTRCXSetOutput, 89
- HTRCXSetPower, 90
- HTRCXSetPriority, 90
- HTRCXSetSensorMode, 90
- HTRCXSetSensorType, 91
- HTRCXSetSleepTime, 91
- HTRCXSetTxPower, 91
- HTRCXSetWatch, 92
- HTRCXStartTask, 92
- HTRCXStopAllTasks, 92
- HTRCXStopTask, 92
- HTRCXToggle, 93
- HTRCXUnmuteSound, 93
- HTScoutCalibrateSensor, 93
- HTScoutMuteSound, 93
- HTScoutSelectSounds, 94
- HTScoutSendVLL, 94
- HTScoutSetEventFeedback, 94
- HTScoutSetLight, 95
- HTScoutSetScoutMode, 95
- HTScoutSetSensorClickTime, 95
- HTScoutSetSensorHysteresis, 96
- HTScoutSetSensorLowerLimit, 96
- HTScoutSetSensorUpperLimit, 96
- HTScoutUnmuteSound, 97
- ReadSensorHTAccel, 97
- ReadSensorHTAngle, 98
- ReadSensorHTBarometric, 98
- ReadSensorHTColor, 99
- ReadSensorHTColor2Active, 99
- ReadSensorHTIRReceiver, 100
- ReadSensorHTIRReceiverEx, 100
- ReadSensorHTIRSeeker, 101
- ReadSensorHTIRSeeker2AC, 101
- ReadSensorHTIRSeeker2DC, 102
- ReadSensorHTNormalizedColor, 103
- ReadSensorHTNormalizedColor2Active, 103
- ReadSensorHTProtoAllAnalog, 104
- ReadSensorHTRawColor, 105
- ReadSensorHTRawColor2, 105
- ReadSensorHTSuperProAllAnalog, 106
- ReadSensorHTSuperProAnalogOut, 106
- ReadSensorHTTouchMultiplexer, 107
- ResetHTBarometricCalibration, 107
- ResetSensorHTAngle, 108
- SensorHTColorNum, 108
- SensorHTCompass, 109
- SensorHTEOPD, 109
- SensorHTGyro, 109
- SensorHTIRSeeker2ACDir, 110
- SensorHTIRSeeker2Addr, 110
- SensorHTIRSeeker2DCDir, 111
- SensorHTIRSeekerDir, 111
- SensorHTMagnet, 112
- SensorHTProtoAnalog, 112
- SensorHTProtoDigital, 113
- SensorHTProtoDigitalControl, 113
- SensorHTSuperProAnalog, 113
- SensorHTSuperProDigital, 114
- SensorHTSuperProDigitalControl, 114
- SensorHTSuperProLED, 115
- SensorHTSuperProProgramControl, 115
- SensorHTSuperProStrobe, 116
- SetHTBarometricCalibration, 116
- SetHTColor2Mode, 116
- SetHTIRSeeker2Mode, 117
- SetSensorHTEOPD, 117
- SetSensorHTGyro, 118
- SetSensorHTMagnet, 118
- SetSensorHTProtoDigital, 118
- SetSensorHTProtoDigitalControl, 119

- SetSensorHTSuperProAnalogOut, 119
- SetSensorHTSuperProDigital, 120
- SetSensorHTSuperProDigitalControl, 120
- SetSensorHTSuperProLED, 121
- SetSensorHTSuperProProgramControl, 121
- SetSensorHTSuperProStrobe, 121
- HiTechnicConstants
  - HT\_ADDR\_ACCEL, 913
  - HT\_ADDR\_ANGLE, 913
  - HT\_ADDR\_BAROMETRIC, 913
  - HT\_ADDR\_COLOR, 913
  - HT\_ADDR\_COLOR2, 913
  - HT\_ADDR\_COMPASS, 913
  - HT\_ADDR\_IRLINK, 913
  - HT\_ADDR\_IRRECEIVER, 913
  - HT\_ADDR\_IRSEEKER, 913
  - HT\_ADDR\_IRSEEKER2, 913
  - HT\_ADDR\_PROTOBOARD, 914
  - HT\_ADDR\_SUPERPRO, 914
- HS\_ADDRESS\_1
  - CommHiSpeedAddressConstants, 850
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_2
  - CommHiSpeedAddressConstants, 850
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_3
  - CommHiSpeedAddressConstants, 850
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_4
  - CommHiSpeedAddressConstants, 850
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_5
  - CommHiSpeedAddressConstants, 851
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_6
  - CommHiSpeedAddressConstants, 851
  - NBCCCommon.h, 1223
- HS\_ADDRESS\_7
  - CommHiSpeedAddressConstants, 851
  - NBCCCommon.h, 1224
- HS\_ADDRESS\_8
  - CommHiSpeedAddressConstants, 851
  - NBCCCommon.h, 1224
- HS\_ADDRESS\_ALL
  - CommHiSpeedAddressConstants, 851
  - NBCCCommon.h, 1224
- HS\_BAUD\_115200
  - CommHiSpeedBaudConstants, 843
  - NBCCCommon.h, 1224
- HS\_BAUD\_1200
  - CommHiSpeedBaudConstants, 843
  - NBCCCommon.h, 1224
- HS\_BAUD\_14400
  - CommHiSpeedBaudConstants, 843
  - NBCCCommon.h, 1224
- HS\_BAUD\_19200
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1224
- HS\_BAUD\_230400
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1224
- HS\_BAUD\_2400
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1224
- HS\_BAUD\_28800
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1224
- HS\_BAUD\_3600
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1225
- HS\_BAUD\_38400
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1225
- HS\_BAUD\_460800
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1225
- HS\_BAUD\_4800
  - CommHiSpeedBaudConstants, 844
  - NBCCCommon.h, 1225
- HS\_BAUD\_57600

- CommHiSpeedBaudConstants, [844](#)
- NBCCCommon.h, [1225](#)
- HS\_BAUD\_7200
  - CommHiSpeedBaudConstants, [844](#)
  - NBCCCommon.h, [1225](#)
- HS\_BAUD\_76800
  - CommHiSpeedBaudConstants, [845](#)
  - NBCCCommon.h, [1225](#)
- HS\_BAUD\_921600
  - CommHiSpeedBaudConstants, [845](#)
  - NBCCCommon.h, [1225](#)
- HS\_BAUD\_9600
  - CommHiSpeedBaudConstants, [845](#)
  - NBCCCommon.h, [1225](#)
- HS\_BAUD\_DEFAULT
  - CommHiSpeedBaudConstants, [845](#)
  - NBCCCommon.h, [1225](#)
- HS\_BYTES\_REMAINING
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1226](#)
- HS\_CMD\_READY
  - CommStatusCodesConstants, [855](#)
  - NBCCCommon.h, [1226](#)
- HS\_CTRL\_EXIT
  - CommHiSpeedCtrlConstants, [842](#)
  - NBCCCommon.h, [1226](#)
- HS\_CTRL\_INIT
  - CommHiSpeedCtrlConstants, [842](#)
  - NBCCCommon.h, [1226](#)
- HS\_CTRL\_UART
  - CommHiSpeedCtrlConstants, [842](#)
  - NBCCCommon.h, [1226](#)
- HS\_DEFAULT
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1226](#)
- HS\_DISABLE
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1226](#)
- HS\_ENABLE
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1226](#)
- HS\_INIT\_RECEIVER
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1227](#)
- HS\_INITIALISE
  - CommHiSpeedStateConstants, [841](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_10\_STOP
  - CommHiSpeedStopBitsConstants, [848](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_15\_STOP
  - CommHiSpeedStopBitsConstants, [848](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_20\_STOP
  - CommHiSpeedStopBitsConstants, [848](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_5\_DATA
  - CommHiSpeedDataBitsConstants, [847](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_6\_DATA
  - CommHiSpeedDataBitsConstants, [847](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_7\_DATA
  - CommHiSpeedDataBitsConstants, [847](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_7E1
  - CommHiSpeedCombinedConstants, [849](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_8\_DATA
  - CommHiSpeedDataBitsConstants, [847](#)
  - NBCCCommon.h, [1227](#)
- HS\_MODE\_8N1
  - CommHiSpeedCombinedConstants, [849](#)
  - NBCCCommon.h, [1228](#)
- HS\_MODE\_DEFAULT
  - CommHiSpeedModeConstants, [846](#)
  - NBCCCommon.h, [1228](#)
- HS\_MODE\_E\_PARITY
  - CommHiSpeedParityConstants, [848](#)
  - NBCCCommon.h, [1228](#)
- HS\_MODE\_M\_PARITY
  - CommHiSpeedParityConstants, [848](#)
  - NBCCCommon.h, [1228](#)

- HS\_MODE\_MASK
  - CommHiSpeedModeConstants, 846
  - NBCCCommon.h, 1228
- HS\_MODE\_N\_PARITY
  - CommHiSpeedParityConstants, 849
  - NBCCCommon.h, 1228
- HS\_MODE\_O\_PARITY
  - CommHiSpeedParityConstants, 849
  - NBCCCommon.h, 1228
- HS\_MODE\_S\_PARITY
  - CommHiSpeedParityConstants, 849
  - NBCCCommon.h, 1228
- HS\_MODE\_UART\_RS232
  - CommHiSpeedModeConstants, 846
  - NBCCCommon.h, 1229
- HS\_MODE\_UART\_RS485
  - CommHiSpeedModeConstants, 846
  - NBCCCommon.h, 1229
- HS\_SEND\_DATA
  - CommHiSpeedStateConstants, 842
  - NBCCCommon.h, 1229
- HS\_UART\_MASK
  - CommHiSpeedModeConstants, 846
  - NBCCCommon.h, 1229
- HS\_UPDATE
  - CommHiSpeedFlagsConstants, 840
  - NBCCCommon.h, 1229
- HSAddress
  - CommModuleFunctions, 483
  - NXCDefs.h, 1601
- HSDataMode
  - CommModuleFunctions, 484
  - NXCDefs.h, 1601
- HSFlags
  - CommModuleFunctions, 484
  - NXCDefs.h, 1601
- HSInputBufferInPtr
  - CommModuleFunctions, 484
  - NXCDefs.h, 1602
- HSInputBufferOutPtr
  - CommModuleFunctions, 485
  - NXCDefs.h, 1602
- HSMode
  - CommModuleFunctions, 485
  - NXCDefs.h, 1602
- HSOutputBufferInPtr
  - CommModuleFunctions, 485
  - NXCDefs.h, 1603
- HSOutputBufferOutPtr
  - CommModuleFunctions, 486
  - NXCDefs.h, 1603
- HSSpeed
  - CommModuleFunctions, 486
  - NXCDefs.h, 1603
- HSSState
  - CommModuleFunctions, 486
  - NXCDefs.h, 1604
- HT\_ADDR\_ACCEL
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1229
- HT\_ADDR\_ANGLE
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1229
- HT\_ADDR\_BAROMETRIC
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1229
- HT\_ADDR\_COLOR
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1229
- HT\_ADDR\_COLOR2
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1229
- HT\_ADDR\_COMPASS
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1230
- HT\_ADDR\_IRLINK
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1230
- HT\_ADDR\_IRRECEIVER
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1230
- HT\_ADDR\_IRSEEKER
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1230
- HT\_ADDR\_IRSEEKER2
  - HiTechnicConstants, 913
  - NBCCCommon.h, 1230
- HT\_ADDR\_PROTOBOARD
  - HiTechnicConstants, 914
  - NBCCCommon.h, 1230
- HT\_ADDR\_SUPERPRO
  - HiTechnicConstants, 914

- NBCCCommon.h, [1230](#)
- HT\_CH1\_A
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1230](#)
- HT\_CH1\_B
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1230](#)
- HT\_CH2\_A
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1231](#)
- HT\_CH2\_B
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1231](#)
- HT\_CH3\_A
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1231](#)
- HT\_CH3\_B
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1231](#)
- HT\_CH4\_A
  - HTIRReceiverConstants, [917](#)
  - NBCCCommon.h, [1231](#)
- HT\_CH4\_B
  - HTIRReceiverConstants, [918](#)
  - NBCCCommon.h, [1231](#)
- HT\_CMD\_COLOR2\_50HZ
  - HTColor2Constants, [918](#)
  - NBCCCommon.h, [1231](#)
- HT\_CMD\_COLOR2\_60HZ
  - HTColor2Constants, [918](#)
  - NBCCCommon.h, [1231](#)
- HT\_CMD\_COLOR2\_ACTIVE
  - HTColor2Constants, [918](#)
  - NBCCCommon.h, [1231](#)
- HT\_CMD\_COLOR2\_BLCAL
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_FAR
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_LED\_HI
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_LED\_LOW
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_NEAR
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_PASSIVE
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_RAW
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HT\_CMD\_COLOR2\_WBCAL
  - HTColor2Constants, [919](#)
  - NBCCCommon.h, [1232](#)
- HTANGLE\_MODE\_CALIBRATE
  - HTAngleConstants, [920](#)
  - NBCCCommon.h, [1232](#)
- HTANGLE\_MODE\_NORMAL
  - HTAngleConstants, [920](#)
  - NBCCCommon.h, [1232](#)
- HTANGLE\_MODE\_RESET
  - HTAngleConstants, [920](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_ACDIR
  - HTAngleConstants, [920](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DC01
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DC02
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DC03
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DC04
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DC05
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DCAVG
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_DCDIR
  - HTAngleConstants, [921](#)
  - NBCCCommon.h, [1233](#)
- HTANGLE\_REG\_MODE

- HTAngleConstants, 921
- NBCCCommon.h, 1234
- HTAngleConstants
  - HTANGLE\_MODE\_CALIBRATE, 920
  - HTANGLE\_MODE\_NORMAL, 920
  - HTANGLE\_MODE\_RESET, 920
  - HTANGLE\_REG\_ACDIR, 920
  - HTANGLE\_REG\_DC01, 921
  - HTANGLE\_REG\_DC02, 921
  - HTANGLE\_REG\_DC03, 921
  - HTANGLE\_REG\_DC04, 921
  - HTANGLE\_REG\_DC05, 921
  - HTANGLE\_REG\_DCAVG, 921
  - HTANGLE\_REG\_DCDIR, 921
  - HTANGLE\_REG\_MODE, 921
- HTBAR\_REG\_CALIBRATION
  - HTBarometricConstants, 922
  - NBCCCommon.h, 1234
- HTBAR\_REG\_COMMAND
  - HTBarometricConstants, 922
  - NBCCCommon.h, 1234
- HTBAR\_REG\_PRESSURE
  - HTBarometricConstants, 922
  - NBCCCommon.h, 1234
- HTBAR\_REG\_TEMPERATURE
  - HTBarometricConstants, 922
  - NBCCCommon.h, 1234
- HTBarometricConstants
  - HTBAR\_REG\_CALIBRATION, 922
  - HTBAR\_REG\_COMMAND, 922
  - HTBAR\_REG\_PRESSURE, 922
  - HTBAR\_REG\_TEMPERATURE, 922
- HTColor2Constants
  - HT\_CMD\_COLOR2\_50HZ, 918
  - HT\_CMD\_COLOR2\_60HZ, 918
  - HT\_CMD\_COLOR2\_ACTIVE, 918
  - HT\_CMD\_COLOR2\_BLCAL, 919
  - HT\_CMD\_COLOR2\_FAR, 919
  - HT\_CMD\_COLOR2\_LED\_HI, 919
  - HT\_CMD\_COLOR2\_LED\_LOW, 919
  - HT\_CMD\_COLOR2\_NEAR, 919
  - HT\_CMD\_COLOR2\_PASSIVE, 919
  - HT\_CMD\_COLOR2\_RAW, 919
  - HT\_CMD\_COLOR2\_WBCAL, 919
- HTIR2\_MODE\_1200
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1234
- HTIR2\_MODE\_600
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1234
- HTIR2\_REG\_AC01
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1234
- HTIR2\_REG\_AC02
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1234
- HTIR2\_REG\_AC03
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_AC04
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_AC05
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_ACDIR
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DC01
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DC02
  - HTIRSeeker2Constants, 915
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DC03
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DC04
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DC05
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1235
- HTIR2\_REG\_DCAVG
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1235

- HTIR2\_REG\_DCDIR
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1236
- HTIR2\_REG\_MODE
  - HTIRSeeker2Constants, 916
  - NBCCCommon.h, 1236
- HTIRReceiverConstants
  - HT\_CH1\_A, 917
  - HT\_CH1\_B, 917
  - HT\_CH2\_A, 917
  - HT\_CH2\_B, 917
  - HT\_CH3\_A, 917
  - HT\_CH3\_B, 917
  - HT\_CH4\_A, 917
  - HT\_CH4\_B, 918
- HTIRSeeker2Constants
  - HTIR2\_MODE\_1200, 915
  - HTIR2\_MODE\_600, 915
  - HTIR2\_REG\_AC01, 915
  - HTIR2\_REG\_AC02, 915
  - HTIR2\_REG\_AC03, 915
  - HTIR2\_REG\_AC04, 915
  - HTIR2\_REG\_AC05, 915
  - HTIR2\_REG\_ACDIR, 915
  - HTIR2\_REG\_DC01, 915
  - HTIR2\_REG\_DC02, 915
  - HTIR2\_REG\_DC03, 916
  - HTIR2\_REG\_DC04, 916
  - HTIR2\_REG\_DC05, 916
  - HTIR2\_REG\_DCAVG, 916
  - HTIR2\_REG\_DCDIR, 916
  - HTIR2\_REG\_MODE, 916
- HTIRTrain
  - HiTechnicAPI, 69
  - NXCDefs.h, 1604
- HTPFComboDirect
  - HiTechnicAPI, 69
  - NXCDefs.h, 1605
- HTPFComboPWM
  - HiTechnicAPI, 70
  - NXCDefs.h, 1605
- HTPFRawOutput
  - HiTechnicAPI, 71
  - NXCDefs.h, 1606
- HTPFRepeat
  - HiTechnicAPI, 71
  - NXCDefs.h, 1606
- HTPFSingleOutputCST
  - HiTechnicAPI, 72
  - NXCDefs.h, 1607
- HTPFSingleOutputPWM
  - HiTechnicAPI, 72
  - NXCDefs.h, 1608
- HTPFSinglePin
  - HiTechnicAPI, 73
  - NXCDefs.h, 1608
- HTPFTrain
  - HiTechnicAPI, 74
  - NXCDefs.h, 1609
- HTPROTO\_A0
  - HTProtoAnalogInputConstants, 924
  - NBCCCommon.h, 1236
- HTPROTO\_A1
  - HTProtoAnalogInputConstants, 924
  - NBCCCommon.h, 1236
- HTPROTO\_A2
  - HTProtoAnalogInputConstants, 925
  - NBCCCommon.h, 1236
- HTPROTO\_A3
  - HTProtoAnalogInputConstants, 925
  - NBCCCommon.h, 1236
- HTPROTO\_A4
  - HTProtoAnalogInputConstants, 925
  - NBCCCommon.h, 1236
- HTPROTO\_REG\_A0
  - HTProtoConstants, 923
  - NBCCCommon.h, 1236
- HTPROTO\_REG\_A1
  - HTProtoConstants, 923
  - NBCCCommon.h, 1237
- HTPROTO\_REG\_A2
  - HTProtoConstants, 923
  - NBCCCommon.h, 1237
- HTPROTO\_REG\_A3
  - HTProtoConstants, 923
  - NBCCCommon.h, 1237
- HTPROTO\_REG\_A4
  - HTProtoConstants, 923
  - NBCCCommon.h, 1237
- HTPROTO\_REG\_DCTRL
  - HTProtoConstants, 923
  - NBCCCommon.h, 1237

- HTPROTO\_REG\_DIN
  - HTProtoConstants, [923](#)
  - NBCCCommon.h, [1237](#)
- HTPROTO\_REG\_DOUT
  - HTProtoConstants, [924](#)
  - NBCCCommon.h, [1237](#)
- HTPROTO\_REG\_SRATE
  - HTProtoConstants, [924](#)
  - NBCCCommon.h, [1237](#)
- HTProtoAnalogInputConstants
  - HTPROTO\_A0, [924](#)
  - HTPROTO\_A1, [924](#)
  - HTPROTO\_A2, [925](#)
  - HTPROTO\_A3, [925](#)
  - HTPROTO\_A4, [925](#)
- HTProtoConstants
  - HTPROTO\_REG\_A0, [923](#)
  - HTPROTO\_REG\_A1, [923](#)
  - HTPROTO\_REG\_A2, [923](#)
  - HTPROTO\_REG\_A3, [923](#)
  - HTPROTO\_REG\_A4, [923](#)
  - HTPROTO\_REG\_DCTRL, [923](#)
  - HTPROTO\_REG\_DIN, [923](#)
  - HTPROTO\_REG\_DOUT, [924](#)
  - HTPROTO\_REG\_SRATE, [924](#)
- HTRCXAddToDatalog
  - HiTechnicAPI, [74](#)
  - NXCDefs.h, [1610](#)
- HTRCXBatteryLevel
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1610](#)
- HTRCXClearAllEvents
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1610](#)
- HTRCXClearCounter
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1610](#)
- HTRCXClearMsg
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1611](#)
- HTRCXClearSensor
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1611](#)
- HTRCXClearSound
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1611](#)
- HTRCXClearTimer
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1611](#)
- HTRCXCreateDatalog
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1612](#)
- HTRCXDecCounter
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1612](#)
- HTRCXDeleteSub
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1612](#)
- HTRCXDeleteSubs
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1613](#)
- HTRCXDeleteTask
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1613](#)
- HTRCXDeleteTasks
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1613](#)
- HTRCXDisableOutput
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1613](#)
- HTRCXEnableOutput
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1614](#)
- HTRCXEvent
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1614](#)
- HTRCXFloat
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1614](#)
- HTRCXFwd
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1615](#)
- HTRCXIncCounter
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1615](#)
- HTRCXInvertOutput
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1615](#)
- HTRCXMuteSound
  - HiTechnicAPI, [81](#)
  - NXCDefs.h, [1616](#)
- HTRCXObvertOutput

- HiTechnicAPI, 81
- NXCDefs.h, 1616
- HTRCXOff
  - HiTechnicAPI, 81
  - NXCDefs.h, 1616
- HTRCXOn
  - HiTechnicAPI, 82
  - NXCDefs.h, 1617
- HTRCXOnFor
  - HiTechnicAPI, 82
  - NXCDefs.h, 1617
- HTRCXOnFwd
  - HiTechnicAPI, 82
  - NXCDefs.h, 1617
- HTRCXOnRev
  - HiTechnicAPI, 83
  - NXCDefs.h, 1618
- HTRCXPBTurnOff
  - HiTechnicAPI, 83
  - NXCDefs.h, 1618
- HTRCXPing
  - HiTechnicAPI, 83
  - NXCDefs.h, 1618
- HTRCXPlaySound
  - HiTechnicAPI, 83
  - NXCDefs.h, 1618
- HTRCXPlayTone
  - HiTechnicAPI, 84
  - NXCDefs.h, 1619
- HTRCXPlayToneVar
  - HiTechnicAPI, 84
  - NXCDefs.h, 1619
- HTRCXPoll
  - HiTechnicAPI, 84
  - NXCDefs.h, 1619
- HTRCXPollMemory
  - HiTechnicAPI, 85
  - NXCDefs.h, 1620
- HTRCXRemote
  - HiTechnicAPI, 85
  - NXCDefs.h, 1620
- HTRCXRev
  - HiTechnicAPI, 85
  - NXCDefs.h, 1620
- HTRCXSelectDisplay
  - HiTechnicAPI, 86
  - NXCDefs.h, 1621
- HTRCXSelectProgram
  - HiTechnicAPI, 86
  - NXCDefs.h, 1621
- HTRCXSendSerial
  - HiTechnicAPI, 86
  - NXCDefs.h, 1621
- HTRCXSetDirection
  - HiTechnicAPI, 87
  - NXCDefs.h, 1622
- HTRCXSetEvent
  - HiTechnicAPI, 87
  - NXCDefs.h, 1622
- HTRCXSetGlobalDirection
  - HiTechnicAPI, 88
  - NXCDefs.h, 1623
- HTRCXSetGlobalOutput
  - HiTechnicAPI, 88
  - NXCDefs.h, 1623
- HTRCXSetIRLinkPort
  - HiTechnicAPI, 88
  - NXCDefs.h, 1623
- HTRCXSetMaxPower
  - HiTechnicAPI, 89
  - NXCDefs.h, 1624
- HTRCXSetMessage
  - HiTechnicAPI, 89
  - NXCDefs.h, 1624
- HTRCXSetOutput
  - HiTechnicAPI, 89
  - NXCDefs.h, 1624
- HTRCXSetPower
  - HiTechnicAPI, 90
  - NXCDefs.h, 1625
- HTRCXSetPriority
  - HiTechnicAPI, 90
  - NXCDefs.h, 1625
- HTRCXSetSensorMode
  - HiTechnicAPI, 90
  - NXCDefs.h, 1625
- HTRCXSetSensorType
  - HiTechnicAPI, 91
  - NXCDefs.h, 1626
- HTRCXSetSleepTime
  - HiTechnicAPI, 91
  - NXCDefs.h, 1626

- HTRCXSetTxPower
  - HiTechnicAPI, 91
  - NXCDefs.h, 1626
- HTRCXSetWatch
  - HiTechnicAPI, 92
  - NXCDefs.h, 1627
- HTRCXStartTask
  - HiTechnicAPI, 92
  - NXCDefs.h, 1627
- HTRCXStopAllTasks
  - HiTechnicAPI, 92
  - NXCDefs.h, 1627
- HTRCXStopTask
  - HiTechnicAPI, 92
  - NXCDefs.h, 1627
- HTRCXToggle
  - HiTechnicAPI, 93
  - NXCDefs.h, 1628
- HTRCXUnmuteSound
  - HiTechnicAPI, 93
  - NXCDefs.h, 1628
- HTScoutCalibrateSensor
  - HiTechnicAPI, 93
  - NXCDefs.h, 1628
- HTScoutMuteSound
  - HiTechnicAPI, 93
  - NXCDefs.h, 1628
- HTScoutSelectSounds
  - HiTechnicAPI, 94
  - NXCDefs.h, 1629
- HTScoutSendVLL
  - HiTechnicAPI, 94
  - NXCDefs.h, 1629
- HTScoutSetEventFeedback
  - HiTechnicAPI, 94
  - NXCDefs.h, 1629
- HTScoutSetLight
  - HiTechnicAPI, 95
  - NXCDefs.h, 1630
- HTScoutSetScoutMode
  - HiTechnicAPI, 95
  - NXCDefs.h, 1630
- HTScoutSetSensorClickTime
  - HiTechnicAPI, 95
  - NXCDefs.h, 1630
- HTScoutSetSensorHysteresis
  - HiTechnicAPI, 96
  - NXCDefs.h, 1631
- HTScoutSetSensorLowerLimit
  - HiTechnicAPI, 96
  - NXCDefs.h, 1631
- HTScoutSetSensorUpperLimit
  - HiTechnicAPI, 96
  - NXCDefs.h, 1631
- HTScoutUnmuteSound
  - HiTechnicAPI, 97
  - NXCDefs.h, 1632
- HTSPRO\_A0
  - HTSProAnalogInputConstants, 933
  - NBCCommon.h, 1237
- HTSPRO\_A1
  - HTSProAnalogInputConstants, 933
  - NBCCommon.h, 1238
- HTSPRO\_A2
  - HTSProAnalogInputConstants, 933
  - NBCCommon.h, 1238
- HTSPRO\_A3
  - HTSProAnalogInputConstants, 933
  - NBCCommon.h, 1238
- HTSPRO\_DAC0
  - HTSProDACIndexConstants, 933
  - NBCCommon.h, 1238
- HTSPRO\_DAC1
  - HTSProDACIndexConstants, 934
  - NBCCommon.h, 1238
- HTSPRO\_REG\_A0
  - HTSuperProConstants, 927
  - NBCCommon.h, 1238
- HTSPRO\_REG\_A1
  - HTSuperProConstants, 927
  - NBCCommon.h, 1238
- HTSPRO\_REG\_A2
  - HTSuperProConstants, 927
  - NBCCommon.h, 1238
- HTSPRO\_REG\_A3
  - HTSuperProConstants, 927
  - NBCCommon.h, 1239
- HTSPRO\_REG\_CTRL
  - HTSuperProConstants, 927
  - NBCCommon.h, 1239
- HTSPRO\_REG\_DAC0\_FREQ
  - HTSuperProConstants, 927

- NBCCCommon.h, 1239
- HTSPRO\_REG\_DAC0\_MODE
  - HTSuperProConstants, 927
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DAC0\_VOLTAGE
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DAC1\_FREQ
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DAC1\_MODE
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DAC1\_VOLTAGE
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DCTRL
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DIN
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1239
- HTSPRO\_REG\_DLADDRESS
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_DLCHKSUM
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_DLCONTROL
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_DLDATA
  - HTSuperProConstants, 928
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_DOUT
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_LED
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_MEMORY\_20
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_MEMORY\_21
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_MEMORY\_22
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_MEMORY\_23
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1240
- HTSPRO\_REG\_MEMORY\_24
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_25
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_26
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_27
  - HTSuperProConstants, 929
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_28
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_29
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_2A
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_2B
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_2C
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_2D
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1241
- HTSPRO\_REG\_MEMORY\_2E
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_2F
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_30
  - HTSuperProConstants, 930
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_31

- HTSuperProConstants, 930
- NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_32
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_33
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_34
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_35
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_36
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_37
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1242
- HTSPRO\_REG\_MEMORY\_38
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_39
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3A
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3B
  - HTSuperProConstants, 931
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3C
  - HTSuperProConstants, 932
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3D
  - HTSuperProConstants, 932
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3E
  - HTSuperProConstants, 932
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_MEMORY\_3F
  - HTSuperProConstants, 932
  - NBCCCommon.h, 1243
- HTSPRO\_REG\_STROBE
  - HTSuperProConstants, 932
- NBCCCommon.h, 1243
- HTSPROAnalogInputConstants
  - HTSPRO\_A0, 933
  - HTSPRO\_A1, 933
  - HTSPRO\_A2, 933
  - HTSPRO\_A3, 933
- HTSPRODACIndexConstants
  - HTSPRO\_DAC0, 933
  - HTSPRO\_DAC1, 934
- HTSuperProConstants
  - HTSPRO\_REG\_A0, 927
  - HTSPRO\_REG\_A1, 927
  - HTSPRO\_REG\_A2, 927
  - HTSPRO\_REG\_A3, 927
  - HTSPRO\_REG\_CTRL, 927
  - HTSPRO\_REG\_DAC0\_FREQ, 927
  - HTSPRO\_REG\_DAC0\_MODE, 927
  - HTSPRO\_REG\_DAC0\_VOLTAGE, 928
  - HTSPRO\_REG\_DAC1\_FREQ, 928
  - HTSPRO\_REG\_DAC1\_MODE, 928
  - HTSPRO\_REG\_DAC1\_VOLTAGE, 928
  - HTSPRO\_REG\_DCTRL, 928
  - HTSPRO\_REG\_DIN, 928
  - HTSPRO\_REG\_DLADDRESS, 928
  - HTSPRO\_REG\_DLCHKSUM, 928
  - HTSPRO\_REG\_DLCONTROL, 928
  - HTSPRO\_REG\_DLDATA, 928
  - HTSPRO\_REG\_DOUT, 929
  - HTSPRO\_REG\_LED, 929
  - HTSPRO\_REG\_MEMORY\_20, 929
  - HTSPRO\_REG\_MEMORY\_21, 929
  - HTSPRO\_REG\_MEMORY\_22, 929
  - HTSPRO\_REG\_MEMORY\_23, 929
  - HTSPRO\_REG\_MEMORY\_24, 929
  - HTSPRO\_REG\_MEMORY\_25, 929
  - HTSPRO\_REG\_MEMORY\_26, 929
  - HTSPRO\_REG\_MEMORY\_27, 929
  - HTSPRO\_REG\_MEMORY\_28, 930
  - HTSPRO\_REG\_MEMORY\_29, 930
  - HTSPRO\_REG\_MEMORY\_2A, 930
  - HTSPRO\_REG\_MEMORY\_2B, 930

- HTSPRO\_REG\_MEMORY\_2C, [930](#)
- HTSPRO\_REG\_MEMORY\_2D, [930](#)
- HTSPRO\_REG\_MEMORY\_2E, [930](#)
- HTSPRO\_REG\_MEMORY\_2F, [930](#)
- HTSPRO\_REG\_MEMORY\_30, [930](#)
- HTSPRO\_REG\_MEMORY\_31, [930](#)
- HTSPRO\_REG\_MEMORY\_32, [931](#)
- HTSPRO\_REG\_MEMORY\_33, [931](#)
- HTSPRO\_REG\_MEMORY\_34, [931](#)
- HTSPRO\_REG\_MEMORY\_35, [931](#)
- HTSPRO\_REG\_MEMORY\_36, [931](#)
- HTSPRO\_REG\_MEMORY\_37, [931](#)
- HTSPRO\_REG\_MEMORY\_38, [931](#)
- HTSPRO\_REG\_MEMORY\_39, [931](#)
- HTSPRO\_REG\_MEMORY\_3A, [931](#)
- HTSPRO\_REG\_MEMORY\_3B, [931](#)
- HTSPRO\_REG\_MEMORY\_3C, [932](#)
- HTSPRO\_REG\_MEMORY\_3D, [932](#)
- HTSPRO\_REG\_MEMORY\_3E, [932](#)
- HTSPRO\_REG\_MEMORY\_3F, [932](#)
- HTSPRO\_REG\_STROBE, [932](#)
- I2C option constants, [810](#)
- I2C\_ADDR\_DEFAULT
  - GenericI2CConstants, [802](#)
  - NBCCCommon.h, [1243](#)
- I2C\_OPTION\_FAST
  - I2COptionConstants, [810](#)
  - NBCCCommon.h, [1244](#)
- I2C\_OPTION\_NORESTART
  - I2COptionConstants, [810](#)
  - NBCCCommon.h, [1244](#)
- I2C\_OPTION\_STANDARD
  - I2COptionConstants, [810](#)
  - NBCCCommon.h, [1244](#)
- I2C\_REG\_CMD
  - GenericI2CConstants, [802](#)
  - NBCCCommon.h, [1244](#)
- I2C\_REG\_DEVICE\_ID
  - GenericI2CConstants, [802](#)
  - NBCCCommon.h, [1244](#)
- I2C\_REG\_VENDOR\_ID
  - GenericI2CConstants, [802](#)
  - NBCCCommon.h, [1244](#)
- I2C\_REG\_VERSION
  - GenericI2CConstants, [802](#)
  - NBCCCommon.h, [1245](#)
- I2CBytes
  - LowSpeedModuleFunctions, [390](#)
  - NXCDefs.h, [1632](#)
- I2CBytesReady
  - LowSpeedModuleFunctions, [391](#)
  - NXCDefs.h, [1633](#)
- I2CCheckStatus
  - LowSpeedModuleFunctions, [391](#)
  - NXCDefs.h, [1634](#)
- I2CDeviceId
  - LowSpeedModuleFunctions, [392](#)
  - NXCDefs.h, [1634](#)
- I2CDeviceInfo
  - LowSpeedModuleFunctions, [392](#)
  - NXCDefs.h, [1635](#)
- I2COptionConstants
  - I2C\_OPTION\_FAST, [810](#)
  - I2C\_OPTION\_NORESTART, [810](#)
  - I2C\_OPTION\_STANDARD, [810](#)
- I2CRead
  - LowSpeedModuleFunctions, [393](#)
  - NXCDefs.h, [1635](#)
- I2CSendCommand
  - LowSpeedModuleFunctions, [394](#)
  - NXCDefs.h, [1636](#)
- I2CStatus
  - LowSpeedModuleFunctions, [394](#)
  - NXCDefs.h, [1637](#)
- I2CVendorId
  - LowSpeedModuleFunctions, [395](#)
  - NXCDefs.h, [1637](#)
- I2CVersion
  - LowSpeedModuleFunctions, [395](#)
  - NXCDefs.h, [1638](#)
- I2CWrite
  - LowSpeedModuleFunctions, [396](#)
  - NXCDefs.h, [1638](#)

- IN\_1
  - NBCCCommon.h, 1245
  - NBCInputPortConstants, 756
- IN\_2
  - NBCCCommon.h, 1245
  - NBCInputPortConstants, 756
- IN\_3
  - NBCCCommon.h, 1245
  - NBCInputPortConstants, 756
- IN\_4
  - NBCCCommon.h, 1245
  - NBCInputPortConstants, 756
- IN\_MODE\_ANGLESTEP
  - NBCCCommon.h, 1245
  - NBCSensorModeConstants, 760
- IN\_MODE\_BOOLEAN
  - NBCCCommon.h, 1245
  - NBCSensorModeConstants, 760
- IN\_MODE\_CELSIUS
  - NBCCCommon.h, 1245
  - NBCSensorModeConstants, 760
- IN\_MODE\_FAHRENHEIT
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 760
- IN\_MODE\_MODEMASK
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 760
- IN\_MODE\_PCTFULLSCALE
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 761
- IN\_MODE\_PERIODCOUNTER
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 761
- IN\_MODE\_RAW
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 761
- IN\_MODE\_SLOPEMASK
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 761
- IN\_MODE\_TRANSITIONCNT
  - NBCCCommon.h, 1246
  - NBCSensorModeConstants, 761
- IN\_TYPE\_ANGLE
  - NBCCCommon.h, 1246
  - NBCSensorTypeConstants, 757
- IN\_TYPE\_COLORBLUE
  - NBCCCommon.h, 1246
  - NBCSensorTypeConstants, 757
- IN\_TYPE\_COLOREXIT
  - NBCCCommon.h, 1246
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_COLORFULL
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_COLORGREEN
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_COLORNONE
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_COLORRED
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_CUSTOM
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_HISPEED
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_LIGHT\_ACTIVE
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_LIGHT\_INACTIVE
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_LOWSPEED
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 758
- IN\_TYPE\_LOWSPEED\_9V
  - NBCCCommon.h, 1247
  - NBCSensorTypeConstants, 759
- IN\_TYPE\_NO\_SENSOR
  - NBCCCommon.h, 1248
  - NBCSensorTypeConstants, 759
- IN\_TYPE\_REFLECTION
  - NBCCCommon.h, 1248
  - NBCSensorTypeConstants, 759
- IN\_TYPE\_SOUND\_DB
  - NBCCCommon.h, 1248
  - NBCSensorTypeConstants, 759
- IN\_TYPE\_SOUND\_DBA
  - NBCCCommon.h, 1248

- NBCSensorTypeConstants, 759
- IN\_TYPE\_SWITCH
  - NBCCCommon.h, 1248
  - NBCSensorTypeConstants, 759
- IN\_TYPE\_TEMPERATURE
  - NBCCCommon.h, 1248
  - NBCSensorTypeConstants, 759
- Index
  - ReadButtonType, 1098
- InPorts
  - S1, 274
  - S2, 276
  - S3, 276
  - S4, 276
- Input field constants, 761
- Input module, 44
- Input module constants, 45
- Input module functions, 285
- Input module IOMAP offsets, 767
- Input module types, 284
- Input port constants, 273
- Input port digital pin constants, 762
- INPUT\_BLACKCOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1248
- INPUT\_BLANK
  - InputColorIdxConstants, 764
  - NBCCCommon.h, 1248
- INPUT\_BLUE
  - InputColorIdxConstants, 764
  - NBCCCommon.h, 1248
- INPUT\_BLUECOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1248
- INPUT\_CAL\_POINT\_0
  - InputColorCalibrationConstants, 767
  - NBCCCommon.h, 1249
- INPUT\_CAL\_POINT\_1
  - InputColorCalibrationConstants, 767
  - NBCCCommon.h, 1249
- INPUT\_CAL\_POINT\_2
  - InputColorCalibrationConstants, 767
  - NBCCCommon.h, 1249
- INPUT\_CUSTOM9V
  - InputModuleConstants, 46
  - NBCCCommon.h, 1249
- INPUT\_CUSTOMACTIVE
  - InputModuleConstants, 46
  - NBCCCommon.h, 1249
- INPUT\_CUSTOMINACTIVE
  - InputModuleConstants, 46
  - NBCCCommon.h, 1249
- INPUT\_DIGI0
  - InputDigiPinConstants, 763
  - NBCCCommon.h, 1249
- INPUT\_DIGI1
  - InputDigiPinConstants, 763
  - NBCCCommon.h, 1249
- INPUT\_GREEN
  - InputColorIdxConstants, 764
  - NBCCCommon.h, 1250
- INPUT\_GREENCOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1250
- INPUT\_INVALID\_DATA
  - InputModuleConstants, 46
  - NBCCCommon.h, 1250
- INPUT\_NO\_OF\_COLORS
  - InputColorIdxConstants, 764
  - NBCCCommon.h, 1250
- INPUT\_NO\_OF\_POINTS
  - InputColorCalibrationConstants, 767
  - NBCCCommon.h, 1250
- INPUT\_PINCMD\_CLEAR
  - InputPinFuncConstants, 771
  - NBCCCommon.h, 1250
- INPUT\_PINCMD\_DIR
  - InputPinFuncConstants, 771
  - NBCCCommon.h, 1250
- INPUT\_PINCMD\_MASK
  - InputPinFuncConstants, 771
  - NBCCCommon.h, 1250
- INPUT\_PINCMD\_READ
  - InputPinFuncConstants, 771
  - NBCCCommon.h, 1251
- INPUT\_PINCMD\_SET
  - InputPinFuncConstants, 772
  - NBCCCommon.h, 1251

- INPUT\_PINCMD\_WAIT
  - InputPinFuncConstants, 772
  - NBCCCommon.h, 1251
- INPUT\_PINDIR\_INPUT
  - InputPinFuncConstants, 772
  - NBCCCommon.h, 1251
- INPUT\_PINDIR\_OUTPUT
  - InputPinFuncConstants, 772
  - NBCCCommon.h, 1251
- INPUT\_RED
  - InputColorIdxConstants, 764
  - NBCCCommon.h, 1251
- INPUT\_REDCOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1252
- INPUT\_RESETCAL
  - InputColorCalibrationStateConstants, 766
  - NBCCCommon.h, 1252
- INPUT\_RUNNINGCAL
  - InputColorCalibrationStateConstants, 766
  - NBCCCommon.h, 1252
- INPUT\_SENSORCAL
  - InputColorCalibrationStateConstants, 766
  - NBCCCommon.h, 1252
- INPUT\_SENSOROFF
  - InputColorCalibrationStateConstants, 766
  - NBCCCommon.h, 1252
- INPUT\_STARTCAL
  - InputColorCalibrationStateConstants, 766
  - NBCCCommon.h, 1252
- INPUT\_WHITECOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1252
- INPUT\_YELLOWCOLOR
  - InputColorValueConstants, 765
  - NBCCCommon.h, 1252
- InputColorCalibrationConstants
  - INPUT\_CAL\_POINT\_0, 767
  - INPUT\_CAL\_POINT\_1, 767
  - INPUT\_CAL\_POINT\_2, 767
  - INPUT\_NO\_OF\_POINTS, 767
- InputColorCalibrationStateConstants
  - INPUT\_RESETCAL, 766
  - INPUT\_RUNNINGCAL, 766
  - INPUT\_SENSORCAL, 766
  - INPUT\_SENSOROFF, 766
  - INPUT\_STARTCAL, 766
- InputColorIdxConstants
  - INPUT\_BLANK, 764
  - INPUT\_BLUE, 764
  - INPUT\_GREEN, 764
  - INPUT\_NO\_OF\_COLORS, 764
  - INPUT\_RED, 764
- InputColorValueConstants
  - INPUT\_BLACKCOLOR, 765
  - INPUT\_BLUECOLOR, 765
  - INPUT\_GREENCOLOR, 765
  - INPUT\_REDCOLOR, 765
  - INPUT\_WHITECOLOR, 765
  - INPUT\_YELLOWCOLOR, 765
- InputDigiPinConstants
  - INPUT\_DIGI0, 763
  - INPUT\_DIGI1, 763
- InputFieldConstants
  - InputModeField, 762
  - InvalidDataField, 762
  - NormalizedValueField, 762
  - RawValueField, 762
  - ScaledValueField, 762
  - TypeField, 762
- InputIOMAP
  - InputOffsetADRaw, 768
  - InputOffsetColorADRaw, 768
  - InputOffsetColorBoolean, 768
  - InputOffsetColorCalibration, 768
  - InputOffsetColorCalibrationState, 769
  - InputOffsetColorCalLimits, 769
  - InputOffsetColorSensorRaw, 769
  - InputOffsetColorSensorValue, 769
  - InputOffsetCustomActiveStatus, 769
  - InputOffsetCustomPctFullScale, 769
  - InputOffsetCustomZeroOffset, 769
  - InputOffsetDigiPinsDir, 769
  - InputOffsetDigiPinsIn, 769
  - InputOffsetDigiPinsOut, 770
  - InputOffsetInvalidData, 770

- InputOffsetSensorBoolean, 770
- InputOffsetSensorMode, 770
- InputOffsetSensorRaw, 770
- InputOffsetSensorType, 770
- InputOffsetSensorValue, 770
- InputModeField
  - InputFieldConstants, 762
  - NBCCCommon.h, 1252
- InputModuleConstants
  - INPUT\_CUSTOM9V, 46
  - INPUT\_CUSTOMACTIVE, 46
  - INPUT\_CUSTOMINACTIVE, 46
  - INPUT\_INVALID\_DATA, 46
- InputModuleFunctions
  - ClearSensor, 289
  - ColorADRaw, 289
  - ColorBoolean, 290
  - ColorCalibration, 290
  - ColorCalibrationState, 291
  - ColorCallimits, 291
  - ColorSensorRaw, 292
  - ColorSensorValue, 292
  - CustomSensorActiveStatus, 293
  - CustomSensorPercentFullScale, 293
  - CustomSensorZeroOffset, 294
  - GetInput, 294
  - ReadSensorColorEx, 295
  - ReadSensorColorRaw, 295
  - ResetSensor, 296
  - Sensor, 296
  - SensorBoolean, 297
  - SensorDigiPinsDirection, 297
  - SensorDigiPinsOutputLevel, 297
  - SensorDigiPinsStatus, 298
  - SensorInvalid, 298
  - SensorMode, 299
  - SensorNormalized, 299
  - SensorRaw, 299
  - SensorScaled, 300
  - SensorType, 300
  - SensorValue, 301
  - SensorValueBool, 301
  - SensorValueRaw, 302
  - SetCustomSensorActiveStatus, 302
  - SetCustomSensorPercentFullScale, 302
  - SetCustomSensorZeroOffset, 303
  - SetInput, 303
  - SetSensor, 304
  - SetSensorBoolean, 304
  - SetSensorColorBlue, 304
  - SetSensorColorFull, 305
  - SetSensorColorGreen, 305
  - SetSensorColorNone, 305
  - SetSensorColorRed, 306
  - SetSensorDigiPinsDirection, 306
  - SetSensorDigiPinsOutputLevel, 307
  - SetSensorDigiPinsStatus, 307
  - SetSensorEMeter, 307
  - SetSensorLight, 308
  - SetSensorLowspeed, 308
  - SetSensorMode, 309
  - SetSensorSound, 309
  - SetSensorTemperature, 309
  - SetSensorTouch, 310
  - SetSensorType, 310
  - SetSensorUltrasonic, 311
  - SysColorSensorRead, 311
  - SysInputPinFunction, 311
- InputModuleID
  - ModuleIDConstants, 259
  - NBCCCommon.h, 1253
- InputModuleName
  - ModuleNameConstants, 257
  - NBCCCommon.h, 1253
- InputOffsetADRaw
  - InputIOMAP, 768
  - NBCCCommon.h, 1253
- InputOffsetColorADRaw
  - InputIOMAP, 768
  - NBCCCommon.h, 1253
- InputOffsetColorBoolean
  - InputIOMAP, 768
  - NBCCCommon.h, 1253
- InputOffsetColorCalibration
  - InputIOMAP, 768
  - NBCCCommon.h, 1253
- InputOffsetColorCalibrationState
  - InputIOMAP, 769
  - NBCCCommon.h, 1253
- InputOffsetColorCallimits
  - InputIOMAP, 769

- NBCCCommon.h, [1253](#)
- InputOffsetColorSensorRaw
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1253](#)
- InputOffsetColorSensorValue
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1253](#)
- InputOffsetCustomActiveStatus
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetCustomPctFullScale
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetCustomZeroOffset
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetDigiPinsDir
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetDigiPinsIn
  - InputIOMAP, [769](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetDigiPinsOut
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetInvalidData
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetSensorBoolean
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetSensorMode
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetSensorRaw
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1254](#)
- InputOffsetSensorType
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1255](#)
- InputOffsetSensorValue
  - InputIOMAP, [770](#)
  - NBCCCommon.h, [1255](#)
- InputPinFuncConstants
  - INPUT\_PINCMD\_CLEAR, [771](#)
  - INPUT\_PINCMD\_DIR, [771](#)
  - INPUT\_PINCMD\_MASK, [771](#)
  - INPUT\_PINCMD\_READ, [771](#)
  - INPUT\_PINCMD\_SET, [772](#)
  - INPUT\_PINCMD\_WAIT, [772](#)
  - INPUT\_PINDIR\_INPUT, [772](#)
  - INPUT\_PINDIR\_OUTPUT, [772](#)
- InputPinFunction
  - NBCCCommon.h, [1255](#)
  - SysCallConstants, [688](#)
- InputPinFunctionType, [1070](#)
  - Cmd, [1071](#)
  - Data, [1071](#)
  - Pin, [1071](#)
  - Port, [1072](#)
  - Result, [1072](#)
- InputValuesType, [1072](#)
  - Calibrated, [1073](#)
  - CalibratedValue, [1073](#)
  - NormalizedValue, [1073](#)
  - Port, [1073](#)
  - RawValue, [1073](#)
  - ScaledValue, [1073](#)
  - SensorMode, [1074](#)
  - SensorType, [1074](#)
  - Valid, [1074](#)
- INT\_MAX
  - NBCCCommon.h, [1255](#)
  - NXTLimits, [1011](#)
- INT\_MIN
  - NBCCCommon.h, [1255](#)
  - NXTLimits, [1011](#)
- INTF\_BTOFF
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1255](#)
- INTF\_BTON
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1255](#)
- INTF\_CONNECT
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1255](#)
- INTF\_CONNECTBYNAME
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1255](#)
- INTF\_CONNECTREQ
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1256](#)

- INTF\_DISCONNECT
  - CommInterfaceConstants, [853](#)
  - NBCCCommon.h, [1256](#)
- INTF\_DISCONNECTALL
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_EXTREAD
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_FACTORYRESET
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_OPENSTREAM
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_PINREQ
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_REMOVEDEVICE
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_SEARCH
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_SENDDATA
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1256](#)
- INTF\_SENDFILE
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1257](#)
- INTF\_SETBTNAME
  - CommInterfaceConstants, [854](#)
  - NBCCCommon.h, [1257](#)
- INTF\_SETCMDMODE
  - CommInterfaceConstants, [855](#)
  - NBCCCommon.h, [1257](#)
- INTF\_STOPSEARCH
  - CommInterfaceConstants, [855](#)
  - NBCCCommon.h, [1257](#)
- INTF\_VISIBILITY
  - CommInterfaceConstants, [855](#)
  - NBCCCommon.h, [1257](#)
- Invalid
  - ColorSensorReadType, [1018](#)
- InvalidDataField
  - InputFieldConstants, [762](#)
  - NBCCCommon.h, [1257](#)
- IOCtrl module, [54](#)
- IOCtrl module constants, [714](#)
- IOCtrl module functions, [459](#)
- IOCtrl module IOMAP offsets, [716](#)
- IOCtrl module types, [459](#)
- IOCTRL\_BOOT
  - IOCtrlIPO, [715](#)
  - NBCCCommon.h, [1257](#)
- IOCTRL\_POWERDOWN
  - IOCtrlIPO, [715](#)
  - NBCCCommon.h, [1257](#)
- IOCtrlIOMAP
  - IOCtrlOffsetPowerOn, [716](#)
- IOCtrlModuleFunctions
  - PowerDown, [460](#)
  - RebootInFirmwareMode, [460](#)
  - SleepNow, [460](#)
- IOCtrlModuleID
  - ModuleIDConstants, [259](#)
  - NBCCCommon.h, [1257](#)
- IOCtrlModuleName
  - ModuleNameConstants, [258](#)
  - NBCCCommon.h, [1257](#)
- IOCtrlOffsetPowerOn
  - IOCtrlIOMAP, [716](#)
  - NBCCCommon.h, [1258](#)
- IOCtrlIPO
  - IOCTRL\_BOOT, [715](#)
  - IOCTRL\_POWERDOWN, [715](#)
- IOMapRead
  - NBCCCommon.h, [1258](#)
  - SysCallConstants, [689](#)
- IOMapReadByID
  - NBCCCommon.h, [1258](#)
  - SysCallConstants, [689](#)
- IOMapReadByIDType, [1074](#)
  - Buffer, [1075](#)
  - Count, [1075](#)
  - ModuleID, [1075](#)
  - Offset, [1075](#)
  - Result, [1075](#)
- IOMapReadType, [1076](#)
  - Buffer, [1076](#)
  - Count, [1076](#)
  - ModuleName, [1077](#)

- Offset, [1077](#)
- Result, [1077](#)
- IOMapWrite
  - NBCCCommon.h, [1258](#)
  - SysCallConstants, [689](#)
- IOMapWriteByID
  - NBCCCommon.h, [1258](#)
  - SysCallConstants, [689](#)
- IOMapWriteByIDType, [1077](#)
  - Buffer, [1078](#)
  - ModuleID, [1078](#)
  - Offset, [1078](#)
  - Result, [1078](#)
- IOMapWriteType, [1079](#)
  - Buffer, [1079](#)
  - ModuleName, [1079](#)
  - Offset, [1080](#)
  - Result, [1080](#)
- IR Train channel constants, [904](#)
- IRTrainChannels
  - TRAIN\_CHANNEL\_1, [905](#)
  - TRAIN\_CHANNEL\_2, [905](#)
  - TRAIN\_CHANNEL\_3, [905](#)
  - TRAIN\_CHANNEL\_ALL, [905](#)
- IRTrainFuncs
  - TRAIN\_FUNC\_DECR\_SPEED, [904](#)
  - TRAIN\_FUNC\_INCR\_SPEED, [904](#)
  - TRAIN\_FUNC\_STOP, [904](#)
  - TRAIN\_FUNC\_TOGGLE\_LIGHT, [904](#)
- isalnum
  - ctypeAPI, [674](#)
  - NXCDefs.h, [1639](#)
- isalpha
  - ctypeAPI, [674](#)
  - NXCDefs.h, [1639](#)
- isctrl
  - ctypeAPI, [675](#)
  - NXCDefs.h, [1640](#)
- isdigit
  - ctypeAPI, [675](#)
  - NXCDefs.h, [1640](#)
- isgraph
  - ctypeAPI, [675](#)
  - NXCDefs.h, [1641](#)
- islower
  - ctypeAPI, [676](#)
  - NXCDefs.h, [1641](#)
- isNAN
  - cmathAPI, [625](#)
  - NXCDefs.h, [1641](#)
- isprint
  - ctypeAPI, [676](#)
  - NXCDefs.h, [1642](#)
- ispunct
  - ctypeAPI, [677](#)
  - NXCDefs.h, [1642](#)
- isspace
  - ctypeAPI, [677](#)
  - NXCDefs.h, [1643](#)
- isupper
  - ctypeAPI, [677](#)
  - NXCDefs.h, [1643](#)
- isxdigit
  - ctypeAPI, [678](#)
  - NXCDefs.h, [1643](#)
- JoystickDir
  - JoystickMessageType, [1081](#)
- JoystickMessageRead
  - CommModuleFunctions, [487](#)
  - NXCDefs.h, [1644](#)
- JoystickMessageType, [1080](#)
  - BothMotors, [1081](#)
  - Buttons, [1081](#)
  - JoystickDir, [1081](#)
  - LeftMotor, [1081](#)
  - LeftSpeed, [1081](#)
  - RightMotor, [1082](#)
  - RightSpeed, [1082](#)
- KeepAlive
  - NBCCCommon.h, [1258](#)
  - SysCallConstants, [689](#)
- KeepAliveType, [1082](#)
  - Result, [1083](#)
- labs
  - cstdlibAPI, [649](#)
  - NXCDefs.h, [1644](#)
- LCD\_LINE1

- LineConstants, [692](#)
- NBCCCommon.h, [1258](#)
- LCD\_LINE2
  - LineConstants, [692](#)
  - NBCCCommon.h, [1259](#)
- LCD\_LINE3
  - LineConstants, [693](#)
  - NBCCCommon.h, [1260](#)
- LCD\_LINE4
  - LineConstants, [693](#)
  - NBCCCommon.h, [1260](#)
- LCD\_LINE5
  - LineConstants, [694](#)
  - NBCCCommon.h, [1260](#)
- LCD\_LINE6
  - LineConstants, [694](#)
  - NBCCCommon.h, [1261](#)
- LCD\_LINE7
  - LineConstants, [694](#)
  - NBCCCommon.h, [1261](#)
- LCD\_LINE8
  - LineConstants, [694](#)
  - NBCCCommon.h, [1261](#)
- ldiv
  - cstdlibAPI, [649](#)
  - NXCDefs.h, [1645](#)
- ldiv\_t, [1083](#)
  - quot, [1084](#)
  - rem, [1084](#)
- LDR\_APPENDNOTPOSSIBLE
  - LoaderErrors, [718](#)
  - NBCCCommon.h, [1261](#)
- LDR\_BTBUSY
  - LoaderErrors, [718](#)
  - NBCCCommon.h, [1262](#)
- LDR\_BTCONNECTFAIL
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1262](#)
- LDR\_BTTIMEOUT
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_BOOTCMD
  - LoaderFunctionConstants, [723](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_BTFACTORYRESET
  - LoaderFunctionConstants, [723](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_BTGETADR
  - LoaderFunctionConstants, [723](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_CLOSE
  - LoaderFunctionConstants, [723](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_CLOSEMODHANDLE
  - LoaderFunctionConstants, [723](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_CROPDATAFILE
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_DELETE
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1262](#)
- LDR\_CMD\_DELETEUSERFLASH
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_DEVICEINFO
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_FINDFIRST
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_FINDFIRSTMODULE
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_FINDNEXT
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_FINDNEXTMODULE
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_IOMAPREAD
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_IOMAPWRITE
  - LoaderFunctionConstants, [724](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_OPENAPPENDDATA
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1263](#)
- LDR\_CMD\_OPENREAD
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1263](#)

- LDR\_CMD\_OPENREADLINEAR
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_OPENWRITE
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_OPENWRITEDATA
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_OPENWRITELINEAR
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_POLLCMD
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_POLLCMDLEN
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_READ
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_RENAMEFILE
  - LoaderFunctionConstants, [725](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_RESIZEDATAFILE
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_SEEKFROMCURRENT
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1264](#)
- LDR\_CMD\_SEEKFROMEND
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1265](#)
- LDR\_CMD\_SEEKFROMSTART
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1265](#)
- LDR\_CMD\_SETBRICKNAME
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1265](#)
- LDR\_CMD\_VERSIONS
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1265](#)
- LDR\_CMD\_WRITE
  - LoaderFunctionConstants, [726](#)
  - NBCCCommon.h, [1265](#)
- LDR\_ENDOFFILE
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1265](#)
- LDR\_EOFEXPECTED
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1265](#)
- LDR\_FILEEXISTS
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1265](#)
- LDR\_FILEISBUSY
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILEISFULL
  - LoaderErrors, [719](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILENOTFOUND
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILETX\_CLOSEERROR
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILETX\_DSTEXISTS
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILETX\_SRCMISSING
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1266](#)
- LDR\_FILETX\_STREAMERROR
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1266](#)
- LDR\_HANDLEALREADYCLOSED
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1267](#)
- LDR\_ILLEGALFILENAME
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1267](#)
- LDR\_ILLEGALHANDLE
  - LoaderErrors, [720](#)
  - NBCCCommon.h, [1267](#)
- LDR\_INPROGRESS
  - LoaderErrors, [721](#)
  - NBCCCommon.h, [1267](#)
- LDR\_INVALIDSEEK
  - LoaderErrors, [721](#)

- NBCCCommon.h, 1267
- LDR\_MODULENOTFOUND
  - LoaderErrors, 721
  - NBCCCommon.h, 1267
- LDR\_NOLINEARSPACE
  - LoaderErrors, 721
  - NBCCCommon.h, 1267
- LDR\_NOMOREFILES
  - LoaderErrors, 721
  - NBCCCommon.h, 1267
- LDR\_NOMOREHANDLES
  - LoaderErrors, 721
  - NBCCCommon.h, 1267
- LDR\_NOSPACE
  - LoaderErrors, 721
  - NBCCCommon.h, 1267
- LDR\_NOTLINEARFILE
  - LoaderErrors, 721
  - NBCCCommon.h, 1268
- LDR\_NOWRITEBUFFERS
  - LoaderErrors, 721
  - NBCCCommon.h, 1268
- LDR\_OUTOFBOUNDARY
  - LoaderErrors, 721
  - NBCCCommon.h, 1268
- LDR\_REQPIN
  - LoaderErrors, 722
  - NBCCCommon.h, 1268
- LDR\_SUCCESS
  - LoaderErrors, 722
  - NBCCCommon.h, 1268
- LDR\_UNDEFINEDERROR
  - LoaderErrors, 722
  - NBCCCommon.h, 1268
- LED\_BLUE
  - LEDCtrlConstants, 124
  - NBCCCommon.h, 1268
- LED\_NONE
  - LEDCtrlConstants, 124
  - NBCCCommon.h, 1268
- LED\_RED
  - LEDCtrlConstants, 124
  - NBCCCommon.h, 1269
- LEDCtrlConstants
  - LED\_BLUE, 124
  - LED\_NONE, 124
  - LED\_RED, 124
- LeftMotor
  - JoystickMessageType, 1081
- LeftSpeed
  - JoystickMessageType, 1081
- LeftStr
  - cstringAPI, 662
  - NXCDefs.h, 1645
- LEGO I2C address constants, 803
- LEGO temperature sensor constants, 805
- LEGO\_ADDR\_EMETER
  - LEGOI2CAddressConstants, 803
  - NBCCCommon.h, 1269
- LEGO\_ADDR\_TEMP
  - LEGOI2CAddressConstants, 803
  - NBCCCommon.h, 1269
- LEGO\_ADDR\_US
  - LEGOI2CAddressConstants, 803
  - NBCCCommon.h, 1269
- LEGOI2CAddressConstants
  - LEGO\_ADDR\_EMETER, 803
  - LEGO\_ADDR\_TEMP, 803
  - LEGO\_ADDR\_US, 803
- Length
  - FileFindType, 1060
  - FileOpenType, 1061
  - FileReadWriteType, 1063
  - FileSeekType, 1068
  - LoaderExecuteFunctionType, 1088
  - ReadLastResponseType, 1099
- Line number constants, 691
- LineConstants
  - LCD\_LINE1, 692
  - LCD\_LINE2, 692
  - LCD\_LINE3, 693
  - LCD\_LINE4, 693
  - LCD\_LINE5, 694
  - LCD\_LINE6, 694
  - LCD\_LINE7, 694
  - LCD\_LINE8, 694
- LineOut
  - DisplayModuleFunctions, 362
  - NXCDefs.h, 1645
- ListFiles
  - NBCCCommon.h, 1269
  - SysCallConstants, 689

- ListFilesType, 1084
  - FileList, 1085
  - Pattern, 1085
  - Result, 1085
- Loader module, 54
- Loader module constants, 716
- Loader module error codes, 717
- Loader module function constants, 722
- Loader module functions, 573
- Loader module IOMAP offsets, 717
- Loader module types, 572
- LoaderErrors
  - LDR\_APPENDNOTPOSSIBLE, 718
  - LDR\_BTBUSY, 718
  - LDR\_BTCONNECTFAIL, 719
  - LDR\_BTTIMEOUT, 719
  - LDR\_ENDOFFILE, 719
  - LDR\_EOFEXPECTED, 719
  - LDR\_FILEEXISTS, 719
  - LDR\_FILEISBUSY, 719
  - LDR\_FILEISFULL, 719
  - LDR\_FILENOTFOUND, 720
  - LDR\_FILETX\_CLOSEERROR, 720
  - LDR\_FILETX\_DSTEXISTS, 720
  - LDR\_FILETX\_SRCMISSING, 720
  - LDR\_FILETX\_STREAMERROR, 720
  - LDR\_FILETX\_TIMEOUT, 720
  - LDR\_-
    - HANDLEALREADYCLOSED, 720
  - LDR\_ILLEGALFILENAME, 720
  - LDR\_ILLEGALHANDLE, 720
  - LDR\_INPROGRESS, 721
  - LDR\_INVALIDSEEK, 721
  - LDR\_MODULENOTFOUND, 721
  - LDR\_NOLINEARSPACE, 721
  - LDR\_NOMOREFILES, 721
  - LDR\_NOMOREHANDLES, 721
  - LDR\_NOSPACE, 721
  - LDR\_NOTLINEARFILE, 721
  - LDR\_NOWRITEBUFFERS, 721
  - LDR\_OUTOFBOUNDARY, 721
  - LDR\_REQPIN, 722
  - LDR\_SUCCESS, 722
  - LDR\_UNDEFINEDERROR, 722
- LoaderExecuteFunction
  - NBCCCommon.h, 1269
  - SysCallConstants, 689
- LoaderExecuteFunctionType, 1085
  - Buffer, 1088
  - Cmd, 1088
  - Filename, 1088
  - Length, 1088
  - Result, 1088
- LoaderFunctionConstants
  - LDR\_CMD\_BOOTCMD, 723
  - LDR\_CMD\_BTFACTORYRESET, 723
  - LDR\_CMD\_BTGETADR, 723
  - LDR\_CMD\_CLOSE, 723
  - LDR\_CMD\_-
    - CLOSEMODHANDLE, 723
  - LDR\_CMD\_CROPDATAFILE, 724
  - LDR\_CMD\_DELETE, 724
  - LDR\_CMD\_-
    - DELETEUSERFLASH, 724
  - LDR\_CMD\_DEVICEINFO, 724
  - LDR\_CMD\_FINDFIRST, 724
  - LDR\_CMD\_-
    - FINDFIRSTMODULE, 724
  - LDR\_CMD\_FINDNEXT, 724
  - LDR\_CMD\_-
    - FINDNEXTMODULE, 724
  - LDR\_CMD\_IOMAPREAD, 724
  - LDR\_CMD\_IOMAPWRITE, 724
  - LDR\_CMD\_OPENAPPENDDATA, 725
  - LDR\_CMD\_OPENREAD, 725
  - LDR\_CMD\_OPENREADLINEAR, 725
  - LDR\_CMD\_OPENWRITE, 725
  - LDR\_CMD\_OPENWRITEDATA, 725
  - LDR\_CMD\_-
    - OPENWRITELINEAR, 725
  - LDR\_CMD\_POLLCMD, 725
  - LDR\_CMD\_POLLCMDLEN, 725
  - LDR\_CMD\_READ, 725

- LDR\_CMD\_RENAMEFILE, 725
- LDR\_CMD\_RESIZEDATAFILE, 726
- LDR\_CMD\_-
  - SEEKFROMCURRENT, 726
- LDR\_CMD\_SEEKFROMEND, 726
- LDR\_CMD\_SEEKFROMSTART, 726
- LDR\_CMD\_SETBRICKNAME, 726
- LDR\_CMD\_VERSIONS, 726
- LDR\_CMD\_WRITE, 726
- LoaderIOMAP
  - LoaderOffsetFreeUserFlash, 717
  - LoaderOffsetPFunc, 717
- LoaderModuleConstants
  - EOF, 717
  - NULL, 717
- LoaderModuleFunctions
  - CloseFile, 576
  - CreateFile, 577
  - CreateFileLinear, 577
  - CreateFileNonLinear, 578
  - DeleteFile, 579
  - FindFirstFile, 579
  - FindNextFile, 579
  - FreeMemory, 580
  - OpenFileAppend, 580
  - OpenFileRead, 581
  - OpenFileReadLinear, 581
  - Read, 582
  - ReadBytes, 582
  - ReadLn, 583
  - ReadLnString, 583
  - RenameFile, 584
  - ResizeFile, 584
  - ResolveHandle, 585
  - SizeOf, 585
  - SysFileClose, 586
  - SysFileDelete, 586
  - SysFileFindFirst, 586
  - SysFileFindNext, 587
  - SysFileOpenAppend, 587
  - SysFileOpenRead, 588
  - SysFileOpenReadLinear, 588
  - SysFileOpenWrite, 588
  - SysFileOpenWriteLinear, 589
  - SysFileOpenWriteNonLinear, 589
  - SysFileRead, 590
  - SysFileRename, 590
  - SysFileResize, 590
  - SysFileResolveHandle, 591
  - SysFileSeek, 591
  - SysFileTell, 591
  - SysFileWrite, 592
  - SysListFiles, 592
  - SysLoaderExecuteFunction, 592
  - Write, 593
  - WriteBytes, 593
  - WriteBytesEx, 594
  - WriteLn, 594
  - WriteLnString, 595
  - WriteString, 595
- LoaderModuleID
  - ModuleIDConstants, 260
  - NBCCCommon.h, 1269
- LoaderModuleName
  - ModuleNameConstants, 258
  - NBCCCommon.h, 1269
- LoaderOffsetFreeUserFlash
  - LoaderIOMAP, 717
  - NBCCCommon.h, 1269
- LoaderOffsetPFunc
  - LoaderIOMAP, 717
  - NBCCCommon.h, 1269
- Location
  - DrawFontType, 1046
  - DrawGraphicArrayType, 1048
  - DrawGraphicType, 1050
  - DrawPointType, 1052
  - DrawRectType, 1055
  - DrawTextType, 1056
- LocationType, 1088
  - X, 1089
  - Y, 1089
- Log
  - cmathAPI, 611
  - NXCDefs.h, 1487
- log
  - cmathAPI, 625
  - NXCDefs.h, 1646

- Log10
  - cmathAPI, [611](#)
  - NXCDefs.h, [1488](#)
- log10
  - cmathAPI, [626](#)
  - NXCDefs.h, [1647](#)
- LONG\_MAX
  - NBCCCommon.h, [1270](#)
  - NXTLimits, [1011](#)
- LONG\_MIN
  - NBCCCommon.h, [1270](#)
  - NXTLimits, [1012](#)
- LongAbort
  - NXCDefs.h, [1647](#)
  - UiModuleFunctions, [563](#)
- Loop
  - SoundPlayFileType, [1106](#)
  - SoundPlayToneType, [1107](#)
- Low level LowSpeed module functions, [403](#)
- Low Speed module, [56](#)
- Low speed module IOMAP offsets, [798](#)
- LowLevelLowSpeedModuleFunctions
  - GetLSInputBuffer, [405](#)
  - GetLSOutputBuffer, [405](#)
  - LSChannelState, [406](#)
  - LSErrorType, [406](#)
  - LSInputBufferBytesToRx, [406](#)
  - LSInputBufferInPtr, [407](#)
  - LSInputBufferOutPtr, [407](#)
  - LSMode, [407](#)
  - LSNoRestartOnRead, [408](#)
  - LSOutputBufferBytesToRx, [408](#)
  - LSOutputBufferInPtr, [408](#)
  - LSOutputBufferOutPtr, [409](#)
  - LSSpeed, [409](#)
  - LSState, [410](#)
  - SetI2COptions, [410](#)
- LowSpeed module constants, [793](#)
- LowSpeed module functions, [387](#)
- LowSpeed module system call functions, [410](#)
- LowSpeed module types, [387](#)
- LOWSPEED\_CH\_NOT\_READY
  - LowSpeedErrorTypeConstants, [797](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_COMMUNICATING
  - LowSpeedChannelStateConstants, [795](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_DATA\_RECEIVED
  - LowSpeedModeConstants, [797](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_DONE
  - LowSpeedChannelStateConstants, [795](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_ERROR
  - LowSpeedChannelStateConstants, [795](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_IDLE
  - LowSpeedChannelStateConstants, [796](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_INIT
  - LowSpeedChannelStateConstants, [796](#)
  - NBCCCommon.h, [1270](#)
- LOWSPEED\_LOAD\_BUFFER
  - LowSpeedChannelStateConstants, [796](#)
  - NBCCCommon.h, [1271](#)
- LOWSPEED\_NO\_ERROR
  - LowSpeedErrorTypeConstants, [797](#)
  - NBCCCommon.h, [1271](#)
- LOWSPEED\_RECEIVING
  - LowSpeedModeConstants, [797](#)
  - NBCCCommon.h, [1271](#)
- LOWSPEED\_RX\_ERROR
  - LowSpeedErrorTypeConstants, [797](#)
  - NBCCCommon.h, [1271](#)
- LOWSPEED\_TRANSMITTING
  - LowSpeedModeConstants, [797](#)
  - NBCCCommon.h, [1271](#)
- LOWSPEED\_TX\_ERROR
  - LowSpeedErrorTypeConstants, [798](#)
  - NBCCCommon.h, [1271](#)
- LowspeedBytesReady
  - LowSpeedModuleFunctions, [397](#)
  - NXCDefs.h, [1648](#)
- LowSpeedChannelStateConstants

- LOWSPEED\_COMMUNICATING, 795
  - LOWSPEED\_DONE, 795
  - LOWSPEED\_ERROR, 795
  - LOWSPEED\_IDLE, 796
  - LOWSPEED\_INIT, 796
  - LOWSPEED\_LOAD\_BUFFER, 796
- LowSpeedCheckStatus
  - LowSpeedModuleFunctions, 397
  - NXCDefs.h, 1648
- LowSpeedErrorTypeConstants
  - LOWSPEED\_CH\_NOT\_READY, 797
  - LOWSPEED\_NO\_ERROR, 797
  - LOWSPEED\_RX\_ERROR, 797
  - LOWSPEED\_TX\_ERROR, 798
- LowSpeedIOMAP
  - LowSpeedOffsetChannelState, 798
  - LowSpeedOffsetErrorType, 798
  - LowSpeedOffsetInBufBuf, 799
  - LowSpeedOffsetInBufBytesToRx, 799
  - LowSpeedOffsetInBufInPtr, 799
  - LowSpeedOffsetInBufOutPtr, 799
  - LowSpeedOffsetMode, 799
  - LowSpeedOffsetNoRestartOnRead, 799
  - LowSpeedOffsetOutBufBuf, 799
  - LowSpeedOffsetOutBufBytesToRx, 799
  - LowSpeedOffsetOutBufInPtr, 799
  - LowSpeedOffsetOutBufOutPtr, 799
  - LowSpeedOffsetSpeed, 800
  - LowSpeedOffsetState, 800
- LowSpeedModeConstants
  - LOWSPEED\_DATA\_RECEIVED, 797
  - LOWSPEED\_RECEIVING, 797
  - LOWSPEED\_TRANSMITTING, 797
- LowSpeedModuleFunctions
  - ConfigureTemperatureSensor, 389
  - I2CBytes, 390
  - I2CBytesReady, 391
  - I2CCheckStatus, 391
  - I2CDeviceId, 392
  - I2CDeviceInfo, 392
  - I2CRead, 393
  - I2CSendCommand, 394
  - I2CStatus, 394
  - I2CVendorId, 395
  - I2CVersion, 395
  - I2CWrite, 396
  - LowSpeedBytesReady, 397
  - LowSpeedCheckStatus, 397
  - LowSpeedRead, 398
  - LowSpeedStatus, 399
  - LowSpeedWrite, 399
  - ReadI2CRegister, 400
  - ReadSensorEMeter, 401
  - ReadSensorUSEx, 401
  - SensorTemperature, 402
  - SensorUS, 402
  - WriteI2CRegister, 403
- LowSpeedModuleID
  - ModuleIDConstants, 260
  - NBCCommon.h, 1271
- LowSpeedModuleName
  - ModuleNameConstants, 258
  - NBCCommon.h, 1271
- LowSpeedModuleSystemCallFunctions
  - SysCommLSCheckStatus, 411
  - SysCommLSRead, 411
  - SysCommLSWrite, 412
  - SysCommLSWriteEx, 412
- LowSpeedNoRestartConstants
  - LSREAD\_NO\_RESTART\_1, 800
  - LSREAD\_NO\_RESTART\_2, 800
  - LSREAD\_NO\_RESTART\_3, 801
  - LSREAD\_NO\_RESTART\_4, 801
  - LSREAD\_NO\_RESTART\_MASK, 801
  - LSREAD\_RESTART\_ALL, 801
  - LSREAD\_RESTART\_NONE, 801
- LowSpeedOffsetChannelState
  - LowSpeedIOMAP, 798
  - NBCCommon.h, 1271
- LowSpeedOffsetErrorType
  - LowSpeedIOMAP, 798
  - NBCCommon.h, 1271
- LowSpeedOffsetInBufBuf
  - LowSpeedIOMAP, 799

- NBCCCommon.h, [1272](#)
- LowSpeedOffsetInBufBytesToRx
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetInBufInPtr
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetInBufOutPtr
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetMode
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetNoRestartOnRead
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetOutBufBuf
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetOutBufBytesToRx
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetOutBufInPtr
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetOutBufOutPtr
  - LowSpeedIOMAP, [799](#)
  - NBCCCommon.h, [1272](#)
- LowSpeedOffsetSpeed
  - LowSpeedIOMAP, [800](#)
  - NBCCCommon.h, [1273](#)
- LowSpeedOffsetState
  - LowSpeedIOMAP, [800](#)
  - NBCCCommon.h, [1273](#)
- LowspeedRead
  - LowSpeedModuleFunctions, [398](#)
  - NXCDefs.h, [1649](#)
- LowSpeedStateConstants
  - COM\_CHANNEL\_FOUR\_-ACTIVE, [794](#)
  - COM\_CHANNEL\_NONE\_-ACTIVE, [794](#)
  - COM\_CHANNEL\_ONE\_ACTIVE, [794](#)
  - COM\_CHANNEL\_THREE\_-ACTIVE, [795](#)
- COM\_CHANNEL\_TWO\_ACTIVE, [795](#)
- LowspeedStatus
  - LowSpeedModuleFunctions, [399](#)
  - NXCDefs.h, [1650](#)
- LowspeedWrite
  - LowSpeedModuleFunctions, [399](#)
  - NXCDefs.h, [1650](#)
- LR\_COULD\_NOT\_SAVE
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1273](#)
- LR\_ENTRY\_REMOVED
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1273](#)
- LR\_STORE\_IS\_FULL
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1273](#)
- LR\_SUCCESS
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1273](#)
- LR\_UNKNOWN\_ADDR
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1273](#)
- LSChannelState
  - LowLevelLowSpeedModuleFunctions, [406](#)
  - NXCDefs.h, [1651](#)
- LSChannelState constants, [795](#)
- LSErrorType
  - LowLevelLowSpeedModuleFunctions, [406](#)
  - NXCDefs.h, [1652](#)
- LSErrorType constants, [797](#)
- LSInputBufferBytesToRx
  - LowLevelLowSpeedModuleFunctions, [406](#)
  - NXCDefs.h, [1652](#)
- LSInputBufferInPtr
  - LowLevelLowSpeedModuleFunctions, [407](#)
  - NXCDefs.h, [1652](#)
- LSInputBufferOutPtr
  - LowLevelLowSpeedModuleFunctions, [407](#)
  - NXCDefs.h, [1653](#)
- LSMode

- LowLevelLowSpeedModuleFunc-  
tions, [407](#)
  - NXCDefs.h, [1653](#)
- LSMode constants, [796](#)
- LSNoRestartOnRead
  - LowLevelLowSpeedModuleFunc-  
tions, [408](#)
  - NXCDefs.h, [1654](#)
- LSNoRestartOnRead constants, [800](#)
- LSOutputBufferBytesToRx
  - LowLevelLowSpeedModuleFunc-  
tions, [408](#)
  - NXCDefs.h, [1654](#)
- LSOutputBufferInPtr
  - LowLevelLowSpeedModuleFunc-  
tions, [408](#)
  - NXCDefs.h, [1654](#)
- LSOutputBufferOutPtr
  - LowLevelLowSpeedModuleFunc-  
tions, [409](#)
  - NXCDefs.h, [1655](#)
- LSREAD\_NO\_RESTART\_1
  - LowSpeedNoRestartConstants, [800](#)
  - NBCCCommon.h, [1273](#)
- LSREAD\_NO\_RESTART\_2
  - LowSpeedNoRestartConstants, [800](#)
  - NBCCCommon.h, [1273](#)
- LSREAD\_NO\_RESTART\_3
  - LowSpeedNoRestartConstants, [801](#)
  - NBCCCommon.h, [1273](#)
- LSREAD\_NO\_RESTART\_4
  - LowSpeedNoRestartConstants, [801](#)
  - NBCCCommon.h, [1274](#)
- LSREAD\_NO\_RESTART\_MASK
  - LowSpeedNoRestartConstants, [801](#)
  - NBCCCommon.h, [1274](#)
- LSREAD\_RESTART\_ALL
  - LowSpeedNoRestartConstants, [801](#)
  - NBCCCommon.h, [1274](#)
- LSREAD\_RESTART\_NONE
  - LowSpeedNoRestartConstants, [801](#)
  - NBCCCommon.h, [1274](#)
- LSSpeed
  - LowLevelLowSpeedModuleFunc-  
tions, [409](#)
  - NXCDefs.h, [1655](#)
- LSState
  - LowLevelLowSpeedModuleFunc-  
tions, [410](#)
  - NXCDefs.h, [1655](#)
- LSState constants, [794](#)
- LT
  - cmpconst, [449](#)
  - NXCDefs.h, [1488](#)
- LTEQ
  - cmpconst, [449](#)
  - NXCDefs.h, [1488](#)
- Mailbox constants, [703](#)
- MAILBOX1
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1274](#)
- MAILBOX10
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1274](#)
- MAILBOX2
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1274](#)
- MAILBOX3
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1274](#)
- MAILBOX4
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1275](#)
- MAILBOX5
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1275](#)
- MAILBOX6
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1275](#)
- MAILBOX7
  - MailboxConstants, [704](#)
  - NBCCCommon.h, [1275](#)
- MAILBOX8
  - MailboxConstants, [705](#)
  - NBCCCommon.h, [1275](#)
- MAILBOX9
  - MailboxConstants, [705](#)
  - NBCCCommon.h, [1275](#)
- MailboxConstants
  - MAILBOX1, [704](#)
  - MAILBOX10, [704](#)

- MAILBOX2, [704](#)
- MAILBOX3, [704](#)
- MAILBOX4, [704](#)
- MAILBOX5, [704](#)
- MAILBOX6, [704](#)
- MAILBOX7, [704](#)
- MAILBOX8, [705](#)
- MAILBOX9, [705](#)
- MAX\_BT\_MSG\_SIZE
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1275](#)
- MaxAccelerationField
  - NBCCCommon.h, [1275](#)
  - OutputFieldConstants, [785](#)
- MaxSpeedField
  - NBCCCommon.h, [1275](#)
  - OutputFieldConstants, [785](#)
- MaxVal
  - UpdateCalibCacheInfoType, [1110](#)
- memcmp
  - cstringAPI, [662](#)
  - NXCDefs.h, [1656](#)
- memcpy
  - cstringAPI, [662](#)
  - NXCDefs.h, [1656](#)
- memmove
  - cstringAPI, [663](#)
  - NXCDefs.h, [1656](#)
- MemoryManager
  - NBCCCommon.h, [1276](#)
  - SysCallConstants, [689](#)
- MemoryManagerType, [1090](#)
  - Compact, [1090](#)
  - DataspaceSize, [1090](#)
  - PoolSize, [1090](#)
  - Result, [1091](#)
- MENUICON\_CENTER
  - DisplayModuleConstants, [813](#)
  - NBCCCommon.h, [1276](#)
- MENUICON\_LEFT
  - DisplayModuleConstants, [814](#)
  - NBCCCommon.h, [1276](#)
- MENUICON\_RIGHT
  - DisplayModuleConstants, [814](#)
  - NBCCCommon.h, [1276](#)
- MENUICONS
  - DisplayModuleConstants, [814](#)
  - NBCCCommon.h, [1276](#)
- MENUTEXT
  - DisplayModuleConstants, [814](#)
  - NBCCCommon.h, [1276](#)
- Message
  - DatalogWriteType, [1039](#)
  - MessageReadType, [1092](#)
  - MessageWriteType, [1093](#)
- MessageRead
  - NBCCCommon.h, [1276](#)
  - SysCallConstants, [689](#)
- MessageReadType, [1091](#)
  - Message, [1092](#)
  - QueueID, [1092](#)
  - Remove, [1092](#)
  - Result, [1092](#)
- MessageWrite
  - NBCCCommon.h, [1276](#)
  - SysCallConstants, [690](#)
- MessageWriteType, [1092](#)
  - Message, [1093](#)
  - QueueID, [1093](#)
  - Result, [1093](#)
- MI\_ADDR\_XG1300L
  - NBCCCommon.h, [1276](#)
  - XG1300LConstants, [1008](#)
- Microinfinity API Functions, [247](#)
- Microinfinity CruizCore XG1300L, [1009](#)
- Microinfinity CruizCore XG1300L sensor constants, [1008](#)
- Microinfinity device constants, [1007](#)
- Microinfinity functions, [596](#)
- Microinfinity types, [596](#)
- MicroinfinityFunctions
  - ReadSensorMIXG1300L, [597](#)
  - ResetMIXG1300L, [597](#)
  - SensorMIXG1300LScale, [598](#)
  - SetSensorMIXG1300LScale, [599](#)
- MidStr
  - cstringAPI, [663](#)
  - NXCDefs.h, [1657](#)
- MIN\_1
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [696](#)
- MindSensors ACCL-Nx constants, [948](#)

- MindSensors ACCL-Nx sensitivity level constants, [951](#)
- MindSensors API Functions, [127](#)
- MindSensors device constants, [934](#)
- MindSensors DIST-Nx constants, [939](#)
- MindSensors nRLink constants, [946](#)
- MindSensors NXTHID commands, [967](#)
- MindSensors NXTHID constants, [964](#)
- MindSensors NXTHID modifier keys, [966](#)
- MindSensors NXTHID registers, [965](#)
- MindSensors NXTLineLeader commands, [975](#)
- MindSensors NXTLineLeader constants, [972](#)
- MindSensors NXTLineLeader registers, [973](#)
- MindSensors NXTPowerMeter commands, [971](#)
- MindSensors NXTPowerMeter constants, [968](#)
- MindSensors NXTPowerMeter registers, [968](#)
- MindSensors NXTServo commands, [963](#)
- MindSensors NXTServo constants, [955](#)
- MindSensors NXTServo position constants, [960](#)
- MindSensors NXTServo quick position constants, [961](#)
- MindSensors NXTServo registers, [956](#)
- MindSensors NXTServo servo numbers, [961](#)
- MindSensors NXTSumoEyes constants, [971](#)
- MindSensors PFMate constants, [952](#)
- MindSensors PSP-Nx button set 1 constants, [942](#)
- MindSensors PSP-Nx button set 2 constants, [944](#)
- MindSensors PSP-Nx constants, [941](#)
- MindSensorsAPI
  - ACCLNxCalibrateX, [143](#)
  - ACCLNxCalibrateXEnd, [144](#)
  - ACCLNxCalibrateY, [144](#)
  - ACCLNxCalibrateYEnd, [145](#)
  - ACCLNxCalibrateZ, [145](#)
  - ACCLNxCalibrateZEnd, [146](#)
  - ACCLNxResetCalibration, [146](#)
  - ACCLNxSensitivity, [146](#)
  - ACCLNxXOffset, [147](#)
  - ACCLNxXRange, [147](#)
  - ACCLNxYOffset, [148](#)
  - ACCLNxYRange, [148](#)
  - ACCLNxZOffset, [149](#)
  - ACCLNxZRange, [149](#)
  - DISTNxDistance, [149](#)
  - DISTNxGP2D12, [150](#)
  - DISTNxGP2D120, [150](#)
  - DISTNxGP2YA02, [151](#)
  - DISTNxGP2YA21, [151](#)
  - DISTNxMaxDistance, [152](#)
  - DISTNxMinDistance, [152](#)
  - DISTNxModuleType, [153](#)
  - DISTNxNumPoints, [153](#)
  - DISTNxVoltage, [153](#)
  - MSADPAOff, [154](#)
  - MSADPAOn, [154](#)
  - MSDeenergize, [155](#)
  - MSnergize, [155](#)
  - MSIRTrain, [156](#)
  - MSPFComboDirect, [156](#)
  - MSPFComboPWM, [157](#)
  - MSPFRawOutput, [157](#)
  - MSPFRepeat, [158](#)
  - MSPFSingleOutputCST, [159](#)
  - MSPFSingleOutputPWM, [159](#)
  - MSPFSinglePin, [160](#)
  - MSPFTrain, [161](#)
  - MSRCXAbsVar, [161](#)
  - MSRCXAddToDatalog, [162](#)
  - MSRCXAndVar, [162](#)
  - MSRCXBatteryLevel, [163](#)
  - MSRCXBoot, [163](#)
  - MSRCXCalibrateEvent, [163](#)
  - MSRCXClearAllEvents, [163](#)
  - MSRCXClearCounter, [164](#)
  - MSRCXClearMsg, [164](#)
  - MSRCXClearSensor, [164](#)
  - MSRCXClearSound, [164](#)
  - MSRCXClearTimer, [165](#)
  - MSRCXCreateDatalog, [165](#)
  - MSRCXDecCounter, [165](#)

- MSRCXDeleteSub, 166
- MSRCXDeleteSubs, 166
- MSRCXDeleteTask, 166
- MSRCXDeleteTasks, 166
- MSRCXDisableOutput, 167
- MSRCXDivVar, 167
- MSRCXEnableOutput, 167
- MSRCXEvent, 168
- MSRCXFloat, 168
- MSRCXFwd, 168
- MSRCXIncCounter, 169
- MSRCXInvertOutput, 169
- MSRCXMulVar, 169
- MSRCXMuteSound, 170
- MSRCXObvertOutput, 170
- MSRCXOff, 170
- MSRCXOn, 170
- MSRCXOnFor, 171
- MSRCXOnFwd, 171
- MSRCXOnRev, 171
- MSRCXOrVar, 172
- MSRCXPBTurnOff, 172
- MSRCXPing, 172
- MSRCXPlaySound, 173
- MSRCXPlayTone, 173
- MSRCXPlayToneVar, 173
- MSRCXPoll, 174
- MSRCXPollMemory, 174
- MSRCXRemote, 174
- MSRCXReset, 175
- MSRCXRev, 175
- MSRCXSelectDisplay, 175
- MSRCXSelectProgram, 176
- MSRCXSendSerial, 176
- MSRCXSet, 176
- MSRCXSetDirection, 177
- MSRCXSetEvent, 177
- MSRCXSetGlobalDirection, 177
- MSRCXSetGlobalOutput, 178
- MSRCXSetMaxPower, 178
- MSRCXSetMessage, 179
- MSRCXSetNRLinkPort, 179
- MSRCXSetOutput, 179
- MSRCXSetPower, 180
- MSRCXSetPriority, 180
- MSRCXSetSensorMode, 180
- MSRCXSetSensorType, 181
- MSRCXSetSleepTime, 181
- MSRCXSetTxPower, 181
- MSRCXSetUserDisplay, 182
- MSRCXSetVar, 182
- MSRCXSetWatch, 182
- MSRCXSgnVar, 183
- MSRCXStartTask, 183
- MSRCXStopAllTasks, 183
- MSRCXStopTask, 184
- MSRCXSubVar, 184
- MSRCXSumVar, 184
- MSRCXToggle, 185
- MSRCXUnlock, 185
- MSRCXUnmuteSound, 185
- MSScoutCalibrateSensor, 186
- MSScoutMuteSound, 186
- MSScoutSelectSounds, 186
- MSScoutSendVLL, 187
- MSScoutSetCounterLimit, 187
- MSScoutSetEventFeedback, 187
- MSScoutSetLight, 188
- MSScoutSetScoutMode, 188
- MSScoutSetScoutRules, 188
- MSScoutSetSensorClickTime, 189
- MSScoutSetSensorHysteresis, 189
- MSScoutSetSensorLowerLimit, 190
- MSScoutSetSensorUpperLimit, 190
- MSScoutSetTimerLimit, 190
- MSScoutUnmuteSound, 191
- NRLink2400, 191
- NRLink4800, 191
- NRLinkFlush, 192
- NRLinkIRLong, 192
- NRLinkIRShort, 193
- NRLinkSetPF, 193
- NRLinkSetRCX, 194
- NRLinkSetTrain, 194
- NRLinkStatus, 194
- NRLinkTxRaw, 195
- NXTHIDAsciiMode, 195
- NXTHIDDirectMode, 196
- NXTHIDLoadCharacter, 196
- NXTHIDTransmit, 197
- NXTLineLeaderAverage, 197

- NXTLineLeaderCalibrateBlack, 198
- NXTLineLeaderCalibrateWhite, 198
- NXTLineLeaderInvert, 199
- NXTLineLeaderPowerDown, 199
- NXTLineLeaderPowerUp, 200
- NXTLineLeaderReset, 200
- NXTLineLeaderResult, 201
- NXTLineLeaderSnapshot, 201
- NXTLineLeaderSteering, 202
- NXTPowerMeterCapacityUsed, 202
- NXTPowerMeterElapsedTime, 203
- NXTPowerMeterErrorCount, 203
- NXTPowerMeterMaxCurrent, 204
- NXTPowerMeterMaxVoltage, 204
- NXTPowerMeterMinCurrent, 205
- NXTPowerMeterMinVoltage, 205
- NXTPowerMeterPresentCurrent, 206
- NXTPowerMeterPresentPower, 206
- NXTPowerMeterPresentVoltage, 206
- NXTPowerMeterResetCounters, 207
- NXTPowerMeterTotalPowerConsumed, 207
- NXTServoBatteryVoltage, 208
- NXTServoEditMacro, 208
- NXTServoGotoMacroAddress, 209
- NXTServoHaltMacro, 209
- NXTServoInit, 210
- NXTServoPauseMacro, 210
- NXTServoPosition, 211
- NXTServoQuitEdit, 211
- NXTServoReset, 212
- NXTServoResumeMacro, 212
- NXTServoSpeed, 213
- PFMateSend, 213
- PFMateSendRaw, 214
- PSPNxAnalog, 215
- PSPNxDigital, 215
- ReadNRLinkBytes, 216
- ReadSensorMSAccel, 216
- ReadSensorMSPlayStation, 217
- ReadSensorMSRTClock, 217
- ReadSensorMSTilt, 218
- RunNRLinkMacro, 219
- SensorMSCompass, 219
- SensorMSDROD, 220
- SensorMSPressure, 220
- SensorMSPressureRaw, 220
- SensorNXTSumoEyes, 221
- SensorNXTSumoEyesRaw, 221
- SetACCLNxSensitivity, 222
- SetNXTLineLeaderKdFactor, 222
- SetNXTLineLeaderKdValue, 223
- SetNXTLineLeaderKiFactor, 223
- SetNXTLineLeaderKiValue, 224
- SetNXTLineLeaderKpFactor, 224
- SetNXTLineLeaderKpValue, 225
- SetNXTLineLeaderSetpoint, 226
- SetNXTServoPosition, 226
- SetNXTServoQuickPosition, 227
- SetNXTServoSpeed, 227
- SetSensorMSDROD, 228
- SetSensorMSPressure, 228
- SetSensorNXTSumoEyes, 228
- WriteNRLinkBytes, 229
- MindSensorsConstants
  - MS\_ADDR\_ACCLNX, 935
  - MS\_ADDR\_CMPSNX, 936
  - MS\_ADDR\_DISTNX, 936
  - MS\_ADDR\_IVSENS, 936
  - MS\_ADDR\_LINELDR, 936
  - MS\_ADDR\_MTRMUX, 937
  - MS\_ADDR\_NRLINK, 937
  - MS\_ADDR\_NXTCAM, 937
  - MS\_ADDR\_NXTHID, 937
  - MS\_ADDR\_NXTMMX, 937
  - MS\_ADDR\_NXTSERVO, 937
  - MS\_ADDR\_NXTSERVO\_EM, 937
  - MS\_ADDR\_PFMATE, 938
  - MS\_ADDR\_PSPNX, 938
  - MS\_ADDR\_RTCLOCK, 938
  - MS\_ADDR\_RXMUX, 938
  - MS\_CMD\_ADPA\_OFF, 938
  - MS\_CMD\_ADPA\_ON, 938
  - MS\_CMD\_DEENERGIZED, 938
  - MS\_CMD\_ENERGIZED, 938
- MinVal
  - UpdateCalibCacheInfoType, 1110
- MiscConstants
  - DEGREES\_PER\_RADIAN, 261

- FALSE, [261](#)
- NA, [261](#)
- PI, [261](#)
- RADIANS\_PER\_DEGREE, [262](#)
- TRUE, [262](#)
- Miscellaneous Comm module constants, [831](#)
- Miscellaneous NBC/NXC constants, [260](#)
- Mode
  - CommHSControlType, [1028](#)
  - OutputStateType, [1094](#)
- ModuleID
  - IOMapReadByIDType, [1075](#)
  - IOMapWriteByIDType, [1078](#)
- ModuleIDConstants
  - ButtonModuleID, [259](#)
  - CommandModuleID, [259](#)
  - CommModuleID, [259](#)
  - DisplayModuleID, [259](#)
  - InputModuleID, [259](#)
  - IOCtrlModuleID, [259](#)
  - LoaderModuleID, [260](#)
  - LowSpeedModuleID, [260](#)
  - OutputModuleID, [260](#)
  - SoundModuleID, [260](#)
  - UIModuleID, [260](#)
- ModuleName
  - IOMapReadType, [1077](#)
  - IOMapWriteType, [1079](#)
- ModuleNameConstants
  - ButtonModuleName, [257](#)
  - CommandModuleName, [257](#)
  - CommModuleName, [257](#)
  - DisplayModuleName, [257](#)
  - InputModuleName, [257](#)
  - IOCtrlModuleName, [258](#)
  - LoaderModuleName, [258](#)
  - LowSpeedModuleName, [258](#)
  - OutputModuleName, [258](#)
  - SoundModuleName, [258](#)
  - UIModuleName, [258](#)
- MotorActualSpeed
  - NXCDefs.h, [1657](#)
  - OutputModuleFunctions, [319](#)
- MotorBlockTachoCount
  - NXCDefs.h, [1658](#)
  - OutputModuleFunctions, [320](#)
- MotorMaxAcceleration
  - NXCDefs.h, [1658](#)
  - OutputModuleFunctions, [320](#)
- MotorMaxSpeed
  - NXCDefs.h, [1659](#)
  - OutputModuleFunctions, [321](#)
- MotorMode
  - NXCDefs.h, [1659](#)
  - OutputModuleFunctions, [321](#)
- MotorOutputOptions
  - NXCDefs.h, [1659](#)
  - OutputModuleFunctions, [322](#)
- MotorOverload
  - NXCDefs.h, [1660](#)
  - OutputModuleFunctions, [322](#)
- MotorPower
  - NXCDefs.h, [1660](#)
  - OutputModuleFunctions, [322](#)
- MotorPwnFreq
  - NXCDefs.h, [1661](#)
  - OutputModuleFunctions, [323](#)
- MotorRegDValue
  - NXCDefs.h, [1661](#)
  - OutputModuleFunctions, [323](#)
- MotorRegIValue
  - NXCDefs.h, [1661](#)
  - OutputModuleFunctions, [324](#)
- MotorRegPValue
  - NXCDefs.h, [1662](#)
  - OutputModuleFunctions, [324](#)
- MotorRegulation
  - NXCDefs.h, [1662](#)
  - OutputModuleFunctions, [324](#)
- MotorRegulationOptions
  - NXCDefs.h, [1663](#)
  - OutputModuleFunctions, [325](#)
- MotorRegulationTime
  - NXCDefs.h, [1663](#)
  - OutputModuleFunctions, [325](#)
- MotorRotationCount
  - NXCDefs.h, [1663](#)
  - OutputModuleFunctions, [326](#)
- MotorRunState
  - NXCDefs.h, [1664](#)
  - OutputModuleFunctions, [326](#)

- MotorTachoCount
  - NXCDefs.h, [1664](#)
  - OutputModuleFunctions, [326](#)
- MotorTachoLimit
  - NXCDefs.h, [1665](#)
  - OutputModuleFunctions, [327](#)
- MotorTurnRatio
  - NXCDefs.h, [1665](#)
  - OutputModuleFunctions, [327](#)
- MS\_1
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [696](#)
- MS\_10
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [696](#)
- MS\_100
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [697](#)
- MS\_150
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [697](#)
- MS\_2
  - NBCCCommon.h, [1277](#)
  - TimeConstants, [697](#)
- MS\_20
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [697](#)
- MS\_200
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [697](#)
- MS\_250
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [697](#)
- MS\_3
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [698](#)
- MS\_30
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [698](#)
- MS\_300
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [698](#)
- MS\_350
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [698](#)
- MS\_4
  - NBCCCommon.h, [1278](#)
  - TimeConstants, [698](#)
- MS\_40
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [698](#)
- MS\_400
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [698](#)
- MS\_450
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [698](#)
- MS\_5
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [698](#)
- MS\_50
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [699](#)
- MS\_500
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [699](#)
- MS\_6
  - NBCCCommon.h, [1279](#)
  - TimeConstants, [699](#)
- MS\_60
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [699](#)
- MS\_600
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [699](#)
- MS\_7
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [699](#)
- MS\_70
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [699](#)
- MS\_700
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [699](#)
- MS\_8
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [700](#)
- MS\_80
  - NBCCCommon.h, [1280](#)
  - TimeConstants, [700](#)
- MS\_800
  - NBCCCommon.h, [1280](#)

- TimeConstants, 700
- MS\_9
  - NBCCCommon.h, 1280
  - TimeConstants, 700
- MS\_90
  - NBCCCommon.h, 1280
  - TimeConstants, 700
- MS\_900
  - NBCCCommon.h, 1281
  - TimeConstants, 700
- MS\_ADDR\_ACCLNX
  - MindSensorsConstants, 935
  - NBCCCommon.h, 1281
- MS\_ADDR\_CMPSNX
  - MindSensorsConstants, 936
  - NBCCCommon.h, 1281
- MS\_ADDR\_DISTNX
  - MindSensorsConstants, 936
  - NBCCCommon.h, 1281
- MS\_ADDR\_IVSENS
  - MindSensorsConstants, 936
  - NBCCCommon.h, 1281
- MS\_ADDR\_LINELDR
  - MindSensorsConstants, 936
  - NBCCCommon.h, 1282
- MS\_ADDR\_MTRMUX
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1282
- MS\_ADDR\_NRLINK
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1282
- MS\_ADDR\_NXTCAM
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1282
- MS\_ADDR\_NXTHID
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1282
- MS\_ADDR\_NXTMMX
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1282
- MS\_ADDR\_NXTSERVO
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1283
- MS\_ADDR\_NXTSERVO\_EM
  - MindSensorsConstants, 937
  - NBCCCommon.h, 1283
- MS\_ADDR\_PFMATE
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1283
- MS\_ADDR\_PSPNX
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1283
- MS\_ADDR\_RTCLOCK
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1283
- MS\_ADDR\_RXMUX
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1283
- MS\_CMD\_ADPA\_OFF
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1283
- MS\_CMD\_ADPA\_ON
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1284
- MS\_CMD\_DEENERGIZED
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1284
- MS\_CMD\_ENERGIZED
  - MindSensorsConstants, 938
  - NBCCCommon.h, 1284
- MSACCLNx
  - ACCL\_CMD\_RESET\_CAL, 949
  - ACCL\_CMD\_X\_CAL, 949
  - ACCL\_CMD\_X\_CAL\_END, 949
  - ACCL\_CMD\_Y\_CAL, 949
  - ACCL\_CMD\_Y\_CAL\_END, 949
  - ACCL\_CMD\_Z\_CAL, 949
  - ACCL\_CMD\_Z\_CAL\_END, 949
  - ACCL\_REG\_SENS\_LVL, 950
  - ACCL\_REG\_X\_ACCEL, 950
  - ACCL\_REG\_X\_OFFSET, 950
  - ACCL\_REG\_X\_RANGE, 950
  - ACCL\_REG\_X\_TILT, 950
  - ACCL\_REG\_Y\_ACCEL, 950
  - ACCL\_REG\_Y\_OFFSET, 950
  - ACCL\_REG\_Y\_RANGE, 950
  - ACCL\_REG\_Y\_TILT, 950
  - ACCL\_REG\_Z\_ACCEL, 950
  - ACCL\_REG\_Z\_OFFSET, 951
  - ACCL\_REG\_Z\_RANGE, 951
  - ACCL\_REG\_Z\_TILT, 951
- MSACCLNxSLevel

- ACCL\_SENSITIVITY\_LEVEL\_1, [951](#)
- ACCL\_SENSITIVITY\_LEVEL\_2, [951](#)
- ACCL\_SENSITIVITY\_LEVEL\_3, [952](#)
- ACCL\_SENSITIVITY\_LEVEL\_4, [952](#)
- MSADPAOff
  - MindSensorsAPI, [154](#)
  - NXCDefs.h, [1665](#)
- MSADPAOn
  - MindSensorsAPI, [154](#)
  - NXCDefs.h, [1666](#)
- MSDeenergize
  - MindSensorsAPI, [155](#)
  - NXCDefs.h, [1666](#)
- MSDistNX
  - DIST\_CMD\_CUSTOM, [939](#)
  - DIST\_CMD\_GP2D12, [939](#)
  - DIST\_CMD\_GP2D120, [939](#)
  - DIST\_CMD\_GP2YA02, [939](#)
  - DIST\_CMD\_GP2YA21, [940](#)
  - DIST\_REG\_DIST, [940](#)
  - DIST\_REG\_DIST1, [940](#)
  - DIST\_REG\_DIST\_MAX, [940](#)
  - DIST\_REG\_DIST\_MIN, [940](#)
  - DIST\_REG\_MODULE\_TYPE, [940](#)
  - DIST\_REG\_NUM\_POINTS, [940](#)
  - DIST\_REG\_VOLT, [940](#)
  - DIST\_REG\_VOLT1, [940](#)
- MSEnergize
  - MindSensorsAPI, [155](#)
  - NXCDefs.h, [1667](#)
- MSIRTrain
  - MindSensorsAPI, [156](#)
  - NXCDefs.h, [1667](#)
- MSNRLink
  - NRLINK\_CMD\_2400, [947](#)
  - NRLINK\_CMD\_4800, [947](#)
  - NRLINK\_CMD\_FLUSH, [947](#)
  - NRLINK\_CMD\_IR\_LONG, [947](#)
  - NRLINK\_CMD\_IR\_SHORT, [947](#)
  - NRLINK\_CMD\_RUN\_MACRO, [947](#)
  - NRLINK\_CMD\_SET\_PF, [947](#)
  - NRLINK\_CMD\_SET\_RCX, [947](#)
  - NRLINK\_CMD\_SET\_TRAIN, [947](#)
  - NRLINK\_CMD\_TX\_RAW, [947](#)
  - NRLINK\_REG\_BYTES, [948](#)
  - NRLINK\_REG\_DATA, [948](#)
  - NRLINK\_REG\_EEPROM, [948](#)
- MSPFComboDirect
  - MindSensorsAPI, [156](#)
  - NXCDefs.h, [1668](#)
- MSPFComboPWM
  - MindSensorsAPI, [157](#)
  - NXCDefs.h, [1668](#)
- MSPFRawOutput
  - MindSensorsAPI, [157](#)
  - NXCDefs.h, [1669](#)
- MSPFRepeat
  - MindSensorsAPI, [158](#)
  - NXCDefs.h, [1670](#)
- MSPFSingleOutputCST
  - MindSensorsAPI, [159](#)
  - NXCDefs.h, [1670](#)
- MSPFSingleOutputPWM
  - MindSensorsAPI, [159](#)
  - NXCDefs.h, [1671](#)
- MSPFSinglePin
  - MindSensorsAPI, [160](#)
  - NXCDefs.h, [1672](#)
- MSPFTrain
  - MindSensorsAPI, [161](#)
  - NXCDefs.h, [1672](#)
- MSPSPNX
  - PSP\_CMD\_ANALOG, [941](#)
  - PSP\_CMD\_DIGITAL, [941](#)
  - PSP\_REG\_BTNSET1, [941](#)
  - PSP\_REG\_BTNSET2, [941](#)
  - PSP\_REG\_XLEFT, [942](#)
  - PSP\_REG\_XRIGHT, [942](#)
  - PSP\_REG\_YLEFT, [942](#)
  - PSP\_REG\_YRIGHT, [942](#)
- MSPSPNXBtnSet1
  - PSP\_BTNSET1\_DOWN, [943](#)
  - PSP\_BTNSET1\_L3, [943](#)
  - PSP\_BTNSET1\_LEFT, [943](#)
  - PSP\_BTNSET1\_R3, [943](#)
  - PSP\_BTNSET1\_RIGHT, [943](#)
  - PSP\_BTNSET1\_SELECT, [943](#)

- PSP\_BTNSET1\_START, [944](#)
- PSP\_BTNSET1\_UP, [944](#)
- MSPSPNXBtnSet2
  - PSP\_BTNSET2\_CIRCLE, [944](#)
  - PSP\_BTNSET2\_CROSS, [945](#)
  - PSP\_BTNSET2\_L1, [945](#)
  - PSP\_BTNSET2\_L2, [945](#)
  - PSP\_BTNSET2\_R1, [945](#)
  - PSP\_BTNSET2\_R2, [945](#)
  - PSP\_BTNSET2\_SQUARE, [945](#)
  - PSP\_BTNSET2\_TRIANGLE, [946](#)
- MSRCXAbsVar
  - MindSensorsAPI, [161](#)
  - NXCDefs.h, [1673](#)
- MSRCXAddToDatalog
  - MindSensorsAPI, [162](#)
  - NXCDefs.h, [1673](#)
- MSRCXAndVar
  - MindSensorsAPI, [162](#)
  - NXCDefs.h, [1674](#)
- MSRCXBatteryLevel
  - MindSensorsAPI, [163](#)
  - NXCDefs.h, [1674](#)
- MSRCXBoot
  - MindSensorsAPI, [163](#)
  - NXCDefs.h, [1674](#)
- MSRCXCalibrateEvent
  - MindSensorsAPI, [163](#)
  - NXCDefs.h, [1675](#)
- MSRCXClearAllEvents
  - MindSensorsAPI, [163](#)
  - NXCDefs.h, [1675](#)
- MSRCXClearCounter
  - MindSensorsAPI, [164](#)
  - NXCDefs.h, [1675](#)
- MSRCXClearMsg
  - MindSensorsAPI, [164](#)
  - NXCDefs.h, [1676](#)
- MSRCXClearSensor
  - MindSensorsAPI, [164](#)
  - NXCDefs.h, [1676](#)
- MSRCXClearSound
  - MindSensorsAPI, [164](#)
  - NXCDefs.h, [1676](#)
- MSRCXClearTimer
  - MindSensorsAPI, [165](#)
- NXCDefs.h, [1676](#)
- MSRCXCreateDatalog
  - MindSensorsAPI, [165](#)
  - NXCDefs.h, [1677](#)
- MSRCXDecCounter
  - MindSensorsAPI, [165](#)
  - NXCDefs.h, [1677](#)
- MSRCXDeleteSub
  - MindSensorsAPI, [166](#)
  - NXCDefs.h, [1677](#)
- MSRCXDeleteSubs
  - MindSensorsAPI, [166](#)
  - NXCDefs.h, [1678](#)
- MSRCXDeleteTask
  - MindSensorsAPI, [166](#)
  - NXCDefs.h, [1678](#)
- MSRCXDeleteTasks
  - MindSensorsAPI, [166](#)
  - NXCDefs.h, [1678](#)
- MSRCXDisableOutput
  - MindSensorsAPI, [167](#)
  - NXCDefs.h, [1678](#)
- MSRCXDivVar
  - MindSensorsAPI, [167](#)
  - NXCDefs.h, [1679](#)
- MSRCXEnableOutput
  - MindSensorsAPI, [167](#)
  - NXCDefs.h, [1679](#)
- MSRCXEvent
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1679](#)
- MSRCXFloat
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1680](#)
- MSRCXFwd
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1680](#)
- MSRCXIncCounter
  - MindSensorsAPI, [169](#)
  - NXCDefs.h, [1680](#)
- MSRCXInvertOutput
  - MindSensorsAPI, [169](#)
  - NXCDefs.h, [1681](#)
- MSRCXMulVar
  - MindSensorsAPI, [169](#)
  - NXCDefs.h, [1681](#)

- MSRCXMuteSound
  - MindSensorsAPI, 170
  - NXCDefs.h, 1681
- MSRCXObvertOutput
  - MindSensorsAPI, 170
  - NXCDefs.h, 1682
- MSRCXOff
  - MindSensorsAPI, 170
  - NXCDefs.h, 1682
- MSRCXOn
  - MindSensorsAPI, 170
  - NXCDefs.h, 1682
- MSRCXOnFor
  - MindSensorsAPI, 171
  - NXCDefs.h, 1683
- MSRCXOnFwd
  - MindSensorsAPI, 171
  - NXCDefs.h, 1683
- MSRCXOnRev
  - MindSensorsAPI, 171
  - NXCDefs.h, 1683
- MSRCXOrVar
  - MindSensorsAPI, 172
  - NXCDefs.h, 1684
- MSRCXPBTurnOff
  - MindSensorsAPI, 172
  - NXCDefs.h, 1684
- MSRCXPing
  - MindSensorsAPI, 172
  - NXCDefs.h, 1684
- MSRCXPlaySound
  - MindSensorsAPI, 173
  - NXCDefs.h, 1684
- MSRCXPlayTone
  - MindSensorsAPI, 173
  - NXCDefs.h, 1685
- MSRCXPlayToneVar
  - MindSensorsAPI, 173
  - NXCDefs.h, 1685
- MSRCXPoll
  - MindSensorsAPI, 174
  - NXCDefs.h, 1685
- MSRCXPollMemory
  - MindSensorsAPI, 174
  - NXCDefs.h, 1686
- MSRCXRemote
  - MindSensorsAPI, 174
  - NXCDefs.h, 1686
- MSRCXReset
  - MindSensorsAPI, 175
  - NXCDefs.h, 1687
- MSRCXRev
  - MindSensorsAPI, 175
  - NXCDefs.h, 1687
- MSRCXSelectDisplay
  - MindSensorsAPI, 175
  - NXCDefs.h, 1687
- MSRCXSelectProgram
  - MindSensorsAPI, 176
  - NXCDefs.h, 1687
- MSRCXSendSerial
  - MindSensorsAPI, 176
  - NXCDefs.h, 1688
- MSRCXSet
  - MindSensorsAPI, 176
  - NXCDefs.h, 1688
- MSRCXSetDirection
  - MindSensorsAPI, 177
  - NXCDefs.h, 1688
- MSRCXSetEvent
  - MindSensorsAPI, 177
  - NXCDefs.h, 1689
- MSRCXSetGlobalDirection
  - MindSensorsAPI, 177
  - NXCDefs.h, 1689
- MSRCXSetGlobalOutput
  - MindSensorsAPI, 178
  - NXCDefs.h, 1690
- MSRCXSetMaxPower
  - MindSensorsAPI, 178
  - NXCDefs.h, 1690
- MSRCXSetMessage
  - MindSensorsAPI, 179
  - NXCDefs.h, 1690
- MSRCXSetNRLinkPort
  - MindSensorsAPI, 179
  - NXCDefs.h, 1691
- MSRCXSetOutput
  - MindSensorsAPI, 179
  - NXCDefs.h, 1691
- MSRCXSetPower
  - MindSensorsAPI, 180

- NXCDefs.h, 1691
- MSRCXSetPriority
  - MindSensorsAPI, 180
  - NXCDefs.h, 1692
- MSRCXSetSensorMode
  - MindSensorsAPI, 180
  - NXCDefs.h, 1692
- MSRCXSetSensorType
  - MindSensorsAPI, 181
  - NXCDefs.h, 1692
- MSRCXSetSleepTime
  - MindSensorsAPI, 181
  - NXCDefs.h, 1693
- MSRCXSetTxPower
  - MindSensorsAPI, 181
  - NXCDefs.h, 1693
- MSRCXSetUserDisplay
  - MindSensorsAPI, 182
  - NXCDefs.h, 1693
- MSRCXSetVar
  - MindSensorsAPI, 182
  - NXCDefs.h, 1694
- MSRCXSetWatch
  - MindSensorsAPI, 182
  - NXCDefs.h, 1694
- MSRCXSgnVar
  - MindSensorsAPI, 183
  - NXCDefs.h, 1695
- MSRCXStartTask
  - MindSensorsAPI, 183
  - NXCDefs.h, 1695
- MSRCXStopAllTasks
  - MindSensorsAPI, 183
  - NXCDefs.h, 1695
- MSRCXStopTask
  - MindSensorsAPI, 184
  - NXCDefs.h, 1695
- MSRCXSubVar
  - MindSensorsAPI, 184
  - NXCDefs.h, 1696
- MSRCXSumVar
  - MindSensorsAPI, 184
  - NXCDefs.h, 1696
- MSRCXToggle
  - MindSensorsAPI, 185
  - NXCDefs.h, 1696
- MSRCXUnlock
  - MindSensorsAPI, 185
  - NXCDefs.h, 1697
- MSRCXUnmuteSound
  - MindSensorsAPI, 185
  - NXCDefs.h, 1697
- MSReadValue
  - MindSensorsAPI, 185
  - NXCDefs.h, 1697
- MSScoutCalibrateSensor
  - MindSensorsAPI, 186
  - NXCDefs.h, 1698
- MSScoutMuteSound
  - MindSensorsAPI, 186
  - NXCDefs.h, 1698
- MSScoutSelectSounds
  - MindSensorsAPI, 186
  - NXCDefs.h, 1698
- MSScoutSendVLL
  - MindSensorsAPI, 187
  - NXCDefs.h, 1698
- MSScoutSetCounterLimit
  - MindSensorsAPI, 187
  - NXCDefs.h, 1699
- MSScoutSetEventFeedback
  - MindSensorsAPI, 187
  - NXCDefs.h, 1699
- MSScoutSetLight
  - MindSensorsAPI, 188
  - NXCDefs.h, 1699
- MSScoutSetScoutMode
  - MindSensorsAPI, 188
  - NXCDefs.h, 1700
- MSScoutSetScoutRules
  - MindSensorsAPI, 188
  - NXCDefs.h, 1700
- MSScoutSetSensorClickTime
  - MindSensorsAPI, 189
  - NXCDefs.h, 1700
- MSScoutSetSensorHysteresis
  - MindSensorsAPI, 189
  - NXCDefs.h, 1701
- MSScoutSetSensorLowerLimit
  - MindSensorsAPI, 190
  - NXCDefs.h, 1701
- MSScoutSetSensorUpperLimit

- MindSensorsAPI, [190](#)
- NXCDefs.h, [1702](#)
- MSScoutSetTimerLimit
  - MindSensorsAPI, [190](#)
  - NXCDefs.h, [1702](#)
- MSScoutUnmuteSound
  - MindSensorsAPI, [191](#)
  - NXCDefs.h, [1702](#)
- MulDiv32
  - cmathAPI, [612](#)
  - NXCDefs.h, [1488](#)
- muldiv32
  - cmathAPI, [626](#)
  - NXCDefs.h, [1702](#)
- NA
  - MiscConstants, [261](#)
  - NBCCCommon.h, [1284](#)
- Name
  - CommBTConnectionType, [1021](#)
  - CommExecuteFunctionType, [1026](#)
  - ComputeCalibValueType, [1036](#)
  - UpdateCalibCacheInfoType, [1111](#)
- NBC Input port constants, [756](#)
- NBC sensor mode constants, [759](#)
- NBC sensor type constants, [757](#)
- NBCCCommon.h, [1115](#)
  - ACCL\_CMD\_RESET\_CAL, [1163](#)
  - ACCL\_CMD\_X\_CAL, [1163](#)
  - ACCL\_CMD\_X\_CAL\_END, [1163](#)
  - ACCL\_CMD\_Y\_CAL, [1163](#)
  - ACCL\_CMD\_Y\_CAL\_END, [1163](#)
  - ACCL\_CMD\_Z\_CAL, [1163](#)
  - ACCL\_CMD\_Z\_CAL\_END, [1163](#)
  - ACCL\_REG\_SENS\_LVL, [1164](#)
  - ACCL\_REG\_X\_ACCEL, [1164](#)
  - ACCL\_REG\_X\_OFFSET, [1164](#)
  - ACCL\_REG\_X\_RANGE, [1164](#)
  - ACCL\_REG\_X\_TILT, [1164](#)
  - ACCL\_REG\_Y\_ACCEL, [1164](#)
  - ACCL\_REG\_Y\_OFFSET, [1164](#)
  - ACCL\_REG\_Y\_RANGE, [1164](#)
  - ACCL\_REG\_Y\_TILT, [1164](#)
  - ACCL\_REG\_Z\_ACCEL, [1164](#)
  - ACCL\_REG\_Z\_OFFSET, [1165](#)
  - ACCL\_REG\_Z\_RANGE, [1165](#)
- ACCL\_REG\_Z\_TILT, [1165](#)
- ACCL\_SENSITIVITY\_LEVEL\_1, [1165](#)
- ACCL\_SENSITIVITY\_LEVEL\_2, [1165](#)
- ACCL\_SENSITIVITY\_LEVEL\_3, [1165](#)
- ACCL\_SENSITIVITY\_LEVEL\_4, [1165](#)
- ActualSpeedField, [1165](#)
- BITMAP\_1, [1166](#)
- BITMAP\_2, [1166](#)
- BITMAP\_3, [1166](#)
- BITMAP\_4, [1166](#)
- BITMAPS, [1166](#)
- BlockTachoCountField, [1166](#)
- BREAKOUT\_REQ, [1166](#)
- BT\_ARM\_CMD\_MODE, [1166](#)
- BT\_ARM\_DATA\_MODE, [1167](#)
- BT\_ARM\_OFF, [1167](#)
- BT\_BRICK\_PORT\_OPEN, [1167](#)
- BT\_BRICK\_VISIBILITY, [1167](#)
- BT\_CMD\_BYTE, [1167](#)
- BT\_CMD\_READY, [1167](#)
- BT\_CONNECTION\_0\_ENABLE, [1167](#)
- BT\_CONNECTION\_1\_ENABLE, [1167](#)
- BT\_CONNECTION\_2\_ENABLE, [1167](#)
- BT\_CONNECTION\_3\_ENABLE, [1167](#)
- BT\_DEFAULT\_INQUIRY\_MAX, [1168](#)
- BT\_DEFAULT\_INQUIRY\_-  
TIMEOUT\_LO, [1168](#)
- BT\_DEVICE\_AWAY, [1168](#)
- BT\_DEVICE\_EMPTY, [1168](#)
- BT\_DEVICE\_KNOWN, [1168](#)
- BT\_DEVICE\_NAME, [1168](#)
- BT\_DEVICE\_UNKNOWN, [1168](#)
- BT\_DISABLE, [1168](#)
- BT\_ENABLE, [1168](#)
- BTN1, [1168](#)
- BTN2, [1169](#)
- BTN3, [1169](#)

- BTN4, 1169
- BTNCENTER, 1169
- BTNEXIT, 1169
- BTNLEFT, 1169
- BTNRIGHT, 1170
- BTNSTATE\_LONG\_PRESSED\_-  
EV, 1170
- BTNSTATE\_LONG\_RELEASED\_-  
EV, 1170
- BTNSTATE\_NONE, 1170
- BTNSTATE\_PRESSED\_EV, 1170
- BTNSTATE\_PRESSED\_STATE,  
1170
- BTNSTATE\_SHORT\_-  
RELEASED\_EV, 1171
- ButtonModuleID, 1171
- ButtonModuleName, 1171
- ButtonOffsetLongPressCnt, 1171
- ButtonOffsetLongRelCnt, 1171
- ButtonOffsetPressedCnt, 1171
- ButtonOffsetRelCnt, 1171
- ButtonOffsetShortRelCnt, 1171
- ButtonOffsetState, 1171
- CHAR\_BIT, 1171
- CHAR\_MAX, 1172
- CHAR\_MIN, 1172
- CLUMP\_DONE, 1172
- CLUMP\_SUSPEND, 1172
- ColorSensorRead, 1172
- COM\_CHANNEL\_FOUR\_-  
ACTIVE, 1172
- COM\_CHANNEL\_NONE\_-  
ACTIVE, 1172
- COM\_CHANNEL\_ONE\_ACTIVE,  
1172
- COM\_CHANNEL\_THREE\_-  
ACTIVE, 1172
- COM\_CHANNEL\_TWO\_ACTIVE,  
1172
- CommandModuleID, 1173
- CommandModuleName, 1173
- CommandOffsetActivateFlag, 1173
- CommandOffsetAwake, 1173
- CommandOffsetDeactivateFlag,  
1173
- CommandOffsetFileName, 1173
- CommandOffsetFormatString, 1173
- CommandOffsetMemoryPool, 1173
- CommandOffsetOffsetDS, 1174
- CommandOffsetOffsetDVA, 1174
- CommandOffsetPRCHandler, 1174
- CommandOffsetProgStatus, 1174
- CommandOffsetSyncTick, 1174
- CommandOffsetSyncTime, 1174
- CommandOffsetTick, 1174
- CommBTCheckStatus, 1175
- CommBTConnection, 1175
- CommBTONOff, 1175
- CommBTRead, 1175
- CommBTWrite, 1175
- CommExecuteFunction, 1175
- CommHSCheckStatus, 1175
- CommHSControl, 1175
- CommHSRead, 1175
- CommHSWrite, 1175
- CommLSCheckStatus, 1176
- CommLSRead, 1176
- CommLSWrite, 1176
- CommLSWriteEx, 1176
- CommModuleID, 1176
- CommModuleName, 1176
- CommOffsetBrickDataBdAddr,  
1176
- CommOffsetBrickDataBluecoreVer-  
sion, 1176
- CommOffsetBrickDataBtHwStatus,  
1176
- CommOffsetBrickDataBtStateSta-  
tus, 1176
- CommOffsetBrickDataName, 1177
- CommOffsetBrickDataTimeOut-  
Value, 1177
- CommOffsetBtConnectTableB-  
dAddr, 1177
- CommOffsetBtConnectTableClas-  
sOfDevice, 1177
- CommOffsetBtConnectTableHan-  
dleNr, 1177
- CommOffsetBtConnect-  
TableLinkQuality, 1177
- CommOffsetBtConnectTableName,  
1177

- CommOffsetBtConnectTablePinCode, 1177
- CommOffsetBtConnectTableStreamStatus, 1177
- CommOffsetBtDataMode, 1177
- CommOffsetBtDeviceCnt, 1178
- CommOffsetBtDeviceNameCnt, 1178
- CommOffsetBtDeviceTableBdAddr, 1178
- CommOffsetBtDeviceTableClassesOfDevice, 1178
- CommOffsetBtDeviceTableDeviceStatus, 1178
- CommOffsetBtDeviceTableName, 1178
- CommOffsetBtInBufBuf, 1178
- CommOffsetBtInBufInPtr, 1178
- CommOffsetBtInBufOutPtr, 1178
- CommOffsetBtOutBufBuf, 1178
- CommOffsetBtOutBufInPtr, 1179
- CommOffsetBtOutBufOutPtr, 1179
- CommOffsetHsAddress, 1179
- CommOffsetHsDataMode, 1179
- CommOffsetHsFlags, 1179
- CommOffsetHsInBufBuf, 1179
- CommOffsetHsInBufInPtr, 1179
- CommOffsetHsInBufOutPtr, 1179
- CommOffsetHsMode, 1179
- CommOffsetHsOutBufBuf, 1179
- CommOffsetHsOutBufInPtr, 1180
- CommOffsetHsOutBufOutPtr, 1180
- CommOffsetHsSpeed, 1180
- CommOffsetHsState, 1180
- CommOffsetPFunc, 1180
- CommOffsetPFuncTwo, 1180
- CommOffsetUsbInBufBuf, 1180
- CommOffsetUsbInBufInPtr, 1180
- CommOffsetUsbInBufOutPtr, 1180
- CommOffsetUsbOutBufBuf, 1180
- CommOffsetUsbOutBufInPtr, 1181
- CommOffsetUsbOutBufOutPtr, 1181
- CommOffsetUsbPollBufBuf, 1181
- CommOffsetUsbPollBufInPtr, 1181
- CommOffsetUsbPollBufOutPtr, 1181
- CommOffsetUsbState, 1181
- ComputeCalibValue, 1181
- CONN\_BT0, 1181
- CONN\_BT1, 1181
- CONN\_BT2, 1182
- CONN\_BT3, 1182
- CONN\_HS4, 1182
- CONN\_HS\_1, 1182
- CONN\_HS\_2, 1182
- CONN\_HS\_3, 1182
- CONN\_HS\_4, 1183
- CONN\_HS\_5, 1183
- CONN\_HS\_6, 1183
- CONN\_HS\_7, 1183
- CONN\_HS\_8, 1183
- CONN\_HS\_ALL, 1183
- CT\_ADDR\_RFID, 1183
- CT\_REG\_DATA, 1183
- CT\_REG\_MODE, 1183
- CT\_REG\_STATUS, 1183
- DAC\_MODE\_DCOUT, 1184
- DAC\_MODE\_PWMVOLTAGE, 1184
- DAC\_MODE\_SAWNEGWAVE, 1184
- DAC\_MODE\_SAWPOSWAVE, 1184
- DAC\_MODE\_SINEWAVE, 1184
- DAC\_MODE\_SQUAREWAVE, 1184
- DAC\_MODE\_TRIANGLEWAVE, 1184
- DATA\_MODE\_GPS, 1184
- DATA\_MODE\_MASK, 1185
- DATA\_MODE\_NXT, 1185
- DATA\_MODE\_RAW, 1185
- DATA\_MODE\_UPDATE, 1185
- DatalogGetTimes, 1185
- DatalogWrite, 1185
- DEGREES\_PER\_RADIAN, 1185
- DGPS\_REG\_DISTANCE, 1185
- DGPS\_REG\_HEADING, 1185
- DGPS\_REG\_LASTANGLE, 1186
- DGPS\_REG\_LATITUDE, 1186

- DGPS\_REG\_LONGITUDE, 1186  
DGPS\_REG\_SETLATITUDE, 1186  
DGPS\_REG\_SETLONGITUDE,  
1186  
DGPS\_REG\_STATUS, 1186  
DGPS\_REG\_TIME, 1186  
DGPS\_REG\_VELOCITY, 1186  
DGPS\_REG\_WAYANGLE, 1186  
DI\_ADDR\_ACCL, 1186  
DI\_ADDR\_DGPS, 1187  
DI\_ADDR\_GYRO, 1187  
DIACCL\_CTRL1\_FILT\_BW125,  
1187  
DIACCL\_CTRL1\_INT2TOINT1,  
1187  
DIACCL\_CTRL1\_LEVELPULSE,  
1187  
DIACCL\_CTRL1\_NO\_XDETECT,  
1187  
DIACCL\_CTRL1\_NO\_YDETECT,  
1187  
DIACCL\_CTRL1\_NO\_ZDETECT,  
1187  
DIACCL\_CTRL1\_PULSELEVEL,  
1187  
DIACCL\_CTRL1\_PULSEPULSE,  
1187  
DIACCL\_CTRL1\_THRESH\_INT,  
1188  
DIACCL\_CTRL2\_DETPOL\_-  
NEGAND, 1188  
DIACCL\_CTRL2\_DRIVE\_-  
STRONG, 1188  
DIACCL\_CTRL2\_LVLPOL\_-  
NEGAND, 1188  
DIACCL\_INTERRUPT\_LATCH\_-  
CLEAR1, 1188  
DIACCL\_INTERRUPT\_LATCH\_-  
CLEAR2, 1188  
DIACCL\_MODE\_GLVL2, 1188  
DIACCL\_MODE\_GLVL4, 1188  
DIACCL\_MODE\_GLVL8, 1188  
DIACCL\_MODE\_LVLDETECT,  
1189  
DIACCL\_MODE\_MEASURE,  
1189  
DIACCL\_MODE\_PLSDetect,  
1189  
DIACCL\_MODE\_STANDBY, 1189  
DIACCL\_REG\_CTRL1, 1189  
DIACCL\_REG\_CTRL2, 1189  
DIACCL\_REG\_DETECTSRC,  
1189  
DIACCL\_REG\_I2CADDR, 1189  
DIACCL\_REG\_INTLATCH, 1189  
DIACCL\_REG\_LATENCYTM,  
1190  
DIACCL\_REG\_LVLDETHR,  
1190  
DIACCL\_REG\_MODECTRL, 1190  
DIACCL\_REG\_OUTTEMP, 1190  
DIACCL\_REG\_PLSDETHR,  
1190  
DIACCL\_REG\_PLSDURVAL,  
1190  
DIACCL\_REG\_STATUS, 1190  
DIACCL\_REG\_TIMEWINDOW,  
1190  
DIACCL\_REG\_USERINFO, 1190  
DIACCL\_REG\_WHOAMI, 1190  
DIACCL\_REG\_X8, 1191  
DIACCL\_REG\_XHIGH, 1191  
DIACCL\_REG\_XHIGHDRIFT,  
1191  
DIACCL\_REG\_XLOW, 1191  
DIACCL\_REG\_XLOWDRIFT,  
1191  
DIACCL\_REG\_Y8, 1191  
DIACCL\_REG\_YHIGH, 1191  
DIACCL\_REG\_YHIGHDRIFT,  
1191  
DIACCL\_REG\_YLOW, 1191  
DIACCL\_REG\_YLOWDRIFT,  
1191  
DIACCL\_REG\_Z8, 1192  
DIACCL\_REG\_ZHIGH, 1192  
DIACCL\_REG\_ZHIGHDRIFT,  
1192  
DIACCL\_REG\_ZLOW, 1192  
DIACCL\_REG\_ZLOWDRIFT,  
1192

- DIACCL\_STATUS\_DATAOVER, [1192](#)
- DIACCL\_STATUS\_DATAREADY, [1192](#)
- DIACCL\_STATUS\_PARITYERR, [1192](#)
- DIGI\_PIN0, [1192](#)
- DIGI\_PIN1, [1193](#)
- DIGI\_PIN2, [1193](#)
- DIGI\_PIN3, [1193](#)
- DIGI\_PIN4, [1193](#)
- DIGI\_PIN5, [1193](#)
- DIGI\_PIN6, [1193](#)
- DIGI\_PIN7, [1193](#)
- DIGYRO\_CTRL1\_-  
BANDWIDTH\_1, [1193](#)
- DIGYRO\_CTRL1\_-  
BANDWIDTH\_2, [1194](#)
- DIGYRO\_CTRL1\_-  
BANDWIDTH\_3, [1194](#)
- DIGYRO\_CTRL1\_-  
BANDWIDTH\_4, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_-  
100, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_-  
200, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_-  
400, [1194](#)
- DIGYRO\_CTRL1\_DATARATE\_-  
800, [1194](#)
- DIGYRO\_CTRL1\_NORMAL, [1194](#)
- DIGYRO\_CTRL1\_POWERDOWN, [1195](#)
- DIGYRO\_CTRL1\_XENABLE, [1195](#)
- DIGYRO\_CTRL1\_YENABLE, [1195](#)
- DIGYRO\_CTRL1\_ZENABLE, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_001, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_002, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_005, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_01, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_02, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_05, [1195](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_1, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_2, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_4, [1196](#)
- DIGYRO\_CTRL2\_CUTOFF\_-  
FREQ\_8, [1196](#)
- DIGYRO\_CTRL2\_HPMODE\_-  
AUTOINT, [1196](#)
- DIGYRO\_CTRL2\_HPMODE\_-  
NORMAL, [1196](#)
- DIGYRO\_CTRL2\_HPMODE\_-  
REFSIG, [1196](#)
- DIGYRO\_CTRL2\_HPMODE\_-  
RESET, [1196](#)
- DIGYRO\_CTRL3\_INT1\_BOOT, [1196](#)
- DIGYRO\_CTRL3\_INT1\_ENABLE, [1196](#)
- DIGYRO\_CTRL3\_INT1\_-  
LOWACTIVE, [1197](#)
- DIGYRO\_CTRL3\_INT2\_-  
DATAREADY, [1197](#)
- DIGYRO\_CTRL3\_INT2\_EMPTY, [1197](#)
- DIGYRO\_CTRL3\_INT2\_-  
OVERRUN, [1197](#)
- DIGYRO\_CTRL3\_INT2\_-  
WATERMARK, [1197](#)
- DIGYRO\_CTRL3\_OPENDRAIN, [1197](#)
- DIGYRO\_CTRL4\_BIGENDIAN, [1197](#)
- DIGYRO\_CTRL4\_BLOCKDATA, [1197](#)
- DIGYRO\_CTRL4\_SCALE\_2000, [1197](#)
- DIGYRO\_CTRL4\_SCALE\_250, [1198](#)

- DIGYRO\_CTRL4\_SCALE\_500, 1198
- DIGYRO\_CTRL5\_FIFOENABLE, 1198
- DIGYRO\_CTRL5\_HPENABLE, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_1, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_2, 1198
- DIGYRO\_CTRL5\_INT1\_SEL\_3, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_1, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_2, 1198
- DIGYRO\_CTRL5\_OUT\_SEL\_3, 1199
- DIGYRO\_CTRL5\_REBOOTMEM, 1199
- DIGYRO\_FIFCTRL\_BYPASS, 1199
- DIGYRO\_FIFCTRL\_-  
  BYPASS2STREAM, 1199
- DIGYRO\_FIFCTRL\_FIFO, 1199
- DIGYRO\_FIFCTRL\_STREAM, 1199
- DIGYRO\_FIFCTRL\_-  
  STREAM2FIFO, 1199
- DIGYRO\_FIFCTRL\_-  
  WATERMARK\_MASK, 1199
- DIGYRO\_REG\_CTRL1, 1199
- DIGYRO\_REG\_CTRL1AUTO, 1199
- DIGYRO\_REG\_CTRL2, 1200
- DIGYRO\_REG\_CTRL3, 1200
- DIGYRO\_REG\_CTRL4, 1200
- DIGYRO\_REG\_CTRL5, 1200
- DIGYRO\_REG\_FIFCTRL, 1200
- DIGYRO\_REG\_FIFOSRC, 1200
- DIGYRO\_REG\_INT1\_CFG, 1200
- DIGYRO\_REG\_INT1\_DUR, 1200
- DIGYRO\_REG\_INT1\_SRC, 1200
- DIGYRO\_REG\_INT1\_XHI, 1200
- DIGYRO\_REG\_INT1\_XLO, 1201
- DIGYRO\_REG\_INT1\_YHI, 1201
- DIGYRO\_REG\_INT1\_YLO, 1201
- DIGYRO\_REG\_INT1\_ZHI, 1201
- DIGYRO\_REG\_INT1\_ZLO, 1201
- DIGYRO\_REG\_OUTTEMP, 1201
- DIGYRO\_REG\_REFERENCE, 1201
- DIGYRO\_REG\_STATUS, 1201
- DIGYRO\_REG\_TEMPAUTO, 1201
- DIGYRO\_REG\_WHOAMI, 1201
- DIGYRO\_REG\_XHIGH, 1202
- DIGYRO\_REG\_XLOW, 1202
- DIGYRO\_REG\_XLOWBURST, 1202
- DIGYRO\_REG\_YHIGH, 1202
- DIGYRO\_REG\_YLOW, 1202
- DIGYRO\_REG\_YLOWBURST, 1202
- DIGYRO\_REG\_ZHIGH, 1202
- DIGYRO\_REG\_ZLOW, 1202
- DIGYRO\_REG\_ZLOWBURST, 1202
- DIGYRO\_STATUS\_XDATA, 1202
- DIGYRO\_STATUS\_XOVER, 1203
- DIGYRO\_STATUS\_XYZDATA, 1203
- DIGYRO\_STATUS\_XYZOVER, 1203
- DIGYRO\_STATUS\_YDATA, 1203
- DIGYRO\_STATUS\_YOVER, 1203
- DIGYRO\_STATUS\_ZDATA, 1203
- DIGYRO\_STATUS\_ZOVER, 1203
- DISPLAY\_BUSY, 1203
- DISPLAY\_CHAR, 1203
- DISPLAY\_CONTRAST\_-  
  DEFAULT, 1203
- DISPLAY\_CONTRAST\_MAX, 1204
- DISPLAY\_ERASE\_ALL, 1204
- DISPLAY\_ERASE\_LINE, 1204
- DISPLAY\_FILL\_REGION, 1204
- DISPLAY\_FRAME, 1204
- DISPLAY\_HEIGHT, 1204
- DISPLAY\_HORIZONTAL\_LINE, 1204

- DISPLAY\_MENUICONS\_X\_DIFF, 1205
- DISPLAY\_MENUICONS\_X\_OFFS, 1205
- DISPLAY\_MENUICONS\_Y, 1205
- DISPLAY\_ON, 1205
- DISPLAY\_PIXEL, 1205
- DISPLAY\_POPUP, 1205
- DISPLAY\_REFRESH, 1205
- DISPLAY\_REFRESH\_DISABLED, 1205
- DISPLAY\_VERTICAL\_LINE, 1206
- DISPLAY\_WIDTH, 1206
- DisplayExecuteFunction, 1206
- DisplayModuleID, 1206
- DisplayModuleName, 1206
- DisplayOffsetContrast, 1206
- DisplayOffsetDisplay, 1206
- DisplayOffsetEraseMask, 1206
- DisplayOffsetFlags, 1206
- DisplayOffsetNormal, 1207
- DisplayOffsetPBitmaps, 1207
- DisplayOffsetPFont, 1207
- DisplayOffsetPFunc, 1207
- DisplayOffsetPMenuIcons, 1207
- DisplayOffsetPMenuText, 1207
- DisplayOffsetPopup, 1207
- DisplayOffsetPScreens, 1207
- DisplayOffsetPStatusIcons, 1207
- DisplayOffsetPStatusText, 1207
- DisplayOffsetPStepIcons, 1208
- DisplayOffsetPTextLines, 1208
- DisplayOffsetStatusIcons, 1208
- DisplayOffsetStepIcons, 1208
- DisplayOffsetTextLinesCenterFlags, 1208
- DisplayOffsetUpdateMask, 1208
- DIST\_CMD\_CUSTOM, 1208
- DIST\_CMD\_GP2D12, 1208
- DIST\_CMD\_GP2D120, 1208
- DIST\_CMD\_GP2YA02, 1208
- DIST\_CMD\_GP2YA21, 1209
- DIST\_REG\_DIST, 1209
- DIST\_REG\_DIST1, 1209
- DIST\_REG\_DIST\_MAX, 1209
- DIST\_REG\_DIST\_MIN, 1209
- DIST\_REG\_MODULE\_TYPE, 1209
- DIST\_REG\_NUM\_POINTS, 1209
- DIST\_REG\_VOLT, 1209
- DIST\_REG\_VOLT1, 1209
- DRAW\_OPT\_CLEAR, 1209
- DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_SCREEN, 1210
- DRAW\_OPT\_CLEAR\_PIXELS, 1210
- DRAW\_OPT\_CLEAR\_SCREEN\_MODES, 1210
- DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN, 1210
- DRAW\_OPT\_FILL\_SHAPE, 1210
- DRAW\_OPT\_FONT\_DIR\_B2TL, 1210
- DRAW\_OPT\_FONT\_DIR\_B2TR, 1210
- DRAW\_OPT\_FONT\_DIR\_L2RB, 1210
- DRAW\_OPT\_FONT\_DIR\_L2RT, 1211
- DRAW\_OPT\_FONT\_DIR\_R2LB, 1211
- DRAW\_OPT\_FONT\_DIR\_R2LT, 1211
- DRAW\_OPT\_FONT\_DIR\_T2BL, 1211
- DRAW\_OPT\_FONT\_DIR\_T2BR, 1211
- DRAW\_OPT\_FONT\_DIRECTIONS, 1211
- DRAW\_OPT\_FONT\_WRAP, 1211
- DRAW\_OPT\_INVERT, 1212
- DRAW\_OPT\_LOGICAL\_AND, 1212
- DRAW\_OPT\_LOGICAL\_COPY, 1212
- DRAW\_OPT\_LOGICAL\_OPERATIONS, 1212
- DRAW\_OPT\_LOGICAL\_OR, 1212
- DRAW\_OPT\_LOGICAL\_XOR, 1212
- DRAW\_OPT\_NORMAL, 1213

- DRAW\_OPT\_POLYGON\_-  
POLYLINE, 1213
- DrawCircle, 1213
- DrawEllipse, 1213
- DrawFont, 1213
- DrawGraphic, 1213
- DrawGraphicArray, 1213
- DrawLine, 1214
- DrawPoint, 1214
- DrawPolygon, 1214
- DrawRect, 1214
- DrawText, 1214
- EMETER\_REG\_AIN, 1214
- EMETER\_REG\_AOUT, 1214
- EMETER\_REG\_JOULES, 1214
- EMETER\_REG\_VIN, 1214
- EMETER\_REG\_VOUT, 1215
- EMETER\_REG\_WIN, 1215
- EMETER\_REG\_WOUT, 1215
- EOF, 1215
- ERR\_ARG, 1215
- ERR\_BAD\_POOL\_SIZE, 1215
- ERR\_BAD\_PTR, 1215
- ERR\_CLUMP\_COUNT, 1215
- ERR\_COMM\_BUFFER\_FULL,  
1215
- ERR\_COMM\_BUS\_ERR, 1215
- ERR\_COMM\_CHAN\_INVALID,  
1216
- ERR\_COMM\_CHAN\_NOT\_-  
READY, 1216
- ERR\_DEFAULT\_OFFSETS, 1216
- ERR\_FILE, 1216
- ERR\_INSANE\_OFFSET, 1216
- ERR\_INSTR, 1216
- ERR\_INVALID\_FIELD, 1216
- ERR\_INVALID\_PORT, 1216
- ERR\_INVALID\_QUEUE, 1216
- ERR\_INVALID\_SIZE, 1216
- ERR\_LOADER\_ERR, 1217
- ERR\_MEM, 1217
- ERR\_MEMMGR\_FAIL, 1217
- ERR\_NO\_ACTIVE\_CLUMP, 1217
- ERR\_NO\_CODE, 1217
- ERR\_NO\_PROG, 1217
- ERR\_NON\_FATAL, 1217
- ERR\_RC\_BAD\_PACKET, 1217
- ERR\_RC\_FAILED, 1217
- ERR\_RC\_ILLEGAL\_VAL, 1217
- ERR\_RC\_UNKNOWN\_CMD, 1218
- ERR\_SPOTCHECK\_FAIL, 1218
- ERR\_VER, 1218
- FALSE, 1218
- FileClose, 1218
- FileDelete, 1218
- FileFindFirst, 1218
- FileFindNext, 1218
- FileOpenAppend, 1218
- FileOpenRead, 1218
- FileOpenReadLinear, 1219
- FileOpenWrite, 1219
- FileOpenWriteLinear, 1219
- FileOpenWriteNonLinear, 1219
- FileRead, 1219
- FileRename, 1219
- FileResize, 1219
- FileResolveHandle, 1219
- FileSeek, 1219
- FileTell, 1219
- FileWrite, 1220
- FRAME\_SELECT, 1220
- FREQUENCY\_MAX, 1220
- FREQUENCY\_MIN, 1220
- GetStartTick, 1220
- GL\_CAMERA\_DEPTH, 1220
- GL\_CIRCLE, 1220
- GL\_CIRCLE\_SIZE, 1220
- GL\_CULL\_BACK, 1220
- GL\_CULL\_FRONT, 1221
- GL\_CULL\_MODE, 1221
- GL\_CULL\_NONE, 1221
- GL\_LINE, 1221
- GL\_POINT, 1221
- GL\_POLYGON, 1221
- GL\_ROTATE\_X, 1221
- GL\_ROTATE\_Y, 1222
- GL\_ROTATE\_Z, 1222
- GL\_SCALE\_X, 1222
- GL\_SCALE\_Y, 1222
- GL\_SCALE\_Z, 1222
- GL\_TRANSLATE\_X, 1222
- GL\_TRANSLATE\_Y, 1222

GL\_TRANSLATE\_Z, 1223  
GL\_ZOOM\_FACTOR, 1223  
HS\_ADDRESS\_1, 1223  
HS\_ADDRESS\_2, 1223  
HS\_ADDRESS\_3, 1223  
HS\_ADDRESS\_4, 1223  
HS\_ADDRESS\_5, 1223  
HS\_ADDRESS\_6, 1223  
HS\_ADDRESS\_7, 1224  
HS\_ADDRESS\_8, 1224  
HS\_ADDRESS\_ALL, 1224  
HS\_BAUD\_115200, 1224  
HS\_BAUD\_1200, 1224  
HS\_BAUD\_14400, 1224  
HS\_BAUD\_19200, 1224  
HS\_BAUD\_230400, 1224  
HS\_BAUD\_2400, 1224  
HS\_BAUD\_28800, 1224  
HS\_BAUD\_3600, 1225  
HS\_BAUD\_38400, 1225  
HS\_BAUD\_460800, 1225  
HS\_BAUD\_4800, 1225  
HS\_BAUD\_57600, 1225  
HS\_BAUD\_7200, 1225  
HS\_BAUD\_76800, 1225  
HS\_BAUD\_921600, 1225  
HS\_BAUD\_9600, 1225  
HS\_BAUD\_DEFAULT, 1225  
HS\_BYTES\_REMAINING, 1226  
HS\_CMD\_READY, 1226  
HS\_CTRL\_EXIT, 1226  
HS\_CTRL\_INIT, 1226  
HS\_CTRL\_UART, 1226  
HS\_DEFAULT, 1226  
HS\_DISABLE, 1226  
HS\_ENABLE, 1226  
HS\_INIT\_RECEIVER, 1227  
HS\_INITIALISE, 1227  
HS\_MODE\_10\_STOP, 1227  
HS\_MODE\_15\_STOP, 1227  
HS\_MODE\_20\_STOP, 1227  
HS\_MODE\_5\_DATA, 1227  
HS\_MODE\_6\_DATA, 1227  
HS\_MODE\_7\_DATA, 1227  
HS\_MODE\_7E1, 1227  
HS\_MODE\_8\_DATA, 1227  
HS\_MODE\_8N1, 1228  
HS\_MODE\_DEFAULT, 1228  
HS\_MODE\_E\_PARITY, 1228  
HS\_MODE\_M\_PARITY, 1228  
HS\_MODE\_MASK, 1228  
HS\_MODE\_N\_PARITY, 1228  
HS\_MODE\_O\_PARITY, 1228  
HS\_MODE\_S\_PARITY, 1228  
HS\_MODE\_UART\_RS232, 1229  
HS\_MODE\_UART\_RS485, 1229  
HS\_SEND\_DATA, 1229  
HS\_UART\_MASK, 1229  
HS\_UPDATE, 1229  
HT\_ADDR\_ACCEL, 1229  
HT\_ADDR\_ANGLE, 1229  
HT\_ADDR\_BAROMETRIC, 1229  
HT\_ADDR\_COLOR, 1229  
HT\_ADDR\_COLOR2, 1229  
HT\_ADDR\_COMPASS, 1230  
HT\_ADDR\_IRLINK, 1230  
HT\_ADDR\_IRRECEIVER, 1230  
HT\_ADDR\_IRSEEKER, 1230  
HT\_ADDR\_IRSEEKER2, 1230  
HT\_ADDR\_PROTOBOARD, 1230  
HT\_ADDR\_SUPERPRO, 1230  
HT\_CH1\_A, 1230  
HT\_CH1\_B, 1230  
HT\_CH2\_A, 1231  
HT\_CH2\_B, 1231  
HT\_CH3\_A, 1231  
HT\_CH3\_B, 1231  
HT\_CH4\_A, 1231  
HT\_CH4\_B, 1231  
HT\_CMD\_COLOR2\_50HZ, 1231  
HT\_CMD\_COLOR2\_60HZ, 1231  
HT\_CMD\_COLOR2\_ACTIVE,  
1231  
HT\_CMD\_COLOR2\_BLCAL, 1232  
HT\_CMD\_COLOR2\_FAR, 1232  
HT\_CMD\_COLOR2\_LED\_HI,  
1232  
HT\_CMD\_COLOR2\_LED\_LOW,  
1232  
HT\_CMD\_COLOR2\_NEAR, 1232  
HT\_CMD\_COLOR2\_PASSIVE,  
1232

HT\_CMD\_COLOR2\_RAW, 1232  
HT\_CMD\_COLOR2\_WBCAL,  
1232  
HTANGLE\_MODE\_CALIBRATE,  
1232  
HTANGLE\_MODE\_NORMAL,  
1232  
HTANGLE\_MODE\_RESET, 1233  
HTANGLE\_REG\_ACDIR, 1233  
HTANGLE\_REG\_DC01, 1233  
HTANGLE\_REG\_DC02, 1233  
HTANGLE\_REG\_DC03, 1233  
HTANGLE\_REG\_DC04, 1233  
HTANGLE\_REG\_DC05, 1233  
HTANGLE\_REG\_DCAVG, 1233  
HTANGLE\_REG\_DCDIR, 1233  
HTANGLE\_REG\_MODE, 1234  
HTBAR\_REG\_CALIBRATION,  
1234  
HTBAR\_REG\_COMMAND, 1234  
HTBAR\_REG\_PRESSURE, 1234  
HTBAR\_REG\_TEMPERATURE,  
1234  
HTIR2\_MODE\_1200, 1234  
HTIR2\_MODE\_600, 1234  
HTIR2\_REG\_AC01, 1234  
HTIR2\_REG\_AC02, 1234  
HTIR2\_REG\_AC03, 1235  
HTIR2\_REG\_AC04, 1235  
HTIR2\_REG\_AC05, 1235  
HTIR2\_REG\_ACDIR, 1235  
HTIR2\_REG\_DC01, 1235  
HTIR2\_REG\_DC02, 1235  
HTIR2\_REG\_DC03, 1235  
HTIR2\_REG\_DC04, 1235  
HTIR2\_REG\_DC05, 1235  
HTIR2\_REG\_DCAVG, 1235  
HTIR2\_REG\_DCDIR, 1236  
HTIR2\_REG\_MODE, 1236  
HTPROTO\_A0, 1236  
HTPROTO\_A1, 1236  
HTPROTO\_A2, 1236  
HTPROTO\_A3, 1236  
HTPROTO\_A4, 1236  
HTPROTO\_REG\_A0, 1236  
HTPROTO\_REG\_A1, 1237  
HTPROTO\_REG\_A2, 1237  
HTPROTO\_REG\_A3, 1237  
HTPROTO\_REG\_A4, 1237  
HTPROTO\_REG\_DCTRL, 1237  
HTPROTO\_REG\_DIN, 1237  
HTPROTO\_REG\_DOUT, 1237  
HTPROTO\_REG\_SRATE, 1237  
HTSPRO\_A0, 1237  
HTSPRO\_A1, 1238  
HTSPRO\_A2, 1238  
HTSPRO\_A3, 1238  
HTSPRO\_DAC0, 1238  
HTSPRO\_DAC1, 1238  
HTSPRO\_REG\_A0, 1238  
HTSPRO\_REG\_A1, 1238  
HTSPRO\_REG\_A2, 1238  
HTSPRO\_REG\_A3, 1239  
HTSPRO\_REG\_CTRL, 1239  
HTSPRO\_REG\_DAC0\_FREQ,  
1239  
HTSPRO\_REG\_DAC0\_MODE,  
1239  
HTSPRO\_REG\_DAC0\_VOLTAGE,  
1239  
HTSPRO\_REG\_DAC1\_FREQ,  
1239  
HTSPRO\_REG\_DAC1\_MODE,  
1239  
HTSPRO\_REG\_DAC1\_VOLTAGE,  
1239  
HTSPRO\_REG\_DCTRL, 1239  
HTSPRO\_REG\_DIN, 1239  
HTSPRO\_REG\_DLADDRESS,  
1240  
HTSPRO\_REG\_DLCHKSUM,  
1240  
HTSPRO\_REG\_DLCONTROL,  
1240  
HTSPRO\_REG\_DLDATA, 1240  
HTSPRO\_REG\_DOUT, 1240  
HTSPRO\_REG\_LED, 1240  
HTSPRO\_REG\_MEMORY\_20,  
1240  
HTSPRO\_REG\_MEMORY\_21,  
1240

HTSPRO\_REG\_MEMORY\_22,  
1240

HTSPRO\_REG\_MEMORY\_23,  
1240

HTSPRO\_REG\_MEMORY\_24,  
1241

HTSPRO\_REG\_MEMORY\_25,  
1241

HTSPRO\_REG\_MEMORY\_26,  
1241

HTSPRO\_REG\_MEMORY\_27,  
1241

HTSPRO\_REG\_MEMORY\_28,  
1241

HTSPRO\_REG\_MEMORY\_29,  
1241

HTSPRO\_REG\_MEMORY\_2A,  
1241

HTSPRO\_REG\_MEMORY\_2B,  
1241

HTSPRO\_REG\_MEMORY\_2C,  
1241

HTSPRO\_REG\_MEMORY\_2D,  
1241

HTSPRO\_REG\_MEMORY\_2E,  
1242

HTSPRO\_REG\_MEMORY\_2F,  
1242

HTSPRO\_REG\_MEMORY\_30,  
1242

HTSPRO\_REG\_MEMORY\_31,  
1242

HTSPRO\_REG\_MEMORY\_32,  
1242

HTSPRO\_REG\_MEMORY\_33,  
1242

HTSPRO\_REG\_MEMORY\_34,  
1242

HTSPRO\_REG\_MEMORY\_35,  
1242

HTSPRO\_REG\_MEMORY\_36,  
1242

HTSPRO\_REG\_MEMORY\_37,  
1242

HTSPRO\_REG\_MEMORY\_38,  
1243

HTSPRO\_REG\_MEMORY\_39,  
1243

HTSPRO\_REG\_MEMORY\_3A,  
1243

HTSPRO\_REG\_MEMORY\_3B,  
1243

HTSPRO\_REG\_MEMORY\_3C,  
1243

HTSPRO\_REG\_MEMORY\_3D,  
1243

HTSPRO\_REG\_MEMORY\_3E,  
1243

HTSPRO\_REG\_MEMORY\_3F,  
1243

HTSPRO\_REG\_STROBE, 1243

I2C\_ADDR\_DEFAULT, 1243

I2C\_OPTION\_FAST, 1244

I2C\_OPTION\_NOESTART, 1244

I2C\_OPTION\_STANDARD, 1244

I2C\_REG\_CMD, 1244

I2C\_REG\_DEVICE\_ID, 1244

I2C\_REG\_VENDOR\_ID, 1244

I2C\_REG\_VERSION, 1245

IN\_1, 1245

IN\_2, 1245

IN\_3, 1245

IN\_4, 1245

IN\_MODE\_ANGLESTEP, 1245

IN\_MODE\_BOOLEAN, 1245

IN\_MODE\_CELSIUS, 1245

IN\_MODE\_FAHRENHEIT, 1246

IN\_MODE\_MODEMASK, 1246

IN\_MODE\_PCTFULLSCALE,  
1246

IN\_MODE\_PERIODCOUNTER,  
1246

IN\_MODE\_RAW, 1246

IN\_MODE\_SLOPEMASK, 1246

IN\_MODE\_TRANSITIONCNT,  
1246

IN\_TYPE\_ANGLE, 1246

IN\_TYPE\_COLORBLUE, 1246

IN\_TYPE\_COLOREXIT, 1246

IN\_TYPE\_COLORFULL, 1247

IN\_TYPE\_COLORGREEN, 1247

IN\_TYPE\_COLORNONE, 1247

- IN\_TYPE\_COLORRED, 1247
- IN\_TYPE\_CUSTOM, 1247
- IN\_TYPE\_HISPEED, 1247
- IN\_TYPE\_LIGHT\_ACTIVE, 1247
- IN\_TYPE\_LIGHT\_INACTIVE, 1247
- IN\_TYPE\_LOWSPEED, 1247
- IN\_TYPE\_LOWSPEED\_9V, 1247
- IN\_TYPE\_NO\_SENSOR, 1248
- IN\_TYPE\_REFLECTION, 1248
- IN\_TYPE\_SOUND\_DB, 1248
- IN\_TYPE\_SOUND\_DBA, 1248
- IN\_TYPE\_SWITCH, 1248
- IN\_TYPE\_TEMPERATURE, 1248
- INPUT\_BLACKCOLOR, 1248
- INPUT\_BLANK, 1248
- INPUT\_BLUE, 1248
- INPUT\_BLUECOLOR, 1248
- INPUT\_CAL\_POINT\_0, 1249
- INPUT\_CAL\_POINT\_1, 1249
- INPUT\_CAL\_POINT\_2, 1249
- INPUT\_CUSTOM9V, 1249
- INPUT\_CUSTOMACTIVE, 1249
- INPUT\_CUSTOMINACTIVE, 1249
- INPUT\_DIGI0, 1249
- INPUT\_DIGI1, 1249
- INPUT\_GREEN, 1250
- INPUT\_GREENCOLOR, 1250
- INPUT\_INVALID\_DATA, 1250
- INPUT\_NO\_OF\_COLORS, 1250
- INPUT\_NO\_OF\_POINTS, 1250
- INPUT\_PINCMD\_CLEAR, 1250
- INPUT\_PINCMD\_DIR, 1250
- INPUT\_PINCMD\_MASK, 1250
- INPUT\_PINCMD\_READ, 1251
- INPUT\_PINCMD\_SET, 1251
- INPUT\_PINCMD\_WAIT, 1251
- INPUT\_PINDIR\_INPUT, 1251
- INPUT\_PINDIR\_OUTPUT, 1251
- INPUT\_RED, 1251
- INPUT\_REDCOLOR, 1252
- INPUT\_RESETCAL, 1252
- INPUT\_RUNNINGCAL, 1252
- INPUT\_SENSORCAL, 1252
- INPUT\_SENSOROFF, 1252
- INPUT\_STARTCAL, 1252
- INPUT\_WHITECOLOR, 1252
- INPUT\_YELLOWCOLOR, 1252
- InputModeField, 1252
- InputModuleID, 1253
- InputModuleName, 1253
- InputOffsetADRaw, 1253
- InputOffsetColorADRaw, 1253
- InputOffsetColorBoolean, 1253
- InputOffsetColorCalibration, 1253
- InputOffsetColorCalibrationState, 1253
- InputOffsetColorCalLimits, 1253
- InputOffsetColorSensorRaw, 1253
- InputOffsetColorSensorValue, 1253
- InputOffsetCustomActiveStatus, 1254
- InputOffsetCustomPctFullScale, 1254
- InputOffsetCustomZeroOffset, 1254
- InputOffsetDigiPinsDir, 1254
- InputOffsetDigiPinsIn, 1254
- InputOffsetDigiPinsOut, 1254
- InputOffsetInvalidData, 1254
- InputOffsetSensorBoolean, 1254
- InputOffsetSensorMode, 1254
- InputOffsetSensorRaw, 1254
- InputOffsetSensorType, 1255
- InputOffsetSensorValue, 1255
- InputPinFunction, 1255
- INT\_MAX, 1255
- INT\_MIN, 1255
- INTF\_BTOFF, 1255
- INTF\_BTON, 1255
- INTF\_CONNECT, 1255
- INTF\_CONNECTBYNAME, 1255
- INTF\_CONNECTREQ, 1256
- INTF\_DISCONNECT, 1256
- INTF\_DISCONNECTALL, 1256
- INTF\_EXTREAD, 1256
- INTF\_FACTORYRESET, 1256
- INTF\_OPENSTREAM, 1256
- INTF\_PINREQ, 1256
- INTF\_REMOVEDEVICE, 1256
- INTF\_SEARCH, 1256
- INTF\_SENDDATA, 1256
- INTF\_SENDFILE, 1257

INTF\_SETBTNAME, 1257  
INTF\_SETCMDMODE, 1257  
INTF\_STOPSEARCH, 1257  
INTF\_VISIBILITY, 1257  
InvalidDataField, 1257  
IOCTRL\_BOOT, 1257  
IOCTRL\_POWERDOWN, 1257  
IOCtrlModuleID, 1257  
IOCtrlModuleName, 1257  
IOCtrlOffsetPowerOn, 1258  
IOMapRead, 1258  
IOMapReadByID, 1258  
IOMapWrite, 1258  
IOMapWriteByID, 1258  
KeepAlive, 1258  
LCD\_LINE1, 1258  
LCD\_LINE2, 1259  
LCD\_LINE3, 1260  
LCD\_LINE4, 1260  
LCD\_LINE5, 1260  
LCD\_LINE6, 1261  
LCD\_LINE7, 1261  
LCD\_LINE8, 1261  
LDR\_APPENDNOTPOSSIBLE,  
1261  
LDR\_BTBUSY, 1262  
LDR\_BTCONNECTFAIL, 1262  
LDR\_BTTIMEOUT, 1262  
LDR\_CMD\_BOOTCMD, 1262  
LDR\_CMD\_BTFACTORYRESET,  
1262  
LDR\_CMD\_BTGETADR, 1262  
LDR\_CMD\_CLOSE, 1262  
LDR\_CMD\_-  
CLOSEMODHANDLE,  
1262  
LDR\_CMD\_CROPDATAFILE,  
1262  
LDR\_CMD\_DELETE, 1262  
LDR\_CMD\_-  
DELETEUSERFLASH, 1263  
LDR\_CMD\_DEVICEINFO, 1263  
LDR\_CMD\_FINDFIRST, 1263  
LDR\_CMD\_-  
FINDFIRSTMODULE, 1263  
LDR\_CMD\_FINDNEXT, 1263  
LDR\_CMD\_-  
FINDNEXTMODULE, 1263  
LDR\_CMD\_IOMAPREAD, 1263  
LDR\_CMD\_IOMAPWRITE, 1263  
LDR\_CMD\_OPENAPPENDDATA,  
1263  
LDR\_CMD\_OPENREAD, 1263  
LDR\_CMD\_OPENREADLINEAR,  
1264  
LDR\_CMD\_OPENWRITE, 1264  
LDR\_CMD\_OPENWRITEDATA,  
1264  
LDR\_CMD\_-  
OPENWRITELINEAR, 1264  
LDR\_CMD\_POLLCMD, 1264  
LDR\_CMD\_POLLCMDLEN, 1264  
LDR\_CMD\_READ, 1264  
LDR\_CMD\_RENAMEFILE, 1264  
LDR\_CMD\_RESIZEDATAFILE,  
1264  
LDR\_CMD\_-  
SEEKFROMCURRENT,  
1264  
LDR\_CMD\_SEEKFROMEND,  
1265  
LDR\_CMD\_SEEKFROMSTART,  
1265  
LDR\_CMD\_SETBRICKNAME,  
1265  
LDR\_CMD\_VERSIONS, 1265  
LDR\_CMD\_WRITE, 1265  
LDR\_ENDOFFILE, 1265  
LDR\_EOFEXPECTED, 1265  
LDR\_FILEEXISTS, 1265  
LDR\_FILEISBUSY, 1266  
LDR\_FILEISFULL, 1266  
LDR\_FILENOTFOUND, 1266  
LDR\_FILETX\_CLOSEERROR,  
1266  
LDR\_FILETX\_DSTEXISTS, 1266  
LDR\_FILETX\_SRCMISSING,  
1266  
LDR\_FILETX\_STREAMERROR,  
1266  
LDR\_FILETX\_TIMEOUT, 1266

- LDR\_-
  - HANDLEALREADYCLOSED, [1267](#)
- LDR\_ILLEGALFILENAME, [1267](#)
- LDR\_ILLEGALHANDLE, [1267](#)
- LDR\_INPROGRESS, [1267](#)
- LDR\_INVALIDSEEK, [1267](#)
- LDR\_MODULENOTFOUND, [1267](#)
- LDR\_NOLINEARSPACE, [1267](#)
- LDR\_NOMOREFILES, [1267](#)
- LDR\_NOMOREHANDLES, [1267](#)
- LDR\_NOSPACE, [1267](#)
- LDR\_NOTLINEARFILE, [1268](#)
- LDR\_NOWRITEBUFFERS, [1268](#)
- LDR\_OUTOFBOUNDARY, [1268](#)
- LDR\_REQPIN, [1268](#)
- LDR\_SUCCESS, [1268](#)
- LDR\_UNDEFINEDERROR, [1268](#)
- LED\_BLUE, [1268](#)
- LED\_NONE, [1268](#)
- LED\_RED, [1269](#)
- LEGO\_ADDR\_EMETER, [1269](#)
- LEGO\_ADDR\_TEMP, [1269](#)
- LEGO\_ADDR\_US, [1269](#)
- ListFiles, [1269](#)
- LoaderExecuteFunction, [1269](#)
- LoaderModuleID, [1269](#)
- LoaderModuleName, [1269](#)
- LoaderOffsetFreeUserFlash, [1269](#)
- LoaderOffsetPFunc, [1269](#)
- LONG\_MAX, [1270](#)
- LONG\_MIN, [1270](#)
- LOWSPEED\_CH\_NOT\_READY, [1270](#)
- LOWSPEED\_COMMUNICATING, [1270](#)
- LOWSPEED\_DATA\_RECEIVED, [1270](#)
- LOWSPEED\_DONE, [1270](#)
- LOWSPEED\_ERROR, [1270](#)
- LOWSPEED\_IDLE, [1270](#)
- LOWSPEED\_INIT, [1270](#)
- LOWSPEED\_LOAD\_BUFFER, [1271](#)
- LOWSPEED\_NO\_ERROR, [1271](#)
- LOWSPEED\_RECEIVING, [1271](#)
- LOWSPEED\_RX\_ERROR, [1271](#)
- LOWSPEED\_TRANSMITTING, [1271](#)
- LOWSPEED\_TX\_ERROR, [1271](#)
- LowSpeedModuleID, [1271](#)
- LowSpeedModuleName, [1271](#)
- LowSpeedOffsetChannelState, [1271](#)
- LowSpeedOffsetErrorType, [1271](#)
- LowSpeedOffsetInBufBuf, [1272](#)
- LowSpeedOffsetInBufBytesToRx, [1272](#)
- LowSpeedOffsetInBufInPtr, [1272](#)
- LowSpeedOffsetInBufOutPtr, [1272](#)
- LowSpeedOffsetMode, [1272](#)
- LowSpeedOffsetNoRestartOnRead, [1272](#)
- LowSpeedOffsetOutBufBuf, [1272](#)
- LowSpeedOffsetOutBufBytesToRx, [1272](#)
- LowSpeedOffsetOutBufInPtr, [1272](#)
- LowSpeedOffsetOutBufOutPtr, [1272](#)
- LowSpeedOffsetSpeed, [1273](#)
- LowSpeedOffsetState, [1273](#)
- LR\_COULD\_NOT\_SAVE, [1273](#)
- LR\_ENTRY\_REMOVED, [1273](#)
- LR\_STORE\_IS\_FULL, [1273](#)
- LR\_SUCCESS, [1273](#)
- LR\_UNKNOWN\_ADDR, [1273](#)
- LSREAD\_NO\_RESTART\_1, [1273](#)
- LSREAD\_NO\_RESTART\_2, [1273](#)
- LSREAD\_NO\_RESTART\_3, [1273](#)
- LSREAD\_NO\_RESTART\_4, [1274](#)
- LSREAD\_NO\_RESTART\_MASK, [1274](#)
- LSREAD\_RESTART\_ALL, [1274](#)
- LSREAD\_RESTART\_NONE, [1274](#)
- MAILBOX1, [1274](#)
- MAILBOX10, [1274](#)
- MAILBOX2, [1274](#)
- MAILBOX3, [1274](#)
- MAILBOX4, [1275](#)
- MAILBOX5, [1275](#)
- MAILBOX6, [1275](#)
- MAILBOX7, [1275](#)
- MAILBOX8, [1275](#)

MAILBOX9, 1275  
MAX\_BT\_MSG\_SIZE, 1275  
MaxAccelerationField, 1275  
MaxSpeedField, 1275  
MemoryManager, 1276  
MENUICON\_CENTER, 1276  
MENUICON\_LEFT, 1276  
MENUICON\_RIGHT, 1276  
MENUICONS, 1276  
MENUTEXT, 1276  
MessageRead, 1276  
MessageWrite, 1276  
MI\_ADDR\_XG1300L, 1276  
MIN\_1, 1277  
MS\_1, 1277  
MS\_10, 1277  
MS\_100, 1277  
MS\_150, 1277  
MS\_2, 1277  
MS\_20, 1278  
MS\_200, 1278  
MS\_250, 1278  
MS\_3, 1278  
MS\_30, 1278  
MS\_300, 1278  
MS\_350, 1278  
MS\_4, 1278  
MS\_40, 1279  
MS\_400, 1279  
MS\_450, 1279  
MS\_5, 1279  
MS\_50, 1279  
MS\_500, 1279  
MS\_6, 1279  
MS\_60, 1280  
MS\_600, 1280  
MS\_7, 1280  
MS\_70, 1280  
MS\_700, 1280  
MS\_8, 1280  
MS\_80, 1280  
MS\_800, 1280  
MS\_9, 1280  
MS\_90, 1280  
MS\_900, 1281  
MS\_ADDR\_ACCLNX, 1281  
MS\_ADDR\_CMPSNX, 1281  
MS\_ADDR\_DISTNX, 1281  
MS\_ADDR\_IVSENS, 1281  
MS\_ADDR\_LINELDR, 1282  
MS\_ADDR\_MTRMUX, 1282  
MS\_ADDR\_NRLINK, 1282  
MS\_ADDR\_NXTCAM, 1282  
MS\_ADDR\_NXTHID, 1282  
MS\_ADDR\_NXTMMX, 1282  
MS\_ADDR\_NXTSERVO, 1283  
MS\_ADDR\_NXTSERVO\_EM,  
1283  
MS\_ADDR\_PFMATE, 1283  
MS\_ADDR\_PSPNX, 1283  
MS\_ADDR\_RTCLOCK, 1283  
MS\_ADDR\_RXMUX, 1283  
MS\_CMD\_ADPA\_OFF, 1283  
MS\_CMD\_ADPA\_ON, 1284  
MS\_CMD\_DEENERGIZED, 1284  
MS\_CMD\_ENERGIZED, 1284  
NA, 1284  
NO\_ERR, 1284  
NO\_OF\_BTNS, 1284  
NormalizedValueField, 1285  
NRLINK\_CMD\_2400, 1285  
NRLINK\_CMD\_4800, 1285  
NRLINK\_CMD\_FLUSH, 1285  
NRLINK\_CMD\_IR\_LONG, 1285  
NRLINK\_CMD\_IR\_SHORT, 1285  
NRLINK\_CMD\_RUN\_MACRO,  
1285  
NRLINK\_CMD\_SET\_PF, 1285  
NRLINK\_CMD\_SET\_RCX, 1285  
NRLINK\_CMD\_SET\_TRAIN,  
1285  
NRLINK\_CMD\_TX\_RAW, 1286  
NRLINK\_REG\_BYTES, 1286  
NRLINK\_REG\_DATA, 1286  
NRLINK\_REG\_EEPROM, 1286  
NULL, 1286  
NXTHID\_CMD\_ASCII, 1286  
NXTHID\_CMD\_DIRECT, 1286  
NXTHID\_CMD\_TRANSMIT, 1286  
NXTHID\_MOD\_LEFT\_ALT, 1286  
NXTHID\_MOD\_LEFT\_CTRL,  
1286

- NXTHID\_MOD\_LEFT\_GUI, [1287](#)
- NXTHID\_MOD\_LEFT\_SHIFT,  
[1287](#)
- NXTHID\_MOD\_NONE, [1287](#)
- NXTHID\_MOD\_RIGHT\_ALT,  
[1287](#)
- NXTHID\_MOD\_RIGHT\_CTRL,  
[1287](#)
- NXTHID\_MOD\_RIGHT\_GUI,  
[1287](#)
- NXTHID\_MOD\_RIGHT\_SHIFT,  
[1287](#)
- NXTHID\_REG\_CMD, [1287](#)
- NXTHID\_REG\_DATA, [1288](#)
- NXTHID\_REG\_MODIFIER, [1288](#)
- NXTLL\_CMD\_BLACK, [1288](#)
- NXTLL\_CMD\_EUROPEAN, [1288](#)
- NXTLL\_CMD\_INVERT, [1288](#)
- NXTLL\_CMD\_POWERDOWN,  
[1288](#)
- NXTLL\_CMD\_POWERUP, [1288](#)
- NXTLL\_CMD\_RESET, [1288](#)
- NXTLL\_CMD\_SNAPSHOT, [1288](#)
- NXTLL\_CMD\_UNIVERSAL, [1288](#)
- NXTLL\_CMD\_USA, [1289](#)
- NXTLL\_CMD\_WHITE, [1289](#)
- NXTLL\_REG\_AVERAGE, [1289](#)
- NXTLL\_REG\_BLACKDATA, [1289](#)
- NXTLL\_REG\_BLACKLIMITS,  
[1289](#)
- NXTLL\_REG\_CALIBRATED,  
[1289](#)
- NXTLL\_REG\_CMD, [1289](#)
- NXTLL\_REG\_KD\_FACTOR, [1289](#)
- NXTLL\_REG\_KD\_VALUE, [1289](#)
- NXTLL\_REG\_KI\_FACTOR, [1290](#)
- NXTLL\_REG\_KI\_VALUE, [1290](#)
- NXTLL\_REG\_KP\_FACTOR, [1290](#)
- NXTLL\_REG\_KP\_VALUE, [1290](#)
- NXTLL\_REG\_RAWVOLTAGE,  
[1290](#)
- NXTLL\_REG\_RESULT, [1290](#)
- NXTLL\_REG\_SETPOINT, [1290](#)
- NXTLL\_REG\_STEERING, [1290](#)
- NXTLL\_REG\_WHITEDATA, [1290](#)
- NXTLL\_REG\_WHITELIMITS,  
[1290](#)
- NXTPM\_CMD\_RESET, [1291](#)
- NXTPM\_REG\_CAPACITY, [1291](#)
- NXTPM\_REG\_CMD, [1291](#)
- NXTPM\_REG\_CURRENT, [1291](#)
- NXTPM\_REG\_ERRORCOUNT,  
[1291](#)
- NXTPM\_REG\_GAIN, [1291](#)
- NXTPM\_REG\_MAXCURRENT,  
[1291](#)
- NXTPM\_REG\_MAXVOLTAGE,  
[1291](#)
- NXTPM\_REG\_MINCURRENT,  
[1291](#)
- NXTPM\_REG\_MINVOLTAGE,  
[1291](#)
- NXTPM\_REG\_POWER, [1292](#)
- NXTPM\_REG\_TIME, [1292](#)
- NXTPM\_REG\_TOTALPOWER,  
[1292](#)
- NXTPM\_REG\_USERGAIN, [1292](#)
- NXTPM\_REG\_VOLTAGE, [1292](#)
- NXTSE\_ZONE\_FRONT, [1292](#)
- NXTSE\_ZONE\_LEFT, [1292](#)
- NXTSE\_ZONE\_NONE, [1292](#)
- NXTSE\_ZONE\_RIGHT, [1293](#)
- NXTSERVO\_CMD\_EDIT1, [1293](#)
- NXTSERVO\_CMD\_EDIT2, [1293](#)
- NXTSERVO\_CMD\_GOTO, [1293](#)
- NXTSERVO\_CMD\_HALT, [1293](#)
- NXTSERVO\_CMD\_INIT, [1293](#)
- NXTSERVO\_CMD\_PAUSE, [1293](#)
- NXTSERVO\_CMD\_RESET, [1293](#)
- NXTSERVO\_CMD\_RESUME,  
[1294](#)
- NXTSERVO\_EM\_CMD\_QUIT,  
[1294](#)
- NXTSERVO\_EM\_REG\_CMD,  
[1294](#)
- NXTSERVO\_EM\_REG\_-  
EEPROM\_END, [1294](#)
- NXTSERVO\_EM\_REG\_-  
EEPROM\_START, [1294](#)
- NXTSERVO\_POS\_CENTER, [1294](#)
- NXTSERVO\_POS\_MAX, [1294](#)

NXTSERVO\_POS\_MIN, 1294  
NXTSERVO\_QPOS\_CENTER,  
1294  
NXTSERVO\_QPOS\_MAX, 1295  
NXTSERVO\_QPOS\_MIN, 1295  
NXTSERVO\_REG\_CMD, 1295  
NXTSERVO\_REG\_S1\_POS, 1295  
NXTSERVO\_REG\_S1\_QPOS,  
1295  
NXTSERVO\_REG\_S1\_SPEED,  
1295  
NXTSERVO\_REG\_S2\_POS, 1295  
NXTSERVO\_REG\_S2\_QPOS,  
1295  
NXTSERVO\_REG\_S2\_SPEED,  
1295  
NXTSERVO\_REG\_S3\_POS, 1296  
NXTSERVO\_REG\_S3\_QPOS,  
1296  
NXTSERVO\_REG\_S3\_SPEED,  
1296  
NXTSERVO\_REG\_S4\_POS, 1296  
NXTSERVO\_REG\_S4\_QPOS,  
1296  
NXTSERVO\_REG\_S4\_SPEED,  
1296  
NXTSERVO\_REG\_S5\_POS, 1296  
NXTSERVO\_REG\_S5\_QPOS,  
1296  
NXTSERVO\_REG\_S5\_SPEED,  
1296  
NXTSERVO\_REG\_S6\_POS, 1296  
NXTSERVO\_REG\_S6\_QPOS,  
1297  
NXTSERVO\_REG\_S6\_SPEED,  
1297  
NXTSERVO\_REG\_S7\_POS, 1297  
NXTSERVO\_REG\_S7\_QPOS,  
1297  
NXTSERVO\_REG\_S7\_SPEED,  
1297  
NXTSERVO\_REG\_S8\_POS, 1297  
NXTSERVO\_REG\_S8\_QPOS,  
1297  
NXTSERVO\_REG\_S8\_SPEED,  
1297  
NXTSERVO\_REG\_VOLTAGE,  
1297  
NXTSERVO\_SERVO\_1, 1297  
NXTSERVO\_SERVO\_2, 1298  
NXTSERVO\_SERVO\_3, 1298  
NXTSERVO\_SERVO\_4, 1298  
NXTSERVO\_SERVO\_5, 1298  
NXTSERVO\_SERVO\_6, 1298  
NXTSERVO\_SERVO\_7, 1298  
NXTSERVO\_SERVO\_8, 1298  
OPARR\_MAX, 1298  
OPARR\_MEAN, 1299  
OPARR\_MIN, 1299  
OPARR\_SORT, 1299  
OPARR\_STD, 1299  
OPARR\_SUM, 1299  
OPARR\_SUMSQ, 1299  
OUT\_A, 1299  
OUT\_AB, 1300  
OUT\_ABC, 1300  
OUT\_AC, 1300  
OUT\_B, 1300  
OUT\_BC, 1300  
OUT\_C, 1300  
OUT\_MODE\_BRAKE, 1300  
OUT\_MODE\_COAST, 1300  
OUT\_MODE\_MOTORON, 1301  
OUT\_MODE\_REGMETHOD, 1301  
OUT\_MODE\_REGULATED, 1301  
OUT\_OPTION\_HOLDATLIMIT,  
1301  
OUT\_OPTION\_-  
RAMPDOWNLIMIT,  
1301  
OUT\_REGMODE\_IDLE, 1301  
OUT\_REGMODE\_POS, 1301  
OUT\_REGMODE\_SPEED, 1301  
OUT\_REGMODE\_SYNC, 1302  
OUT\_REGOPTION\_NO\_-  
SATURATION, 1302  
OUT\_RUNSTATE\_HOLD, 1302  
OUT\_RUNSTATE\_IDLE, 1302  
OUT\_RUNSTATE\_RAMPDOWN,  
1302  
OUT\_RUNSTATE\_RAMPUP, 1302

- OUT\_RUNSTATE\_RUNNING, 1302
- OutputModeField, 1303
- OutputModuleID, 1303
- OutputModuleName, 1303
- OutputOffsetActualSpeed, 1303
- OutputOffsetBlockTachoCount, 1303
- OutputOffsetFlags, 1303
- OutputOffsetMaxAccel, 1303
- OutputOffsetMaxSpeed, 1304
- OutputOffsetMode, 1304
- OutputOffsetMotorRPM, 1304
- OutputOffsetOptions, 1304
- OutputOffsetOverloaded, 1304
- OutputOffsetRegDParameter, 1304
- OutputOffsetRegIParameter, 1304
- OutputOffsetRegMode, 1304
- OutputOffsetRegPParameter, 1304
- OutputOffsetRegulationOptions, 1305
- OutputOffsetRegulationTime, 1305
- OutputOffsetRotationCount, 1305
- OutputOffsetRunState, 1305
- OutputOffsetSpeed, 1305
- OutputOffsetSyncTurnParameter, 1305
- OutputOffsetTachoCount, 1305
- OutputOffsetTachoLimit, 1305
- OutputOptionsField, 1305
- OverloadField, 1306
- PF\_CHANNEL\_1, 1306
- PF\_CHANNEL\_2, 1306
- PF\_CHANNEL\_3, 1306
- PF\_CHANNEL\_4, 1306
- PF\_CMD\_BRAKE, 1307
- PF\_CMD\_FLOAT, 1307
- PF\_CMD\_FWD, 1307
- PF\_CMD\_REV, 1307
- PF\_CMD\_STOP, 1307
- PF\_CST\_CLEAR1\_CLEAR2, 1307
- PF\_CST\_CLEAR1\_SET2, 1307
- PF\_CST\_DECREMENT\_PWM, 1308
- PF\_CST\_FULL\_FWD, 1308
- PF\_CST\_FULL\_REV, 1308
- PF\_CST\_INCREMENT\_PWM, 1308
- PF\_CST\_SET1\_CLEAR2, 1308
- PF\_CST\_SET1\_SET2, 1308
- PF\_CST\_TOGGLE\_DIR, 1308
- PF\_FUNC\_CLEAR, 1308
- PF\_FUNC\_NOCHANGE, 1308
- PF\_FUNC\_SET, 1309
- PF\_FUNC\_TOGGLE, 1309
- PF\_MODE\_COMBO\_DIRECT, 1309
- PF\_MODE\_COMBO\_PWM, 1309
- PF\_MODE\_SINGLE\_OUTPUT\_CST, 1309
- PF\_MODE\_SINGLE\_OUTPUT\_PWM, 1309
- PF\_MODE\_SINGLE\_PIN\_CONT, 1309
- PF\_MODE\_SINGLE\_PIN\_TIME, 1309
- PF\_MODE\_TRAIN, 1309
- PF\_OUT\_A, 1310
- PF\_OUT\_B, 1310
- PF\_PIN\_C1, 1310
- PF\_PIN\_C2, 1310
- PF\_PWM\_BRAKE, 1310
- PF\_PWM\_FLOAT, 1310
- PF\_PWM\_FWD1, 1310
- PF\_PWM\_FWD2, 1310
- PF\_PWM\_FWD3, 1311
- PF\_PWM\_FWD4, 1311
- PF\_PWM\_FWD5, 1311
- PF\_PWM\_FWD6, 1311
- PF\_PWM\_FWD7, 1311
- PF\_PWM\_REV1, 1311
- PF\_PWM\_REV2, 1311
- PF\_PWM\_REV3, 1311
- PF\_PWM\_REV4, 1311
- PF\_PWM\_REV5, 1312
- PF\_PWM\_REV6, 1312
- PF\_PWM\_REV7, 1312
- PFMATE\_CHANNEL\_1, 1312
- PFMATE\_CHANNEL\_2, 1312
- PFMATE\_CHANNEL\_3, 1312
- PFMATE\_CHANNEL\_4, 1312
- PFMATE\_CMD\_GO, 1312

- PFMATE\_CMD\_RAW, 1313
- PFMATE\_MOTORS\_A, 1313
- PFMATE\_MOTORS\_B, 1313
- PFMATE\_MOTORS\_BOTH, 1313
- PFMATE\_REG\_A\_CMD, 1313
- PFMATE\_REG\_A\_SPEED, 1313
- PFMATE\_REG\_B\_CMD, 1313
- PFMATE\_REG\_B\_SPEED, 1313
- PFMATE\_REG\_CHANNEL, 1313
- PFMATE\_REG\_CMD, 1314
- PFMATE\_REG\_MOTORS, 1314
- PI, 1314
- PID\_0, 1314
- PID\_1, 1314
- PID\_2, 1314
- PID\_3, 1314
- PID\_4, 1314
- PID\_5, 1314
- PID\_6, 1315
- PID\_7, 1315
- POOL\_MAX\_SIZE, 1315
- PowerField, 1315
- PROG\_ABORT, 1315
- PROG\_ERROR, 1315
- PROG\_IDLE, 1315
- PROG\_OK, 1315
- PROG\_RESET, 1316
- PROG\_RUNNING, 1316
- PSP\_BTNSET1\_DOWN, 1316
- PSP\_BTNSET1\_L3, 1316
- PSP\_BTNSET1\_LEFT, 1316
- PSP\_BTNSET1\_R3, 1316
- PSP\_BTNSET1\_RIGHT, 1316
- PSP\_BTNSET1\_SELECT, 1317
- PSP\_BTNSET1\_START, 1317
- PSP\_BTNSET1\_UP, 1317
- PSP\_BTNSET2\_CIRCLE, 1317
- PSP\_BTNSET2\_CROSS, 1317
- PSP\_BTNSET2\_L1, 1317
- PSP\_BTNSET2\_L2, 1318
- PSP\_BTNSET2\_R1, 1318
- PSP\_BTNSET2\_R2, 1318
- PSP\_BTNSET2\_SQUARE, 1318
- PSP\_BTNSET2\_TRIANGLE, 1318
- PSP\_CMD\_ANALOG, 1319
- PSP\_CMD\_DIGITAL, 1319
- PSP\_REG\_BTNSET1, 1319
- PSP\_REG\_BTNSET2, 1319
- PSP\_REG\_XLEFT, 1319
- PSP\_REG\_XRIGHT, 1319
- PSP\_REG\_YLEFT, 1319
- PSP\_REG\_YRIGHT, 1319
- RADIANS\_PER\_DEGREE, 1319
- RAND\_MAX, 1320
- RandomEx, 1320
- RandomNumber, 1320
- RawValueField, 1320
- RC\_PROP\_BTNOFF, 1320
- RC\_PROP\_DEBUGGING, 1320
- RC\_PROP\_SLEEP\_TIMEOUT, 1320
- RC\_PROP\_SOUND\_LEVEL, 1320
- RCX\_AbsVarOp, 1321
- RCX\_AndVarOp, 1321
- RCX\_AutoOffOp, 1321
- RCX\_BatteryLevelOp, 1321
- RCX\_BatteryLevelSrc, 1321
- RCX\_BootModeOp, 1321
- RCX\_CalibrateEventOp, 1321
- RCX\_ClearAllEventsOp, 1321
- RCX\_ClearCounterOp, 1321
- RCX\_ClearMsgOp, 1321
- RCX\_ClearSensorOp, 1322
- RCX\_ClearSoundOp, 1322
- RCX\_ClearTimerOp, 1322
- RCX\_ClickCounterSrc, 1322
- RCX\_ConstantSrc, 1322
- RCX\_CounterSrc, 1322
- RCX\_DatalogOp, 1322
- RCX\_DatalogRawDirectSrc, 1323
- RCX\_DatalogRawIndirectSrc, 1323
- RCX\_DatalogSrcDirectSrc, 1323
- RCX\_DatalogSrcIndirectSrc, 1323
- RCX\_DatalogValueDirectSrc, 1323
- RCX\_DatalogValueIndirectSrc, 1323
- RCX\_DecCounterOp, 1323
- RCX\_DeleteSubOp, 1323
- RCX\_DeleteSubsOp, 1323
- RCX\_DeleteTaskOp, 1323
- RCX\_DeleteTasksOp, 1324
- RCX\_DirectEventOp, 1324

- RCX\_DisplayOp, [1324](#)
- RCX\_DivVarOp, [1324](#)
- RCX\_DurationSrc, [1324](#)
- RCX\_EventStateSrc, [1324](#)
- RCX\_FirmwareVersionSrc, [1324](#)
- RCX\_GlobalMotorStatusSrc, [1324](#)
- RCX\_GOutputDirOp, [1324](#)
- RCX\_GOutputModeOp, [1324](#)
- RCX\_GOutputPowerOp, [1325](#)
- RCX\_HysteresisSrc, [1325](#)
- RCX\_IncCounterOp, [1325](#)
- RCX\_IndirectVarSrc, [1325](#)
- RCX\_InputBooleanSrc, [1325](#)
- RCX\_InputModeOp, [1325](#)
- RCX\_InputModeSrc, [1325](#)
- RCX\_InputRawSrc, [1325](#)
- RCX\_InputTypeOp, [1325](#)
- RCX\_InputTypeSrc, [1325](#)
- RCX\_InputValueSrc, [1326](#)
- RCX\_IRModeOp, [1326](#)
- RCX\_LightOp, [1326](#)
- RCX\_LowerThresholdSrc, [1326](#)
- RCX\_LSblinkTimeOp, [1326](#)
- RCX\_LSCalibrateOp, [1326](#)
- RCX\_LSHysteresisOp, [1326](#)
- RCX\_LSLowerThreshOp, [1326](#)
- RCX\_LSupperThreshOp, [1326](#)
- RCX\_MessageOp, [1327](#)
- RCX\_MessageSrc, [1327](#)
- RCX\_MulVarOp, [1327](#)
- RCX\_MuteSoundOp, [1327](#)
- RCX\_OnOffFloatOp, [1327](#)
- RCX\_OrVarOp, [1327](#)
- RCX\_OUT\_A, [1327](#)
- RCX\_OUT\_AB, [1328](#)
- RCX\_OUT\_ABC, [1328](#)
- RCX\_OUT\_AC, [1328](#)
- RCX\_OUT\_B, [1328](#)
- RCX\_OUT\_BC, [1328](#)
- RCX\_OUT\_C, [1328](#)
- RCX\_OUT\_FLOAT, [1328](#)
- RCX\_OUT\_FULL, [1328](#)
- RCX\_OUT\_FWD, [1329](#)
- RCX\_OUT\_HALF, [1329](#)
- RCX\_OUT\_LOW, [1329](#)
- RCX\_OUT\_OFF, [1329](#)
- RCX\_OUT\_ON, [1329](#)
- RCX\_OUT\_REV, [1329](#)
- RCX\_OUT\_TOGGLE, [1329](#)
- RCX\_OutputDirOp, [1330](#)
- RCX\_OutputPowerOp, [1330](#)
- RCX\_OutputStatusSrc, [1330](#)
- RCX\_PBTurnOffOp, [1330](#)
- RCX\_PingOp, [1330](#)
- RCX\_PlaySoundOp, [1330](#)
- RCX\_PlayToneOp, [1330](#)
- RCX\_PlayToneVarOp, [1330](#)
- RCX\_PollMemoryOp, [1330](#)
- RCX\_PollOp, [1330](#)
- RCX\_ProgramSlotSrc, [1331](#)
- RCX\_RandomSrc, [1331](#)
- RCX\_RemoteKeysReleased, [1331](#)
- RCX\_RemoteOp, [1331](#)
- RCX\_RemoteOutABackward, [1331](#)
- RCX\_RemoteOutAForward, [1331](#)
- RCX\_RemoteOutBBackward, [1331](#)
- RCX\_RemoteOutBForward, [1331](#)
- RCX\_RemoteOutCBackward, [1331](#)
- RCX\_RemoteOutCForward, [1332](#)
- RCX\_RemotePBMessage1, [1332](#)
- RCX\_RemotePBMessage2, [1332](#)
- RCX\_RemotePBMessage3, [1332](#)
- RCX\_RemotePlayASound, [1332](#)
- RCX\_RemoteSelProgram1, [1332](#)
- RCX\_RemoteSelProgram2, [1332](#)
- RCX\_RemoteSelProgram3, [1332](#)
- RCX\_RemoteSelProgram4, [1332](#)
- RCX\_RemoteSelProgram5, [1333](#)
- RCX\_RemoteStopOutOff, [1333](#)
- RCX\_ScoutCounterLimitSrc, [1333](#)
- RCX\_ScoutEventFBsrc, [1333](#)
- RCX\_ScoutLightParamsSrc, [1333](#)
- RCX\_ScoutOp, [1333](#)
- RCX\_ScoutRulesOp, [1333](#)
- RCX\_ScoutRulesSrc, [1333](#)
- RCX\_ScoutTimerLimitSrc, [1333](#)
- RCX\_SelectProgramOp, [1333](#)
- RCX\_SendUARTDataOp, [1334](#)
- RCX\_SetCounterOp, [1334](#)
- RCX\_SetDatalogOp, [1334](#)
- RCX\_SetEventOp, [1334](#)
- RCX\_SetFeedbackOp, [1334](#)

- RCX\_SetPriorityOp, 1334
- RCX\_SetSourceValueOp, 1334
- RCX\_SetTimerLimitOp, 1334
- RCX\_SetVarOp, 1334
- RCX\_SetWatchOp, 1334
- RCX\_SgnVarOp, 1335
- RCX\_SoundOp, 1335
- RCX\_StartTaskOp, 1335
- RCX\_StopAllTasksOp, 1335
- RCX\_StopTaskOp, 1335
- RCX\_SubVarOp, 1335
- RCX\_SumVarOp, 1335
- RCX\_TaskEventsSrc, 1335
- RCX\_TenMSTimerSrc, 1335
- RCX\_TimerSrc, 1335
- RCX\_UARTSetupSrc, 1336
- RCX\_UnlockFirmOp, 1336
- RCX\_UnlockOp, 1336
- RCX\_UnmuteSoundOp, 1336
- RCX\_UploadDatalogOp, 1336
- RCX\_UpperThresholdSrc, 1336
- RCX\_VariableSrc, 1336
- RCX\_ViewSourceValOp, 1336
- RCX\_VLLOp, 1337
- RCX\_WatchSrc, 1337
- ReadButton, 1337
- ReadLastResponse, 1337
- ReadSemData, 1337
- RegDValueField, 1337
- RegIValueField, 1337
- RegModeField, 1337
- RegPValueField, 1338
- RESET\_ALL, 1338
- RESET\_BLOCK\_COUNT, 1338
- RESET\_BLOCKKANDTACHO,  
1338
- RESET\_COUNT, 1338
- RESET\_NONE, 1339
- RESET\_ROTATION\_COUNT,  
1339
- RFID\_MODE\_CONTINUOUS,  
1339
- RFID\_MODE\_SINGLE, 1339
- RFID\_MODE\_STOP, 1339
- RICArg, 1339
- RICImgPoint, 1340
- RICImgRect, 1340
- RICMapArg, 1340
- RICMapElement, 1341
- RICMapFunction, 1341
- RICOpCircle, 1341
- RICOpCopyBits, 1342
- RICOpDescription, 1342
- RICOpEllipse, 1342
- RICOpLine, 1343
- RICOpNumBox, 1343
- RICOpPixel, 1343
- RICOpPolygon, 1344
- RICOpRect, 1344
- RICOpSprite, 1344
- RICOpVarMap, 1345
- RICPolygonPoints, 1345
- RICSpriteData, 1346
- ROTATE\_QUEUE, 1346
- RotationCountField, 1346
- RunStateField, 1346
- SAMPLERATE\_DEFAULT, 1347
- SAMPLERATE\_MAX, 1347
- SAMPLERATE\_MIN, 1347
- ScaledValueField, 1347
- SCHAR\_MAX, 1347
- SCHAR\_MIN, 1347
- SCOUT\_FXR\_ALARM, 1347
- SCOUT\_FXR\_BUG, 1347
- SCOUT\_FXR\_NONE, 1347
- SCOUT\_FXR\_RANDOM, 1348
- SCOUT\_FXR\_SCIENCE, 1348
- SCOUT\_LIGHT\_OFF, 1348
- SCOUT\_LIGHT\_ON, 1348
- SCOUT\_LR\_AVOID, 1348
- SCOUT\_LR\_IGNORE, 1348
- SCOUT\_LR\_OFF\_WHEN, 1348
- SCOUT\_LR\_SEEK\_DARK, 1348
- SCOUT\_LR\_SEEK\_LIGHT, 1349
- SCOUT\_LR\_WAIT\_FOR, 1349
- SCOUT\_MODE\_POWER, 1349
- SCOUT\_MODE\_STANDALONE,  
1349
- SCOUT\_MR\_CIRCLE\_LEFT, 1349
- SCOUT\_MR\_CIRCLE\_RIGHT,  
1349
- SCOUT\_MR\_FORWARD, 1349

- SCOUT\_MR\_LOOP\_A, [1349](#)
- SCOUT\_MR\_LOOP\_AB, [1350](#)
- SCOUT\_MR\_LOOP\_B, [1350](#)
- SCOUT\_MR\_NO\_MOTION, [1350](#)
- SCOUT\_MR\_ZIGZAG, [1350](#)
- SCOUT\_SNDSET\_ALARM, [1350](#)
- SCOUT\_SNDSET\_BASIC, [1350](#)
- SCOUT\_SNDSET\_BUG, [1350](#)
- SCOUT\_SNDSET\_NONE, [1350](#)
- SCOUT\_SNDSET\_RANDOM, [1350](#)
- SCOUT\_SNDSET\_SCIENCE, [1350](#)
- SCOUT\_SOUND\_1\_BLINK, [1351](#)
- SCOUT\_SOUND\_2\_BLINK, [1351](#)
- SCOUT\_SOUND\_COUNTER1, [1351](#)
- SCOUT\_SOUND\_COUNTER2, [1351](#)
- SCOUT\_SOUND\_ENTER\_-  
BRIGHT, [1351](#)
- SCOUT\_SOUND\_ENTER\_DARK, [1351](#)
- SCOUT\_SOUND\_ENTER\_-  
NORMAL, [1351](#)
- SCOUT\_SOUND\_ENTERSA, [1351](#)
- SCOUT\_SOUND\_KEYERROR, [1351](#)
- SCOUT\_SOUND\_MAIL\_-  
RECEIVED, [1351](#)
- SCOUT\_SOUND\_NONE, [1352](#)
- SCOUT\_SOUND\_REMOTE, [1352](#)
- SCOUT\_SOUND\_SPECIAL1, [1352](#)
- SCOUT\_SOUND\_SPECIAL2, [1352](#)
- SCOUT\_SOUND\_SPECIAL3, [1352](#)
- SCOUT\_SOUND\_TIMER1, [1352](#)
- SCOUT\_SOUND\_TIMER2, [1352](#)
- SCOUT\_SOUND\_TIMER3, [1352](#)
- SCOUT\_SOUND\_TOUCH1\_-  
PRES, [1352](#)
- SCOUT\_SOUND\_TOUCH1\_REL, [1352](#)
- SCOUT\_SOUND\_TOUCH2\_-  
PRES, [1353](#)
- SCOUT\_SOUND\_TOUCH2\_REL, [1353](#)
- SCOUT\_TGS\_LONG, [1353](#)
- SCOUT\_TGS\_MEDIUM, [1353](#)
- SCOUT\_TGS\_SHORT, [1353](#)
- SCOUT\_TR\_AVOID, [1353](#)
- SCOUT\_TR\_IGNORE, [1353](#)
- SCOUT\_TR\_OFF\_WHEN, [1353](#)
- SCOUT\_TR\_REVERSE, [1353](#)
- SCOUT\_TR\_WAIT\_FOR, [1354](#)
- SCREEN\_BACKGROUND, [1354](#)
- SCREEN\_LARGE, [1354](#)
- SCREEN\_MODE\_CLEAR, [1354](#)
- SCREEN\_MODE\_RESTORE, [1354](#)
- SCREEN\_SMALL, [1354](#)
- SCREENS, [1354](#)
- SEC\_1, [1355](#)
- SEC\_10, [1355](#)
- SEC\_15, [1355](#)
- SEC\_2, [1355](#)
- SEC\_20, [1355](#)
- SEC\_3, [1356](#)
- SEC\_30, [1356](#)
- SEC\_4, [1356](#)
- SEC\_5, [1356](#)
- SEC\_6, [1356](#)
- SEC\_7, [1357](#)
- SEC\_8, [1357](#)
- SEC\_9, [1357](#)
- SetScreenMode, [1357](#)
- SetSleepTimeoutVal, [1357](#)
- SHRT\_MAX, [1357](#)
- SHRT\_MIN, [1357](#)
- SIZE\_OF\_BDADDR, [1358](#)
- SIZE\_OF\_BRICK\_NAME, [1358](#)
- SIZE\_OF\_BT\_CONNECT\_-  
TABLE, [1358](#)
- SIZE\_OF\_BT\_DEVICE\_TABLE, [1358](#)
- SIZE\_OF\_BT\_NAME, [1358](#)
- SIZE\_OF\_BT\_PINCODE, [1358](#)
- SIZE\_OF\_BTBUF, [1358](#)
- SIZE\_OF\_CLASS\_OF\_DEVICE, [1358](#)
- SIZE\_OF\_HSBUF, [1358](#)
- SIZE\_OF\_USBBUF, [1358](#)
- SIZE\_OF\_USBDATA, [1359](#)
- SOUND\_CLICK, [1359](#)
- SOUND\_DOUBLE\_BEEP, [1359](#)

SOUND\_DOWN, 1359  
SOUND\_FAST\_UP, 1359  
SOUND\_FLAGS\_IDLE, 1359  
SOUND\_FLAGS\_RUNNING, 1359  
SOUND\_FLAGS\_UPDATE, 1359  
SOUND\_LOW\_BEEP, 1360  
SOUND\_MODE\_LOOP, 1360  
SOUND\_MODE\_ONCE, 1360  
SOUND\_MODE\_TONE, 1360  
SOUND\_STATE\_FILE, 1360  
SOUND\_STATE\_IDLE, 1360  
SOUND\_STATE\_STOP, 1360  
SOUND\_STATE\_TONE, 1361  
SOUND\_UP, 1361  
SoundGetState, 1361  
SoundModuleID, 1361  
SoundModuleName, 1361  
SoundOffsetDuration, 1361  
SoundOffsetFlags, 1362  
SoundOffsetFreq, 1362  
SoundOffsetMode, 1362  
SoundOffsetSampleRate, 1362  
SoundOffsetSoundFilename, 1362  
SoundOffsetState, 1362  
SoundOffsetVolume, 1362  
SoundPlayFile, 1362  
SoundPlayTone, 1362  
SoundSetState, 1363  
SPECIALS, 1363  
STAT\_COMM\_PENDING, 1363  
STAT\_MSG\_EMPTY\_MAILBOX,  
1363  
STATUSICON\_BATTERY, 1363  
STATUSICON\_BLUETOOTH,  
1363  
STATUSICON\_USB, 1363  
STATUSICON\_VM, 1363  
STATUSICONS, 1363  
STATUSTEXT, 1363  
STEPICON\_1, 1364  
STEPICON\_2, 1364  
STEPICON\_3, 1364  
STEPICON\_4, 1364  
STEPICON\_5, 1364  
STEPICONS, 1364  
STEPLINE, 1364  
STOP\_REQ, 1364  
STROBE\_READ, 1364  
STROBE\_S0, 1364  
STROBE\_S1, 1365  
STROBE\_S2, 1365  
STROBE\_S3, 1365  
STROBE\_WRITE, 1365  
TachoCountField, 1365  
TachoLimitField, 1365  
TEMP\_FQ\_1, 1366  
TEMP\_FQ\_2, 1366  
TEMP\_FQ\_4, 1366  
TEMP\_FQ\_6, 1366  
TEMP\_OS\_ONESHOT, 1366  
TEMP\_POL\_HIGH, 1366  
TEMP\_POL\_LOW, 1366  
TEMP\_REG\_CONFIG, 1366  
TEMP\_REG\_TEMP, 1366  
TEMP\_REG\_THIGH, 1367  
TEMP\_REG\_TLOW, 1367  
TEMP\_RES\_10BIT, 1367  
TEMP\_RES\_11BIT, 1367  
TEMP\_RES\_12BIT, 1367  
TEMP\_RES\_9BIT, 1367  
TEMP\_SD\_CONTINUOUS, 1367  
TEMP\_SD\_SHUTDOWN, 1367  
TEMP\_TM\_COMPARATOR, 1367  
TEMP\_TM\_INTERRUPT, 1367  
TEXTLINE\_1, 1368  
TEXTLINE\_2, 1368  
TEXTLINE\_3, 1368  
TEXTLINE\_4, 1368  
TEXTLINE\_5, 1368  
TEXTLINE\_6, 1368  
TEXTLINE\_7, 1368  
TEXTLINE\_8, 1368  
TEXTLINES, 1368  
TIMES\_UP, 1369  
TONE\_A3, 1369  
TONE\_A4, 1369  
TONE\_A5, 1369  
TONE\_A6, 1369  
TONE\_A7, 1369  
TONE\_AS3, 1369  
TONE\_AS4, 1369  
TONE\_AS5, 1369

TONE\_AS6, [1370](#)  
TONE\_AS7, [1370](#)  
TONE\_B3, [1370](#)  
TONE\_B4, [1370](#)  
TONE\_B5, [1370](#)  
TONE\_B6, [1370](#)  
TONE\_B7, [1370](#)  
TONE\_C4, [1370](#)  
TONE\_C5, [1370](#)  
TONE\_C6, [1371](#)  
TONE\_C7, [1371](#)  
TONE\_CS4, [1371](#)  
TONE\_CS5, [1371](#)  
TONE\_CS6, [1371](#)  
TONE\_CS7, [1371](#)  
TONE\_D4, [1371](#)  
TONE\_D5, [1371](#)  
TONE\_D6, [1372](#)  
TONE\_D7, [1372](#)  
TONE\_DS4, [1372](#)  
TONE\_DS5, [1372](#)  
TONE\_DS6, [1372](#)  
TONE\_DS7, [1372](#)  
TONE\_E4, [1372](#)  
TONE\_E5, [1372](#)  
TONE\_E6, [1373](#)  
TONE\_E7, [1373](#)  
TONE\_F4, [1373](#)  
TONE\_F5, [1373](#)  
TONE\_F6, [1373](#)  
TONE\_F7, [1373](#)  
TONE\_FS4, [1373](#)  
TONE\_FS5, [1373](#)  
TONE\_FS6, [1373](#)  
TONE\_FS7, [1373](#)  
TONE\_G4, [1374](#)  
TONE\_G5, [1374](#)  
TONE\_G6, [1374](#)  
TONE\_G7, [1374](#)  
TONE\_GS4, [1374](#)  
TONE\_GS5, [1374](#)  
TONE\_GS6, [1374](#)  
TONE\_GS7, [1374](#)  
TOPLINE, [1375](#)  
TRAIN\_CHANNEL\_1, [1375](#)  
TRAIN\_CHANNEL\_2, [1375](#)  
TRAIN\_CHANNEL\_3, [1375](#)  
TRAIN\_CHANNEL\_ALL, [1375](#)  
TRAIN\_FUNC\_DECR\_SPEED,  
[1375](#)  
TRAIN\_FUNC\_INCR\_SPEED,  
[1375](#)  
TRAIN\_FUNC\_STOP, [1375](#)  
TRAIN\_FUNC\_TOGGLE\_LIGHT,  
[1376](#)  
TRUE, [1376](#)  
TurnRatioField, [1376](#)  
TypeField, [1376](#)  
UCHAR\_MAX, [1376](#)  
UF\_PENDING\_UPDATES, [1376](#)  
UF\_UPDATE\_MODE, [1377](#)  
UF\_UPDATE\_PID\_VALUES, [1377](#)  
UF\_UPDATE\_RESET\_BLOCK\_-  
COUNT, [1377](#)  
UF\_UPDATE\_RESET\_COUNT,  
[1377](#)  
UF\_UPDATE\_RESET\_-  
ROTATION\_COUNT, [1377](#)  
UF\_UPDATE\_SPEED, [1377](#)  
UF\_UPDATE\_TACHO\_LIMIT,  
[1377](#)  
UI\_BT\_CONNECT\_REQUEST,  
[1377](#)  
UI\_BT\_ERROR\_ATTENTION,  
[1377](#)  
UI\_BT\_PIN\_REQUEST, [1377](#)  
UI\_BT\_STATE\_CONNECTED,  
[1378](#)  
UI\_BT\_STATE\_OFF, [1378](#)  
UI\_BT\_STATE\_VISIBLE, [1378](#)  
UI\_BUTTON\_ENTER, [1378](#)  
UI\_BUTTON\_EXIT, [1378](#)  
UI\_BUTTON\_LEFT, [1378](#)  
UI\_BUTTON\_NONE, [1378](#)  
UI\_BUTTON\_RIGHT, [1378](#)  
UI\_FLAGS\_BUSY, [1379](#)  
UI\_FLAGS\_DISABLE\_EXIT, [1379](#)  
UI\_FLAGS\_DISABLE\_LEFT\_-  
RIGHT\_ENTER, [1379](#)  
UI\_FLAGS\_ENABLE\_STATUS\_-  
UPDATE, [1379](#)

- UI\_FLAGS\_EXECUTE\_LMS\_-  
FILE, 1379
- UI\_FLAGS\_REDRAW\_STATUS,  
1379
- UI\_FLAGS\_RESET\_SLEEP\_-  
TIMER, 1379
- UI\_FLAGS\_UPDATE, 1379
- UI\_STATE\_BT\_ERROR, 1379
- UI\_STATE\_CONNECT\_-  
REQUEST, 1380
- UI\_STATE\_DRAW\_MENU, 1380
- UI\_STATE\_ENTER\_PRESSED,  
1380
- UI\_STATE\_EXECUTE\_FILE, 1380
- UI\_STATE\_EXECUTING\_FILE,  
1380
- UI\_STATE\_EXIT\_PRESSED, 1380
- UI\_STATE\_INIT\_DISPLAY, 1380
- UI\_STATE\_INIT\_INTRO, 1380
- UI\_STATE\_INIT\_LOW\_-  
BATTERY, 1380
- UI\_STATE\_INIT\_MENU, 1380
- UI\_STATE\_INIT\_WAIT, 1381
- UI\_STATE\_LEFT\_PRESSED, 1381
- UI\_STATE\_LOW\_BATTERY, 1381
- UI\_STATE\_NEXT\_MENU, 1381
- UI\_STATE\_RIGHT\_PRESSED,  
1381
- UI\_STATE\_TEST\_BUTTONS,  
1381
- UI\_VM\_IDLE, 1381
- UI\_VM\_RESET1, 1381
- UI\_VM\_RESET2, 1381
- UI\_VM\_RUN\_FREE, 1382
- UI\_VM\_RUN\_PAUSE, 1382
- UI\_VM\_RUN\_SINGLE, 1382
- UIModuleID, 1382
- UIModuleName, 1382
- UINT\_MAX, 1382
- UIOffsetAbortFlag, 1382
- UIOffsetBatteryState, 1382
- UIOffsetBatteryVoltage, 1382
- UIOffsetBluetoothState, 1382
- UIOffsetButton, 1383
- UIOffsetError, 1383
- UIOffsetFlags, 1383
- UIOffsetForceOff, 1383
- UIOffsetLMSfilename, 1383
- UIOffsetOBPPPointer, 1383
- UIOffsetPMenu, 1383
- UIOffsetRechargeable, 1383
- UIOffsetRunState, 1383
- UIOffsetSleepTimeout, 1383
- UIOffsetSleepTimer, 1384
- UIOffsetState, 1384
- UIOffsetUsbState, 1384
- UIOffsetVolume, 1384
- ULONG\_MAX, 1384
- UpdateCalibCacheInfo, 1384
- UpdateFlagsField, 1384
- US\_CMD\_CONTINUOUS, 1384
- US\_CMD\_EVENTCAPTURE,  
1384
- US\_CMD\_OFF, 1385
- US\_CMD\_SINGLESHOT, 1385
- US\_CMD\_WARMRESET, 1385
- US\_REG\_ACTUAL\_ZERO, 1385
- US\_REG\_CM\_INTERVAL, 1385
- US\_REG\_FACTORY\_ACTUAL\_-  
ZERO, 1385
- US\_REG\_FACTORY\_SCALE\_-  
DIVISOR, 1385
- US\_REG\_FACTORY\_SCALE\_-  
FACTOR, 1385
- US\_REG\_MEASUREMENT\_-  
UNITS, 1385
- US\_REG\_SCALE\_DIVISOR, 1386
- US\_REG\_SCALE\_FACTOR, 1386
- USB\_CMD\_READY, 1386
- USB\_PROTOCOL\_OVERHEAD,  
1386
- USHRT\_MAX, 1386
- WriteSemData, 1386
- XG1300L\_REG\_2G, 1386
- XG1300L\_REG\_4G, 1386
- XG1300L\_REG\_8G, 1386
- XG1300L\_REG\_ANGLE, 1386
- XG1300L\_REG\_RESET, 1387
- XG1300L\_REG\_TURNRATE, 1387
- XG1300L\_REG\_XAXIS, 1387
- XG1300L\_REG\_YAXIS, 1387
- XG1300L\_REG\_ZAXIS, 1387

- XG1300L\_SCALE\_2G, [1387](#)
- XG1300L\_SCALE\_4G, [1387](#)
- XG1300L\_SCALE\_8G, [1388](#)
- NBCInputPortConstants
  - IN\_1, [756](#)
  - IN\_2, [756](#)
  - IN\_3, [756](#)
  - IN\_4, [756](#)
- NBCSensorModeConstants
  - IN\_MODE\_ANGLESTEP, [760](#)
  - IN\_MODE\_BOOLEAN, [760](#)
  - IN\_MODE\_CELSIUS, [760](#)
  - IN\_MODE\_FAHRENHEIT, [760](#)
  - IN\_MODE\_MODEMASK, [760](#)
  - IN\_MODE\_PCTFULLSCALE, [761](#)
  - IN\_MODE\_PERIODCOUNTER, [761](#)
  - IN\_MODE\_RAW, [761](#)
  - IN\_MODE\_SLOPEMASK, [761](#)
  - IN\_MODE\_TRANSITIONCNT, [761](#)
- NBCSensorTypeConstants
  - IN\_TYPE\_ANGLE, [757](#)
  - IN\_TYPE\_COLORBLUE, [757](#)
  - IN\_TYPE\_COLOREXIT, [758](#)
  - IN\_TYPE\_COLORFULL, [758](#)
  - IN\_TYPE\_COLORGREEN, [758](#)
  - IN\_TYPE\_COLORNONE, [758](#)
  - IN\_TYPE\_COLORRED, [758](#)
  - IN\_TYPE\_CUSTOM, [758](#)
  - IN\_TYPE\_HISPEED, [758](#)
  - IN\_TYPE\_LIGHT\_ACTIVE, [758](#)
  - IN\_TYPE\_LIGHT\_INACTIVE, [758](#)
  - IN\_TYPE\_LOWSPEED, [758](#)
  - IN\_TYPE\_LOWSPEED\_9V, [759](#)
  - IN\_TYPE\_NO\_SENSOR, [759](#)
  - IN\_TYPE\_REFLECTION, [759](#)
  - IN\_TYPE\_SOUND\_DB, [759](#)
  - IN\_TYPE\_SOUND\_DBA, [759](#)
  - IN\_TYPE\_SWITCH, [759](#)
  - IN\_TYPE\_TEMPERATURE, [759](#)
- NEQ
  - cmpconst, [449](#)
  - NXCDefs.h, [1489](#)
- NewFilename
  - FileRenameType, [1064](#)
- NewSize
  - FileResizeType, [1065](#)
- NewVal
  - WriteSemDataType, [1113](#)
- NO\_ERR
  - CommandModuleConstants, [51](#)
  - NBCCommon.h, [1284](#)
- NO\_OF\_BTNS
  - ButtonNameConstants, [742](#)
  - NBCCommon.h, [1284](#)
- NoRestartOnRead
  - CommLSWriteExType, [1033](#)
- NormalizedArray
  - ColorSensorReadType, [1019](#)
- NormalizedValue
  - InputValuesType, [1073](#)
- NormalizedValueField
  - InputFieldConstants, [762](#)
  - NBCCommon.h, [1285](#)
- NRLink2400
  - MindSensorsAPI, [191](#)
  - NXCDefs.h, [1703](#)
- NRLink4800
  - MindSensorsAPI, [191](#)
  - NXCDefs.h, [1703](#)
- NRLINK\_CMD\_2400
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_4800
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_FLUSH
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_IR\_LONG
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_IR\_SHORT
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_RUN\_MACRO
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)
- NRLINK\_CMD\_SET\_PF
  - MSNRLink, [947](#)
  - NBCCommon.h, [1285](#)

- NRLINK\_CMD\_SET\_RCX
  - MSNRLink, 947
  - NBCCCommon.h, 1285
- NRLINK\_CMD\_SET\_TRAIN
  - MSNRLink, 947
  - NBCCCommon.h, 1285
- NRLINK\_CMD\_TX\_RAW
  - MSNRLink, 947
  - NBCCCommon.h, 1286
- NRLINK\_REG\_BYTES
  - MSNRLink, 948
  - NBCCCommon.h, 1286
- NRLINK\_REG\_DATA
  - MSNRLink, 948
  - NBCCCommon.h, 1286
- NRLINK\_REG\_EEPROM
  - MSNRLink, 948
  - NBCCCommon.h, 1286
- NRLinkFlush
  - MindSensorsAPI, 192
  - NXCDefs.h, 1704
- NRLinkIRLong
  - MindSensorsAPI, 192
  - NXCDefs.h, 1704
- NRLinkIRShort
  - MindSensorsAPI, 193
  - NXCDefs.h, 1705
- NRLinkSetPF
  - MindSensorsAPI, 193
  - NXCDefs.h, 1705
- NRLinkSetRCX
  - MindSensorsAPI, 194
  - NXCDefs.h, 1706
- NRLinkSetTrain
  - MindSensorsAPI, 194
  - NXCDefs.h, 1706
- NRLinkStatus
  - MindSensorsAPI, 194
  - NXCDefs.h, 1706
- NRLinkTxRaw
  - MindSensorsAPI, 195
  - NXCDefs.h, 1707
- NULL
  - LoaderModuleConstants, 717
  - NBCCCommon.h, 1286
- NumOut
  - DisplayModuleFunctions, 362
  - NXCDefs.h, 1707
- NumToStr
  - cstringAPI, 664
  - NXCDefs.h, 1708
- NXCAPIDocs.h, 1388
- NXCDefs.h, 1389
  - \_SENSOR\_CFG, 1480
  - abort, 1503
  - AbortFlag, 1504
  - abs, 1504
  - ACCLNxCalibrateX, 1504
  - ACCLNxCalibrateXEnd, 1505
  - ACCLNxCalibrateY, 1505
  - ACCLNxCalibrateYEnd, 1506
  - ACCLNxCalibrateZ, 1506
  - ACCLNxCalibrateZEnd, 1507
  - ACCLNxResetCalibration, 1507
  - ACCLNxSensitivity, 1507
  - ACCLNxXOffset, 1508
  - ACCLNxXRange, 1508
  - ACCLNxYOffset, 1509
  - ACCLNxYRange, 1509
  - ACCLNxZOffset, 1510
  - ACCLNxZRange, 1510
  - Acos, 1480
  - acos, 1510
  - AcosD, 1480
  - acosd, 1511
  - Acquire, 1511
  - addressOf, 1512
  - addressOfEx, 1512
  - ArrayBuild, 1513
  - ArrayIndex, 1513
  - ArrayInit, 1514
  - ArrayLen, 1514
  - ArrayMax, 1515
  - ArrayMean, 1516
  - ArrayMin, 1516
  - ArrayOp, 1517
  - ArrayReplace, 1518
  - ArraySort, 1518
  - ArrayStd, 1519
  - ArraySubset, 1520
  - ArraySum, 1520
  - ArraySumSqr, 1521

- Asin, 1481
- asin, 1521
- AsinD, 1481
- asind, 1522
- Atan, 1481
- atan, 1522
- Atan2, 1482
- atan2, 1523
- Atan2D, 1482
- atan2d, 1524
- AtanD, 1483
- atand, 1524
- atof, 1525
- atoi, 1526
- atol, 1526
- BatteryLevel, 1527
- BatteryState, 1527
- bcd2dec, 1527
- BluetoothState, 1528
- BluetoothStatus, 1528
- BluetoothWrite, 1529
- BranchComp, 1529
- BranchTest, 1530
- BrickDataBluecoreVersion, 1530
- BrickDataBtHardwareStatus, 1530
- BrickDataBtStateStatus, 1531
- BrickDataName, 1531
- BrickDataTimeoutValue, 1531
- BTConnectionClass, 1532
- BTConnectionHandleNum, 1532
- BTConnectionLinkQuality, 1532
- BTConnectionName, 1533
- BTConnectionPinCode, 1533
- BTConnectionStreamStatus, 1534
- BTDataMode, 1534
- BTDeviceClass, 1534
- BTDeviceCount, 1535
- BTDeviceName, 1535
- BTDeviceNameCount, 1535
- BTDeviceStatus, 1536
- BTInputBufferInPtr, 1536
- BTInputBufferOutPtr, 1536
- BTOutputBufferInPtr, 1537
- BTOutputBufferOutPtr, 1537
- ButtonCount, 1537
- ButtonLongPressCount, 1538
- ButtonLongReleaseCount, 1538
- ButtonPressCount, 1539
- ButtonPressed, 1539
- ButtonReleaseCount, 1539
- ButtonShortReleaseCount, 1540
- ButtonState, 1540
- ByteArrayToStr, 1541
- ByteArrayToStrEx, 1541
- Ceil, 1483
- ceil, 1541
- CircleOut, 1542
- ClearLine, 1542
- ClearScreen, 1543
- ClearSensor, 1543
- CloseFile, 1543
- Coast, 1544
- CoastEx, 1544
- ColorADRaw, 1545
- ColorBoolean, 1545
- ColorCalibration, 1546
- ColorCalibrationState, 1546
- ColorCalLimits, 1547
- ColorSensorRaw, 1547
- ColorSensorValue, 1548
- CommandFlags, 1548
- ConfigureTemperatureSensor, 1549
- Copy, 1549
- Cos, 1483
- cos, 1550
- CosD, 1484
- cosd, 1550
- Cosh, 1484
- cosh, 1551
- CoshD, 1485
- coshd, 1551
- CreateFile, 1552
- CreateFileLinear, 1552
- CreateFileNonLinear, 1553
- CurrentTick, 1553
- CustomSensorActiveStatus, 1554
- CustomSensorPercentFullScale, 1554
- CustomSensorZeroOffset, 1554
- DeleteFile, 1555
- DisplayContrast, 1555
- DisplayDisplay, 1556

DisplayEraseMask, 1556  
DisplayFlags, 1556  
DisplayFont, 1557  
DisplayTextLinesCenterFlags, 1557  
DisplayUpdateMask, 1557  
DISTNxDistance, 1558  
DISTN×GP2D12, 1558  
DISTN×GP2D120, 1558  
DISTN×GP2YA02, 1559  
DISTN×GP2YA21, 1559  
DISTN×MaxDistance, 1560  
DISTN×MinDistance, 1560  
DISTN×ModuleType, 1561  
DISTN×NumPoints, 1561  
DISTN×Voltage, 1562  
div, 1562  
EllipseOut, 1563  
EQ, 1485  
ExitTo, 1563  
Exp, 1485  
exp, 1564  
fclose, 1564  
feof, 1565  
fflush, 1565  
fgetc, 1565  
fgets, 1566  
FindFirstFile, 1566  
FindNextFile, 1567  
FirstTick, 1567  
Flatten, 1568  
FlattenVar, 1568  
Float, 1569  
Floor, 1486  
floor, 1569  
Follows, 1569  
FontNumOut, 1570  
FontTextOut, 1571  
fopen, 1571  
ForceOff, 1572  
FormatNum, 1572  
fprintf, 1573  
fputc, 1573  
fputs, 1574  
Frac, 1486  
frac, 1574  
FreeMemory, 1575  
fseek, 1575  
ftell, 1576  
GetBrickDataAddress, 1576  
GetBTConnectionAddress, 1576  
GetBTDeviceAddress, 1577  
GetBTInputBuffer, 1577  
GetBTOutputBuffer, 1578  
GetButtonModuleValue, 1578  
getc, 1486  
getchar, 1578  
GetCommandModuleBytes, 1579  
GetCommandModuleValue, 1579  
GetCommModuleBytes, 1580  
GetCommModuleValue, 1580  
GetDisplayModuleBytes, 1580  
GetDisplayModuleValue, 1581  
GetDisplayNormal, 1581  
GetDisplayPopup, 1582  
GetHSInputBuffer, 1582  
GetHSOutputBuffer, 1583  
GetInput, 1583  
GetInputModuleValue, 1583  
GetIOMapBytes, 1584  
GetIOMapBytesByID, 1584  
GetIOMapValue, 1585  
GetIOMapValueByID, 1585  
GetLastResponseInfo, 1586  
GetLoaderModuleValue, 1586  
GetLowSpeedModuleBytes, 1587  
GetLowSpeedModuleValue, 1587  
GetLSInputBuffer, 1587  
GetLSOutputBuffer, 1588  
GetMemoryInfo, 1588  
GetOutput, 1589  
GetOutputModuleValue, 1589  
GetSoundModuleValue, 1590  
GetUIModuleValue, 1590  
GetUSBInputBuffer, 1590  
GetUSBOutputBuffer, 1591  
GetUSBPollBuffer, 1591  
glAddToAngleX, 1592  
glAddToAngleY, 1592  
glAddToAngleZ, 1592  
glAddVertex, 1593  
glBegin, 1593  
glBeginObject, 1593

glBeginRender, 1593  
glBox, 1594  
glCallObject, 1594  
glCos32768, 1594  
glCube, 1595  
glEnd, 1595  
glEndObject, 1595  
glFinishRender, 1595  
glInit, 1596  
glObjectAction, 1596  
glPyramid, 1596  
glSet, 1597  
glSetAngleX, 1597  
glSetAngleY, 1597  
glSetAngleZ, 1598  
glSin32768, 1598  
GraphicArrayOut, 1598  
GraphicArrayOutEx, 1599  
GraphicOut, 1600  
GraphicOutEx, 1600  
GT, 1487  
GTEQ, 1487  
HSAddress, 1601  
HSDataMode, 1601  
HSFlags, 1601  
HSInputBufferInPtr, 1602  
HSInputBufferOutPtr, 1602  
HSMMode, 1602  
HSOutputBufferInPtr, 1603  
HSOutputBufferOutPtr, 1603  
HSSpeed, 1603  
HSState, 1604  
HTIRTrain, 1604  
HTPFComboDirect, 1605  
HTPFComboPWM, 1605  
HTPFRawOutput, 1606  
HTPFRepeat, 1606  
HTPFSingleOutputCST, 1607  
HTPFSingleOutputPWM, 1608  
HTPFSinglePin, 1608  
HTPFTrain, 1609  
HTRCXAddToDatalog, 1610  
HTRCXBatteryLevel, 1610  
HTRCXCLEARAllEvents, 1610  
HTRCXCLEARCounter, 1610  
HTRCXCLEARMsg, 1611  
HTRCXCLEARSensor, 1611  
HTRCXCLEARSound, 1611  
HTRCXCLEARTimer, 1611  
HTRCXCreateDatalog, 1612  
HTRCXDecCounter, 1612  
HTRCXDeleteSub, 1612  
HTRCXDeleteSubs, 1613  
HTRCXDeleteTask, 1613  
HTRCXDeleteTasks, 1613  
HTRCXDisableOutput, 1613  
HTRCXEnableOutput, 1614  
HTRCXEvent, 1614  
HTRCXFloat, 1614  
HTRCXFwd, 1615  
HTRCXIncCounter, 1615  
HTRCXInvertOutput, 1615  
HTRCXMuteSound, 1616  
HTRCXObvertOutput, 1616  
HTRCXOff, 1616  
HTRCXOn, 1617  
HTRCXOnFor, 1617  
HTRCXOnFwd, 1617  
HTRCXOnRev, 1618  
HTRCXPBTurnOff, 1618  
HTRCXPing, 1618  
HTRCXPlaySound, 1618  
HTRCXPlayTone, 1619  
HTRCXPlayToneVar, 1619  
HTRCXPoll, 1619  
HTRCXPollMemory, 1620  
HTRCXRemote, 1620  
HTRCXRev, 1620  
HTRCXSelectDisplay, 1621  
HTRCXSelectProgram, 1621  
HTRCXSendSerial, 1621  
HTRCXSetDirection, 1622  
HTRCXSetEvent, 1622  
HTRCXSetGlobalDirection, 1623  
HTRCXSetGlobalOutput, 1623  
HTRCXSetIRLinkPort, 1623  
HTRCXSetMaxPower, 1624  
HTRCXSetMessage, 1624  
HTRCXSetOutput, 1624  
HTRCXSetPower, 1625  
HTRCXSetPriority, 1625  
HTRCXSetSensorMode, 1625

HTRCXSetSensorType, 1626  
HTRCXSetSleepTime, 1626  
HTRCXSetTxPower, 1626  
HTRCXSetWatch, 1627  
HTRCXStartTask, 1627  
HTRCXStopAllTasks, 1627  
HTRCXStopTask, 1627  
HTRCXToggle, 1628  
HTRCXUnmuteSound, 1628  
HTScoutCalibrateSensor, 1628  
HTScoutMuteSound, 1628  
HTScoutSelectSounds, 1629  
HTScoutSendVLL, 1629  
HTScoutSetEventFeedback, 1629  
HTScoutSetLight, 1630  
HTScoutSetScoutMode, 1630  
HTScoutSetSensorClickTime, 1630  
HTScoutSetSensorHysteresis, 1631  
HTScoutSetSensorLowerLimit,  
1631  
HTScoutSetSensorUpperLimit,  
1631  
HTScoutUnmuteSound, 1632  
I2CBytes, 1632  
I2CBytesReady, 1633  
I2CCheckStatus, 1634  
I2CDeviceId, 1634  
I2CDeviceInfo, 1635  
I2CRead, 1635  
I2CSendCommand, 1636  
I2CStatus, 1637  
I2CVendorId, 1637  
I2CVersion, 1638  
I2CWrite, 1638  
isalnum, 1639  
isalpha, 1639  
iscntrl, 1640  
isdigit, 1640  
isgraph, 1641  
islower, 1641  
isNAN, 1641  
isprint, 1642  
ispunct, 1642  
isspace, 1643  
isupper, 1643  
isxdigit, 1643  
JoystickMessageRead, 1644  
labs, 1644  
ldiv, 1645  
LeftStr, 1645  
LineOut, 1645  
Log, 1487  
log, 1646  
Log10, 1488  
log10, 1647  
LongAbort, 1647  
LowSpeedBytesReady, 1648  
LowSpeedCheckStatus, 1648  
LowSpeedRead, 1649  
LowSpeedStatus, 1650  
LowSpeedWrite, 1650  
LSChannelState, 1651  
LSErrorType, 1652  
LSInputBufferBytesToRx, 1652  
LSInputBufferInPtr, 1652  
LSInputBufferOutPtr, 1653  
LSMode, 1653  
LSNoRestartOnRead, 1654  
LSOutputBufferBytesToRx, 1654  
LSOutputBufferInPtr, 1654  
LSOutputBufferOutPtr, 1655  
LSSpeed, 1655  
LSState, 1655  
LT, 1488  
LTEQ, 1488  
memcmp, 1656  
memcpy, 1656  
memmove, 1656  
MidStr, 1657  
MotorActualSpeed, 1657  
MotorBlockTachoCount, 1658  
MotorMaxAcceleration, 1658  
MotorMaxSpeed, 1659  
MotorMode, 1659  
MotorOutputOptions, 1659  
MotorOverload, 1660  
MotorPower, 1660  
MotorPwnFreq, 1661  
MotorRegDValue, 1661  
MotorRegIValue, 1661  
MotorRegPValue, 1662  
MotorRegulation, 1662

MotorRegulationOptions, 1663  
MotorRegulationTime, 1663  
MotorRotationCount, 1663  
MotorRunState, 1664  
MotorTachoCount, 1664  
MotorTachoLimit, 1665  
MotorTurnRatio, 1665  
MSADPAMute, 1666  
MSADPAOn, 1666  
MSDeenergize, 1666  
MSEnergize, 1667  
MSIRTrain, 1667  
MSPFComboDirect, 1668  
MSPFComboPWM, 1668  
MSPFRawOutput, 1669  
MSPFRepeat, 1670  
MSPFSingleOutputCST, 1670  
MSPFSingleOutputPWM, 1671  
MSPFSinglePin, 1672  
MSPFTrain, 1672  
MSRCXAbsVar, 1673  
MSRCXAddToDatalog, 1673  
MSRCXAndVar, 1674  
MSRCXBatteryLevel, 1674  
MSRCXBoot, 1674  
MSRCXCalibrateEvent, 1675  
MSRCXClearAllEvents, 1675  
MSRCXClearCounter, 1675  
MSRCXClearMsg, 1676  
MSRCXClearSensor, 1676  
MSRCXClearSound, 1676  
MSRCXClearTimer, 1676  
MSRCXCreateDatalog, 1677  
MSRCXDecCounter, 1677  
MSRCXDeleteSub, 1677  
MSRCXDeleteSubs, 1678  
MSRCXDeleteTask, 1678  
MSRCXDeleteTasks, 1678  
MSRCXDisableOutput, 1678  
MSRCXDivVar, 1679  
MSRCXEnableOutput, 1679  
MSRCXEvent, 1679  
MSRCXFloat, 1680  
MSRCXFwd, 1680  
MSRCXIncCounter, 1680  
MSRCXInvertOutput, 1681  
MSRCXMulVar, 1681  
MSRCXMuteSound, 1681  
MSRCXObvertOutput, 1682  
MSRCXOff, 1682  
MSRCXOn, 1682  
MSRCXOnFor, 1683  
MSRCXOnFwd, 1683  
MSRCXOnRev, 1683  
MSRCXOrVar, 1684  
MSRCXPBTurnOff, 1684  
MSRCXPing, 1684  
MSRCXPlaySound, 1684  
MSRCXPlayTone, 1685  
MSRCXPlayToneVar, 1685  
MSRCXPoll, 1685  
MSRCXPollMemory, 1686  
MSRCXRemote, 1686  
MSRCXReset, 1687  
MSRCXRev, 1687  
MSRCXSelectDisplay, 1687  
MSRCXSelectProgram, 1687  
MSRCXSendSerial, 1688  
MSRCXSet, 1688  
MSRCXSetDirection, 1688  
MSRCXSetEvent, 1689  
MSRCXSetGlobalDirection, 1689  
MSRCXSetGlobalOutput, 1690  
MSRCXSetMaxPower, 1690  
MSRCXSetMessage, 1690  
MSRCXSetNRLinkPort, 1691  
MSRCXSetOutput, 1691  
MSRCXSetPower, 1691  
MSRCXSetPriority, 1692  
MSRCXSetSensorMode, 1692  
MSRCXSetSensorType, 1692  
MSRCXSetSleepTime, 1693  
MSRCXSetTxPower, 1693  
MSRCXSetUserDisplay, 1693  
MSRCXSetVar, 1694  
MSRCXSetWatch, 1694  
MSRCXSignVar, 1695  
MSRCXStartTask, 1695  
MSRCXStopAllTasks, 1695  
MSRCXStopTask, 1695  
MSRCXSubVar, 1696  
MSRCXSumVar, 1696

- MSRCXToggle, [1696](#)
- MSRCXUnlock, [1697](#)
- MSRCXUnmuteSound, [1697](#)
- MSReadValue, [1697](#)
- MSScoutCalibrateSensor, [1698](#)
- MSScoutMuteSound, [1698](#)
- MSScoutSelectSounds, [1698](#)
- MSScoutSendVLL, [1698](#)
- MSScoutSetCounterLimit, [1699](#)
- MSScoutSetEventFeedback, [1699](#)
- MSScoutSetLight, [1699](#)
- MSScoutSetScoutMode, [1700](#)
- MSScoutSetScoutRules, [1700](#)
- MSScoutSetSensorClickTime, [1700](#)
- MSScoutSetSensorHysteresis, [1701](#)
- MSScoutSetSensorLowerLimit, [1701](#)
- MSScoutSetSensorUpperLimit, [1702](#)
- MSScoutSetTimerLimit, [1702](#)
- MSScoutUnmuteSound, [1702](#)
- MulDiv32, [1488](#)
- muldiv32, [1702](#)
- NEQ, [1489](#)
- NRLink2400, [1703](#)
- NRLink4800, [1703](#)
- NRLinkFlush, [1704](#)
- NRLinkIRLong, [1704](#)
- NRLinkIRShort, [1705](#)
- NRLinkSetPF, [1705](#)
- NRLinkSetRCX, [1706](#)
- NRLinkSetTrain, [1706](#)
- NRLinkStatus, [1706](#)
- NRLinkTxRaw, [1707](#)
- NumOut, [1707](#)
- NumToStr, [1708](#)
- NXTHIDAsciiMode, [1709](#)
- NXTHIDDirectMode, [1709](#)
- NXTHIDLoadCharacter, [1710](#)
- NXTHIDTransmit, [1710](#)
- NXTLineLeaderAverage, [1711](#)
- NXTLineLeaderCalibrateBlack, [1711](#)
- NXTLineLeaderCalibrateWhite, [1712](#)
- NXTLineLeaderInvert, [1712](#)
- NXTLineLeaderPowerDown, [1713](#)
- NXTLineLeaderPowerUp, [1713](#)
- NXTLineLeaderReset, [1714](#)
- NXTLineLeaderResult, [1714](#)
- NXTLineLeaderSnapshot, [1715](#)
- NXTLineLeaderSteering, [1715](#)
- NXTPowerMeterCapacityUsed, [1716](#)
- NXTPowerMeterElapsedTime, [1716](#)
- NXTPowerMeterErrorCount, [1717](#)
- NXTPowerMeterMaxCurrent, [1717](#)
- NXTPowerMeterMaxVoltage, [1718](#)
- NXTPowerMeterMinCurrent, [1718](#)
- NXTPowerMeterMinVoltage, [1719](#)
- NXTPowerMeterPresentCurrent, [1719](#)
- NXTPowerMeterPresentPower, [1719](#)
- NXTPowerMeterPresentVoltage, [1720](#)
- NXTPowerMeterResetCounters, [1720](#)
- NXTPowerMeterTotalPowerConsumed, [1721](#)
- NXTServoBatteryVoltage, [1721](#)
- NXTServoEditMacro, [1722](#)
- NXTServoGotoMacroAddress, [1722](#)
- NXTServoHaltMacro, [1723](#)
- NXTServoInit, [1723](#)
- NXTServoPauseMacro, [1724](#)
- NXTServoPosition, [1724](#)
- NXTServoQuitEdit, [1725](#)
- NXTServoReset, [1725](#)
- NXTServoResumeMacro, [1726](#)
- NXTServoSpeed, [1726](#)
- Off, [1727](#)
- OffEx, [1727](#)
- OnBrickProgramPointer, [1728](#)
- OnFwd, [1728](#)
- OnFwdEx, [1728](#)
- OnFwdReg, [1729](#)
- OnFwdRegEx, [1729](#)
- OnFwdRegExPID, [1730](#)
- OnFwdRegPID, [1731](#)
- OnFwdSync, [1731](#)
- OnFwdSyncEx, [1732](#)

- OnFwdSyncExPID, 1732
- OnFwdSyncPID, 1733
- OnRev, 1734
- OnRevEx, 1734
- OnRevReg, 1734
- OnRevRegEx, 1735
- OnRevRegExPID, 1735
- OnRevRegPID, 1736
- OnRevSync, 1737
- OnRevSyncEx, 1737
- OnRevSyncExPID, 1738
- OnRevSyncPID, 1738
- OpenFileAppend, 1739
- OpenFileRead, 1740
- OpenFileReadLinear, 1740
- PFMateSend, 1741
- PFMateSendRaw, 1741
- PlayFile, 1742
- PlayFileEx, 1742
- PlaySound, 1743
- PlayTone, 1743
- PlayToneEx, 1744
- PlayTones, 1744
- PointOut, 1745
- PolyOut, 1745
- Pos, 1746
- PosRegAddAngle, 1746
- PosRegEnable, 1747
- PosRegSetAngle, 1748
- PosRegSetMax, 1748
- Pow, 1489
- pow, 1749
- PowerDown, 1749
- Precedes, 1749
- printf, 1750
- PSPNxAnalog, 1750
- PSPNxDigital, 1751
- putc, 1489
- rand, 1751
- Random, 1751
- Read, 1752
- ReadButtonEx, 1752
- ReadBytes, 1753
- ReadI2CRegister, 1753
- ReadLn, 1754
- ReadLnString, 1754
- ReadNRLinkBytes, 1755
- ReadSensorColorEx, 1755
- ReadSensorColorRaw, 1756
- ReadSensorDIAccl, 1757
- ReadSensorDIAccl8, 1757
- ReadSensorDIAccl8Raw, 1757
- ReadSensorDIAcclDrift, 1758
- ReadSensorDIAcclRaw, 1758
- ReadSensorDIGyro, 1759
- ReadSensorDIGyroRaw, 1759
- ReadSensorEMeter, 1760
- ReadSensorHTAccel, 1761
- ReadSensorHTAngle, 1761
- ReadSensorHTBarometric, 1762
- ReadSensorHTColor, 1762
- ReadSensorHTColor2Active, 1763
- ReadSensorHTIRReceiver, 1763
- ReadSensorHTIRReceiverEx, 1764
- ReadSensorHTIRSeeker, 1764
- ReadSensorHTIRSeeker2AC, 1765
- ReadSensorHTIRSeeker2DC, 1766
- ReadSensorHTNormalizedColor, 1766
- ReadSensorHTNormalized-Color2Active, 1767
- ReadSensorHTProtoAllAnalog, 1768
- ReadSensorHTRawColor, 1768
- ReadSensorHTRawColor2, 1769
- ReadSensorHTSuperProAllAnalog, 1769
- ReadSensorHTSuperProAnalogOut, 1770
- ReadSensorHTTouchMultiplexer, 1771
- ReadSensorMIXG1300L, 1771
- ReadSensorMSAccel, 1772
- ReadSensorMSPlayStation, 1772
- ReadSensorMSRTClock, 1773
- ReadSensorMSTilt, 1774
- ReadSensorUSEx, 1774
- RebootInFirmwareMode, 1775
- ReceiveMessage, 1775
- ReceiveRemoteBool, 1775
- ReceiveRemoteMessageEx, 1776
- ReceiveRemoteNumber, 1776

- ReceiveRemoteString, 1777
- RechargeableBattery, 1777
- RectOut, 1778
- reladdressOf, 1778
- Release, 1779
- RemoteBluetoothFactoryReset, 1779
- RemoteCloseFile, 1780
- RemoteConnectionIdle, 1780
- RemoteConnectionWrite, 1781
- RemoteDatalogRead, 1781
- RemoteDatalogSetTimes, 1782
- RemoteDeleteFile, 1783
- RemoteDeleteUserFlash, 1783
- RemoteFindFirstFile, 1784
- RemoteFindNextFile, 1784
- RemoteGetBatteryLevel, 1785
- RemoteGetBluetoothAddress, 1785
- RemoteGetConnectionCount, 1786
- RemoteGetConnectionName, 1787
- RemoteGetContactCount, 1787
- RemoteGetContactName, 1788
- RemoteGetCurrentProgramName, 1788
- RemoteGetDeviceInfo, 1789
- RemoteGetFirmwareVersion, 1789
- RemoteGetInputValues, 1790
- RemoteGetOutputState, 1791
- RemoteGetProperty, 1791
- RemoteIOMapRead, 1792
- RemoteIOMapWriteBytes, 1793
- RemoteIOMapWriteValue, 1793
- RemoteKeepAlive, 1794
- RemoteLowspeedGetStatus, 1794
- RemoteLowspeedRead, 1795
- RemoteLowspeedWrite, 1795
- RemoteMessageRead, 1796
- RemoteMessageWrite, 1797
- RemoteOpenAppendData, 1797
- RemoteOpenRead, 1798
- RemoteOpenWrite, 1798
- RemoteOpenWriteData, 1799
- RemoteOpenWriteLinear, 1800
- RemotePlaySoundFile, 1800
- RemotePlayTone, 1801
- RemotePollCommand, 1801
- RemotePollCommandLength, 1802
- RemoteRead, 1803
- RemoteRenameFile, 1803
- RemoteResetMotorPosition, 1804
- RemoteResetScaledValue, 1805
- RemoteResetTachoCount, 1805
- RemoteSetBrickName, 1806
- RemoteSetInputMode, 1806
- RemoteSetOutputState, 1807
- RemoteSetProperty, 1807
- RemoteStartProgram, 1808
- RemoteStopProgram, 1808
- RemoteStopSound, 1809
- RemoteWrite, 1809
- remove, 1810
- rename, 1810
- RenameFile, 1811
- ResetAllTachoCounts, 1811
- ResetBlockTachoCount, 1812
- ResetHTBarometricCalibration, 1812
- ResetMIXG1300L, 1812
- ResetRotationCount, 1813
- ResetScreen, 1813
- ResetSensor, 1814
- ResetSensorHTAngle, 1814
- ResetSleepTimer, 1814
- ResetTachoCount, 1815
- ResizeFile, 1815
- ResolveHandle, 1816
- rewind, 1816
- RFIDInit, 1816
- RFIDMode, 1817
- RFIDRead, 1817
- RFIDReadContinuous, 1818
- RFIDReadSingle, 1818
- RFIDStatus, 1819
- RFIDStop, 1819
- RICSetValue, 1490
- RightStr, 1819
- RotateMotor, 1820
- RotateMotorEx, 1820
- RotateMotorExPID, 1821
- RotateMotorPID, 1822
- RS485Control, 1822
- RS485DataAvailable, 1823

- RS485Disable, 1823
- RS485Enable, 1823
- RS485Initialize, 1824
- RS485Read, 1824
- RS485ReadEx, 1825
- RS485SendingData, 1825
- RS485Status, 1825
- RS485Uart, 1826
- RS485Write, 1826
- RunNRLinkMacro, 1827
- S1, 1490
- s16, 1492
- S2, 1492
- S3, 1492
- s32, 1493
- S4, 1493
- s8, 1493
- SEEK\_CUR, 1493
- SEEK\_END, 1493
- SEEK\_SET, 1493
- SendMessage, 1827
- SendRemoteBool, 1828
- SendRemoteNumber, 1828
- SendRemoteString, 1829
- SendResponseBool, 1829
- SendResponseNumber, 1830
- SendResponseString, 1830
- SendRS485Bool, 1831
- SendRS485Number, 1831
- SendRS485String, 1831
- Sensor, 1832
- SENSOR\_1, 1493
- SENSOR\_2, 1493
- SENSOR\_3, 1494
- SENSOR\_4, 1494
- SENSOR\_CELSIUS, 1494
- SENSOR\_COLORBLUE, 1494
- SENSOR\_COLORFULL, 1494
- SENSOR\_COLORGREEN, 1494
- SENSOR\_COLORNONE, 1494
- SENSOR\_COLORRED, 1494
- SENSOR\_EDGE, 1495
- SENSOR\_FAHRENHEIT, 1495
- SENSOR\_LIGHT, 1495
- SENSOR\_LOWSPEED, 1495
- SENSOR\_LOWSPEED\_9V, 1495
- SENSOR\_MODE\_BOOL, 1495
- SENSOR\_MODE\_CELSIUS, 1495
- SENSOR\_MODE\_EDGE, 1495
- SENSOR\_MODE\_FAHRENHEIT, 1496
- SENSOR\_MODE\_PERCENT, 1496
- SENSOR\_MODE\_PULSE, 1496
- SENSOR\_MODE\_RAW, 1496
- SENSOR\_MODE\_ROTATION, 1496
- SENSOR\_NXTLIGHT, 1496
- SENSOR\_PULSE, 1496
- SENSOR\_ROTATION, 1496
- SENSOR\_SOUND, 1497
- SENSOR\_TOUCH, 1497
- SENSOR\_TYPE\_COLORBLUE, 1497
- SENSOR\_TYPE\_COLORFULL, 1497
- SENSOR\_TYPE\_COLORGREEN, 1497
- SENSOR\_TYPE\_COLORNONE, 1497
- SENSOR\_TYPE\_COLORRED, 1497
- SENSOR\_TYPE\_CUSTOM, 1497
- SENSOR\_TYPE\_HIGHSPEED, 1498
- SENSOR\_TYPE\_LIGHT, 1498
- SENSOR\_TYPE\_LIGHT\_ACTIVE, 1498
- SENSOR\_TYPE\_LIGHT\_-  
INACTIVE, 1498
- SENSOR\_TYPE\_LOWSPEED, 1498
- SENSOR\_TYPE\_LOWSPEED\_9V, 1498
- SENSOR\_TYPE\_NONE, 1498
- SENSOR\_TYPE\_ROTATION, 1498
- SENSOR\_TYPE\_SOUND\_DB, 1499
- SENSOR\_TYPE\_SOUND\_DBA, 1499
- SENSOR\_TYPE\_-  
TEMPERATURE, 1499
- SENSOR\_TYPE\_TOUCH, 1499

- SensorBoolean, [1832](#)
- SensorDIAccelStatus, [1833](#)
- SensorDigiPinsDirection, [1833](#)
- SensorDigiPinsOutputLevel, [1834](#)
- SensorDigiPinsStatus, [1834](#)
- SensorDIGPSDistanceToWaypoint, [1834](#)
- SensorDIGPSHeading, [1835](#)
- SensorDIGPSHeadingToWaypoint, [1835](#)
- SensorDIGPSLatitude, [1836](#)
- SensorDIGPSLongitude, [1836](#)
- SensorDIGPSRelativeHeading, [1837](#)
- SensorDIGPSStatus, [1837](#)
- SensorDIGPSTime, [1837](#)
- SensorDIGPSVelocity, [1838](#)
- SensorDIGyroStatus, [1838](#)
- SensorDIGyroTemperature, [1839](#)
- SensorHTColorNum, [1839](#)
- SensorHTCompass, [1839](#)
- SensorHTEOPD, [1840](#)
- SensorHTGyro, [1840](#)
- SensorHTIRSeeker2ACDir, [1841](#)
- SensorHTIRSeeker2Addr, [1841](#)
- SensorHTIRSeeker2DCDir, [1842](#)
- SensorHTIRSeekerDir, [1842](#)
- SensorHTMagnet, [1842](#)
- SensorHTProtoAnalog, [1843](#)
- SensorHTProtoDigital, [1843](#)
- SensorHTProtoDigitalControl, [1844](#)
- SensorHTSuperProAnalog, [1844](#)
- SensorHTSuperProDigital, [1845](#)
- SensorHTSuperProDigitalControl, [1845](#)
- SensorHTSuperProLED, [1846](#)
- SensorHTSuperProProgramControl, [1846](#)
- SensorHTSuperProStrobe, [1846](#)
- SensorInvalid, [1847](#)
- SensorMIXG1300LScale, [1847](#)
- SensorMode, [1848](#)
- SensorMSCompass, [1848](#)
- SensorMSDROD, [1848](#)
- SensorMSPressure, [1849](#)
- SensorMSPressureRaw, [1849](#)
- SensorNormalized, [1850](#)
- SensorNXTSumoEyes, [1850](#)
- SensorNXTSumoEyesRaw, [1850](#)
- SensorRaw, [1851](#)
- SensorScaled, [1851](#)
- SensorTemperature, [1852](#)
- SensorType, [1852](#)
- SensorUS, [1853](#)
- SensorValue, [1853](#)
- SensorValueBool, [1853](#)
- SensorValueRaw, [1854](#)
- set\_fopen\_size, [1854](#)
- SetAbortFlag, [1854](#)
- SetACCLNxSensitivity, [1855](#)
- SetBatteryState, [1855](#)
- SetBluetoothState, [1856](#)
- SetBTDataMode, [1856](#)
- SetBTInputBuffer, [1856](#)
- SetBTInputBufferInPtr, [1857](#)
- SetBTInputBufferOutPtr, [1857](#)
- SetBTOutputBuffer, [1857](#)
- SetBTOutputBufferInPtr, [1858](#)
- SetBTOutputBufferOutPtr, [1858](#)
- SetButtonLongPressCount, [1858](#)
- SetButtonLongReleaseCount, [1859](#)
- SetButtonModuleValue, [1859](#)
- SetButtonPressCount, [1859](#)
- SetButtonReleaseCount, [1860](#)
- SetButtonShortReleaseCount, [1860](#)
- SetButtonState, [1860](#)
- SetCommandFlags, [1861](#)
- SetCommandModuleBytes, [1861](#)
- SetCommandModuleValue, [1861](#)
- SetCommModuleBytes, [1862](#)
- SetCommModuleValue, [1862](#)
- SetCustomSensorActiveStatus, [1863](#)
- SetCustomSensorPercentFullScale, [1863](#)
- SetCustomSensorZeroOffset, [1863](#)
- SetDisplayContrast, [1864](#)
- SetDisplayDisplay, [1864](#)
- SetDisplayEraseMask, [1864](#)
- SetDisplayFlags, [1865](#)
- SetDisplayFont, [1865](#)
- SetDisplayModuleBytes, [1865](#)
- SetDisplayModuleValue, [1866](#)
- SetDisplayNormal, [1866](#)

- SetDisplayPopup, 1867
- SetDisplayTextLinesCenterFlags, 1867
- SetDisplayUpdateMask, 1867
- SetHSAddress, 1868
- SetHSDataMode, 1868
- SetHSFlags, 1868
- SetHSInputBuffer, 1869
- SetHSInputBufferInPtr, 1869
- SetHSInputBufferOutPtr, 1869
- SetHSMMode, 1870
- SetHSOutputBuffer, 1870
- SetHSOutputBufferInPtr, 1871
- SetHSOutputBufferOutPtr, 1871
- SetHSSpeed, 1871
- SetHSState, 1872
- SetHTBarometricCalibration, 1872
- SetHTColor2Mode, 1872
- SetHTIRSeeker2Mode, 1873
- SetI2COptions, 1873
- SetInput, 1874
- SetInputModuleValue, 1874
- SetIOCtrlModuleValue, 1874
- SetIOMapBytes, 1875
- SetIOMapBytesByID, 1875
- SetIOMapValue, 1876
- SetIOMapValueByID, 1876
- SetLoaderModuleValue, 1877
- SetLongAbort, 1877
- SetLowSpeedModuleBytes, 1877
- SetLowSpeedModuleValue, 1878
- SetMotorPwnFreq, 1878
- SetMotorRegulationOptions, 1879
- SetMotorRegulationTime, 1879
- SetNXTRLineLeaderKdFactor, 1879
- SetNXTRLineLeaderKdValue, 1880
- SetNXTRLineLeaderKiFactor, 1880
- SetNXTRLineLeaderKiValue, 1881
- SetNXTRLineLeaderKpFactor, 1881
- SetNXTRLineLeaderKpValue, 1882
- SetNXTRLineLeaderSetpoint, 1883
- SetNXTServoPosition, 1883
- SetNXTServoQuickPosition, 1884
- SetNXTServoSpeed, 1884
- SetOnBrickProgramPointer, 1885
- SetOutput, 1885
- SetOutputModuleValue, 1886
- SetSensor, 1886
- SetSensorBoolean, 1887
- SetSensorColorBlue, 1887
- SetSensorColorFull, 1887
- SetSensorColorGreen, 1888
- SetSensorColorNone, 1888
- SetSensorColorRed, 1888
- SetSensorDIAccl, 1889
- SetSensorDIAcclDrift, 1889
- SetSensorDIAcclEx, 1890
- SetSensorDigiPinsDirection, 1890
- SetSensorDigiPinsOutputLevel, 1891
- SetSensorDigiPinsStatus, 1891
- SetSensorDIGPSWaypoint, 1891
- SetSensorDIGyro, 1892
- SetSensorDIGyroEx, 1892
- SetSensorEMeter, 1893
- SetSensorHTEOPD, 1893
- SetSensorHTGyro, 1894
- SetSensorHTMagnet, 1894
- SetSensorHTProtoDigital, 1894
- SetSensorHTProtoDigitalControl, 1895
- SetSensorHTSuperProAnalogOut, 1895
- SetSensorHTSuperProDigital, 1896
- SetSensorHTSuperProDigitalControl, 1896
- SetSensorHTSuperProLED, 1897
- SetSensorHTSuperProProgramControl, 1897
- SetSensorHTSuperProStrobe, 1897
- SetSensorLight, 1898
- SetSensorLowspeed, 1898
- SetSensorMIXG1300LScale, 1899
- SetSensorMode, 1899
- SetSensorMSDROD, 1900
- SetSensorMSPressure, 1900
- SetSensorNXTSumoEyes, 1900
- SetSensorSound, 1901
- SetSensorTemperature, 1901
- SetSensorTouch, 1902
- SetSensorType, 1902
- SetSensorUltrasonic, 1902

- SetSleepTime, 1903
- SetSleepTimeout, 1903
- SetSleepTimer, 1903
- SetSoundDuration, 1904
- SetSoundFlags, 1904
- SetSoundFrequency, 1904
- SetSoundMode, 1905
- SetSoundModuleState, 1905
- SetSoundModuleValue, 1906
- SetSoundSampleRate, 1906
- SetSoundVolume, 1906
- SetUIButton, 1907
- SetUIModuleValue, 1907
- SetUIState, 1907
- SetUSBInputBuffer, 1908
- SetUSBInputBufferInPtr, 1908
- SetUSBInputBufferOutPtr, 1908
- SetUSBOutputBuffer, 1909
- SetUSBOutputBufferInPtr, 1909
- SetUSBOutputBufferOutPtr, 1909
- SetUSBPollBuffer, 1910
- SetUSBPollBufferInPtr, 1910
- SetUSBPollBufferOutPtr, 1910
- SetUSBState, 1911
- SetVMRunState, 1911
- SetVolume, 1911
- sign, 1912
- Sin, 1499
- sin, 1912
- SinD, 1500
- sind, 1912
- Sinh, 1500
- sinh, 1913
- SinhD, 1500
- sinhd, 1913
- SizeOf, 1914
- SleepNow, 1914
- SleepTime, 1914
- SleepTimeout, 1915
- SleepTimer, 1915
- SoundDuration, 1915
- SoundFlags, 1916
- SoundFrequency, 1916
- SoundMode, 1917
- SoundSampleRate, 1917
- SoundState, 1917
- SoundVolume, 1918
- sprintf, 1918
- Sqrt, 1501
- sqrt, 1919
- srand, 1919
- StartTask, 1920
- Stop, 1920
- StopAllTasks, 1920
- StopSound, 1920
- StopTask, 1921
- StrCat, 1922
- strcat, 1921
- strcmp, 1922
- strcpy, 1923
- StrIndex, 1923
- StrLen, 1924
- strlen, 1924
- strncat, 1924
- strncmp, 1925
- strncpy, 1925
- StrReplace, 1926
- StrToByteArray, 1926
- strtod, 1927
- strtol, 1928
- StrToNum, 1929
- strtoul, 1929
- SubStr, 1930
- SysCall, 1930
- SysColorSensorRead, 1931
- SysCommBTCheckStatus, 1931
- SysCommBTConnection, 1932
- SysCommBTONOff, 1932
- SysCommBTWrite, 1932
- SysCommExecuteFunction, 1933
- SysCommHSCheckStatus, 1933
- SysCommHSControl, 1934
- SysCommHSRead, 1934
- SysCommHSWrite, 1934
- SysCommLSCheckStatus, 1935
- SysCommLSRead, 1935
- SysCommLSWrite, 1936
- SysCommLSWriteEx, 1936
- SysComputeCalibValue, 1936
- SysDatalogGetTimes, 1937
- SysDatalogWrite, 1937
- SysDisplayExecuteFunction, 1938

SysDrawCircle, 1938  
SysDrawEllipse, 1938  
SysDrawFont, 1939  
SysDrawGraphic, 1939  
SysDrawGraphicArray, 1940  
SysDrawLine, 1940  
SysDrawPoint, 1940  
SysDrawPolygon, 1941  
SysDrawRect, 1941  
SysDrawText, 1941  
SysFileClose, 1942  
SysFileDelete, 1942  
SysFileFindFirst, 1942  
SysFileFindNext, 1943  
SysFileOpenAppend, 1943  
SysFileOpenRead, 1944  
SysFileOpenReadLinear, 1944  
SysFileOpenWrite, 1944  
SysFileOpenWriteLinear, 1945  
SysFileOpenWriteNonLinear, 1945  
SysFileRead, 1946  
SysFileRename, 1946  
SysFileResize, 1946  
SysFileResolveHandle, 1947  
SysFileSeek, 1947  
SysFileTell, 1947  
SysFileWrite, 1948  
SysGetStartTick, 1948  
SysInputPinFunction, 1948  
SysIOMapRead, 1949  
SysIOMapReadByID, 1949  
SysIOMapWrite, 1950  
SysIOMapWriteByID, 1950  
SysKeepAlive, 1950  
SysListFiles, 1951  
SysLoaderExecuteFunction, 1951  
SysMemoryManager, 1952  
SysMessageRead, 1952  
SysMessageWrite, 1952  
SysRandomEx, 1953  
SysRandomNumber, 1953  
SysReadButton, 1953  
SysReadLastResponse, 1954  
SysReadSemData, 1954  
SysSetScreenMode, 1955  
SysSetSleepTimeout, 1955  
SysSoundGetState, 1955  
SysSoundPlayFile, 1956  
SysSoundPlayTone, 1956  
SysSoundSetState, 1956  
SysUpdateCalibCacheInfo, 1957  
SysWriteSemData, 1957  
Tan, 1501  
tan, 1958  
TanD, 1501  
tand, 1958  
Tanh, 1502  
tanh, 1959  
TanhD, 1502  
tanhd, 1959  
TextOut, 1959  
tolower, 1961  
toupper, 1961  
Trunc, 1503  
trunc, 1961  
u16, 1503  
u32, 1503  
u8, 1503  
UIButton, 1962  
UIState, 1962  
UnflattenVar, 1962  
USBInputBufferInPtr, 1963  
USBInputBufferOutPtr, 1963  
USBOutputBufferInPtr, 1964  
USBOutputBufferOutPtr, 1964  
USBPollBufferInPtr, 1964  
USBPollBufferOutPtr, 1965  
USBState, 1965  
UsbState, 1965  
UseRS485, 1965  
VectorCross, 1966  
VectorDot, 1966  
VectorNormalize, 1966  
VMRunState, 1966  
Volume, 1967  
Wait, 1967  
Write, 1968  
WriteBytes, 1968  
WriteBytesEx, 1969  
WriteI2CRegister, 1969  
WriteLn, 1970  
WriteLnString, 1970

- WriteNRLinkBytes, 1971
- WriteString, 1971
- Yield, 1972
- NXT firmware module IDs, 258
- NXT firmware module names, 256
- NXT Firmware Modules, 43
- NXTHID\_CMD\_ASCII
  - NBCCCommon.h, 1286
  - NXTHIDCommands, 968
- NXTHID\_CMD\_DIRECT
  - NBCCCommon.h, 1286
  - NXTHIDCommands, 968
- NXTHID\_CMD\_TRANSMIT
  - NBCCCommon.h, 1286
  - NXTHIDCommands, 968
- NXTHID\_MOD\_LEFT\_ALT
  - NBCCCommon.h, 1286
  - NXTHIDModifiers, 966
- NXTHID\_MOD\_LEFT\_CTRL
  - NBCCCommon.h, 1286
  - NXTHIDModifiers, 966
- NXTHID\_MOD\_LEFT\_GUI
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 966
- NXTHID\_MOD\_LEFT\_SHIFT
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 966
- NXTHID\_MOD\_NONE
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 967
- NXTHID\_MOD\_RIGHT\_ALT
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 967
- NXTHID\_MOD\_RIGHT\_CTRL
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 967
- NXTHID\_MOD\_RIGHT\_GUI
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 967
- NXTHID\_MOD\_RIGHT\_SHIFT
  - NBCCCommon.h, 1287
  - NXTHIDModifiers, 967
- NXTHID\_REG\_CMD
  - NBCCCommon.h, 1287
  - NXTHIDRegisters, 965
- NXTHID\_REG\_DATA
  - NBCCCommon.h, 1288
  - NXTHIDRegisters, 965
- NXTHID\_REG\_MODIFIER
  - NBCCCommon.h, 1288
  - NXTHIDRegisters, 965
- NXTHIDAsciiMode
  - MindSensorsAPI, 195
  - NXCDefs.h, 1709
- NXTHIDCommands
  - NXTHID\_CMD\_ASCII, 968
  - NXTHID\_CMD\_DIRECT, 968
  - NXTHID\_CMD\_TRANSMIT, 968
- NXTHIDDirectMode
  - MindSensorsAPI, 196
  - NXCDefs.h, 1709
- NXTHIDLoadCharacter
  - MindSensorsAPI, 196
  - NXCDefs.h, 1710
- NXTHIDModifiers
  - NXTHID\_MOD\_LEFT\_ALT, 966
  - NXTHID\_MOD\_LEFT\_CTRL, 966
  - NXTHID\_MOD\_LEFT\_GUI, 966
  - NXTHID\_MOD\_LEFT\_SHIFT, 966
  - NXTHID\_MOD\_NONE, 967
  - NXTHID\_MOD\_RIGHT\_ALT, 967
  - NXTHID\_MOD\_RIGHT\_CTRL, 967
  - NXTHID\_MOD\_RIGHT\_GUI, 967
  - NXTHID\_MOD\_RIGHT\_SHIFT, 967
- NXTHIDRegisters
  - NXTHID\_REG\_CMD, 965
  - NXTHID\_REG\_DATA, 965
  - NXTHID\_REG\_MODIFIER, 965
- NXTHIDTransmit
  - MindSensorsAPI, 197
  - NXCDefs.h, 1710
- NXTLimits
  - CHAR\_BIT, 1011
  - CHAR\_MAX, 1011
  - CHAR\_MIN, 1011
  - INT\_MAX, 1011
  - INT\_MIN, 1011
  - LONG\_MAX, 1011
  - LONG\_MIN, 1012

- RAND\_MAX, 1012
- SCHAR\_MAX, 1012
- SCHAR\_MIN, 1012
- SHRT\_MAX, 1012
- SHRT\_MIN, 1012
- UCHAR\_MAX, 1012
- UINT\_MAX, 1012
- ULONG\_MAX, 1012
- USHRT\_MAX, 1012
- NXTLineLeaderAverage
  - MindSensorsAPI, 197
  - NXCDefs.h, 1711
- NXTLineLeaderCalibrateBlack
  - MindSensorsAPI, 198
  - NXCDefs.h, 1711
- NXTLineLeaderCalibrateWhite
  - MindSensorsAPI, 198
  - NXCDefs.h, 1712
- NXTLineLeaderCommands
  - NXTLL\_CMD\_BLACK, 976
  - NXTLL\_CMD\_EUROPEAN, 976
  - NXTLL\_CMD\_INVERT, 976
  - NXTLL\_CMD\_POWERDOWN, 976
  - NXTLL\_CMD\_POWERUP, 976
  - NXTLL\_CMD\_RESET, 976
  - NXTLL\_CMD\_SNAPSHOT, 976
  - NXTLL\_CMD\_UNIVERSAL, 976
  - NXTLL\_CMD\_USA, 976
  - NXTLL\_CMD\_WHITE, 976
- NXTLineLeaderInvert
  - MindSensorsAPI, 199
  - NXCDefs.h, 1712
- NXTLineLeaderPowerDown
  - MindSensorsAPI, 199
  - NXCDefs.h, 1713
- NXTLineLeaderPowerUp
  - MindSensorsAPI, 200
  - NXCDefs.h, 1713
- NXTLineLeaderRegisters
  - NXTLL\_REG\_AVERAGE, 973
  - NXTLL\_REG\_BLACKDATA, 973
  - NXTLL\_REG\_BLACKLIMITS, 973
  - NXTLL\_REG\_CALIBRATED, 973
  - NXTLL\_REG\_CMD, 974
  - NXTLL\_REG\_KD\_FACTOR, 974
  - NXTLL\_REG\_KD\_VALUE, 974
  - NXTLL\_REG\_KI\_FACTOR, 974
  - NXTLL\_REG\_KI\_VALUE, 974
  - NXTLL\_REG\_KP\_FACTOR, 974
  - NXTLL\_REG\_KP\_VALUE, 974
  - NXTLL\_REG\_RAWVOLTAGE, 974
  - NXTLL\_REG\_RESULT, 974
  - NXTLL\_REG\_SETPOINT, 974
  - NXTLL\_REG\_STEERING, 975
  - NXTLL\_REG\_WHITEDATA, 975
  - NXTLL\_REG\_WHITELIMITS, 975
- NXTLineLeaderReset
  - MindSensorsAPI, 200
  - NXCDefs.h, 1714
- NXTLineLeaderResult
  - MindSensorsAPI, 201
  - NXCDefs.h, 1714
- NXTLineLeaderSnapshot
  - MindSensorsAPI, 201
  - NXCDefs.h, 1715
- NXTLineLeaderSteering
  - MindSensorsAPI, 202
  - NXCDefs.h, 1715
- NXTLL\_CMD\_BLACK
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_EUROPEAN
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_INVERT
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_POWERDOWN
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_POWERUP
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_RESET
  - NBCCommon.h, 1288
  - NXTLineLeaderCommands, 976
- NXTLL\_CMD\_SNAPSHOT
  - NBCCommon.h, 1288

- NXTLineLeaderCommands, 976
- NXTLL\_CMD\_UNIVERSAL
  - NBCCCommon.h, 1288
- NXTLineLeaderCommands, 976
- NXTLL\_CMD\_USA
  - NBCCCommon.h, 1289
- NXTLineLeaderCommands, 976
- NXTLL\_CMD\_WHITE
  - NBCCCommon.h, 1289
- NXTLineLeaderCommands, 976
- NXTLL\_REG\_AVERAGE
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 973
- NXTLL\_REG\_BLACKDATA
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 973
- NXTLL\_REG\_BLACKLIMITS
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 973
- NXTLL\_REG\_CALIBRATED
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 973
- NXTLL\_REG\_CMD
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KD\_FACTOR
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KD\_VALUE
  - NBCCCommon.h, 1289
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KI\_FACTOR
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KI\_VALUE
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KP\_FACTOR
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_KP\_VALUE
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_RAWVOLTAGE
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_RESULT
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_SETPOINT
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 974
- NXTLL\_REG\_STEERING
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 975
- NXTLL\_REG\_WHITEDATA
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 975
- NXTLL\_REG\_WHITELIMITS
  - NBCCCommon.h, 1290
  - NXTLineLeaderRegisters, 975
- NXTPM\_CMD\_RESET
  - NBCCCommon.h, 1291
  - NXTPowerMeterCommands, 971
- NXTPM\_REG\_CAPACITY
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 969
- NXTPM\_REG\_CMD
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 969
- NXTPM\_REG\_CURRENT
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 969
- NXTPM\_REG\_ERRORCOUNT
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 969
- NXTPM\_REG\_GAIN
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 969
- NXTPM\_REG\_MAXCURRENT
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_MAXVOLTAGE
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_MINCURRENT
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_MINVOLTAGE
  - NBCCCommon.h, 1291
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_POWER

- NBCCCommon.h, 1292
- NXTPowerMeterRegisters, 970
- NXTPM\_REG\_TIME
  - NBCCCommon.h, 1292
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_TOTALPOWER
  - NBCCCommon.h, 1292
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_USERGAIN
  - NBCCCommon.h, 1292
  - NXTPowerMeterRegisters, 970
- NXTPM\_REG\_VOLTAGE
  - NBCCCommon.h, 1292
  - NXTPowerMeterRegisters, 970
- NXTPowerMeterCapacityUsed
  - MindSensorsAPI, 202
  - NXCDefs.h, 1716
- NXTPowerMeterCommands
  - NXTPM\_CMD\_RESET, 971
- NXTPowerMeterElapsedTime
  - MindSensorsAPI, 203
  - NXCDefs.h, 1716
- NXTPowerMeterErrorCount
  - MindSensorsAPI, 203
  - NXCDefs.h, 1717
- NXTPowerMeterMaxCurrent
  - MindSensorsAPI, 204
  - NXCDefs.h, 1717
- NXTPowerMeterMaxVoltage
  - MindSensorsAPI, 204
  - NXCDefs.h, 1718
- NXTPowerMeterMinCurrent
  - MindSensorsAPI, 205
  - NXCDefs.h, 1718
- NXTPowerMeterMinVoltage
  - MindSensorsAPI, 205
  - NXCDefs.h, 1719
- NXTPowerMeterPresentCurrent
  - MindSensorsAPI, 206
  - NXCDefs.h, 1719
- NXTPowerMeterPresentPower
  - MindSensorsAPI, 206
  - NXCDefs.h, 1719
- NXTPowerMeterPresentVoltage
  - MindSensorsAPI, 206
  - NXCDefs.h, 1720
- NXTPowerMeterRegisters
  - NXTPM\_REG\_CAPACITY, 969
  - NXTPM\_REG\_CMD, 969
  - NXTPM\_REG\_CURRENT, 969
  - NXTPM\_REG\_ERRORCOUNT, 969
  - NXTPM\_REG\_GAIN, 969
  - NXTPM\_REG\_MAXCURRENT, 970
  - NXTPM\_REG\_MAXVOLTAGE, 970
  - NXTPM\_REG\_MINCURRENT, 970
  - NXTPM\_REG\_MINVOLTAGE, 970
  - NXTPM\_REG\_POWER, 970
  - NXTPM\_REG\_TIME, 970
  - NXTPM\_REG\_TOTALPOWER, 970
  - NXTPM\_REG\_USERGAIN, 970
  - NXTPM\_REG\_VOLTAGE, 970
- NXTPowerMeterResetCounters
  - MindSensorsAPI, 207
  - NXCDefs.h, 1720
- NXTPowerMeterTotalPowerConsumed
  - MindSensorsAPI, 207
  - NXCDefs.h, 1721
- NXTSE\_ZONE\_FRONT
  - NBCCCommon.h, 1292
  - NXTSumoEyesConstants, 971
- NXTSE\_ZONE\_LEFT
  - NBCCCommon.h, 1292
  - NXTSumoEyesConstants, 972
- NXTSE\_ZONE\_NONE
  - NBCCCommon.h, 1292
  - NXTSumoEyesConstants, 972
- NXTSE\_ZONE\_RIGHT
  - NBCCCommon.h, 1293
  - NXTSumoEyesConstants, 972
- NXTSERVO\_CMD\_EDIT1
  - NBCCCommon.h, 1293
  - NXTServoCommands, 963
- NXTSERVO\_CMD\_EDIT2
  - NBCCCommon.h, 1293
  - NXTServoCommands, 963
- NXTSERVO\_CMD\_GOTO

- NBCCCommon.h, [1293](#)
- NXTServoCommands, [964](#)
- NXTSERVO\_CMD\_HALT
  - NBCCCommon.h, [1293](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_CMD\_INIT
  - NBCCCommon.h, [1293](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_CMD\_PAUSE
  - NBCCCommon.h, [1293](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_CMD\_RESET
  - NBCCCommon.h, [1293](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_CMD\_RESUME
  - NBCCCommon.h, [1294](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_EM\_CMD\_QUIT
  - NBCCCommon.h, [1294](#)
  - NXTServoCommands, [964](#)
- NXTSERVO\_EM\_REG\_CMD
  - NBCCCommon.h, [1294](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_EM\_REG\_EEPROM\_-  
END
  - NBCCCommon.h, [1294](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_EM\_REG\_EEPROM\_-  
START
  - NBCCCommon.h, [1294](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_POS\_CENTER
  - NBCCCommon.h, [1294](#)
  - NXTServoPos, [960](#)
- NXTSERVO\_POS\_MAX
  - NBCCCommon.h, [1294](#)
  - NXTServoPos, [960](#)
- NXTSERVO\_POS\_MIN
  - NBCCCommon.h, [1294](#)
  - NXTServoPos, [960](#)
- NXTSERVO\_QPOS\_CENTER
  - NBCCCommon.h, [1294](#)
  - NXTServoQPos, [961](#)
- NXTSERVO\_QPOS\_MAX
  - NBCCCommon.h, [1295](#)
  - NXTServoQPos, [961](#)
- NXTSERVO\_QPOS\_MIN
  - NBCCCommon.h, [1295](#)
  - NXTServoQPos, [961](#)
- NXTSERVO\_REG\_CMD
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_REG\_S1\_POS
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_REG\_S1\_QPOS
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_REG\_S1\_SPEED
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [957](#)
- NXTSERVO\_REG\_S2\_POS
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S2\_QPOS
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S2\_SPEED
  - NBCCCommon.h, [1295](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S3\_POS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S3\_QPOS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S3\_SPEED
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S4\_POS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S4\_QPOS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S4\_SPEED
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S5\_POS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [958](#)
- NXTSERVO\_REG\_S5\_QPOS

- NBCCCommon.h, [1296](#)
- NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S5\_SPEED
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S6\_POS
  - NBCCCommon.h, [1296](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S6\_QPOS
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S6\_SPEED
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S7\_POS
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S7\_QPOS
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S7\_SPEED
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S8\_POS
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S8\_QPOS
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [959](#)
- NXTSERVO\_REG\_S8\_SPEED
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [960](#)
- NXTSERVO\_REG\_VOLTAGE
  - NBCCCommon.h, [1297](#)
  - NXTServoRegisters, [960](#)
- NXTSERVO\_SERVO\_1
  - NBCCCommon.h, [1297](#)
  - NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_2
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_3
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_4
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [962](#)
- NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_5
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_6
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [962](#)
- NXTSERVO\_SERVO\_7
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [963](#)
- NXTSERVO\_SERVO\_8
  - NBCCCommon.h, [1298](#)
  - NXTServoNumbers, [963](#)
- NXTServoBatteryVoltage
  - MindSensorsAPI, [208](#)
- NXCDefs.h, [1721](#)
- NXTServoCommands
  - NXTSERVO\_CMD\_EDIT1, [963](#)
  - NXTSERVO\_CMD\_EDIT2, [963](#)
  - NXTSERVO\_CMD\_GOTO, [964](#)
  - NXTSERVO\_CMD\_HALT, [964](#)
  - NXTSERVO\_CMD\_INIT, [964](#)
  - NXTSERVO\_CMD\_PAUSE, [964](#)
  - NXTSERVO\_CMD\_RESET, [964](#)
  - NXTSERVO\_CMD\_RESUME, [964](#)
  - NXTSERVO\_EM\_CMD\_QUIT, [964](#)
- NXTServoEditMacro
  - MindSensorsAPI, [208](#)
- NXCDefs.h, [1722](#)
- NXTServoGotoMacroAddress
  - MindSensorsAPI, [209](#)
- NXCDefs.h, [1722](#)
- NXTServoHaltMacro
  - MindSensorsAPI, [209](#)
- NXCDefs.h, [1723](#)
- NXTServoInit
  - MindSensorsAPI, [210](#)
- NXCDefs.h, [1723](#)
- NXTServoNumbers
  - NXTSERVO\_SERVO\_1, [962](#)
  - NXTSERVO\_SERVO\_2, [962](#)
  - NXTSERVO\_SERVO\_3, [962](#)
  - NXTSERVO\_SERVO\_4, [962](#)
  - NXTSERVO\_SERVO\_5, [962](#)
  - NXTSERVO\_SERVO\_6, [962](#)

- NXTSERVO\_SERVO\_7, [963](#)
- NXTSERVO\_SERVO\_8, [963](#)
- NXTServoPauseMacro
  - MindSensorsAPI, [210](#)
  - NXCDefs.h, [1724](#)
- NXTServoPos
  - NXTSERVO\_POS\_CENTER, [960](#)
  - NXTSERVO\_POS\_MAX, [960](#)
  - NXTSERVO\_POS\_MIN, [960](#)
- NXTServoPosition
  - MindSensorsAPI, [211](#)
  - NXCDefs.h, [1724](#)
- NXTServoQPos
  - NXTSERVO\_QPOS\_CENTER, [961](#)
  - NXTSERVO\_QPOS\_MAX, [961](#)
  - NXTSERVO\_QPOS\_MIN, [961](#)
- NXTServoQuitEdit
  - MindSensorsAPI, [211](#)
  - NXCDefs.h, [1725](#)
- NXTServoRegisters
  - NXTSERVO\_EM\_REG\_CMD, [957](#)
  - NXTSERVO\_EM\_REG\_-
    - EEPROM\_END, [957](#)
  - NXTSERVO\_EM\_REG\_-
    - EEPROM\_START, [957](#)
  - NXTSERVO\_REG\_CMD, [957](#)
  - NXTSERVO\_REG\_S1\_POS, [957](#)
  - NXTSERVO\_REG\_S1\_QPOS, [957](#)
  - NXTSERVO\_REG\_S1\_SPEED,
    - [957](#)
  - NXTSERVO\_REG\_S2\_POS, [958](#)
  - NXTSERVO\_REG\_S2\_QPOS, [958](#)
  - NXTSERVO\_REG\_S2\_SPEED,
    - [958](#)
  - NXTSERVO\_REG\_S3\_POS, [958](#)
  - NXTSERVO\_REG\_S3\_QPOS, [958](#)
  - NXTSERVO\_REG\_S3\_SPEED,
    - [958](#)
  - NXTSERVO\_REG\_S4\_POS, [958](#)
  - NXTSERVO\_REG\_S4\_QPOS, [958](#)
  - NXTSERVO\_REG\_S4\_SPEED,
    - [958](#)
  - NXTSERVO\_REG\_S5\_POS, [958](#)
  - NXTSERVO\_REG\_S5\_QPOS, [959](#)
  - NXTSERVO\_REG\_S5\_SPEED,
    - [959](#)
  - NXTSERVO\_REG\_S6\_POS, [959](#)
  - NXTSERVO\_REG\_S6\_QPOS, [959](#)
  - NXTSERVO\_REG\_S6\_SPEED,
    - [959](#)
  - NXTSERVO\_REG\_S7\_POS, [959](#)
  - NXTSERVO\_REG\_S7\_QPOS, [959](#)
  - NXTSERVO\_REG\_S7\_SPEED,
    - [959](#)
  - NXTSERVO\_REG\_S8\_POS, [959](#)
  - NXTSERVO\_REG\_S8\_QPOS, [959](#)
  - NXTSERVO\_REG\_S8\_SPEED,
    - [960](#)
  - NXTSERVO\_REG\_VOLTAGE, [960](#)
- NXTServoReset
  - MindSensorsAPI, [212](#)
  - NXCDefs.h, [1725](#)
- NXTServoResumeMacro
  - MindSensorsAPI, [212](#)
  - NXCDefs.h, [1726](#)
- NXTServoSpeed
  - MindSensorsAPI, [213](#)
  - NXCDefs.h, [1726](#)
- NXTSumoEyesConstants
  - NXTSE\_ZONE\_FRONT, [971](#)
  - NXTSE\_ZONE\_LEFT, [972](#)
  - NXTSE\_ZONE\_NONE, [972](#)
  - NXTSE\_ZONE\_RIGHT, [972](#)
- Off
  - NXCDefs.h, [1727](#)
  - OutputModuleFunctions, [328](#)
- OffEx
  - NXCDefs.h, [1727](#)
  - OutputModuleFunctions, [328](#)
- Offset
  - IOMapReadByIDType, [1075](#)
  - IOMapReadType, [1077](#)
  - IOMapWriteByIDType, [1078](#)
  - IOMapWriteType, [1080](#)
- OldFilename
  - FileRenameType, [1064](#)
- On
  - DisplayExecuteFunctionType, [1040](#)
- OnBrickProgramPointer
  - NXCDefs.h, [1728](#)
  - UiModuleFunctions, [564](#)

- OnFwd
  - NXCDefs.h, 1728
  - OutputModuleFunctions, 328
- OnFwdEx
  - NXCDefs.h, 1728
  - OutputModuleFunctions, 329
- OnFwdReg
  - NXCDefs.h, 1729
  - OutputModuleFunctions, 329
- OnFwdRegEx
  - NXCDefs.h, 1729
  - OutputModuleFunctions, 330
- OnFwdRegExPID
  - NXCDefs.h, 1730
  - OutputModuleFunctions, 330
- OnFwdRegPID
  - NXCDefs.h, 1731
  - OutputModuleFunctions, 331
- OnFwdSync
  - NXCDefs.h, 1731
  - OutputModuleFunctions, 332
- OnFwdSyncEx
  - NXCDefs.h, 1732
  - OutputModuleFunctions, 332
- OnFwdSyncExPID
  - NXCDefs.h, 1732
  - OutputModuleFunctions, 333
- OnFwdSyncPID
  - NXCDefs.h, 1733
  - OutputModuleFunctions, 333
- OnRev
  - NXCDefs.h, 1734
  - OutputModuleFunctions, 334
- OnRevEx
  - NXCDefs.h, 1734
  - OutputModuleFunctions, 334
- OnRevReg
  - NXCDefs.h, 1734
  - OutputModuleFunctions, 335
- OnRevRegEx
  - NXCDefs.h, 1735
  - OutputModuleFunctions, 335
- OnRevRegExPID
  - NXCDefs.h, 1735
  - OutputModuleFunctions, 336
- OnRevRegPID
  - NXCDefs.h, 1736
  - OutputModuleFunctions, 336
- OnRevSync
  - NXCDefs.h, 1737
  - OutputModuleFunctions, 337
- OnRevSyncEx
  - NXCDefs.h, 1737
  - OutputModuleFunctions, 338
- OnRevSyncExPID
  - NXCDefs.h, 1738
  - OutputModuleFunctions, 338
- OnRevSyncPID
  - NXCDefs.h, 1738
  - OutputModuleFunctions, 339
- OPARR\_MAX
  - ArrayOpConstants, 681
  - NBCCommon.h, 1298
- OPARR\_MEAN
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OPARR\_MIN
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OPARR\_SORT
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OPARR\_STD
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OPARR\_SUM
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OPARR\_SUMSQR
  - ArrayOpConstants, 681
  - NBCCommon.h, 1299
- OpenFileAppend
  - LoaderModuleFunctions, 580
  - NXCDefs.h, 1739
- OpenFileRead
  - LoaderModuleFunctions, 581
  - NXCDefs.h, 1740
- OpenFileReadLinear
  - LoaderModuleFunctions, 581
  - NXCDefs.h, 1740
- Options
  - DrawCircleType, 1043

- DrawEllipseType, 1045
- DrawFontType, 1047
- DrawGraphicArrayType, 1048
- DrawGraphicType, 1050
- DrawLineType, 1051
- DrawPointType, 1053
- DrawPolygonType, 1054
- DrawRectType, 1055
- DrawTextType, 1056
- Origin
  - FileSeekType, 1068
- OUT\_A
  - NBCCCommon.h, 1299
  - OutputPortConstants, 773
- OUT\_AB
  - NBCCCommon.h, 1300
  - OutputPortConstants, 773
- OUT\_ABC
  - NBCCCommon.h, 1300
  - OutputPortConstants, 774
- OUT\_AC
  - NBCCCommon.h, 1300
  - OutputPortConstants, 774
- OUT\_B
  - NBCCCommon.h, 1300
  - OutputPortConstants, 774
- OUT\_BC
  - NBCCCommon.h, 1300
  - OutputPortConstants, 774
- OUT\_C
  - NBCCCommon.h, 1300
  - OutputPortConstants, 774
- OUT\_MODE\_BRAKE
  - NBCCCommon.h, 1300
  - OutModeConstants, 779
- OUT\_MODE\_COAST
  - NBCCCommon.h, 1300
  - OutModeConstants, 779
- OUT\_MODE\_MOTORON
  - NBCCCommon.h, 1301
  - OutModeConstants, 779
- OUT\_MODE\_REGMETHOD
  - NBCCCommon.h, 1301
  - OutModeConstants, 779
- OUT\_MODE\_REGULATED
  - NBCCCommon.h, 1301
- OutModeConstants, 779
  - OUT\_OPTION\_HOLDATLIMIT
    - NBCCCommon.h, 1301
    - OutOptionConstants, 780
  - OUT\_OPTION\_RAMPDOWNLIMIT
    - NBCCCommon.h, 1301
    - OutOptionConstants, 780
  - OUT\_REGMODE\_IDLE
    - NBCCCommon.h, 1301
    - OutRegModeConstants, 783
  - OUT\_REGMODE\_POS
    - NBCCCommon.h, 1301
    - OutRegModeConstants, 783
  - OUT\_REGMODE\_SPEED
    - NBCCCommon.h, 1301
    - OutRegModeConstants, 783
  - OUT\_REGMODE\_SYNC
    - NBCCCommon.h, 1302
    - OutRegModeConstants, 783
  - OUT\_REGOPTION\_NO\_-SATURATION
    - NBCCCommon.h, 1302
    - OutRegOptionConstants, 781
  - OUT\_RUNSTATE\_HOLD
    - NBCCCommon.h, 1302
    - OutRunStateConstants, 781
  - OUT\_RUNSTATE\_IDLE
    - NBCCCommon.h, 1302
    - OutRunStateConstants, 781
  - OUT\_RUNSTATE\_RAMPDOWN
    - NBCCCommon.h, 1302
    - OutRunStateConstants, 781
  - OUT\_RUNSTATE\_RAMPUP
    - NBCCCommon.h, 1302
    - OutRunStateConstants, 782
  - OUT\_RUNSTATE\_RUNNING
    - NBCCCommon.h, 1302
    - OutRunStateConstants, 782
- OutOptionConstants
  - OUT\_MODE\_BRAKE, 779
  - OUT\_MODE\_COAST, 779
  - OUT\_MODE\_MOTORON, 779
  - OUT\_MODE\_REGMETHOD, 779
  - OUT\_MODE\_REGULATED, 779

- OUT\_OPTION\_HOLDATLIMIT, 780
- OUT\_OPTION\_-
  - RAMPDOWNTOLIMIT, 780
- Output field constants, 783
- Output module, 48
- Output module constants, 49
- Output module functions, 313
- Output module IOMAP offsets, 790
- Output module types, 313
- Output port constants, 772
- Output port mode constants, 778
- Output port option constants, 780
- Output port regulation mode constants, 782
- Output port run state constants, 781
- Output port update flag constants, 776
- Output regulation option constants, 780
- OutputFieldConstants
  - ActualSpeedField, 785
  - BlockTachoCountField, 785
  - MaxAccelerationField, 785
  - MaxSpeedField, 785
  - OutputModeField, 786
  - OutputOptionsField, 786
  - OverloadField, 786
  - PowerField, 786
  - RegDValueField, 787
  - RegIValueField, 787
  - RegModeField, 787
  - RegPValueField, 788
  - RotationCountField, 788
  - RunStateField, 788
  - TachoCountField, 788
  - TachoLimitField, 789
  - TurnRatioField, 789
  - UpdateFlagsField, 789
- OutputIOMAP
  - OutputOffsetActualSpeed, 790
  - OutputOffsetBlockTachoCount, 790
  - OutputOffsetFlags, 790
  - OutputOffsetMaxAccel, 791
  - OutputOffsetMaxSpeed, 791
  - OutputOffsetMode, 791
  - OutputOffsetMotorRPM, 791
  - OutputOffsetOptions, 791
  - OutputOffsetOverloaded, 791
  - OutputOffsetRegDParameter, 791
  - OutputOffsetRegIParameter, 791
  - OutputOffsetRegMode, 791
  - OutputOffsetRegPParameter, 792
  - OutputOffsetRegulationOptions, 792
  - OutputOffsetRegulationTime, 792
  - OutputOffsetRotationCount, 792
  - OutputOffsetRunState, 792
  - OutputOffsetSpeed, 792
  - OutputOffsetSyncTurnParameter, 792
  - OutputOffsetTachoCount, 792
  - OutputOffsetTachoLimit, 792
- OutputModeField
  - NBCCCommon.h, 1303
  - OutputFieldConstants, 786
- OutputModuleFunctions
  - Coast, 318
  - CoastEx, 318
  - Float, 319
  - GetOutput, 319
  - MotorActualSpeed, 319
  - MotorBlockTachoCount, 320
  - MotorMaxAcceleration, 320
  - MotorMaxSpeed, 321
  - MotorMode, 321
  - MotorOutputOptions, 322
  - MotorOverload, 322
  - MotorPower, 322
  - MotorPwnFreq, 323
  - MotorRegDValue, 323
  - MotorRegIValue, 324
  - MotorRegPValue, 324
  - MotorRegulation, 324
  - MotorRegulationOptions, 325
  - MotorRegulationTime, 325
  - MotorRotationCount, 326
  - MotorRunState, 326
  - MotorTachoCount, 326
  - MotorTachoLimit, 327
  - MotorTurnRatio, 327
  - Off, 328
  - OffEx, 328
  - OnFwd, 328

- OnFwdEx, [329](#)
- OnFwdReg, [329](#)
- OnFwdRegEx, [330](#)
- OnFwdRegExPID, [330](#)
- OnFwdRegPID, [331](#)
- OnFwdSync, [332](#)
- OnFwdSyncEx, [332](#)
- OnFwdSyncExPID, [333](#)
- OnFwdSyncPID, [333](#)
- OnRev, [334](#)
- OnRevEx, [334](#)
- OnRevReg, [335](#)
- OnRevRegEx, [335](#)
- OnRevRegExPID, [336](#)
- OnRevRegPID, [336](#)
- OnRevSync, [337](#)
- OnRevSyncEx, [338](#)
- OnRevSyncExPID, [338](#)
- OnRevSyncPID, [339](#)
- PosRegAddAngle, [339](#)
- PosRegEnable, [340](#)
- PosRegSetAngle, [341](#)
- PosRegSetMax, [341](#)
- ResetAllTachoCounts, [342](#)
- ResetBlockTachoCount, [342](#)
- ResetRotationCount, [342](#)
- ResetTachoCount, [343](#)
- RotateMotor, [343](#)
- RotateMotorEx, [343](#)
- RotateMotorExPID, [344](#)
- RotateMotorPID, [345](#)
- SetMotorPwnFreq, [345](#)
- SetMotorRegulationOptions, [346](#)
- SetMotorRegulationTime, [346](#)
- SetOutput, [346](#)
- OutputModuleID
  - ModuleIDConstants, [260](#)
  - NBCCCommon.h, [1303](#)
- OutputModuleName
  - ModuleNameConstants, [258](#)
  - NBCCCommon.h, [1303](#)
- OutputOffsetActualSpeed
  - NBCCCommon.h, [1303](#)
  - OutputIOMAP, [790](#)
- OutputOffsetBlockTachoCount
  - NBCCCommon.h, [1303](#)
- OutputIOMAP, [790](#)
- OutputOffsetFlags
  - NBCCCommon.h, [1303](#)
  - OutputIOMAP, [790](#)
- OutputOffsetMaxAccel
  - NBCCCommon.h, [1303](#)
  - OutputIOMAP, [791](#)
- OutputOffsetMaxSpeed
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetMode
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetMotorRPM
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetOptions
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetOverloaded
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetRegDParameter
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetRegIPParameter
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetRegMode
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [791](#)
- OutputOffsetRegPParameter
  - NBCCCommon.h, [1304](#)
  - OutputIOMAP, [792](#)
- OutputOffsetRegulationOptions
  - NBCCCommon.h, [1305](#)
  - OutputIOMAP, [792](#)
- OutputOffsetRegulationTime
  - NBCCCommon.h, [1305](#)
  - OutputIOMAP, [792](#)
- OutputOffsetRotationCount
  - NBCCCommon.h, [1305](#)
  - OutputIOMAP, [792](#)
- OutputOffsetRunState
  - NBCCCommon.h, [1305](#)
  - OutputIOMAP, [792](#)

- OutputOffsetSpeed
  - NBCCCommon.h, 1305
  - OutputIOMAP, 792
- OutputOffsetSyncTurnParameter
  - NBCCCommon.h, 1305
  - OutputIOMAP, 792
- OutputOffsetTachoCount
  - NBCCCommon.h, 1305
  - OutputIOMAP, 792
- OutputOffsetTachoLimit
  - NBCCCommon.h, 1305
  - OutputIOMAP, 792
- OutputOptionsField
  - NBCCCommon.h, 1305
  - OutputFieldConstants, 786
- OutputPortConstants
  - OUT\_A, 773
  - OUT\_AB, 773
  - OUT\_ABC, 774
  - OUT\_AC, 774
  - OUT\_B, 774
  - OUT\_BC, 774
  - OUT\_C, 774
- OutputStateType, 1094
  - BlockTachoCount, 1094
  - Mode, 1094
  - Port, 1094
  - Power, 1094
  - RegMode, 1095
  - RotationCount, 1095
  - RunState, 1095
  - TachoCount, 1095
  - TachoLimit, 1095
  - TurnRatio, 1095
- OutRegModeConstants
  - OUT\_REGMODE\_IDLE, 783
  - OUT\_REGMODE\_POS, 783
  - OUT\_REGMODE\_SPEED, 783
  - OUT\_REGMODE\_SYNC, 783
- OutRegOptionConstants
  - OUT\_REGOPTION\_NO\_-SATURATION, 781
- OutRunStateConstants
  - OUT\_RUNSTATE\_HOLD, 781
  - OUT\_RUNSTATE\_IDLE, 781
  - OUT\_RUNSTATE\_RAMPDOWN, 781
  - OUT\_RUNSTATE\_RAMPIP, 782
  - OUT\_RUNSTATE\_RUNNING, 782
- OutUFConstants
  - UF\_PENDING\_UPDATES, 776
  - UF\_UPDATE\_MODE, 776
  - UF\_UPDATE\_PID\_VALUES, 776
  - UF\_UPDATE\_RESET\_BLOCK\_COUNT, 776
  - UF\_UPDATE\_RESET\_COUNT, 777
  - UF\_UPDATE\_RESET\_ROTATION\_COUNT, 777
  - UF\_UPDATE\_SPEED, 777
  - UF\_UPDATE\_TACHO\_LIMIT, 777
- OverloadField
  - NBCCCommon.h, 1306
  - OutputFieldConstants, 786
- Param1
  - CommExecuteFunctionType, 1026
- Param2
  - CommExecuteFunctionType, 1026
- Param3
  - CommExecuteFunctionType, 1026
- Pattern
  - ListFilesType, 1085
- PF/IR Train function constants, 903
- PF\_CHANNEL\_1
  - NBCCCommon.h, 1306
  - PFChannelConstants, 901
- PF\_CHANNEL\_2
  - NBCCCommon.h, 1306
  - PFChannelConstants, 902
- PF\_CHANNEL\_3
  - NBCCCommon.h, 1306
  - PFChannelConstants, 902
- PF\_CHANNEL\_4
  - NBCCCommon.h, 1306
  - PFChannelConstants, 902
- PF\_CMD\_BRAKE
  - NBCCCommon.h, 1307
  - PFCmdConstants, 900
- PF\_CMD\_FLOAT
  - NBCCCommon.h, 1307

- PFCmdConstants, 900
- PF\_CMD\_FWD
  - NBCCCommon.h, 1307
  - PFCmdConstants, 900
- PF\_CMD\_REV
  - NBCCCommon.h, 1307
  - PFCmdConstants, 901
- PF\_CMD\_STOP
  - NBCCCommon.h, 1307
  - PFCmdConstants, 901
- PF\_CST\_CLEAR1\_CLEAR2
  - NBCCCommon.h, 1307
  - PFCSTOptions, 908
- PF\_CST\_CLEAR1\_SET2
  - NBCCCommon.h, 1307
  - PFCSTOptions, 908
- PF\_CST\_DECREMENT\_PWM
  - NBCCCommon.h, 1308
  - PFCSTOptions, 908
- PF\_CST\_FULL\_FWD
  - NBCCCommon.h, 1308
  - PFCSTOptions, 908
- PF\_CST\_FULL\_REV
  - NBCCCommon.h, 1308
  - PFCSTOptions, 908
- PF\_CST\_INCREMENT\_PWM
  - NBCCCommon.h, 1308
  - PFCSTOptions, 908
- PF\_CST\_SET1\_CLEAR2
  - NBCCCommon.h, 1308
  - PFCSTOptions, 908
- PF\_CST\_SET1\_SET2
  - NBCCCommon.h, 1308
  - PFCSTOptions, 909
- PF\_CST\_TOGGLE\_DIR
  - NBCCCommon.h, 1308
  - PFCSTOptions, 909
- PF\_FUNC\_CLEAR
  - NBCCCommon.h, 1308
  - PFPinFuncs, 907
- PF\_FUNC\_NOCHANGE
  - NBCCCommon.h, 1308
  - PFPinFuncs, 907
- PF\_FUNC\_SET
  - NBCCCommon.h, 1309
  - PFPinFuncs, 907
- PF\_FUNC\_TOGGLE
  - NBCCCommon.h, 1309
  - PFPinFuncs, 907
- PF\_MODE\_COMBO\_DIRECT
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_COMBO\_PWM
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_SINGLE\_OUTPUT\_CST
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_SINGLE\_OUTPUT\_PWM
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_SINGLE\_PIN\_CONT
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_SINGLE\_PIN\_TIME
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_MODE\_TRAIN
  - NBCCCommon.h, 1309
  - PFModeConstants, 903
- PF\_OUT\_A
  - NBCCCommon.h, 1310
  - PFOutputs, 905
- PF\_OUT\_B
  - NBCCCommon.h, 1310
  - PFOutputs, 906
- PF\_PIN\_C1
  - NBCCCommon.h, 1310
  - PFPinConstants, 906
- PF\_PIN\_C2
  - NBCCCommon.h, 1310
  - PFPinConstants, 906
- PF\_PWM\_BRAKE
  - NBCCCommon.h, 1310
  - PFPWMOptions, 910
- PF\_PWM\_FLOAT
  - NBCCCommon.h, 1310
  - PFPWMOptions, 910
- PF\_PWM\_FWD1
  - NBCCCommon.h, 1310
  - PFPWMOptions, 910
- PF\_PWM\_FWD2

- NBCCCommon.h, [1310](#)
- PFPWMOptions, [910](#)
- PF\_PWM\_FWD3
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [910](#)
- PF\_PWM\_FWD4
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [910](#)
- PF\_PWM\_FWD5
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [910](#)
- PF\_PWM\_FWD6
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [910](#)
- PF\_PWM\_FWD7
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [910](#)
- PF\_PWM\_REV1
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV2
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV3
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV4
  - NBCCCommon.h, [1311](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV5
  - NBCCCommon.h, [1312](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV6
  - NBCCCommon.h, [1312](#)
  - PFPWMOptions, [911](#)
- PF\_PWM\_REV7
  - NBCCCommon.h, [1312](#)
  - PFPWMOptions, [911](#)
- PFChannelConstants
  - PF\_CHANNEL\_1, [901](#)
  - PF\_CHANNEL\_2, [902](#)
  - PF\_CHANNEL\_3, [902](#)
  - PF\_CHANNEL\_4, [902](#)
- PFCmdConstants
  - PF\_CMD\_BRAKE, [900](#)
  - PF\_CMD\_FLOAT, [900](#)
  - PF\_CMD\_FWD, [900](#)
  - PF\_CMD\_REV, [901](#)
  - PF\_CMD\_STOP, [901](#)
- PFCSTOptions
  - PF\_CST\_CLEAR1\_CLEAR2, [908](#)
  - PF\_CST\_CLEAR1\_SET2, [908](#)
  - PF\_CST\_DECREMENT\_PWM, [908](#)
  - PF\_CST\_FULL\_FWD, [908](#)
  - PF\_CST\_FULL\_REV, [908](#)
  - PF\_CST\_INCREMENT\_PWM, [908](#)
  - PF\_CST\_SET1\_CLEAR2, [908](#)
  - PF\_CST\_SET1\_SET2, [909](#)
  - PF\_CST\_TOGGLE\_DIR, [909](#)
- PFMate channel constants, [954](#)
- PFMate motor constants, [954](#)
- PFMATE\_CHANNEL\_1
  - NBCCCommon.h, [1312](#)
  - PFMateChannelConstants, [955](#)
- PFMATE\_CHANNEL\_2
  - NBCCCommon.h, [1312](#)
  - PFMateChannelConstants, [955](#)
- PFMATE\_CHANNEL\_3
  - NBCCCommon.h, [1312](#)
  - PFMateChannelConstants, [955](#)
- PFMATE\_CHANNEL\_4
  - NBCCCommon.h, [1312](#)
  - PFMateChannelConstants, [955](#)
- PFMATE\_CMD\_GO
  - NBCCCommon.h, [1312](#)
  - PFMateConstants, [953](#)
- PFMATE\_CMD\_RAW
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)
- PFMATE\_MOTORS\_A
  - NBCCCommon.h, [1313](#)
  - PFMateMotorConstants, [954](#)
- PFMATE\_MOTORS\_B
  - NBCCCommon.h, [1313](#)
  - PFMateMotorConstants, [954](#)
- PFMATE\_MOTORS\_BOTH
  - NBCCCommon.h, [1313](#)
  - PFMateMotorConstants, [954](#)
- PFMATE\_REG\_A\_CMD
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)

- PFMATE\_REG\_A\_SPEED
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)
- PFMATE\_REG\_B\_CMD
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)
- PFMATE\_REG\_B\_SPEED
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)
- PFMATE\_REG\_CHANNEL
  - NBCCCommon.h, [1313](#)
  - PFMateConstants, [953](#)
- PFMATE\_REG\_CMD
  - NBCCCommon.h, [1314](#)
  - PFMateConstants, [953](#)
- PFMATE\_REG\_MOTORS
  - NBCCCommon.h, [1314](#)
  - PFMateConstants, [953](#)
- PFMateChannelConstants
  - PFMATE\_CHANNEL\_1, [955](#)
  - PFMATE\_CHANNEL\_2, [955](#)
  - PFMATE\_CHANNEL\_3, [955](#)
  - PFMATE\_CHANNEL\_4, [955](#)
- PFMateConstants
  - PFMATE\_CMD\_GO, [953](#)
  - PFMATE\_CMD\_RAW, [953](#)
  - PFMATE\_REG\_A\_CMD, [953](#)
  - PFMATE\_REG\_A\_SPEED, [953](#)
  - PFMATE\_REG\_B\_CMD, [953](#)
  - PFMATE\_REG\_B\_SPEED, [953](#)
  - PFMATE\_REG\_CHANNEL, [953](#)
  - PFMATE\_REG\_CMD, [953](#)
  - PFMATE\_REG\_MOTORS, [953](#)
- PFMateMotorConstants
  - PFMATE\_MOTORS\_A, [954](#)
  - PFMATE\_MOTORS\_B, [954](#)
  - PFMATE\_MOTORS\_BOTH, [954](#)
- PFMateSend
  - MindSensorsAPI, [213](#)
  - NXCDefs.h, [1741](#)
- PFMateSendRaw
  - MindSensorsAPI, [214](#)
  - NXCDefs.h, [1741](#)
- PFModeConstants
  - PF\_MODE\_COMBO\_DIRECT, [903](#)
- PF\_MODE\_COMBO\_PWM, [903](#)
- PF\_MODE\_SINGLE\_OUTPUT\_CST, [903](#)
- PF\_MODE\_SINGLE\_OUTPUT\_PWM, [903](#)
- PF\_MODE\_SINGLE\_PIN\_CONT, [903](#)
- PF\_MODE\_SINGLE\_PIN\_TIME, [903](#)
- PF\_MODE\_TRAIN, [903](#)
- PFOutputs
  - PF\_OUT\_A, [905](#)
  - PF\_OUT\_B, [906](#)
- PFPinConstants
  - PF\_PIN\_C1, [906](#)
  - PF\_PIN\_C2, [906](#)
- PFPinFuncs
  - PF\_FUNC\_CLEAR, [907](#)
  - PF\_FUNC\_NOCHANGE, [907](#)
  - PF\_FUNC\_SET, [907](#)
  - PF\_FUNC\_TOGGLE, [907](#)
- PFPWMOptions
  - PF\_PWM\_BRAKE, [910](#)
  - PF\_PWM\_FLOAT, [910](#)
  - PF\_PWM\_FWD1, [910](#)
  - PF\_PWM\_FWD2, [910](#)
  - PF\_PWM\_FWD3, [910](#)
  - PF\_PWM\_FWD4, [910](#)
  - PF\_PWM\_FWD5, [910](#)
  - PF\_PWM\_FWD6, [910](#)
  - PF\_PWM\_FWD7, [910](#)
  - PF\_PWM\_REV1, [911](#)
  - PF\_PWM\_REV2, [911](#)
  - PF\_PWM\_REV3, [911](#)
  - PF\_PWM\_REV4, [911](#)
  - PF\_PWM\_REV5, [911](#)
  - PF\_PWM\_REV6, [911](#)
  - PF\_PWM\_REV7, [911](#)
- PI
  - MiscConstants, [261](#)
  - NBCCCommon.h, [1314](#)
- PID constants, [774](#)
- PID\_0
  - NBCCCommon.h, [1314](#)
  - PIDConstants, [775](#)
- PID\_1

- NBCCCommon.h, 1314
- PIDConstants, 775
- PID\_2
  - NBCCCommon.h, 1314
  - PIDConstants, 775
- PID\_3
  - NBCCCommon.h, 1314
  - PIDConstants, 775
- PID\_4
  - NBCCCommon.h, 1314
  - PIDConstants, 775
- PID\_5
  - NBCCCommon.h, 1314
  - PIDConstants, 775
- PID\_6
  - NBCCCommon.h, 1315
  - PIDConstants, 775
- PID\_7
  - NBCCCommon.h, 1315
  - PIDConstants, 775
- PIDConstants
  - PID\_0, 775
  - PID\_1, 775
  - PID\_2, 775
  - PID\_3, 775
  - PID\_4, 775
  - PID\_5, 775
  - PID\_6, 775
  - PID\_7, 775
- Pin
  - InputPinFunctionType, 1071
- PlayFile
  - NXCDefs.h, 1742
  - SoundModuleFunctions, 377
- PlayFileEx
  - NXCDefs.h, 1742
  - SoundModuleFunctions, 377
- PlaySound
  - NXCDefs.h, 1743
  - SoundModuleFunctions, 378
- PlayTone
  - NXCDefs.h, 1743
  - SoundModuleFunctions, 378
- PlayToneEx
  - NXCDefs.h, 1744
  - SoundModuleFunctions, 379
- PlayTones
  - NXCDefs.h, 1744
  - SoundModuleFunctions, 379
- PointOut
  - DisplayModuleFunctions, 363
  - NXCDefs.h, 1745
- Points
  - DrawPolygonType, 1054
- PolyOut
  - DisplayModuleFunctions, 364
  - NXCDefs.h, 1745
- POOL\_MAX\_SIZE
  - CommandModuleConstants, 52
  - NBCCCommon.h, 1315
- PoolSize
  - MemoryManagerType, 1090
- Port
  - ColorSensorReadType, 1019
  - CommLSCheckStatusType, 1031
  - CommLSReadType, 1032
  - CommLSWriteExType, 1034
  - CommLSWriteType, 1035
  - InputPinFunctionType, 1072
  - InputValuesType, 1073
  - OutputStateType, 1094
- Pos
  - cstringAPI, 664
  - NXCDefs.h, 1746
- Position
  - FileTellType, 1069
- PosRegAddAngle
  - NXCDefs.h, 1746
  - OutputModuleFunctions, 339
- PosRegEnable
  - NXCDefs.h, 1747
  - OutputModuleFunctions, 340
- PosRegSetAngle
  - NXCDefs.h, 1748
  - OutputModuleFunctions, 341
- PosRegSetMax
  - NXCDefs.h, 1748
  - OutputModuleFunctions, 341
- Pow
  - cmathAPI, 612
  - NXCDefs.h, 1489
- pow

- cmathAPI, [627](#)
- NXCDefs.h, [1749](#)
- Power
  - OutputStateType, [1094](#)
- Power Function channel constants, [901](#)
- Power Function command constants, [900](#)
- Power Function CST options constants, [907](#)
- Power Function mode constants, [902](#)
- Power Function output constants, [905](#)
- Power Function pin constants, [906](#)
- Power Function PWM option constants, [909](#)
- Power Function single pin function constants, [906](#)
- PowerDown
  - IOCtrlModuleFunctions, [460](#)
  - NXCDefs.h, [1749](#)
- PowerField
  - NBCCCommon.h, [1315](#)
  - OutputFieldConstants, [786](#)
- PowerOn constants, [715](#)
- PowerState
  - CommBTONOffType, [1022](#)
- Precedes
  - CommandModuleFunctions, [430](#)
  - NXCDefs.h, [1749](#)
- Pressed
  - ReadButtonType, [1098](#)
- printf
  - cstdioAPI, [641](#)
  - NXCDefs.h, [1750](#)
- PROG\_ABORT
  - CommandProgStatus, [712](#)
  - NBCCCommon.h, [1315](#)
- PROG\_ERROR
  - CommandProgStatus, [712](#)
  - NBCCCommon.h, [1315](#)
- PROG\_IDLE
  - CommandProgStatus, [712](#)
  - NBCCCommon.h, [1315](#)
- PROG\_OK
  - CommandProgStatus, [712](#)
  - NBCCCommon.h, [1315](#)
- PROG\_RESET
  - CommandProgStatus, [712](#)
- NBCCCommon.h, [1316](#)
- PROG\_RUNNING
  - CommandProgStatus, [712](#)
  - NBCCCommon.h, [1316](#)
- Program status constants, [711](#)
- Property constants, [679](#)
- PSP\_BTNSET1\_DOWN
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1316](#)
- PSP\_BTNSET1\_L3
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1316](#)
- PSP\_BTNSET1\_LEFT
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1316](#)
- PSP\_BTNSET1\_R3
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1316](#)
- PSP\_BTNSET1\_RIGHT
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1316](#)
- PSP\_BTNSET1\_SELECT
  - MSPSPNXBtnSet1, [943](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET1\_START
  - MSPSPNXBtnSet1, [944](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET1\_UP
  - MSPSPNXBtnSet1, [944](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET2\_CIRCLE
  - MSPSPNXBtnSet2, [944](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET2\_CROSS
  - MSPSPNXBtnSet2, [945](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET2\_L1
  - MSPSPNXBtnSet2, [945](#)
  - NBCCCommon.h, [1317](#)
- PSP\_BTNSET2\_L2
  - MSPSPNXBtnSet2, [945](#)
  - NBCCCommon.h, [1318](#)
- PSP\_BTNSET2\_R1
  - MSPSPNXBtnSet2, [945](#)
  - NBCCCommon.h, [1318](#)
- PSP\_BTNSET2\_R2

- MSPSPNXBtnSet2, 945
- NBCCCommon.h, 1318
- PSP\_BTNSET2\_SQUARE
  - MSPSPNXBtnSet2, 945
  - NBCCCommon.h, 1318
- PSP\_BTNSET2\_TRIANGLE
  - MSPSPNXBtnSet2, 946
  - NBCCCommon.h, 1318
- PSP\_CMD\_ANALOG
  - MSPSPNX, 941
  - NBCCCommon.h, 1319
- PSP\_CMD\_DIGITAL
  - MSPSPNX, 941
  - NBCCCommon.h, 1319
- PSP\_REG\_BTNSET1
  - MSPSPNX, 941
  - NBCCCommon.h, 1319
- PSP\_REG\_BTNSET2
  - MSPSPNX, 941
  - NBCCCommon.h, 1319
- PSP\_REG\_XLEFT
  - MSPSPNX, 942
  - NBCCCommon.h, 1319
- PSP\_REG\_XRIGHT
  - MSPSPNX, 942
  - NBCCCommon.h, 1319
- PSP\_REG\_YLEFT
  - MSPSPNX, 942
  - NBCCCommon.h, 1319
- PSP\_REG\_YRIGHT
  - MSPSPNX, 942
  - NBCCCommon.h, 1319
- PSPNXAnalog
  - MindSensorsAPI, 215
  - NXCDefs.h, 1750
- PSPNXDigital
  - MindSensorsAPI, 215
  - NXCDefs.h, 1751
- putc
  - cstdioAPI, 635
  - NXCDefs.h, 1489
- QueueID
  - MessageReadType, 1092
  - MessageWriteType, 1093
- quot
- div\_t, 1042
- ldiv\_t, 1084
- RADIANS\_PER\_DEGREE
  - MiscConstants, 262
  - NBCCCommon.h, 1319
- rand
  - cstdlibAPI, 650
  - NXCDefs.h, 1751
- RAND\_MAX
  - NBCCCommon.h, 1320
  - NXTLimits, 1012
- Random
  - cstdlibAPI, 650
  - NXCDefs.h, 1751
- RandomEx
  - NBCCCommon.h, 1320
  - SysCallConstants, 690
- RandomExType, 1095
  - ReSeed, 1096
  - Seed, 1096
- RandomNumber
  - NBCCCommon.h, 1320
  - SysCallConstants, 690
- RandomNumberType, 1096
  - Result, 1097
- RawArray
  - ColorSensorReadType, 1019
- RawVal
  - ComputeCalibValueType, 1037
- RawValue
  - InputValuesType, 1073
- RawValueField
  - InputFieldConstants, 762
  - NBCCCommon.h, 1320
- RC\_PROP\_BTONOFF
  - NBCCCommon.h, 1320
  - RCPropertyConstants, 680
- RC\_PROP\_DEBUGGING
  - NBCCCommon.h, 1320
  - RCPropertyConstants, 680
- RC\_PROP\_SLEEP\_TIMEOUT
  - NBCCCommon.h, 1320
  - RCPropertyConstants, 680
- RC\_PROP\_SOUND\_LEVEL
  - NBCCCommon.h, 1320

- RCPropertyConstants, 680
- RCPropertyConstants
  - RC\_PROP\_BTONOFF, 680
  - RC\_PROP\_DEBUGGING, 680
  - RC\_PROP\_SLEEP\_TIMEOUT, 680
  - RC\_PROP\_SOUND\_LEVEL, 680
- RCX and Scout opcode constants, 889
- RCX and Scout sound constants, 870
- RCX and Scout source constants, 883
- RCX constants, 863
- RCX IR remote constants, 868
- RCX output constants, 864
- RCX output direction constants, 866
- RCX output mode constants, 865
- RCX output power constants, 867
- RCX\_AbsVarOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 891
- RCX\_AndVarOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 891
- RCX\_AutoOffOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 891
- RCX\_BatteryLevelOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 891
- RCX\_BatteryLevelSrc
  - NBCCCommon.h, 1321
  - RCXSourceConstants, 885
- RCX\_BootModeOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 892
- RCX\_CalibrateEventOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 892
- RCX\_ClearAllEventsOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 892
- RCX\_ClearCounterOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 892
- RCX\_ClearMsgOp
  - NBCCCommon.h, 1321
  - RCXOpcodeConstants, 892
- RCX\_ClearSensorOp
  - NBCCCommon.h, 1322
  - RCXOpcodeConstants, 892
- RCX\_ClearSoundOp
  - NBCCCommon.h, 1322
  - RCXOpcodeConstants, 892
- RCX\_ClearTimerOp
  - NBCCCommon.h, 1322
  - RCXOpcodeConstants, 892
- RCX\_ClickCounterSrc
  - NBCCCommon.h, 1322
  - RCXSourceConstants, 885
- RCX\_ConstantSrc
  - NBCCCommon.h, 1322
  - RCXSourceConstants, 885
- RCX\_CounterSrc
  - NBCCCommon.h, 1322
  - RCXSourceConstants, 885
- RCX\_DatalogOp
  - NBCCCommon.h, 1322
  - RCXOpcodeConstants, 892
- RCX\_DatalogRawDirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 885
- RCX\_DatalogRawIndirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 885
- RCX\_DatalogSrcDirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 885
- RCX\_DatalogSrcIndirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 885
- RCX\_DatalogValueDirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 886
- RCX\_DatalogValueIndirectSrc
  - NBCCCommon.h, 1323
  - RCXSourceConstants, 886
- RCX\_DecCounterOp
  - NBCCCommon.h, 1323
  - RCXOpcodeConstants, 892
- RCX\_DeleteSubOp
  - NBCCCommon.h, 1323
  - RCXOpcodeConstants, 893
- RCX\_DeleteSubsOp
  - NBCCCommon.h, 1323

- RCXOpcodeConstants, 893
- RCX\_DeleteTaskOp
  - NBCCCommon.h, 1323
  - RCXOpcodeConstants, 893
- RCX\_DeleteTasksOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_DirectEventOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_DisplayOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_DivVarOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_DurationSrc
  - NBCCCommon.h, 1324
  - RCXSourceConstants, 886
- RCX\_EventStateSrc
  - NBCCCommon.h, 1324
  - RCXSourceConstants, 886
- RCX\_FirmwareVersionSrc
  - NBCCCommon.h, 1324
  - RCXSourceConstants, 886
- RCX\_GlobalMotorStatusSrc
  - NBCCCommon.h, 1324
  - RCXSourceConstants, 886
- RCX\_GOutputDirOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_GOutputModeOp
  - NBCCCommon.h, 1324
  - RCXOpcodeConstants, 893
- RCX\_GOutputPowerOp
  - NBCCCommon.h, 1325
  - RCXOpcodeConstants, 893
- RCX\_HysteresisSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 886
- RCX\_IncCounterOp
  - NBCCCommon.h, 1325
  - RCXOpcodeConstants, 894
- RCX\_IndirectVarSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 886
- RCX\_InputBooleanSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 886
- RCX\_InputModeOp
  - NBCCCommon.h, 1325
  - RCXOpcodeConstants, 894
- RCX\_InputModeSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 886
- RCX\_InputRawSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 887
- RCX\_InputTypeOp
  - NBCCCommon.h, 1325
  - RCXOpcodeConstants, 894
- RCX\_InputTypeSrc
  - NBCCCommon.h, 1325
  - RCXSourceConstants, 887
- RCX\_InputValueSrc
  - NBCCCommon.h, 1326
  - RCXSourceConstants, 887
- RCX\_IRModeOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LightOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LowerThresholdSrc
  - NBCCCommon.h, 1326
  - RCXSourceConstants, 887
- RCX\_LSblinkTimeOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LSCalibrateOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LSHysteresisOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LSLowerThreshOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_LSUpperThreshOp
  - NBCCCommon.h, 1326
  - RCXOpcodeConstants, 894
- RCX\_MessageOp

- NBCCCommon.h, 1327
- RCXOpcodeConstants, 895
- RCX\_MessageSrc
  - NBCCCommon.h, 1327
  - RCXSourceConstants, 887
- RCX\_MulVarOp
  - NBCCCommon.h, 1327
  - RCXOpcodeConstants, 895
- RCX\_MuteSoundOp
  - NBCCCommon.h, 1327
  - RCXOpcodeConstants, 895
- RCX\_OnOffFloatOp
  - NBCCCommon.h, 1327
  - RCXOpcodeConstants, 895
- RCX\_OrVarOp
  - NBCCCommon.h, 1327
  - RCXOpcodeConstants, 895
- RCX\_OUT\_A
  - NBCCCommon.h, 1327
  - RCXOutputConstants, 864
- RCX\_OUT\_AB
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_ABC
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_AC
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_B
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_BC
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_C
  - NBCCCommon.h, 1328
  - RCXOutputConstants, 865
- RCX\_OUT\_FLOAT
  - NBCCCommon.h, 1328
  - RCXOutputMode, 866
- RCX\_OUT\_FULL
  - NBCCCommon.h, 1328
  - RCXOutputPower, 867
- RCX\_OUT\_FWD
  - NBCCCommon.h, 1329
  - RCXOutputDirection, 866
- RCX\_OUT\_HALF
  - NBCCCommon.h, 1329
  - RCXOutputPower, 867
- RCX\_OUT\_LOW
  - NBCCCommon.h, 1329
  - RCXOutputPower, 867
- RCX\_OUT\_OFF
  - NBCCCommon.h, 1329
  - RCXOutputMode, 866
- RCX\_OUT\_ON
  - NBCCCommon.h, 1329
  - RCXOutputMode, 866
- RCX\_OUT\_REV
  - NBCCCommon.h, 1329
  - RCXOutputDirection, 867
- RCX\_OUT\_TOGGLE
  - NBCCCommon.h, 1329
  - RCXOutputDirection, 867
- RCX\_OutputDirOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 895
- RCX\_OutputPowerOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 895
- RCX\_OutputStatusSrc
  - NBCCCommon.h, 1330
  - RCXSourceConstants, 887
- RCX\_PBTurnOffOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 895
- RCX\_PingOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 895
- RCX\_PlaySoundOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 895
- RCX\_PlayToneOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 896
- RCX\_PlayToneVarOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 896
- RCX\_PollMemoryOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 896

- RCX\_PollOp
  - NBCCCommon.h, 1330
  - RCXOpcodeConstants, 896
- RCX\_ProgramSlotSrc
  - NBCCCommon.h, 1331
  - RCXSourceConstants, 887
- RCX\_RandomSrc
  - NBCCCommon.h, 1331
  - RCXSourceConstants, 887
- RCX\_RemoteKeysReleased
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 868
- RCX\_RemoteOp
  - NBCCCommon.h, 1331
  - RCXOpcodeConstants, 896
- RCX\_RemoteOutABackward
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 868
- RCX\_RemoteOutAForward
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 868
- RCX\_RemoteOutBBackward
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 869
- RCX\_RemoteOutBForward
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 869
- RCX\_RemoteOutCBackward
  - NBCCCommon.h, 1331
  - RCXRemoteConstants, 869
- RCX\_RemoteOutCForward
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemotePBMessage1
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemotePBMessage2
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemotePBMessage3
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemotePlayASound
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemoteSelProgram1
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 869
- RCX\_RemoteSelProgram2
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 870
- RCX\_RemoteSelProgram3
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 870
- RCX\_RemoteSelProgram4
  - NBCCCommon.h, 1332
  - RCXRemoteConstants, 870
- RCX\_RemoteSelProgram5
  - NBCCCommon.h, 1333
  - RCXRemoteConstants, 870
- RCX\_RemoteStopOutOff
  - NBCCCommon.h, 1333
  - RCXRemoteConstants, 870
- RCX\_ScoutCounterLimitSrc
  - NBCCCommon.h, 1333
  - RCXSourceConstants, 888
- RCX\_ScoutEventFBSrc
  - NBCCCommon.h, 1333
  - RCXSourceConstants, 888
- RCX\_ScoutLightParamsSrc
  - NBCCCommon.h, 1333
  - RCXSourceConstants, 888
- RCX\_ScoutOp
  - NBCCCommon.h, 1333
  - RCXOpcodeConstants, 896
- RCX\_ScoutRulesOp
  - NBCCCommon.h, 1333
  - RCXOpcodeConstants, 896
- RCX\_ScoutRulesSrc
  - NBCCCommon.h, 1333
  - RCXSourceConstants, 888
- RCX\_ScoutTimerLimitSrc
  - NBCCCommon.h, 1333
  - RCXSourceConstants, 888
- RCX\_SelectProgramOp
  - NBCCCommon.h, 1333
  - RCXOpcodeConstants, 896
- RCX\_SendUARTDataOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 896
- RCX\_SetCounterOp
  - NBCCCommon.h, 1334

- RCXOpcodeConstants, 896
- RCX\_SetDatalogOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetEventOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetFeedbackOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetPriorityOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetSourceValueOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetTimerLimitOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetVarOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SetWatchOp
  - NBCCCommon.h, 1334
  - RCXOpcodeConstants, 897
- RCX\_SgnVarOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 897
- RCX\_SoundOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 897
- RCX\_StartTaskOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 898
- RCX\_StopAllTasksOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 898
- RCX\_StopTaskOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 898
- RCX\_SubVarOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 898
- RCX\_SumVarOp
  - NBCCCommon.h, 1335
  - RCXOpcodeConstants, 898
- RCX\_TaskEventsSrc
  - NBCCCommon.h, 1335
  - RCXSourceConstants, 888
- RCX\_TenMSTimerSrc
  - NBCCCommon.h, 1335
  - RCXSourceConstants, 888
- RCX\_TimerSrc
  - NBCCCommon.h, 1335
  - RCXSourceConstants, 888
- RCX\_UARTSetupSrc
  - NBCCCommon.h, 1336
  - RCXSourceConstants, 888
- RCX\_UnlockFirmOp
  - NBCCCommon.h, 1336
  - RCXOpcodeConstants, 898
- RCX\_UnlockOp
  - NBCCCommon.h, 1336
  - RCXOpcodeConstants, 898
- RCX\_UnmuteSoundOp
  - NBCCCommon.h, 1336
  - RCXOpcodeConstants, 898
- RCX\_UploadDatalogOp
  - NBCCCommon.h, 1336
  - RCXOpcodeConstants, 898
- RCX\_UpperThresholdSrc
  - NBCCCommon.h, 1336
  - RCXSourceConstants, 889
- RCX\_VariableSrc
  - NBCCCommon.h, 1336
  - RCXSourceConstants, 889
- RCX\_ViewSourceValOp
  - NBCCCommon.h, 1336
  - RCXOpcodeConstants, 898
- RCX\_VLLOp
  - NBCCCommon.h, 1337
  - RCXOpcodeConstants, 899
- RCX\_WatchSrc
  - NBCCCommon.h, 1337
  - RCXSourceConstants, 889
- RCXOpcodeConstants
  - RCX\_AbsVarOp, 891
  - RCX\_AndVarOp, 891
  - RCX\_AutoOffOp, 891
  - RCX\_BatteryLevelOp, 891
  - RCX\_BootModeOp, 892
  - RCX\_CalibrateEventOp, 892

- RCX\_ClearAllEventsOp, 892
- RCX\_ClearCounterOp, 892
- RCX\_ClearMsgOp, 892
- RCX\_ClearSensorOp, 892
- RCX\_ClearSoundOp, 892
- RCX\_ClearTimerOp, 892
- RCX\_DatalogOp, 892
- RCX\_DecCounterOp, 892
- RCX\_DeleteSubOp, 893
- RCX\_DeleteSubsOp, 893
- RCX\_DeleteTaskOp, 893
- RCX\_DeleteTasksOp, 893
- RCX\_DirectEventOp, 893
- RCX\_DisplayOp, 893
- RCX\_DivVarOp, 893
- RCX\_GOutputDirOp, 893
- RCX\_GOutputModeOp, 893
- RCX\_GOutputPowerOp, 893
- RCX\_IncCounterOp, 894
- RCX\_InputModeOp, 894
- RCX\_InputTypeOp, 894
- RCX\_IRModeOp, 894
- RCX\_LightOp, 894
- RCX\_LSblinkTimeOp, 894
- RCX\_LSCalibrateOp, 894
- RCX\_LSHysteresisOp, 894
- RCX\_LSLowerThreshOp, 894
- RCX\_LSUpperThreshOp, 894
- RCX\_MessageOp, 895
- RCX\_MulVarOp, 895
- RCX\_MuteSoundOp, 895
- RCX\_OnOffFloatOp, 895
- RCX\_OrVarOp, 895
- RCX\_OutputDirOp, 895
- RCX\_OutputPowerOp, 895
- RCX\_PBTurnOffOp, 895
- RCX\_PingOp, 895
- RCX\_PlaySoundOp, 895
- RCX\_PlayToneOp, 896
- RCX\_PlayToneVarOp, 896
- RCX\_PollMemoryOp, 896
- RCX\_PollOp, 896
- RCX\_RemoteOp, 896
- RCX\_ScoutOp, 896
- RCX\_ScoutRulesOp, 896
- RCX\_SelectProgramOp, 896
- RCX\_SendUARTDataOp, 896
- RCX\_SetCounterOp, 896
- RCX\_SetDatalogOp, 897
- RCX\_SetEventOp, 897
- RCX\_SetFeedbackOp, 897
- RCX\_SetPriorityOp, 897
- RCX\_SetSourceValueOp, 897
- RCX\_SetTimerLimitOp, 897
- RCX\_SetVarOp, 897
- RCX\_SetWatchOp, 897
- RCX\_SgnVarOp, 897
- RCX\_SoundOp, 897
- RCX\_StartTaskOp, 898
- RCX\_StopAllTasksOp, 898
- RCX\_StopTaskOp, 898
- RCX\_SubVarOp, 898
- RCX\_SumVarOp, 898
- RCX\_UnlockFirmOp, 898
- RCX\_UnlockOp, 898
- RCX\_UnmuteSoundOp, 898
- RCX\_UploadDatalogOp, 898
- RCX\_ViewSourceValOp, 898
- RCX\_VLLOp, 899
- RCXOutputConstants
  - RCX\_OUT\_A, 864
  - RCX\_OUT\_AB, 865
  - RCX\_OUT\_ABC, 865
  - RCX\_OUT\_AC, 865
  - RCX\_OUT\_B, 865
  - RCX\_OUT\_BC, 865
  - RCX\_OUT\_C, 865
- RCXOutputDirection
  - RCX\_OUT\_FWD, 866
  - RCX\_OUT\_REV, 867
  - RCX\_OUT\_TOGGLE, 867
- RCXOutputMode
  - RCX\_OUT\_FLOAT, 866
  - RCX\_OUT\_OFF, 866
  - RCX\_OUT\_ON, 866
- RCXOutputPower
  - RCX\_OUT\_FULL, 867
  - RCX\_OUT\_HALF, 867
  - RCX\_OUT\_LOW, 867
- RCXRemoteConstants
  - RCX\_RemoteKeysReleased, 868
  - RCX\_RemoteOutABackward, 868

- RCX\_RemoteOutAForward, 868
- RCX\_RemoteOutBBackward, 869
- RCX\_RemoteOutBForward, 869
- RCX\_RemoteOutCBackward, 869
- RCX\_RemoteOutCForward, 869
- RCX\_RemotePBMessage1, 869
- RCX\_RemotePBMessage2, 869
- RCX\_RemotePBMessage3, 869
- RCX\_RemotePlayASound, 869
- RCX\_RemoteSelProgram1, 869
- RCX\_RemoteSelProgram2, 870
- RCX\_RemoteSelProgram3, 870
- RCX\_RemoteSelProgram4, 870
- RCX\_RemoteSelProgram5, 870
- RCX\_RemoteStopOutOff, 870
- RCXSoundConstants
  - SOUND\_CLICK, 871
  - SOUND\_DOUBLE\_BEEP, 871
  - SOUND\_DOWN, 871
  - SOUND\_FAST\_UP, 871
  - SOUND\_LOW\_BEEP, 871
  - SOUND\_UP, 871
- RCXSourceConstants
  - RCX\_BatteryLevelSrc, 885
  - RCX\_ClickCounterSrc, 885
  - RCX\_ConstantSrc, 885
  - RCX\_CounterSrc, 885
  - RCX\_DatalogRawDirectSrc, 885
  - RCX\_DatalogRawIndirectSrc, 885
  - RCX\_DatalogSrcDirectSrc, 885
  - RCX\_DatalogSrcIndirectSrc, 885
  - RCX\_DatalogValueDirectSrc, 886
  - RCX\_DatalogValueIndirectSrc, 886
  - RCX\_DurationSrc, 886
  - RCX\_EventStateSrc, 886
  - RCX\_FirmwareVersionSrc, 886
  - RCX\_GlobalMotorStatusSrc, 886
  - RCX\_HysteresisSrc, 886
  - RCX\_IndirectVarSrc, 886
  - RCX\_InputBooleanSrc, 886
  - RCX\_InputModeSrc, 886
  - RCX\_InputRawSrc, 887
  - RCX\_InputTypeSrc, 887
  - RCX\_InputValueSrc, 887
  - RCX\_LowerThresholdSrc, 887
  - RCX\_MessageSrc, 887
  - RCX\_OutputStatusSrc, 887
  - RCX\_ProgramSlotSrc, 887
  - RCX\_RandomSrc, 887
  - RCX\_ScoutCounterLimitSrc, 888
  - RCX\_ScoutEventFBSrc, 888
  - RCX\_ScoutLightParamsSrc, 888
  - RCX\_ScoutRulesSrc, 888
  - RCX\_ScoutTimerLimitSrc, 888
  - RCX\_TaskEventsSrc, 888
  - RCX\_TenMSTimerSrc, 888
  - RCX\_TimerSrc, 888
  - RCX\_UARTSetupSrc, 888
  - RCX\_UpperThresholdSrc, 889
  - RCX\_VariableSrc, 889
  - RCX\_WatchSrc, 889
- Read
  - LoaderModuleFunctions, 582
  - NXCDefs.h, 1752
- ReadButton
  - NBCCommon.h, 1337
  - SysCallConstants, 690
- ReadButtonEx
  - ButtonModuleFunctions, 556
  - NXCDefs.h, 1752
- ReadButtonType, 1097
  - Count, 1098
  - Index, 1098
  - Pressed, 1098
  - Reset, 1098
  - Result, 1098
- ReadBytes
  - LoaderModuleFunctions, 582
  - NXCDefs.h, 1753
- ReadI2CRegister
  - LowSpeedModuleFunctions, 400
  - NXCDefs.h, 1753
- ReadLastResponse
  - NBCCommon.h, 1337
  - SysCallConstants, 690
- ReadLastResponseType, 1098
  - Buffer, 1099
  - Clear, 1099
  - Command, 1099
  - Length, 1099
  - Result, 1100
- ReadLn

- LoaderModuleFunctions, 583
- NXCDefs.h, 1754
- ReadLnString
  - LoaderModuleFunctions, 583
  - NXCDefs.h, 1754
- ReadNRLinkBytes
  - MindSensorsAPI, 216
  - NXCDefs.h, 1755
- ReadSemData
  - NBCCCommon.h, 1337
  - SysCallConstants, 690
- ReadSemDataType, 1100
  - Request, 1101
  - SemData, 1101
- ReadSensorColorEx
  - InputModuleFunctions, 295
  - NXCDefs.h, 1755
- ReadSensorColorRaw
  - InputModuleFunctions, 295
  - NXCDefs.h, 1756
- ReadSensorDIAccl
  - DexterIndustriesAPI, 236
  - NXCDefs.h, 1757
- ReadSensorDIAccl8
  - DexterIndustriesAPI, 236
  - NXCDefs.h, 1757
- ReadSensorDIAccl8Raw
  - DexterIndustriesAPI, 236
  - NXCDefs.h, 1757
- ReadSensorDIAcclDrift
  - DexterIndustriesAPI, 237
  - NXCDefs.h, 1758
- ReadSensorDIAcclRaw
  - DexterIndustriesAPI, 237
  - NXCDefs.h, 1758
- ReadSensorDIGyro
  - DexterIndustriesAPI, 238
  - NXCDefs.h, 1759
- ReadSensorDIGyroRaw
  - DexterIndustriesAPI, 238
  - NXCDefs.h, 1759
- ReadSensorEMeter
  - LowSpeedModuleFunctions, 401
  - NXCDefs.h, 1760
- ReadSensorHTAccel
  - HiTechnicAPI, 97
  - NXCDefs.h, 1761
- ReadSensorHTAngle
  - HiTechnicAPI, 98
  - NXCDefs.h, 1761
- ReadSensorHTBarometric
  - HiTechnicAPI, 98
  - NXCDefs.h, 1762
- ReadSensorHTColor
  - HiTechnicAPI, 99
  - NXCDefs.h, 1762
- ReadSensorHTColor2Active
  - HiTechnicAPI, 99
  - NXCDefs.h, 1763
- ReadSensorHTIRReceiver
  - HiTechnicAPI, 100
  - NXCDefs.h, 1763
- ReadSensorHTIRReceiverEx
  - HiTechnicAPI, 100
  - NXCDefs.h, 1764
- ReadSensorHTIRSeeker
  - HiTechnicAPI, 101
  - NXCDefs.h, 1764
- ReadSensorHTIRSeeker2AC
  - HiTechnicAPI, 101
  - NXCDefs.h, 1765
- ReadSensorHTIRSeeker2DC
  - HiTechnicAPI, 102
  - NXCDefs.h, 1766
- ReadSensorHTNormalizedColor
  - HiTechnicAPI, 103
  - NXCDefs.h, 1766
- ReadSensorHTNormalizedColor2Active
  - HiTechnicAPI, 103
  - NXCDefs.h, 1767
- ReadSensorHTProtoAllAnalog
  - HiTechnicAPI, 104
  - NXCDefs.h, 1768
- ReadSensorHTRawColor
  - HiTechnicAPI, 105
  - NXCDefs.h, 1768
- ReadSensorHTRawColor2
  - HiTechnicAPI, 105
  - NXCDefs.h, 1769
- ReadSensorHTSuperProAllAnalog
  - HiTechnicAPI, 106
  - NXCDefs.h, 1769

- ReadSensorHTSuperProAnalogOut
  - HiTechnicAPI, 106
  - NXCDefs.h, 1770
- ReadSensorHTTouchMultiplexer
  - HiTechnicAPI, 107
  - NXCDefs.h, 1771
- ReadSensorMIXG1300L
  - MicrominifinityFunctions, 597
  - NXCDefs.h, 1771
- ReadSensorMSAccel
  - MindSensorsAPI, 216
  - NXCDefs.h, 1772
- ReadSensorMSPlayStation
  - MindSensorsAPI, 217
  - NXCDefs.h, 1772
- ReadSensorMSRTClock
  - MindSensorsAPI, 217
  - NXCDefs.h, 1773
- ReadSensorMSTilt
  - MindSensorsAPI, 218
  - NXCDefs.h, 1774
- ReadSensorUSEx
  - LowSpeedModuleFunctions, 401
  - NXCDefs.h, 1774
- RebootInFirmwareMode
  - IOCtrlModuleFunctions, 460
  - NXCDefs.h, 1775
- ReceiveMessage
  - CommModuleFunctions, 487
  - NXCDefs.h, 1775
- ReceiveRemoteBool
  - CommModuleFunctions, 488
  - NXCDefs.h, 1775
- ReceiveRemoteMessageEx
  - CommModuleFunctions, 488
  - NXCDefs.h, 1776
- ReceiveRemoteNumber
  - CommModuleFunctions, 489
  - NXCDefs.h, 1776
- ReceiveRemoteString
  - CommModuleFunctions, 489
  - NXCDefs.h, 1777
- RechargeableBattery
  - NXCDefs.h, 1777
  - UiModuleFunctions, 564
- RectOut
  - DisplayModuleFunctions, 365
  - NXCDefs.h, 1778
- RegDValueField
  - NBCCCommon.h, 1337
  - OutputFieldConstants, 787
- RegIValueField
  - NBCCCommon.h, 1337
  - OutputFieldConstants, 787
- RegMode
  - OutputStateType, 1095
- RegModeField
  - NBCCCommon.h, 1337
  - OutputFieldConstants, 787
- RegPValueField
  - NBCCCommon.h, 1338
  - OutputFieldConstants, 788
- reladdressOf
  - cstringAPI, 665
  - NXCDefs.h, 1778
- Release
  - CommandModuleFunctions, 431
  - NXCDefs.h, 1779
- rem
  - div\_t, 1042
  - ldiv\_t, 1084
- Remote connection constants, 836
- Remote control (direct commands) errors, 710
- RemoteBluetoothFactoryReset
  - CommModuleSCFunctions, 537
  - NXCDefs.h, 1779
- RemoteCloseFile
  - CommModuleSCFunctions, 537
  - NXCDefs.h, 1780
- RemoteConnectionIdle
  - CommModuleFunctions, 490
  - NXCDefs.h, 1780
- RemoteConnectionWrite
  - CommModuleFunctions, 490
  - NXCDefs.h, 1781
- RemoteDatalogRead
  - CommModuleDCFunctions, 519
  - NXCDefs.h, 1781
- RemoteDatalogSetTimes
  - CommModuleDCFunctions, 520
  - NXCDefs.h, 1782

- RemoteDeleteFile
  - CommModuleSCFunctions, 538
  - NXCDefs.h, 1783
- RemoteDeleteUserFlash
  - CommModuleSCFunctions, 538
  - NXCDefs.h, 1783
- RemoteFindFirstFile
  - CommModuleSCFunctions, 539
  - NXCDefs.h, 1784
- RemoteFindNextFile
  - CommModuleSCFunctions, 540
  - NXCDefs.h, 1784
- RemoteGetBatteryLevel
  - CommModuleDCFunctions, 520
  - NXCDefs.h, 1785
- RemoteGetBluetoothAddress
  - CommModuleSCFunctions, 540
  - NXCDefs.h, 1785
- RemoteGetConnectionCount
  - CommModuleDCFunctions, 521
  - NXCDefs.h, 1786
- RemoteGetConnectionName
  - CommModuleDCFunctions, 522
  - NXCDefs.h, 1787
- RemoteGetContactCount
  - CommModuleDCFunctions, 522
  - NXCDefs.h, 1787
- RemoteGetContactName
  - CommModuleDCFunctions, 523
  - NXCDefs.h, 1788
- RemoteGetCurrentProgramName
  - CommModuleDCFunctions, 523
  - NXCDefs.h, 1788
- RemoteGetDeviceInfo
  - CommModuleSCFunctions, 541
  - NXCDefs.h, 1789
- RemoteGetFirmwareVersion
  - CommModuleSCFunctions, 541
  - NXCDefs.h, 1789
- RemoteGetInputValues
  - CommModuleDCFunctions, 524
  - NXCDefs.h, 1790
- RemoteGetOutputState
  - CommModuleDCFunctions, 524
  - NXCDefs.h, 1791
- RemoteGetProperty
  - CommModuleDCFunctions, 525
  - NXCDefs.h, 1791
- RemoteIOMapRead
  - CommModuleSCFunctions, 542
  - NXCDefs.h, 1792
- RemoteIOMapWriteBytes
  - CommModuleSCFunctions, 543
  - NXCDefs.h, 1793
- RemoteIOMapWriteValue
  - CommModuleSCFunctions, 543
  - NXCDefs.h, 1793
- RemoteKeepAlive
  - CommModuleDCFunctions, 526
  - NXCDefs.h, 1794
- RemoteLowSpeedGetStatus
  - CommModuleDCFunctions, 526
  - NXCDefs.h, 1794
- RemoteLowSpeedRead
  - CommModuleDCFunctions, 527
  - NXCDefs.h, 1795
- RemoteLowSpeedWrite
  - CommModuleDCFunctions, 527
  - NXCDefs.h, 1795
- RemoteMessageRead
  - CommModuleDCFunctions, 528
  - NXCDefs.h, 1796
- RemoteMessageWrite
  - CommModuleDCFunctions, 528
  - NXCDefs.h, 1797
- RemoteOpenAppendData
  - CommModuleSCFunctions, 544
  - NXCDefs.h, 1797
- RemoteOpenRead
  - CommModuleSCFunctions, 545
  - NXCDefs.h, 1798
- RemoteOpenWrite
  - CommModuleSCFunctions, 545
  - NXCDefs.h, 1798
- RemoteOpenWriteData
  - CommModuleSCFunctions, 546
  - NXCDefs.h, 1799
- RemoteOpenWriteLinear
  - CommModuleSCFunctions, 546
  - NXCDefs.h, 1800
- RemotePlaySoundFile
  - CommModuleDCFunctions, 529

- NXCDefs.h, 1800
- RemotePlayTone
  - CommModuleDCFunctions, 529
  - NXCDefs.h, 1801
- RemotePollCommand
  - CommModuleSCFunctions, 547
  - NXCDefs.h, 1801
- RemotePollCommandLength
  - CommModuleSCFunctions, 548
  - NXCDefs.h, 1802
- RemoteRead
  - CommModuleSCFunctions, 548
  - NXCDefs.h, 1803
- RemoteRenameFile
  - CommModuleSCFunctions, 549
  - NXCDefs.h, 1803
- RemoteResetMotorPosition
  - CommModuleDCFunctions, 530
  - NXCDefs.h, 1804
- RemoteResetScaledValue
  - CommModuleDCFunctions, 530
  - NXCDefs.h, 1805
- RemoteResetTachoCount
  - CommModuleDCFunctions, 531
  - NXCDefs.h, 1805
- RemoteSetBrickName
  - CommModuleSCFunctions, 550
  - NXCDefs.h, 1806
- RemoteSetInputMode
  - CommModuleDCFunctions, 531
  - NXCDefs.h, 1806
- RemoteSetOutputState
  - CommModuleDCFunctions, 532
  - NXCDefs.h, 1807
- RemoteSetProperty
  - CommModuleDCFunctions, 533
  - NXCDefs.h, 1807
- RemoteStartProgram
  - CommModuleDCFunctions, 533
  - NXCDefs.h, 1808
- RemoteStopProgram
  - CommModuleDCFunctions, 534
  - NXCDefs.h, 1808
- RemoteStopSound
  - CommModuleDCFunctions, 534
  - NXCDefs.h, 1809
- RemoteWrite
  - CommModuleSCFunctions, 550
  - NXCDefs.h, 1809
- Remove
  - MessageReadType, 1092
- remove
  - cstdioAPI, 642
  - NXCDefs.h, 1810
- rename
  - cstdioAPI, 642
  - NXCDefs.h, 1810
- RenameFile
  - LoaderModuleFunctions, 584
  - NXCDefs.h, 1811
- Request
  - ReadSemDataType, 1101
  - WriteSemDataType, 1113
- ReSeed
  - RandomExType, 1096
- Reset
  - ReadButtonType, 1098
- RESET\_ALL
  - NBCCCommon.h, 1338
  - TachoResetConstants, 778
- RESET\_BLOCK\_COUNT
  - NBCCCommon.h, 1338
  - TachoResetConstants, 778
- RESET\_BLOCKANDTACHO
  - NBCCCommon.h, 1338
  - TachoResetConstants, 778
- RESET\_COUNT
  - NBCCCommon.h, 1338
  - TachoResetConstants, 778
- RESET\_NONE
  - NBCCCommon.h, 1339
  - TachoResetConstants, 778
- RESET\_ROTATION\_COUNT
  - NBCCCommon.h, 1339
  - TachoResetConstants, 778
- ResetAllTachoCounts
  - NXCDefs.h, 1811
  - OutputModuleFunctions, 342
- ResetBlockTachoCount
  - NXCDefs.h, 1812
  - OutputModuleFunctions, 342
- ResetHTBarometricCalibration

- HiTechnicAPI, 107
- NXCDefs.h, 1812
- ResetMIXG1300L
  - MicroinfinitiyFunctions, 597
  - NXCDefs.h, 1812
- ResetRotationCount
  - NXCDefs.h, 1813
  - OutputModuleFunctions, 342
- ResetScreen
  - DisplayModuleFunctions, 365
  - NXCDefs.h, 1813
- ResetSensor
  - InputModuleFunctions, 296
  - NXCDefs.h, 1814
- ResetSensorHTAngle
  - HiTechnicAPI, 108
  - NXCDefs.h, 1814
- ResetSleepTimer
  - CommandModuleFunctions, 431
  - NXCDefs.h, 1814
- ResetTachoCount
  - NXCDefs.h, 1815
  - OutputModuleFunctions, 343
- ResizeFile
  - LoaderModuleFunctions, 584
  - NXCDefs.h, 1815
- ResolveHandle
  - LoaderModuleFunctions, 585
  - NXCDefs.h, 1816
- Result
  - ColorSensorReadType, 1019
  - CommBTCheckStatusType, 1020
  - CommBTConnectionType, 1021
  - CommBTOnOffType, 1022
  - CommBTWriteType, 1024
  - CommExecuteFunctionType, 1026
  - CommHSControlType, 1028
  - CommLSCheckStatusType, 1031
  - CommLSReadType, 1032
  - CommLSWriteExType, 1034
  - CommLSWriteType, 1035
  - ComputeCalibValueType, 1037
  - DatalogWriteType, 1039
  - DrawCircleType, 1044
  - DrawEllipseType, 1045
  - DrawFontType, 1047
  - DrawGraphicArrayType, 1048
  - DrawGraphicType, 1050
  - DrawLineType, 1051
  - DrawPointType, 1053
  - DrawPolygonType, 1054
  - DrawRectType, 1055
  - DrawTextType, 1057
  - FileCloseType, 1058
  - FileDeleteType, 1059
  - FileFindType, 1060
  - FileOpenType, 1061
  - FileReadWriteType, 1063
  - FileRenameType, 1064
  - FileResizeType, 1065
  - FileResolveHandleType, 1067
  - FileSeekType, 1068
  - FileTellType, 1069
  - GetStartTickType, 1070
  - InputPinFunctionType, 1072
  - IOMapReadByIDType, 1075
  - IOMapReadType, 1077
  - IOMapWriteByIDType, 1078
  - IOMapWriteType, 1080
  - KeepAliveType, 1083
  - ListFilesType, 1085
  - LoaderExecuteFunctionType, 1088
  - MemoryManagerType, 1091
  - MessageReadType, 1092
  - MessageWriteType, 1093
  - RandomNumberType, 1097
  - ReadButtonType, 1098
  - ReadLastResponseType, 1100
  - SetScreenModeType, 1102
  - SetSleepTimeoutType, 1103
  - SoundPlayFileType, 1106
  - SoundPlayToneType, 1107
  - SoundSetStateType, 1108
  - UpdateCalibCacheInfoType, 1111
- ReturnLen
  - CommLSWriteExType, 1034
  - CommLSWriteType, 1035
- RetVal
  - CommExecuteFunctionType, 1026
- rewind
  - cstdioAPI, 642
  - NXCDefs.h, 1816

- RFID\_MODE\_CONTINUOUS
  - CTRFIDModeConstants, 978
  - NBCCCommon.h, 1339
- RFID\_MODE\_SINGLE
  - CTRFIDModeConstants, 978
  - NBCCCommon.h, 1339
- RFID\_MODE\_STOP
  - CTRFIDModeConstants, 979
  - NBCCCommon.h, 1339
- RFIDInit
  - CodatexAPI, 230
  - NXCDefs.h, 1816
- RFIDMode
  - CodatexAPI, 231
  - NXCDefs.h, 1817
- RFIDRead
  - CodatexAPI, 231
  - NXCDefs.h, 1817
- RFIDReadContinuous
  - CodatexAPI, 232
  - NXCDefs.h, 1818
- RFIDReadSingle
  - CodatexAPI, 232
  - NXCDefs.h, 1818
- RFIDStatus
  - CodatexAPI, 232
  - NXCDefs.h, 1819
- RFIDStop
  - CodatexAPI, 233
  - NXCDefs.h, 1819
- RIC Macro Wrappers, 247
- RICArg
  - NBCCCommon.h, 1339
  - RICMacros, 250
- RICImgPoint
  - NBCCCommon.h, 1340
  - RICMacros, 250
- RICImgRect
  - NBCCCommon.h, 1340
  - RICMacros, 250
- RICMacros
  - RICArg, 250
  - RICImgPoint, 250
  - RICImgRect, 250
  - RICMapArg, 251
  - RICMapElement, 251
  - RICMapFunction, 251
  - RICOpCircle, 252
  - RICOpCopyBits, 252
  - RICOpDescription, 252
  - RICOpEllipse, 253
  - RICOpLine, 253
  - RICOpNumBox, 253
  - RICOpPixel, 254
  - RICOpPolygon, 254
  - RICOpRect, 254
  - RICOpSprite, 255
  - RICOpVarMap, 255
  - RICPolygonPoints, 256
  - RICSetValue, 256
  - RICSpriteData, 256
- RICMapArg
  - NBCCCommon.h, 1340
  - RICMacros, 251
- RICMapElement
  - NBCCCommon.h, 1341
  - RICMacros, 251
- RICMapFunction
  - NBCCCommon.h, 1341
  - RICMacros, 251
- RICOpCircle
  - NBCCCommon.h, 1341
  - RICMacros, 252
- RICOpCopyBits
  - NBCCCommon.h, 1342
  - RICMacros, 252
- RICOpDescription
  - NBCCCommon.h, 1342
  - RICMacros, 252
- RICOpEllipse
  - NBCCCommon.h, 1342
  - RICMacros, 253
- RICOpLine
  - NBCCCommon.h, 1343
  - RICMacros, 253
- RICOpNumBox
  - NBCCCommon.h, 1343
  - RICMacros, 253
- RICOpPixel
  - NBCCCommon.h, 1343
  - RICMacros, 254
- RICOpPolygon

- NBCCCommon.h, [1344](#)
- RICMacros, [254](#)
- RICOpRect
  - NBCCCommon.h, [1344](#)
  - RICMacros, [254](#)
- RICOpSprite
  - NBCCCommon.h, [1344](#)
  - RICMacros, [255](#)
- RICOpVarMap
  - NBCCCommon.h, [1345](#)
  - RICMacros, [255](#)
- RICPolygonPoints
  - NBCCCommon.h, [1345](#)
  - RICMacros, [256](#)
- RICSetValue
  - NXCDefs.h, [1490](#)
  - RICMacros, [256](#)
- RICSpriteData
  - NBCCCommon.h, [1346](#)
  - RICMacros, [256](#)
- RightMotor
  - JoystickMessageType, [1082](#)
- RightSpeed
  - JoystickMessageType, [1082](#)
- RightStr
  - cstringAPI, [665](#)
  - NXCDefs.h, [1819](#)
- ROTATE\_QUEUE
  - CommandVMState, [706](#)
  - NBCCCommon.h, [1346](#)
- RotateMotor
  - NXCDefs.h, [1820](#)
  - OutputModuleFunctions, [343](#)
- RotateMotorEx
  - NXCDefs.h, [1820](#)
  - OutputModuleFunctions, [343](#)
- RotateMotorExPID
  - NXCDefs.h, [1821](#)
  - OutputModuleFunctions, [344](#)
- RotateMotorPID
  - NXCDefs.h, [1822](#)
  - OutputModuleFunctions, [345](#)
- RotationCount
  - OutputStateType, [1095](#)
- RotationCountField
  - NBCCCommon.h, [1346](#)
- OutputFieldConstants, [788](#)
- RS485Control
  - CommModuleFunctions, [491](#)
  - NXCDefs.h, [1822](#)
- RS485DataAvailable
  - CommModuleFunctions, [491](#)
  - NXCDefs.h, [1823](#)
- RS485Disable
  - CommModuleFunctions, [492](#)
  - NXCDefs.h, [1823](#)
- RS485Enable
  - CommModuleFunctions, [492](#)
  - NXCDefs.h, [1823](#)
- RS485Initialize
  - CommModuleFunctions, [493](#)
  - NXCDefs.h, [1824](#)
- RS485Read
  - CommModuleFunctions, [493](#)
  - NXCDefs.h, [1824](#)
- RS485ReadEx
  - CommModuleFunctions, [493](#)
  - NXCDefs.h, [1825](#)
- RS485SendingData
  - CommModuleFunctions, [494](#)
  - NXCDefs.h, [1825](#)
- RS485Status
  - CommModuleFunctions, [494](#)
  - NXCDefs.h, [1825](#)
- RS485Uart
  - CommModuleFunctions, [495](#)
  - NXCDefs.h, [1826](#)
- RS485Write
  - CommModuleFunctions, [495](#)
  - NXCDefs.h, [1826](#)
- RunNRLinkMacro
  - MindSensorsAPI, [219](#)
  - NXCDefs.h, [1827](#)
- RunState
  - OutputStateType, [1095](#)
- RunStateField
  - NBCCCommon.h, [1346](#)
  - OutputFieldConstants, [788](#)
- S1
  - InPorts, [274](#)
  - NXCDefs.h, [1490](#)

- s16
  - NXCDefs.h, [1492](#)
  - TypeAliases, [273](#)
- S2
  - InPorts, [276](#)
  - NXCDefs.h, [1492](#)
- S3
  - InPorts, [276](#)
  - NXCDefs.h, [1492](#)
- s32
  - NXCDefs.h, [1493](#)
  - TypeAliases, [273](#)
- S4
  - InPorts, [276](#)
  - NXCDefs.h, [1493](#)
- s8
  - NXCDefs.h, [1493](#)
  - TypeAliases, [273](#)
- SAMPLERATE\_DEFAULT
  - NBCCCommon.h, [1347](#)
  - SoundMisc, [732](#)
- SAMPLERATE\_MAX
  - NBCCCommon.h, [1347](#)
  - SoundMisc, [732](#)
- SAMPLERATE\_MIN
  - NBCCCommon.h, [1347](#)
  - SoundMisc, [732](#)
- ScaledArray
  - ColorSensorReadType, [1019](#)
- ScaledValue
  - InputValueType, [1073](#)
- ScaledValueField
  - InputFieldConstants, [762](#)
  - NBCCCommon.h, [1347](#)
- SCHAR\_MAX
  - NBCCCommon.h, [1347](#)
  - NXTLimits, [1012](#)
- SCHAR\_MIN
  - NBCCCommon.h, [1347](#)
  - NXTLimits, [1012](#)
- Scout constants, [872](#)
- Scout light constants, [872](#)
- Scout light rule constants, [880](#)
- Scout mode constants, [877](#)
- Scout motion rule constants, [878](#)
- Scout sound constants, [873](#)
- Scout sound set constants, [876](#)
- Scout special effect constants, [882](#)
- Scout touch rule constants, [879](#)
- Scout transmit rule constants, [881](#)
- SCOUT\_FXR\_ALARM
  - NBCCCommon.h, [1347](#)
  - ScoutSpecialEffectConstants, [883](#)
- SCOUT\_FXR\_BUG
  - NBCCCommon.h, [1347](#)
  - ScoutSpecialEffectConstants, [883](#)
- SCOUT\_FXR\_NONE
  - NBCCCommon.h, [1347](#)
  - ScoutSpecialEffectConstants, [883](#)
- SCOUT\_FXR\_RANDOM
  - NBCCCommon.h, [1348](#)
  - ScoutSpecialEffectConstants, [883](#)
- SCOUT\_FXR\_SCIENCE
  - NBCCCommon.h, [1348](#)
  - ScoutSpecialEffectConstants, [883](#)
- SCOUT\_LIGHT\_OFF
  - NBCCCommon.h, [1348](#)
  - ScoutLightConstants, [873](#)
- SCOUT\_LIGHT\_ON
  - NBCCCommon.h, [1348](#)
  - ScoutLightConstants, [873](#)
- SCOUT\_LR\_AVOID
  - NBCCCommon.h, [1348](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_LR\_IGNORE
  - NBCCCommon.h, [1348](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_LR\_OFF\_WHEN
  - NBCCCommon.h, [1348](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_LR\_SEEK\_DARK
  - NBCCCommon.h, [1348](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_LR\_SEEK\_LIGHT
  - NBCCCommon.h, [1349](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_LR\_WAIT\_FOR
  - NBCCCommon.h, [1349](#)
  - ScoutLightRuleConstants, [881](#)
- SCOUT\_MODE\_POWER
  - NBCCCommon.h, [1349](#)
  - ScoutModeConstants, [878](#)

- SCOUT\_MODE\_STANDALONE
  - NBCCCommon.h, 1349
  - ScoutModeConstants, 878
- SCOUT\_MR\_CIRCLE\_LEFT
  - NBCCCommon.h, 1349
  - ScoutMotionRuleConstants, 878
- SCOUT\_MR\_CIRCLE\_RIGHT
  - NBCCCommon.h, 1349
  - ScoutMotionRuleConstants, 878
- SCOUT\_MR\_FORWARD
  - NBCCCommon.h, 1349
  - ScoutMotionRuleConstants, 878
- SCOUT\_MR\_LOOP\_A
  - NBCCCommon.h, 1349
  - ScoutMotionRuleConstants, 879
- SCOUT\_MR\_LOOP\_AB
  - NBCCCommon.h, 1350
  - ScoutMotionRuleConstants, 879
- SCOUT\_MR\_LOOP\_B
  - NBCCCommon.h, 1350
  - ScoutMotionRuleConstants, 879
- SCOUT\_MR\_NO\_MOTION
  - NBCCCommon.h, 1350
  - ScoutMotionRuleConstants, 879
- SCOUT\_MR\_ZIGZAG
  - NBCCCommon.h, 1350
  - ScoutMotionRuleConstants, 879
- SCOUT\_SNDSET\_ALARM
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SNDSET\_BASIC
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SNDSET\_BUG
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SNDSET\_NONE
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SNDSET\_RANDOM
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SNDSET\_SCIENCE
  - NBCCCommon.h, 1350
  - ScoutSndSetConstants, 877
- SCOUT\_SOUND\_1\_BLINK
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_2\_BLINK
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_COUNTER1
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_COUNTER2
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_ENTER\_BRIGHT
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_ENTER\_DARK
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_ENTER\_NORMAL
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 874
- SCOUT\_SOUND\_ENTERSA
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_KEYERROR
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_MAIL\_RECEIVED
  - NBCCCommon.h, 1351
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_NONE
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_REMOTE
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_SPECIAL1
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_SPECIAL2
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_SPECIAL3
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_TIMER1
  - NBCCCommon.h, 1352

- ScoutSoundConstants, 875
- SCOUT\_SOUND\_TIMER2
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 875
- SCOUT\_SOUND\_TIMER3
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 876
- SCOUT\_SOUND\_TOUCH1\_PRES
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 876
- SCOUT\_SOUND\_TOUCH1\_REL
  - NBCCCommon.h, 1352
  - ScoutSoundConstants, 876
- SCOUT\_SOUND\_TOUCH2\_PRES
  - NBCCCommon.h, 1353
  - ScoutSoundConstants, 876
- SCOUT\_SOUND\_TOUCH2\_REL
  - NBCCCommon.h, 1353
  - ScoutSoundConstants, 876
- SCOUT\_TGS\_LONG
  - NBCCCommon.h, 1353
  - ScoutTransmitRuleConstants, 882
- SCOUT\_TGS\_MEDIUM
  - NBCCCommon.h, 1353
  - ScoutTransmitRuleConstants, 882
- SCOUT\_TGS\_SHORT
  - NBCCCommon.h, 1353
  - ScoutTransmitRuleConstants, 882
- SCOUT\_TR\_AVOID
  - NBCCCommon.h, 1353
  - ScoutTouchRuleConstants, 880
- SCOUT\_TR\_IGNORE
  - NBCCCommon.h, 1353
  - ScoutTouchRuleConstants, 880
- SCOUT\_TR\_OFF\_WHEN
  - NBCCCommon.h, 1353
  - ScoutTouchRuleConstants, 880
- SCOUT\_TR\_REVERSE
  - NBCCCommon.h, 1353
  - ScoutTouchRuleConstants, 880
- SCOUT\_TR\_WAIT\_FOR
  - NBCCCommon.h, 1354
  - ScoutTouchRuleConstants, 880
- ScoutLightConstants
  - SCOUT\_LIGHT\_OFF, 873
  - SCOUT\_LIGHT\_ON, 873
- ScoutLightRuleConstants
  - SCOUT\_LR\_AVOID, 881
  - SCOUT\_LR\_IGNORE, 881
  - SCOUT\_LR\_OFF\_WHEN, 881
  - SCOUT\_LR\_SEEK\_DARK, 881
  - SCOUT\_LR\_SEEK\_LIGHT, 881
  - SCOUT\_LR\_WAIT\_FOR, 881
- ScoutModeConstants
  - SCOUT\_MODE\_POWER, 878
  - SCOUT\_MODE\_STANDALONE, 878
- ScoutMotionRuleConstants
  - SCOUT\_MR\_CIRCLE\_LEFT, 878
  - SCOUT\_MR\_CIRCLE\_RIGHT, 878
  - SCOUT\_MR\_FORWARD, 878
  - SCOUT\_MR\_LOOP\_A, 879
  - SCOUT\_MR\_LOOP\_AB, 879
  - SCOUT\_MR\_LOOP\_B, 879
  - SCOUT\_MR\_NO\_MOTION, 879
  - SCOUT\_MR\_ZIGZAG, 879
- ScoutSndSetConstants
  - SCOUT\_SNDSET\_ALARM, 877
  - SCOUT\_SNDSET\_BASIC, 877
  - SCOUT\_SNDSET\_BUG, 877
  - SCOUT\_SNDSET\_NONE, 877
  - SCOUT\_SNDSET\_RANDOM, 877
  - SCOUT\_SNDSET\_SCIENCE, 877
- ScoutSoundConstants
  - SCOUT\_SOUND\_1\_BLINK, 874
  - SCOUT\_SOUND\_2\_BLINK, 874
  - SCOUT\_SOUND\_COUNTER1, 874
  - SCOUT\_SOUND\_COUNTER2, 874
  - SCOUT\_SOUND\_ENTER\_- BRIGHT, 874
  - SCOUT\_SOUND\_ENTER\_DARK, 874
  - SCOUT\_SOUND\_ENTER\_- NORMAL, 874
  - SCOUT\_SOUND\_ENTERSA, 875
  - SCOUT\_SOUND\_KEYERROR, 875
  - SCOUT\_SOUND\_MAIL\_- RECEIVED, 875

- SCOUT\_SOUND\_NONE, 875
- SCOUT\_SOUND\_REMOTE, 875
- SCOUT\_SOUND\_SPECIAL1, 875
- SCOUT\_SOUND\_SPECIAL2, 875
- SCOUT\_SOUND\_SPECIAL3, 875
- SCOUT\_SOUND\_TIMER1, 875
- SCOUT\_SOUND\_TIMER2, 875
- SCOUT\_SOUND\_TIMER3, 876
- SCOUT\_SOUND\_TOUCH1\_-  
PRES, 876
- SCOUT\_SOUND\_TOUCH1\_REL,  
876
- SCOUT\_SOUND\_TOUCH2\_-  
PRES, 876
- SCOUT\_SOUND\_TOUCH2\_REL,  
876
- ScoutSpecialEffectConstants
  - SCOUT\_FXR\_ALARM, 883
  - SCOUT\_FXR\_BUG, 883
  - SCOUT\_FXR\_NONE, 883
  - SCOUT\_FXR\_RANDOM, 883
  - SCOUT\_FXR\_SCIENCE, 883
- ScoutTouchRuleConstants
  - SCOUT\_TR\_AVOID, 880
  - SCOUT\_TR\_IGNORE, 880
  - SCOUT\_TR\_OFF\_WHEN, 880
  - SCOUT\_TR\_REVERSE, 880
  - SCOUT\_TR\_WAIT\_FOR, 880
- ScoutTransmitRuleConstants
  - SCOUT\_TGS\_LONG, 882
  - SCOUT\_TGS\_MEDIUM, 882
  - SCOUT\_TGS\_SHORT, 882
- SCREEN\_BACKGROUND
  - DisplayModuleConstants, 814
  - NBCCCommon.h, 1354
- SCREEN\_LARGE
  - DisplayModuleConstants, 814
  - NBCCCommon.h, 1354
- SCREEN\_MODE\_CLEAR
  - DisplayModuleConstants, 814
  - NBCCCommon.h, 1354
- SCREEN\_MODE\_RESTORE
  - DisplayModuleConstants, 814
  - NBCCCommon.h, 1354
- SCREEN\_SMALL
  - DisplayModuleConstants, 815
- NBCCCommon.h, 1354
- ScreenMode
  - SetScreenModeType, 1102
- SCREENS
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1354
- SEC\_1
  - NBCCCommon.h, 1355
  - TimeConstants, 700
- SEC\_10
  - NBCCCommon.h, 1355
  - TimeConstants, 700
- SEC\_15
  - NBCCCommon.h, 1355
  - TimeConstants, 701
- SEC\_2
  - NBCCCommon.h, 1355
  - TimeConstants, 701
- SEC\_20
  - NBCCCommon.h, 1355
  - TimeConstants, 701
- SEC\_3
  - NBCCCommon.h, 1356
  - TimeConstants, 701
- SEC\_30
  - NBCCCommon.h, 1356
  - TimeConstants, 701
- SEC\_4
  - NBCCCommon.h, 1356
  - TimeConstants, 702
- SEC\_5
  - NBCCCommon.h, 1356
  - TimeConstants, 702
- SEC\_6
  - NBCCCommon.h, 1356
  - TimeConstants, 702
- SEC\_7
  - NBCCCommon.h, 1357
  - TimeConstants, 702
- SEC\_8
  - NBCCCommon.h, 1357
  - TimeConstants, 702
- SEC\_9
  - NBCCCommon.h, 1357
  - TimeConstants, 703
- Seed

- RandomExType, 1096
- SEEK\_CUR
  - fseekConstants, 644
  - NXCDefs.h, 1493
- SEEK\_END
  - fseekConstants, 644
  - NXCDefs.h, 1493
- SEEK\_SET
  - fseekConstants, 644
  - NXCDefs.h, 1493
- SemData
  - ReadSemDataType, 1101
  - WriteSemDataType, 1113
- SendingData
  - CommHSCheckStatusType, 1027
- SendMessage
  - CommModuleFunctions, 496
  - NXCDefs.h, 1827
- SendRemoteBool
  - CommModuleFunctions, 496
  - NXCDefs.h, 1828
- SendRemoteNumber
  - CommModuleFunctions, 497
  - NXCDefs.h, 1828
- SendRemoteString
  - CommModuleFunctions, 497
  - NXCDefs.h, 1829
- SendResponseBool
  - CommModuleFunctions, 498
  - NXCDefs.h, 1829
- SendResponseNumber
  - CommModuleFunctions, 498
  - NXCDefs.h, 1830
- SendResponseString
  - CommModuleFunctions, 499
  - NXCDefs.h, 1830
- SendRS485Bool
  - CommModuleFunctions, 499
  - NXCDefs.h, 1831
- SendRS485Number
  - CommModuleFunctions, 499
  - NXCDefs.h, 1831
- SendRS485String
  - CommModuleFunctions, 500
  - NXCDefs.h, 1831
- Sensor
  - InputModuleFunctions, 296
  - NXCDefs.h, 1832
  - Sensor mode constants, 279
  - Sensor type constants, 276
  - Sensor types and modes, 46
  - SENSOR\_1
    - BasicSensorValues, 312
    - NXCDefs.h, 1493
  - SENSOR\_2
    - BasicSensorValues, 312
    - NXCDefs.h, 1493
  - SENSOR\_3
    - BasicSensorValues, 312
    - NXCDefs.h, 1494
  - SENSOR\_4
    - BasicSensorValues, 312
    - NXCDefs.h, 1494
  - SENSOR\_CELSIUS
    - NXCDefs.h, 1494
    - SensorTypeModes, 282
  - SENSOR\_COLORBLUE
    - NXCDefs.h, 1494
    - SensorTypeModes, 282
  - SENSOR\_COLORFULL
    - NXCDefs.h, 1494
    - SensorTypeModes, 282
  - SENSOR\_COLORGREEN
    - NXCDefs.h, 1494
    - SensorTypeModes, 283
  - SENSOR\_COLORNONE
    - NXCDefs.h, 1494
    - SensorTypeModes, 283
  - SENSOR\_COLORRED
    - NXCDefs.h, 1494
    - SensorTypeModes, 283
  - SENSOR\_EDGE
    - NXCDefs.h, 1495
    - SensorTypeModes, 283
  - SENSOR\_FAHRENHEIT
    - NXCDefs.h, 1495
    - SensorTypeModes, 283
  - SENSOR\_LIGHT
    - NXCDefs.h, 1495
    - SensorTypeModes, 283
  - SENSOR\_LOWSPEED
    - NXCDefs.h, 1495

- SensorTypeModes, [283](#)
- SENSOR\_LOWSPEED\_9V
  - NXCDefs.h, [1495](#)
  - SensorTypeModes, [283](#)
- SENSOR\_MODE\_BOOL
  - NXCDefs.h, [1495](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_CELSIUS
  - NXCDefs.h, [1495](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_EDGE
  - NXCDefs.h, [1495](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_FAHRENHEIT
  - NXCDefs.h, [1496](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_PERCENT
  - NXCDefs.h, [1496](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_PULSE
  - NXCDefs.h, [1496](#)
  - SensorModes, [280](#)
- SENSOR\_MODE\_RAW
  - NXCDefs.h, [1496](#)
  - SensorModes, [281](#)
- SENSOR\_MODE\_ROTATION
  - NXCDefs.h, [1496](#)
  - SensorModes, [281](#)
- SENSOR\_NXTLIGHT
  - NXCDefs.h, [1496](#)
  - SensorTypeModes, [284](#)
- SENSOR\_PULSE
  - NXCDefs.h, [1496](#)
  - SensorTypeModes, [284](#)
- SENSOR\_ROTATION
  - NXCDefs.h, [1496](#)
  - SensorTypeModes, [284](#)
- SENSOR\_SOUND
  - NXCDefs.h, [1497](#)
  - SensorTypeModes, [284](#)
- SENSOR\_TOUCH
  - NXCDefs.h, [1497](#)
  - SensorTypeModes, [284](#)
- SENSOR\_TYPE\_COLORBLUE
  - NXCDefs.h, [1497](#)
  - SensorTypes, [277](#)
- SENSOR\_TYPE\_COLORFULL
  - NXCDefs.h, [1497](#)
  - SensorTypes, [277](#)
- SENSOR\_TYPE\_COLORGREEN
  - NXCDefs.h, [1497](#)
  - SensorTypes, [277](#)
- SENSOR\_TYPE\_COLORNONE
  - NXCDefs.h, [1497](#)
  - SensorTypes, [277](#)
- SENSOR\_TYPE\_COLORRED
  - NXCDefs.h, [1497](#)
  - SensorTypes, [277](#)
- SENSOR\_TYPE\_CUSTOM
  - NXCDefs.h, [1497](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_HIGHSPEED
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_LIGHT
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_LIGHT\_ACTIVE
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_LIGHT\_INACTIVE
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_LOWSPEED
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_LOWSPEED\_9V
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_NONE
  - NXCDefs.h, [1498](#)
  - SensorTypes, [278](#)
- SENSOR\_TYPE\_ROTATION
  - NXCDefs.h, [1498](#)
  - SensorTypes, [279](#)
- SENSOR\_TYPE\_SOUND\_DB
  - NXCDefs.h, [1499](#)
  - SensorTypes, [279](#)
- SENSOR\_TYPE\_SOUND\_DBA
  - NXCDefs.h, [1499](#)
  - SensorTypes, [279](#)
- SENSOR\_TYPE\_TEMPERATURE

- NXCDefs.h, 1499
- SensorTypes, 279
- SENSOR\_TYPE\_TOUCH
  - NXCDefs.h, 1499
  - SensorTypes, 279
- SensorBoolean
  - InputModuleFunctions, 297
  - NXCDefs.h, 1832
- SensorDIAccelStatus
  - DexterIndustriesAPI, 239
  - NXCDefs.h, 1833
- SensorDigiPinsDirection
  - InputModuleFunctions, 297
  - NXCDefs.h, 1833
- SensorDigiPinsOutputLevel
  - InputModuleFunctions, 297
  - NXCDefs.h, 1834
- SensorDigiPinsStatus
  - InputModuleFunctions, 298
  - NXCDefs.h, 1834
- SensorDIGPSDistanceToWaypoint
  - DexterIndustriesAPI, 239
  - NXCDefs.h, 1834
- SensorDIGPSHeading
  - DexterIndustriesAPI, 240
  - NXCDefs.h, 1835
- SensorDIGPSHeadingToWaypoint
  - DexterIndustriesAPI, 240
  - NXCDefs.h, 1835
- SensorDIGPSLatitude
  - DexterIndustriesAPI, 241
  - NXCDefs.h, 1836
- SensorDIGPSLongitude
  - DexterIndustriesAPI, 241
  - NXCDefs.h, 1836
- SensorDIGPSRelativeHeading
  - DexterIndustriesAPI, 241
  - NXCDefs.h, 1837
- SensorDIGPSStatus
  - DexterIndustriesAPI, 242
  - NXCDefs.h, 1837
- SensorDIGPSTime
  - DexterIndustriesAPI, 242
  - NXCDefs.h, 1837
- SensorDIGPSVelocity
  - DexterIndustriesAPI, 243
  - NXCDefs.h, 1838
- SensorDIGyroStatus
  - DexterIndustriesAPI, 243
  - NXCDefs.h, 1838
- SensorDIGyroTemperature
  - DexterIndustriesAPI, 243
  - NXCDefs.h, 1839
- SensorHTColorNum
  - HiTechnicAPI, 108
  - NXCDefs.h, 1839
- SensorHTCompass
  - HiTechnicAPI, 109
  - NXCDefs.h, 1839
- SensorHTEOPD
  - HiTechnicAPI, 109
  - NXCDefs.h, 1840
- SensorHTGyro
  - HiTechnicAPI, 109
  - NXCDefs.h, 1840
- SensorHTIRSeeker2ACDir
  - HiTechnicAPI, 110
  - NXCDefs.h, 1841
- SensorHTIRSeeker2Addr
  - HiTechnicAPI, 110
  - NXCDefs.h, 1841
- SensorHTIRSeeker2DCDir
  - HiTechnicAPI, 111
  - NXCDefs.h, 1842
- SensorHTIRSeekerDir
  - HiTechnicAPI, 111
  - NXCDefs.h, 1842
- SensorHTMagnet
  - HiTechnicAPI, 112
  - NXCDefs.h, 1842
- SensorHTProtoAnalog
  - HiTechnicAPI, 112
  - NXCDefs.h, 1843
- SensorHTProtoDigital
  - HiTechnicAPI, 113
  - NXCDefs.h, 1843
- SensorHTProtoDigitalControl
  - HiTechnicAPI, 113
  - NXCDefs.h, 1844
- SensorHTSuperProAnalog
  - HiTechnicAPI, 113
  - NXCDefs.h, 1844

- SensorHTSuperProDigital
  - HiTechnicAPI, 114
  - NXCDefs.h, 1845
- SensorHTSuperProDigitalControl
  - HiTechnicAPI, 114
  - NXCDefs.h, 1845
- SensorHTSuperProLED
  - HiTechnicAPI, 115
  - NXCDefs.h, 1846
- SensorHTSuperProProgramControl
  - HiTechnicAPI, 115
  - NXCDefs.h, 1846
- SensorHTSuperProStrobe
  - HiTechnicAPI, 116
  - NXCDefs.h, 1846
- SensorInvalid
  - InputModuleFunctions, 298
  - NXCDefs.h, 1847
- SensorMIXG1300LScale
  - MicroinfinityFunctions, 598
  - NXCDefs.h, 1847
- SensorMode
  - InputModuleFunctions, 299
  - InputValueType, 1074
  - NXCDefs.h, 1848
- SensorModes
  - SENSOR\_MODE\_BOOL, 280
  - SENSOR\_MODE\_CELSIUS, 280
  - SENSOR\_MODE\_EDGE, 280
  - SENSOR\_MODE\_FAHRENHEIT, 280
  - SENSOR\_MODE\_PERCENT, 280
  - SENSOR\_MODE\_PULSE, 280
  - SENSOR\_MODE\_RAW, 281
  - SENSOR\_MODE\_ROTATION, 281
- SensorMSCompass
  - MindSensorsAPI, 219
  - NXCDefs.h, 1848
- SensorMSDROD
  - MindSensorsAPI, 220
  - NXCDefs.h, 1848
- SensorMSPressure
  - MindSensorsAPI, 220
  - NXCDefs.h, 1849
- SensorMSPressureRaw
  - MindSensorsAPI, 220
  - NXCDefs.h, 1849
- SensorNormalized
  - InputModuleFunctions, 299
  - NXCDefs.h, 1850
- SensorNXTSumoEyes
  - MindSensorsAPI, 221
  - NXCDefs.h, 1850
- SensorNXTSumoEyesRaw
  - MindSensorsAPI, 221
  - NXCDefs.h, 1850
- SensorRaw
  - InputModuleFunctions, 299
  - NXCDefs.h, 1851
- SensorScaled
  - InputModuleFunctions, 300
  - NXCDefs.h, 1851
- SensorTemperature
  - LowSpeedModuleFunctions, 402
  - NXCDefs.h, 1852
- SensorType
  - InputModuleFunctions, 300
  - InputValueType, 1074
  - NXCDefs.h, 1852
- SensorTypeModes
  - \_SENSOR\_CFG, 282
  - SENSOR\_CELSIUS, 282
  - SENSOR\_COLORBLUE, 282
  - SENSOR\_COLORFULL, 282
  - SENSOR\_COLORGREEN, 283
  - SENSOR\_COLORNONE, 283
  - SENSOR\_COLORRED, 283
  - SENSOR\_EDGE, 283
  - SENSOR\_FAHRENHEIT, 283
  - SENSOR\_LIGHT, 283
  - SENSOR\_LOWSPEED, 283
  - SENSOR\_LOWSPEED\_9V, 283
  - SENSOR\_NXTLIGHT, 284
  - SENSOR\_PULSE, 284
  - SENSOR\_ROTATION, 284
  - SENSOR\_SOUND, 284
  - SENSOR\_TOUCH, 284
- SensorTypes
  - SENSOR\_TYPE\_COLORBLUE, 277
  - SENSOR\_TYPE\_COLORFULL, 277

- SENSOR\_TYPE\_COLORGREEN, 277
- SENSOR\_TYPE\_COLORNONE, 277
- SENSOR\_TYPE\_COLORRED, 277
- SENSOR\_TYPE\_CUSTOM, 278
- SENSOR\_TYPE\_HIGHSPEED, 278
- SENSOR\_TYPE\_LIGHT, 278
- SENSOR\_TYPE\_LIGHT\_ACTIVE, 278
- SENSOR\_TYPE\_LIGHT\_-  
INACTIVE, 278
- SENSOR\_TYPE\_LOWSPEED, 278
- SENSOR\_TYPE\_LOWSPEED\_9V, 278
- SENSOR\_TYPE\_NONE, 278
- SENSOR\_TYPE\_ROTATION, 279
- SENSOR\_TYPE\_SOUND\_DB, 279
- SENSOR\_TYPE\_SOUND\_DBA, 279
- SENSOR\_TYPE\_-  
TEMPERATURE, 279
- SENSOR\_TYPE\_TOUCH, 279
- SensorUS
  - LowSpeedModuleFunctions, 402
  - NXCDefs.h, 1853
- SensorValue
  - InputModuleFunctions, 301
  - NXCDefs.h, 1853
- SensorValueBool
  - InputModuleFunctions, 301
  - NXCDefs.h, 1853
- SensorValueRaw
  - InputModuleFunctions, 302
  - NXCDefs.h, 1854
- set\_fopen\_size
  - cstdioAPI, 643
  - NXCDefs.h, 1854
- SetAbortFlag
  - NXCDefs.h, 1854
  - UiModuleFunctions, 564
- SetACCLNxSensitivity
  - MindSensorsAPI, 222
  - NXCDefs.h, 1855
- SetBatteryState
  - NXCDefs.h, 1855
  - UiModuleFunctions, 565
- SetBluetoothState
  - NXCDefs.h, 1856
  - UiModuleFunctions, 565
- SetBTDataMode
  - CommModuleFunctions, 500
  - NXCDefs.h, 1856
- SetBTInputBuffer
  - CommModuleFunctions, 501
  - NXCDefs.h, 1856
- SetBTInputBufferInPtr
  - CommModuleFunctions, 501
  - NXCDefs.h, 1857
- SetBTInputBufferOutPtr
  - CommModuleFunctions, 502
  - NXCDefs.h, 1857
- SetBTOutputBuffer
  - CommModuleFunctions, 502
  - NXCDefs.h, 1857
- SetBTOutputBufferInPtr
  - CommModuleFunctions, 502
  - NXCDefs.h, 1858
- SetBTOutputBufferOutPtr
  - CommModuleFunctions, 503
  - NXCDefs.h, 1858
- SetButtonLongPressCount
  - ButtonModuleFunctions, 556
  - NXCDefs.h, 1858
- SetButtonLongReleaseCount
  - ButtonModuleFunctions, 556
  - NXCDefs.h, 1859
- SetButtonModuleValue
  - CommandModuleFunctions, 432
  - NXCDefs.h, 1859
- SetButtonPressCount
  - ButtonModuleFunctions, 557
  - NXCDefs.h, 1859
- SetButtonReleaseCount
  - ButtonModuleFunctions, 557
  - NXCDefs.h, 1860
- SetButtonShortReleaseCount
  - ButtonModuleFunctions, 557
  - NXCDefs.h, 1860
- SetButtonState
  - ButtonModuleFunctions, 558

- NXCDefs.h, 1860
- SetCommandFlags
  - NXCDefs.h, 1861
  - UiModuleFunctions, 565
- SetCommandModuleBytes
  - CommandModuleFunctions, 432
  - NXCDefs.h, 1861
- SetCommandModuleValue
  - CommandModuleFunctions, 432
  - NXCDefs.h, 1861
- SetCommModuleBytes
  - CommandModuleFunctions, 433
  - NXCDefs.h, 1862
- SetCommModuleValue
  - CommandModuleFunctions, 433
  - NXCDefs.h, 1862
- SetCustomSensorActiveStatus
  - InputModuleFunctions, 302
  - NXCDefs.h, 1863
- SetCustomSensorPercentFullScale
  - InputModuleFunctions, 302
  - NXCDefs.h, 1863
- SetCustomSensorZeroOffset
  - InputModuleFunctions, 303
  - NXCDefs.h, 1863
- SetDisplayContrast
  - DisplayModuleFunctions, 366
  - NXCDefs.h, 1864
- SetDisplayDisplay
  - DisplayModuleFunctions, 366
  - NXCDefs.h, 1864
- SetDisplayEraseMask
  - DisplayModuleFunctions, 366
  - NXCDefs.h, 1864
- SetDisplayFlags
  - DisplayModuleFunctions, 367
  - NXCDefs.h, 1865
- SetDisplayFont
  - DisplayModuleFunctions, 367
  - NXCDefs.h, 1865
- SetDisplayModuleBytes
  - CommandModuleFunctions, 433
  - NXCDefs.h, 1865
- SetDisplayModuleValue
  - CommandModuleFunctions, 434
  - NXCDefs.h, 1866
- SetDisplayNormal
  - DisplayModuleFunctions, 367
  - NXCDefs.h, 1866
- SetDisplayPopup
  - DisplayModuleFunctions, 368
  - NXCDefs.h, 1867
- SetDisplayTextLinesCenterFlags
  - DisplayModuleFunctions, 368
  - NXCDefs.h, 1867
- SetDisplayUpdateMask
  - DisplayModuleFunctions, 368
  - NXCDefs.h, 1867
- SetHSAddress
  - CommModuleFunctions, 503
  - NXCDefs.h, 1868
- SetHSDataMode
  - CommModuleFunctions, 503
  - NXCDefs.h, 1868
- SetHSFlags
  - CommModuleFunctions, 503
  - NXCDefs.h, 1868
- SetHSInputBuffer
  - CommModuleFunctions, 504
  - NXCDefs.h, 1869
- SetHSInputBufferInPtr
  - CommModuleFunctions, 504
  - NXCDefs.h, 1869
- SetHSInputBufferOutPtr
  - CommModuleFunctions, 504
  - NXCDefs.h, 1869
- SetHSMMode
  - CommModuleFunctions, 505
  - NXCDefs.h, 1870
- SetHSOutputBuffer
  - CommModuleFunctions, 505
  - NXCDefs.h, 1870
- SetHSOutputBufferInPtr
  - CommModuleFunctions, 506
  - NXCDefs.h, 1871
- SetHSOutputBufferOutPtr
  - CommModuleFunctions, 506
  - NXCDefs.h, 1871
- SetHSSpeed
  - CommModuleFunctions, 506
  - NXCDefs.h, 1871
- SetHSState

- CommModuleFunctions, [507](#)
  - NXCDefs.h, [1872](#)
- SetHTBarometricCalibration
  - HiTechnicAPI, [116](#)
  - NXCDefs.h, [1872](#)
- SetHTColor2Mode
  - HiTechnicAPI, [116](#)
  - NXCDefs.h, [1872](#)
- SetHTIRSeeker2Mode
  - HiTechnicAPI, [117](#)
  - NXCDefs.h, [1873](#)
- SetI2COptions
  - LowLevelLowSpeedModuleFunctions, [410](#)
  - NXCDefs.h, [1873](#)
- SetInput
  - InputModuleFunctions, [303](#)
  - NXCDefs.h, [1874](#)
- SetInputModuleValue
  - CommandModuleFunctions, [434](#)
  - NXCDefs.h, [1874](#)
- SetIOCtrlModuleValue
  - CommandModuleFunctions, [435](#)
  - NXCDefs.h, [1874](#)
- SetIOMapBytes
  - CommandModuleFunctions, [435](#)
  - NXCDefs.h, [1875](#)
- SetIOMapBytesByID
  - CommandModuleFunctions, [435](#)
  - NXCDefs.h, [1875](#)
- SetIOMapValue
  - CommandModuleFunctions, [436](#)
  - NXCDefs.h, [1876](#)
- SetIOMapValueByID
  - CommandModuleFunctions, [436](#)
  - NXCDefs.h, [1876](#)
- SetLoaderModuleValue
  - CommandModuleFunctions, [437](#)
  - NXCDefs.h, [1877](#)
- SetLongAbort
  - NXCDefs.h, [1877](#)
  - UiModuleFunctions, [566](#)
- SetLowSpeedModuleBytes
  - CommandModuleFunctions, [437](#)
  - NXCDefs.h, [1877](#)
- SetLowSpeedModuleValue
  - CommandModuleFunctions, [438](#)
  - NXCDefs.h, [1878](#)
- SetMotorPwnFreq
  - NXCDefs.h, [1878](#)
  - OutputModuleFunctions, [345](#)
- SetMotorRegulationOptions
  - NXCDefs.h, [1879](#)
  - OutputModuleFunctions, [346](#)
- SetMotorRegulationTime
  - NXCDefs.h, [1879](#)
  - OutputModuleFunctions, [346](#)
- SetNXTLineLeaderKdFactor
  - MindSensorsAPI, [222](#)
  - NXCDefs.h, [1879](#)
- SetNXTLineLeaderKdValue
  - MindSensorsAPI, [223](#)
  - NXCDefs.h, [1880](#)
- SetNXTLineLeaderKiFactor
  - MindSensorsAPI, [223](#)
  - NXCDefs.h, [1880](#)
- SetNXTLineLeaderKiValue
  - MindSensorsAPI, [224](#)
  - NXCDefs.h, [1881](#)
- SetNXTLineLeaderKpFactor
  - MindSensorsAPI, [224](#)
  - NXCDefs.h, [1881](#)
- SetNXTLineLeaderKpValue
  - MindSensorsAPI, [225](#)
  - NXCDefs.h, [1882](#)
- SetNXTLineLeaderSetpoint
  - MindSensorsAPI, [226](#)
  - NXCDefs.h, [1883](#)
- SetNXTServoPosition
  - MindSensorsAPI, [226](#)
  - NXCDefs.h, [1883](#)
- SetNXTServoQuickPosition
  - MindSensorsAPI, [227](#)
  - NXCDefs.h, [1884](#)
- SetNXTServoSpeed
  - MindSensorsAPI, [227](#)
  - NXCDefs.h, [1884](#)
- SetOnBrickProgramPointer
  - NXCDefs.h, [1885](#)
  - UiModuleFunctions, [566](#)
- SetOutput
  - NXCDefs.h, [1885](#)

- OutputModuleFunctions, [346](#)
- SetOutputModuleValue
  - CommandModuleFunctions, [438](#)
  - NXCDefs.h, [1886](#)
- SetScreenMode
  - NBCCCommon.h, [1357](#)
  - SysCallConstants, [690](#)
- SetScreenModeType, [1101](#)
  - Result, [1102](#)
  - ScreenMode, [1102](#)
- SetSensor
  - InputModuleFunctions, [304](#)
  - NXCDefs.h, [1886](#)
- SetSensorBoolean
  - InputModuleFunctions, [304](#)
  - NXCDefs.h, [1887](#)
- SetSensorColorBlue
  - InputModuleFunctions, [304](#)
  - NXCDefs.h, [1887](#)
- SetSensorColorFull
  - InputModuleFunctions, [305](#)
  - NXCDefs.h, [1887](#)
- SetSensorColorGreen
  - InputModuleFunctions, [305](#)
  - NXCDefs.h, [1888](#)
- SetSensorColorNone
  - InputModuleFunctions, [305](#)
  - NXCDefs.h, [1888](#)
- SetSensorColorRed
  - InputModuleFunctions, [306](#)
  - NXCDefs.h, [1888](#)
- SetSensorDIAccl
  - DexterIndustriesAPI, [244](#)
  - NXCDefs.h, [1889](#)
- SetSensorDIAcclDrift
  - DexterIndustriesAPI, [244](#)
  - NXCDefs.h, [1889](#)
- SetSensorDIAcclEx
  - DexterIndustriesAPI, [245](#)
  - NXCDefs.h, [1890](#)
- SetSensorDigiPinsDirection
  - InputModuleFunctions, [306](#)
  - NXCDefs.h, [1890](#)
- SetSensorDigiPinsOutputLevel
  - InputModuleFunctions, [307](#)
  - NXCDefs.h, [1891](#)
- SetSensorDigiPinsStatus
  - InputModuleFunctions, [307](#)
  - NXCDefs.h, [1891](#)
- SetSensorDIGPSWaypoint
  - DexterIndustriesAPI, [245](#)
  - NXCDefs.h, [1891](#)
- SetSensorDIGyro
  - DexterIndustriesAPI, [246](#)
  - NXCDefs.h, [1892](#)
- SetSensorDIGyroEx
  - DexterIndustriesAPI, [246](#)
  - NXCDefs.h, [1892](#)
- SetSensorEMeter
  - InputModuleFunctions, [307](#)
  - NXCDefs.h, [1893](#)
- SetSensorHTEOPD
  - HiTechnicAPI, [117](#)
  - NXCDefs.h, [1893](#)
- SetSensorHTGyro
  - HiTechnicAPI, [118](#)
  - NXCDefs.h, [1894](#)
- SetSensorHTMagnet
  - HiTechnicAPI, [118](#)
  - NXCDefs.h, [1894](#)
- SetSensorHTProtoDigital
  - HiTechnicAPI, [118](#)
  - NXCDefs.h, [1894](#)
- SetSensorHTProtoDigitalControl
  - HiTechnicAPI, [119](#)
  - NXCDefs.h, [1895](#)
- SetSensorHTSuperProAnalogOut
  - HiTechnicAPI, [119](#)
  - NXCDefs.h, [1895](#)
- SetSensorHTSuperProDigital
  - HiTechnicAPI, [120](#)
  - NXCDefs.h, [1896](#)
- SetSensorHTSuperProDigitalControl
  - HiTechnicAPI, [120](#)
  - NXCDefs.h, [1896](#)
- SetSensorHTSuperProLED
  - HiTechnicAPI, [121](#)
  - NXCDefs.h, [1897](#)
- SetSensorHTSuperProProgramControl
  - HiTechnicAPI, [121](#)
  - NXCDefs.h, [1897](#)
- SetSensorHTSuperProStrobe

- HiTechnicAPI, [121](#)
- NXCDefs.h, [1897](#)
- SetSensorLight
  - InputModuleFunctions, [308](#)
  - NXCDefs.h, [1898](#)
- SetSensorLowspeed
  - InputModuleFunctions, [308](#)
  - NXCDefs.h, [1898](#)
- SetSensorMIXG1300LScale
  - Micr infinityFunctions, [599](#)
  - NXCDefs.h, [1899](#)
- SetSensorMode
  - InputModuleFunctions, [309](#)
  - NXCDefs.h, [1899](#)
- SetSensorMSDROD
  - MindSensorsAPI, [228](#)
  - NXCDefs.h, [1900](#)
- SetSensorMSPressure
  - MindSensorsAPI, [228](#)
  - NXCDefs.h, [1900](#)
- SetSensorNXTSumoEyes
  - MindSensorsAPI, [228](#)
  - NXCDefs.h, [1900](#)
- SetSensorSound
  - InputModuleFunctions, [309](#)
  - NXCDefs.h, [1901](#)
- SetSensorTemperature
  - InputModuleFunctions, [309](#)
  - NXCDefs.h, [1901](#)
- SetSensorTouch
  - InputModuleFunctions, [310](#)
  - NXCDefs.h, [1902](#)
- SetSensorType
  - InputModuleFunctions, [310](#)
  - NXCDefs.h, [1902](#)
- SetSensorUltrasonic
  - InputModuleFunctions, [311](#)
  - NXCDefs.h, [1902](#)
- SetSleepTime
  - NXCDefs.h, [1903](#)
  - UiModuleFunctions, [566](#)
- SetSleepTimeout
  - NXCDefs.h, [1903](#)
  - UiModuleFunctions, [567](#)
- SetSleepTimeoutType, [1102](#)
  - Result, [1103](#)
- TheSleepTimeoutMS, [1103](#)
- SetSleepTimeoutVal
  - NBCCommon.h, [1357](#)
  - SysCallConstants, [690](#)
- SetSleepTimer
  - NXCDefs.h, [1903](#)
  - UiModuleFunctions, [567](#)
- SetSoundDuration
  - NXCDefs.h, [1904](#)
  - SoundModuleFunctions, [379](#)
- SetSoundFlags
  - NXCDefs.h, [1904](#)
  - SoundModuleFunctions, [380](#)
- SetSoundFrequency
  - NXCDefs.h, [1904](#)
  - SoundModuleFunctions, [380](#)
- SetSoundMode
  - NXCDefs.h, [1905](#)
  - SoundModuleFunctions, [381](#)
- SetSoundModuleState
  - NXCDefs.h, [1905](#)
  - SoundModuleFunctions, [381](#)
- SetSoundModuleValue
  - CommandModuleFunctions, [438](#)
  - NXCDefs.h, [1906](#)
- SetSoundSampleRate
  - NXCDefs.h, [1906](#)
  - SoundModuleFunctions, [381](#)
- SetSoundVolume
  - NXCDefs.h, [1906](#)
  - SoundModuleFunctions, [382](#)
- SetUIButton
  - NXCDefs.h, [1907](#)
  - UiModuleFunctions, [567](#)
- SetUIModuleValue
  - CommandModuleFunctions, [439](#)
  - NXCDefs.h, [1907](#)
- SetUIState
  - NXCDefs.h, [1907](#)
  - UiModuleFunctions, [568](#)
- SetUSBInputBuffer
  - CommModuleFunctions, [507](#)
  - NXCDefs.h, [1908](#)
- SetUSBInputBufferInPtr
  - CommModuleFunctions, [507](#)
  - NXCDefs.h, [1908](#)

- SetUSBInputBufferOutPtr
  - CommModuleFunctions, [507](#)
  - NXCDefs.h, [1908](#)
- SetUSBOutputBuffer
  - CommModuleFunctions, [508](#)
  - NXCDefs.h, [1909](#)
- SetUSBOutputBufferInPtr
  - CommModuleFunctions, [508](#)
  - NXCDefs.h, [1909](#)
- SetUSBOutputBufferOutPtr
  - CommModuleFunctions, [508](#)
  - NXCDefs.h, [1909](#)
- SetUSBPollBuffer
  - CommModuleFunctions, [509](#)
  - NXCDefs.h, [1910](#)
- SetUSBPollBufferInPtr
  - CommModuleFunctions, [509](#)
  - NXCDefs.h, [1910](#)
- SetUSBPollBufferOutPtr
  - CommModuleFunctions, [509](#)
  - NXCDefs.h, [1910](#)
- SetUSBState
  - CommModuleFunctions, [510](#)
  - NXCDefs.h, [1911](#)
- SetVMRunState
  - NXCDefs.h, [1911](#)
  - UiModuleFunctions, [568](#)
- SetVolume
  - NXCDefs.h, [1911](#)
  - UiModuleFunctions, [568](#)
- SHRT\_MAX
  - NBCCCommon.h, [1357](#)
  - NXTLimits, [1012](#)
- SHRT\_MIN
  - NBCCCommon.h, [1357](#)
  - NXTLimits, [1012](#)
- sign
  - cmathAPI, [627](#)
  - NXCDefs.h, [1912](#)
- Sin
  - cmathAPI, [612](#)
  - NXCDefs.h, [1499](#)
- sin
  - cmathAPI, [628](#)
  - NXCDefs.h, [1912](#)
- SinD
  - cmathAPI, [613](#)
  - NXCDefs.h, [1500](#)
- sind
  - cmathAPI, [628](#)
  - NXCDefs.h, [1912](#)
- Sinh
  - cmathAPI, [613](#)
  - NXCDefs.h, [1500](#)
- sinh
  - cmathAPI, [629](#)
  - NXCDefs.h, [1913](#)
- SinhD
  - cmathAPI, [614](#)
  - NXCDefs.h, [1500](#)
- sinhd
  - cmathAPI, [629](#)
  - NXCDefs.h, [1913](#)
- Size
  - DrawCircleType, [1044](#)
  - DrawRectType, [1055](#)
- SIZE\_OF\_BDADDR
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BRICK\_NAME
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BT\_CONNECT\_TABLE
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BT\_DEVICE\_TABLE
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BT\_NAME
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BT\_PINCODE
  - CommMiscConstants, [832](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_BTBUF
  - CommMiscConstants, [833](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_CLASS\_OF\_DEVICE
  - CommMiscConstants, [833](#)
  - NBCCCommon.h, [1358](#)
- SIZE\_OF\_HSBUF
  - CommMiscConstants, [833](#)

- NBCCCommon.h, 1358
- SIZE\_OF\_USBBUF
  - CommMiscConstants, 833
  - NBCCCommon.h, 1358
- SIZE\_OF\_USBDATA
  - CommMiscConstants, 833
  - NBCCCommon.h, 1359
- SizeOf
  - LoaderModuleFunctions, 585
  - NXCDefs.h, 1914
- SizeType, 1103
  - Height, 1103
  - Width, 1104
- SizeX
  - DrawEllipseType, 1045
- SizeY
  - DrawEllipseType, 1045
- SleepNow
  - IOCtrlModuleFunctions, 460
  - NXCDefs.h, 1914
- SleepTime
  - NXCDefs.h, 1914
  - UiModuleFunctions, 569
- SleepTimeout
  - NXCDefs.h, 1915
  - UiModuleFunctions, 569
- SleepTimer
  - NXCDefs.h, 1915
  - UiModuleFunctions, 569
- Sound module, 55
- Sound module constants, 726
- Sound module functions, 375
- Sound module IOMAP offsets, 730
- Sound module miscellaneous constants, 731
- Sound module types, 374
- SOUND\_CLICK
  - NBCCCommon.h, 1359
  - RCXSoundConstants, 871
- SOUND\_DOUBLE\_BEEP
  - NBCCCommon.h, 1359
  - RCXSoundConstants, 871
- SOUND\_DOWN
  - NBCCCommon.h, 1359
  - RCXSoundConstants, 871
- SOUND\_FAST\_UP
  - NBCCCommon.h, 1359
  - RCXSoundConstants, 871
- SOUND\_FLAGS\_IDLE
  - NBCCCommon.h, 1359
  - SoundFlagsConstants, 728
- SOUND\_FLAGS\_RUNNING
  - NBCCCommon.h, 1359
  - SoundFlagsConstants, 728
- SOUND\_FLAGS\_UPDATE
  - NBCCCommon.h, 1359
  - SoundFlagsConstants, 728
- SOUND\_LOW\_BEEP
  - NBCCCommon.h, 1360
  - RCXSoundConstants, 871
- SOUND\_MODE\_LOOP
  - NBCCCommon.h, 1360
  - SoundModeConstants, 729
- SOUND\_MODE\_ONCE
  - NBCCCommon.h, 1360
  - SoundModeConstants, 729
- SOUND\_MODE\_TONE
  - NBCCCommon.h, 1360
  - SoundModeConstants, 730
- SOUND\_STATE\_FILE
  - NBCCCommon.h, 1360
  - SoundStateConstants, 728
- SOUND\_STATE\_IDLE
  - NBCCCommon.h, 1360
  - SoundStateConstants, 728
- SOUND\_STATE\_STOP
  - NBCCCommon.h, 1360
  - SoundStateConstants, 729
- SOUND\_STATE\_TONE
  - NBCCCommon.h, 1361
  - SoundStateConstants, 729
- SOUND\_UP
  - NBCCCommon.h, 1361
  - RCXSoundConstants, 871
- SoundDuration
  - NXCDefs.h, 1915
  - SoundModuleFunctions, 382
- SoundFlags
  - NXCDefs.h, 1916
  - SoundModuleFunctions, 383
- SoundFlags constants, 727
- SoundFlagsConstants

- SOUND\_FLAGS\_IDLE, 728
  - SOUND\_FLAGS\_RUNNING, 728
  - SOUND\_FLAGS\_UPDATE, 728
  - SoundFrequency
    - NXCDefs.h, 1916
    - SoundModuleFunctions, 383
  - SoundGetState
    - NBCCCommon.h, 1361
    - SysCallConstants, 690
  - SoundGetStateType, 1104
    - Flags, 1104
    - State, 1104
  - SoundIOMAP
    - SoundOffsetDuration, 730
    - SoundOffsetFlags, 730
    - SoundOffsetFreq, 730
    - SoundOffsetMode, 731
    - SoundOffsetSampleRate, 731
    - SoundOffsetSoundFilename, 731
    - SoundOffsetState, 731
    - SoundOffsetVolume, 731
  - SoundLevel
    - SoundPlayFileType, 1106
    - SoundPlayToneType, 1107
  - SoundMisc
    - FREQUENCY\_MAX, 732
    - FREQUENCY\_MIN, 732
    - SAMPLERATE\_DEFAULT, 732
    - SAMPLERATE\_MAX, 732
    - SAMPLERATE\_MIN, 732
  - SoundMode
    - NXCDefs.h, 1917
    - SoundModuleFunctions, 383
  - SoundMode constants, 729
  - SoundModeConstants
    - SOUND\_MODE\_LOOP, 729
    - SOUND\_MODE\_ONCE, 729
    - SOUND\_MODE\_TONE, 730
  - SoundModuleFunctions
    - PlayFile, 377
    - PlayFileEx, 377
    - PlaySound, 378
    - PlayTone, 378
    - PlayToneEx, 379
    - PlayTones, 379
    - SetSoundDuration, 379
    - SetSoundFlags, 380
    - SetSoundFrequency, 380
    - SetSoundMode, 381
    - SetSoundModuleState, 381
    - SetSoundSampleRate, 381
    - SetSoundVolume, 382
  - SoundDuration, 382
  - SoundFlags, 383
  - SoundFrequency, 383
  - SoundMode, 383
  - SoundSampleRate, 384
  - SoundState, 384
  - SoundVolume, 384
  - StopSound, 385
  - SysSoundGetState, 385
  - SysSoundPlayFile, 385
  - SysSoundPlayTone, 386
  - SysSoundSetState, 386
- SoundModuleID
    - ModuleIDConstants, 260
    - NBCCCommon.h, 1361
  - SoundModuleName
    - ModuleNameConstants, 258
    - NBCCCommon.h, 1361
  - SoundOffsetDuration
    - NBCCCommon.h, 1361
    - SoundIOMAP, 730
  - SoundOffsetFlags
    - NBCCCommon.h, 1362
    - SoundIOMAP, 730
  - SoundOffsetFreq
    - NBCCCommon.h, 1362
    - SoundIOMAP, 730
  - SoundOffsetMode
    - NBCCCommon.h, 1362
    - SoundIOMAP, 731
  - SoundOffsetSampleRate
    - NBCCCommon.h, 1362
    - SoundIOMAP, 731
  - SoundOffsetSoundFilename
    - NBCCCommon.h, 1362
    - SoundIOMAP, 731
  - SoundOffsetState
    - NBCCCommon.h, 1362
    - SoundIOMAP, 731
  - SoundOffsetVolume

- NBCCCommon.h, 1362
- SoundIOMAP, 731
- SoundPlayFile
  - NBCCCommon.h, 1362
  - SysCallConstants, 690
- SoundPlayFileType, 1105
  - Filename, 1105
  - Loop, 1106
  - Result, 1106
  - SoundLevel, 1106
- SoundPlayTone
  - NBCCCommon.h, 1362
  - SysCallConstants, 691
- SoundPlayToneType, 1106
  - Duration, 1107
  - Frequency, 1107
  - Loop, 1107
  - Result, 1107
  - SoundLevel, 1107
- SoundSampleRate
  - NXCDefs.h, 1917
  - SoundModuleFunctions, 384
- SoundSetState
  - NBCCCommon.h, 1363
  - SysCallConstants, 691
- SoundSetStateType, 1108
  - Flags, 1108
  - Result, 1108
  - State, 1109
- SoundState
  - NXCDefs.h, 1917
  - SoundModuleFunctions, 384
- SoundState constants, 728
- SoundStateConstants
  - SOUND\_STATE\_FILE, 728
  - SOUND\_STATE\_IDLE, 728
  - SOUND\_STATE\_STOP, 729
  - SOUND\_STATE\_TONE, 729
- SoundVolume
  - NXCDefs.h, 1918
  - SoundModuleFunctions, 384
- SPECIALS
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- sprintf
  - cstdioAPI, 643
- NXCDefs.h, 1918
- Sqrt
  - cmathAPI, 614
  - NXCDefs.h, 1501
- sqrt
  - cmathAPI, 629
  - NXCDefs.h, 1919
- srand
  - cstdlibAPI, 651
  - NXCDefs.h, 1919
- Standard I2C constants, 801
- Standard-C API functions, 263
- StartLoc
  - DrawLineType, 1051
- StartTask
  - CommandModuleFunctions, 439
  - NXCDefs.h, 1920
- STAT\_COMM\_PENDING
  - CommandModuleConstants, 52
  - NBCCCommon.h, 1363
- STAT\_MSG\_EMPTY\_MAILBOX
  - CommandModuleConstants, 52
  - NBCCCommon.h, 1363
- State
  - SoundGetStateType, 1104
  - SoundSetStateType, 1109
- Status
  - CommHSReadWriteType, 1030
  - DisplayExecuteFunctionType, 1041
- STATUSICON\_BATTERY
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- STATUSICON\_BLUETOOTH
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- STATUSICON\_USB
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- STATUSICON\_VM
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- STATUSICONS
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1363
- STATUSTEXT
  - DisplayModuleConstants, 815

- NBCCCommon.h, 1363
- STEPICON\_1
  - DisplayModuleConstants, 815
  - NBCCCommon.h, 1364
- STEPICON\_2
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- STEPICON\_3
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- STEPICON\_4
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- STEPICON\_5
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- STEPICONS
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- STEPLINE
  - DisplayModuleConstants, 816
  - NBCCCommon.h, 1364
- Stop
  - CommandModuleFunctions, 439
  - NXCDefs.h, 1920
- STOP\_REQ
  - CommandVMState, 706
  - NBCCCommon.h, 1364
- StopAllTasks
  - CommandModuleFunctions, 440
  - NXCDefs.h, 1920
- StopSound
  - NXCDefs.h, 1920
  - SoundModuleFunctions, 385
- StopTask
  - CommandModuleFunctions, 440
  - NXCDefs.h, 1921
- StrCat
  - cstringAPI, 666
  - NXCDefs.h, 1922
- strcat
  - cstringAPI, 665
  - NXCDefs.h, 1921
- strcmp
  - cstringAPI, 666
  - NXCDefs.h, 1922
- strcpy
  - cstringAPI, 667
  - NXCDefs.h, 1923
- StrIndex
  - cstringAPI, 667
  - NXCDefs.h, 1923
- StrLen
  - cstringAPI, 668
  - NXCDefs.h, 1924
- strlen
  - cstringAPI, 668
  - NXCDefs.h, 1924
- strncat
  - cstringAPI, 669
  - NXCDefs.h, 1924
- strncmp
  - cstringAPI, 669
  - NXCDefs.h, 1925
- strncpy
  - cstringAPI, 670
  - NXCDefs.h, 1925
- STROBE\_READ
  - NBCCCommon.h, 1364
  - StrobeCtrlConstants, 127
- STROBE\_S0
  - NBCCCommon.h, 1364
  - StrobeCtrlConstants, 127
- STROBE\_S1
  - NBCCCommon.h, 1365
  - StrobeCtrlConstants, 127
- STROBE\_S2
  - NBCCCommon.h, 1365
  - StrobeCtrlConstants, 127
- STROBE\_S3
  - NBCCCommon.h, 1365
  - StrobeCtrlConstants, 127
- STROBE\_WRITE
  - NBCCCommon.h, 1365
  - StrobeCtrlConstants, 127
- StrobeCtrlConstants
  - STROBE\_READ, 127
  - STROBE\_S0, 127
  - STROBE\_S1, 127
  - STROBE\_S2, 127
  - STROBE\_S3, 127
  - STROBE\_WRITE, 127

- StrReplace
  - cstringAPI, 670
  - NXCDefs.h, 1926
- StrToByteArray
  - cstringAPI, 671
  - NXCDefs.h, 1926
- strtod
  - cstdlibAPI, 651
  - NXCDefs.h, 1927
- strtol
  - cstdlibAPI, 652
  - NXCDefs.h, 1928
- StrToNum
  - cstringAPI, 671
  - NXCDefs.h, 1929
- strtoul
  - cstdlibAPI, 653
  - NXCDefs.h, 1929
- SubStr
  - cstringAPI, 672
  - NXCDefs.h, 1930
- SuperPro analog output mode constants, 122
- SuperPro digital pin constants, 124
- SuperPro LED control constants, 124
- SuperPro Strobe control constants, 126
- SyncTick
  - DatalogGetTimesType, 1038
- SyncTime
  - DatalogGetTimesType, 1038
- SysCall
  - CommandModuleFunctions, 440
  - NXCDefs.h, 1930
- SysCallConstants
  - ColorSensorRead, 684
  - CommBTCheckStatus, 684
  - CommBTConnection, 684
  - CommBTOnOff, 684
  - CommBTRead, 684
  - CommBTWrite, 684
  - CommExecuteFunction, 684
  - CommHSCheckStatus, 684
  - CommHSControl, 684
  - CommHSRead, 684
  - CommHSWrite, 685
  - CommLSCheckStatus, 685
  - CommLSRead, 685
  - CommLSWrite, 685
  - CommLSWriteEx, 685
  - ComputeCalibValue, 685
  - DatalogGetTimes, 685
  - DatalogWrite, 685
  - DisplayExecuteFunction, 685
  - DrawCircle, 685
  - DrawEllipse, 686
  - DrawFont, 686
  - DrawGraphic, 686
  - DrawGraphicArray, 686
  - DrawLine, 686
  - DrawPoint, 686
  - DrawPolygon, 686
  - DrawRect, 686
  - DrawText, 686
  - FileClose, 687
  - FileDelete, 687
  - FileFindFirst, 687
  - FileFindNext, 687
  - FileOpenAppend, 687
  - FileOpenRead, 687
  - FileOpenReadLinear, 687
  - FileOpenWrite, 687
  - FileOpenWriteLinear, 687
  - FileOpenWriteNonLinear, 688
  - FileRead, 688
  - FileRename, 688
  - FileResize, 688
  - FileResolveHandle, 688
  - FileSeek, 688
  - FileTell, 688
  - FileWrite, 688
  - GetStartTick, 688
  - InputPinFunction, 688
  - IOMapRead, 689
  - IOMapReadByID, 689
  - IOMapWrite, 689
  - IOMapWriteByID, 689
  - KeepAlive, 689
  - ListFiles, 689
  - LoaderExecuteFunction, 689
  - MemoryManager, 689
  - MessageRead, 689
  - MessageWrite, 690

- RandomEx, [690](#)
- RandomNumber, [690](#)
- ReadButton, [690](#)
- ReadLastResponse, [690](#)
- ReadSemData, [690](#)
- SetScreenMode, [690](#)
- SetSleepTimeoutVal, [690](#)
- SoundGetState, [690](#)
- SoundPlayFile, [690](#)
- SoundPlayTone, [691](#)
- SoundSetState, [691](#)
- UpdateCalibCacheInfo, [691](#)
- WriteSemData, [691](#)
- SysColorSensorRead
  - InputModuleFunctions, [311](#)
  - NXCDefs.h, [1931](#)
- SysCommBTCheckStatus
  - CommModuleFunctions, [510](#)
  - NXCDefs.h, [1931](#)
- SysCommBTConnection
  - CommModuleFunctions, [510](#)
  - NXCDefs.h, [1932](#)
- SysCommBTOnOff
  - CommModuleFunctions, [511](#)
  - NXCDefs.h, [1932](#)
- SysCommBTWrite
  - CommModuleFunctions, [511](#)
  - NXCDefs.h, [1932](#)
- SysCommExecuteFunction
  - CommModuleFunctions, [512](#)
  - NXCDefs.h, [1933](#)
- SysCommHSCheckStatus
  - CommModuleFunctions, [512](#)
  - NXCDefs.h, [1933](#)
- SysCommHSControl
  - CommModuleFunctions, [512](#)
  - NXCDefs.h, [1934](#)
- SysCommHSRead
  - CommModuleFunctions, [513](#)
  - NXCDefs.h, [1934](#)
- SysCommHSWrite
  - CommModuleFunctions, [513](#)
  - NXCDefs.h, [1934](#)
- SysCommLSCheckStatus
  - LowSpeedModuleSystemCallFunc-tions, [411](#)
  - NXCDefs.h, [1935](#)
- SysCommLSRead
  - LowSpeedModuleSystemCallFunc-tions, [411](#)
  - NXCDefs.h, [1935](#)
- SysCommLSWrite
  - LowSpeedModuleSystemCallFunc-tions, [412](#)
  - NXCDefs.h, [1936](#)
- SysCommLSWriteEx
  - LowSpeedModuleSystemCallFunc-tions, [412](#)
  - NXCDefs.h, [1936](#)
- SysComputeCalibValue
  - CommandModuleFunctions, [441](#)
  - NXCDefs.h, [1936](#)
- SysDatalogGetTimes
  - CommandModuleFunctions, [441](#)
  - NXCDefs.h, [1937](#)
- SysDatalogWrite
  - CommandModuleFunctions, [442](#)
  - NXCDefs.h, [1937](#)
- SysDisplayExecuteFunction
  - DisplayModuleFunctions, [369](#)
  - NXCDefs.h, [1938](#)
- SysDrawCircle
  - DisplayModuleFunctions, [369](#)
  - NXCDefs.h, [1938](#)
- SysDrawEllipse
  - DisplayModuleFunctions, [369](#)
  - NXCDefs.h, [1938](#)
- SysDrawFont
  - DisplayModuleFunctions, [370](#)
  - NXCDefs.h, [1939](#)
- SysDrawGraphic
  - DisplayModuleFunctions, [370](#)
  - NXCDefs.h, [1939](#)
- SysDrawGraphicArray
  - DisplayModuleFunctions, [371](#)
  - NXCDefs.h, [1940](#)
- SysDrawLine
  - DisplayModuleFunctions, [371](#)
  - NXCDefs.h, [1940](#)
- SysDrawPoint
  - DisplayModuleFunctions, [371](#)
  - NXCDefs.h, [1940](#)

- SysDrawPolygon
  - DisplayModuleFunctions, 372
  - NXCDefs.h, 1941
- SysDrawRect
  - DisplayModuleFunctions, 372
  - NXCDefs.h, 1941
- SysDrawText
  - DisplayModuleFunctions, 372
  - NXCDefs.h, 1941
- SysFileClose
  - LoaderModuleFunctions, 586
  - NXCDefs.h, 1942
- SysFileDelete
  - LoaderModuleFunctions, 586
  - NXCDefs.h, 1942
- SysFileFindFirst
  - LoaderModuleFunctions, 586
  - NXCDefs.h, 1942
- SysFileFindNext
  - LoaderModuleFunctions, 587
  - NXCDefs.h, 1943
- SysFileOpenAppend
  - LoaderModuleFunctions, 587
  - NXCDefs.h, 1943
- SysFileOpenRead
  - LoaderModuleFunctions, 588
  - NXCDefs.h, 1944
- SysFileOpenReadLinear
  - LoaderModuleFunctions, 588
  - NXCDefs.h, 1944
- SysFileOpenWrite
  - LoaderModuleFunctions, 588
  - NXCDefs.h, 1944
- SysFileOpenWriteLinear
  - LoaderModuleFunctions, 589
  - NXCDefs.h, 1945
- SysFileOpenWriteNonLinear
  - LoaderModuleFunctions, 589
  - NXCDefs.h, 1945
- SysFileRead
  - LoaderModuleFunctions, 590
  - NXCDefs.h, 1946
- SysFileRename
  - LoaderModuleFunctions, 590
  - NXCDefs.h, 1946
- SysFileResize
  - LoaderModuleFunctions, 590
  - NXCDefs.h, 1946
- SysFileResolveHandle
  - LoaderModuleFunctions, 591
  - NXCDefs.h, 1947
- SysFileSeek
  - LoaderModuleFunctions, 591
  - NXCDefs.h, 1947
- SysFileTell
  - LoaderModuleFunctions, 591
  - NXCDefs.h, 1947
- SysFileWrite
  - LoaderModuleFunctions, 592
  - NXCDefs.h, 1948
- SysGetStartTick
  - CommandModuleFunctions, 442
  - NXCDefs.h, 1948
- SysInputPinFunction
  - InputModuleFunctions, 311
  - NXCDefs.h, 1948
- SysIOMapRead
  - CommandModuleFunctions, 443
  - NXCDefs.h, 1949
- SysIOMapReadByID
  - CommandModuleFunctions, 443
  - NXCDefs.h, 1949
- SysIOMapWrite
  - CommandModuleFunctions, 444
  - NXCDefs.h, 1950
- SysIOMapWriteByID
  - CommandModuleFunctions, 444
  - NXCDefs.h, 1950
- SysKeepAlive
  - CommandModuleFunctions, 444
  - NXCDefs.h, 1950
- SysListFiles
  - LoaderModuleFunctions, 592
  - NXCDefs.h, 1951
- SysLoaderExecuteFunction
  - LoaderModuleFunctions, 592
  - NXCDefs.h, 1951
- SysMemoryManager
  - CommandModuleFunctions, 445
  - NXCDefs.h, 1952
- SysMessageRead
  - CommModuleFunctions, 514

- NXCDefs.h, 1952
- SysMessageWrite
  - CommModuleFunctions, 514
  - NXCDefs.h, 1952
- SysRandomEx
  - cstdlibAPI, 654
  - NXCDefs.h, 1953
- SysRandomNumber
  - cstdlibAPI, 654
  - NXCDefs.h, 1953
- SysReadButton
  - ButtonModuleFunctions, 558
  - NXCDefs.h, 1953
- SysReadLastResponse
  - CommandModuleFunctions, 445
  - NXCDefs.h, 1954
- SysReadSemData
  - CommandModuleFunctions, 445
  - NXCDefs.h, 1954
- SysSetScreenMode
  - DisplayModuleFunctions, 373
  - NXCDefs.h, 1955
- SysSetSleepTimeout
  - NXCDefs.h, 1955
  - UiModuleFunctions, 570
- SysSoundGetState
  - NXCDefs.h, 1955
  - SoundModuleFunctions, 385
- SysSoundPlayFile
  - NXCDefs.h, 1956
  - SoundModuleFunctions, 385
- SysSoundPlayTone
  - NXCDefs.h, 1956
  - SoundModuleFunctions, 386
- SysSoundSetState
  - NXCDefs.h, 1956
  - SoundModuleFunctions, 386
- System Call function constants, 682
- System Command functions, 535
- SysUpdateCalibCacheInfo
  - CommandModuleFunctions, 446
  - NXCDefs.h, 1957
- SysWriteSemData
  - CommandModuleFunctions, 446
  - NXCDefs.h, 1957
- TachoCount
  - OutputStateType, 1095
- TachoCountField
  - NBCCCommon.h, 1365
  - OutputFieldConstants, 788
- TachoLimit
  - OutputStateType, 1095
- TachoLimitField
  - NBCCCommon.h, 1365
  - OutputFieldConstants, 789
- Tachometer counter reset flags, 777
- TachoResetConstants
  - RESET\_ALL, 778
  - RESET\_BLOCK\_COUNT, 778
  - RESET\_BLOCKANDTACHO, 778
  - RESET\_COUNT, 778
  - RESET\_NONE, 778
  - RESET\_ROTATION\_COUNT, 778
- Tan
  - cmathAPI, 614
  - NXCDefs.h, 1501
- tan
  - cmathAPI, 630
  - NXCDefs.h, 1958
- TanD
  - cmathAPI, 615
  - NXCDefs.h, 1501
- tand
  - cmathAPI, 630
  - NXCDefs.h, 1958
- Tanh
  - cmathAPI, 615
  - NXCDefs.h, 1502
- tanh
  - cmathAPI, 631
  - NXCDefs.h, 1959
- TanhD
  - cmathAPI, 616
  - NXCDefs.h, 1502
- tanhd
  - cmathAPI, 631
  - NXCDefs.h, 1959
- TEMP\_FQ\_1
  - NBCCCommon.h, 1366
  - TempI2CConstants, 806
- TEMP\_FQ\_2

- NBCCCommon.h, [1366](#)
- TempI2CConstants, [806](#)
- TEMP\_FQ\_4
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [806](#)
- TEMP\_FQ\_6
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [806](#)
- TEMP\_OS\_ONESHOT
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_POL\_HIGH
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_POL\_LOW
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_REG\_CONFIG
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_REG\_TEMP
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_REG\_THIGH
  - NBCCCommon.h, [1366](#)
  - TempI2CConstants, [807](#)
- TEMP\_REG\_TLOW
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [807](#)
- TEMP\_RES\_10BIT
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [807](#)
- TEMP\_RES\_11BIT
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [807](#)
- TEMP\_RES\_12BIT
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TEMP\_RES\_9BIT
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TEMP\_SD\_CONTINUOUS
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TEMP\_SD\_SHUTDOWN
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TempI2CConstants, [808](#)
- TEMP\_TM\_COMPARATOR
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TEMP\_TM\_INTERRUPT
  - NBCCCommon.h, [1367](#)
  - TempI2CConstants, [808](#)
- TempI2CConstants
  - TEMP\_FQ\_1, [806](#)
  - TEMP\_FQ\_2, [806](#)
  - TEMP\_FQ\_4, [806](#)
  - TEMP\_FQ\_6, [806](#)
  - TEMP\_OS\_ONESHOT, [807](#)
  - TEMP\_POL\_HIGH, [807](#)
  - TEMP\_POL\_LOW, [807](#)
  - TEMP\_REG\_CONFIG, [807](#)
  - TEMP\_REG\_TEMP, [807](#)
  - TEMP\_REG\_THIGH, [807](#)
  - TEMP\_REG\_TLOW, [807](#)
  - TEMP\_RES\_10BIT, [807](#)
  - TEMP\_RES\_11BIT, [807](#)
  - TEMP\_RES\_12BIT, [808](#)
  - TEMP\_RES\_9BIT, [808](#)
  - TEMP\_SD\_CONTINUOUS, [808](#)
  - TEMP\_SD\_SHUTDOWN, [808](#)
  - TEMP\_TM\_COMPARATOR, [808](#)
  - TEMP\_TM\_INTERRUPT, [808](#)
- Text
  - DrawFontType, [1047](#)
  - DrawTextType, [1057](#)
- Text line constants, [825](#)
- TEXTLINE\_1
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_2
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_3
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_4
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_5
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)

- TEXTLINE\_6
  - DisplayTextLineConstants, [826](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_7
  - DisplayTextLineConstants, [827](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINE\_8
  - DisplayTextLineConstants, [827](#)
  - NBCCCommon.h, [1368](#)
- TEXTLINES
  - DisplayTextLineConstants, [827](#)
  - NBCCCommon.h, [1368](#)
- TextOut
  - DisplayModuleFunctions, [373](#)
  - NXCDefs.h, [1959](#)
- TheSleepTimeoutMS
  - SetSleepTimeoutType, [1103](#)
- Third-party NXT devices, [262](#)
- Time constants, [695](#)
- TimeConstants
  - MIN\_1, [696](#)
  - MS\_1, [696](#)
  - MS\_10, [696](#)
  - MS\_100, [697](#)
  - MS\_150, [697](#)
  - MS\_2, [697](#)
  - MS\_20, [697](#)
  - MS\_200, [697](#)
  - MS\_250, [697](#)
  - MS\_3, [698](#)
  - MS\_30, [698](#)
  - MS\_300, [698](#)
  - MS\_350, [698](#)
  - MS\_4, [698](#)
  - MS\_40, [698](#)
  - MS\_400, [698](#)
  - MS\_450, [698](#)
  - MS\_5, [698](#)
  - MS\_50, [699](#)
  - MS\_500, [699](#)
  - MS\_6, [699](#)
  - MS\_60, [699](#)
  - MS\_600, [699](#)
  - MS\_7, [699](#)
  - MS\_70, [699](#)
  - MS\_700, [699](#)
  - MS\_8, [700](#)
  - MS\_80, [700](#)
  - MS\_800, [700](#)
  - MS\_9, [700](#)
  - MS\_90, [700](#)
  - MS\_900, [700](#)
  - SEC\_1, [700](#)
  - SEC\_10, [700](#)
  - SEC\_15, [701](#)
  - SEC\_2, [701](#)
  - SEC\_20, [701](#)
  - SEC\_3, [701](#)
  - SEC\_30, [701](#)
  - SEC\_4, [702](#)
  - SEC\_5, [702](#)
  - SEC\_6, [702](#)
  - SEC\_7, [702](#)
  - SEC\_8, [702](#)
  - SEC\_9, [703](#)
- TIMES\_UP
  - CommandVMState, [706](#)
  - NBCCCommon.h, [1369](#)
- tolower
  - ctypeAPI, [678](#)
  - NXCDefs.h, [1961](#)
- Tone, [1109](#)
  - Duration, [1109](#)
  - Frequency, [1109](#)
- Tone constants, [732](#)
- TONE\_A3
  - NBCCCommon.h, [1369](#)
  - ToneConstants, [734](#)
- TONE\_A4
  - NBCCCommon.h, [1369](#)
  - ToneConstants, [734](#)
- TONE\_A5
  - NBCCCommon.h, [1369](#)
  - ToneConstants, [734](#)
- TONE\_A6
  - NBCCCommon.h, [1369](#)
  - ToneConstants, [734](#)
- TONE\_A7
  - NBCCCommon.h, [1369](#)
  - ToneConstants, [734](#)
- TONE\_AS3
  - NBCCCommon.h, [1369](#)

- ToneConstants, 734
- TONE\_AS4
  - NBCCCommon.h, 1369
  - ToneConstants, 734
- TONE\_AS5
  - NBCCCommon.h, 1369
  - ToneConstants, 734
- TONE\_AS6
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_AS7
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_B3
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_B4
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_B5
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_B6
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_B7
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_C4
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_C5
  - NBCCCommon.h, 1370
  - ToneConstants, 735
- TONE\_C6
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_C7
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_CS4
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_CS5
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_CS6
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_CS7
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_D4
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_D5
  - NBCCCommon.h, 1371
  - ToneConstants, 736
- TONE\_D6
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_D7
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_DS4
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_DS5
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_DS6
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_DS7
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_E4
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_E5
  - NBCCCommon.h, 1372
  - ToneConstants, 737
- TONE\_E6
  - NBCCCommon.h, 1373
  - ToneConstants, 738
- TONE\_E7
  - NBCCCommon.h, 1373
  - ToneConstants, 738
- TONE\_F4
  - NBCCCommon.h, 1373
  - ToneConstants, 738
- TONE\_F5

NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_F6  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_F7  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_FS4  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_FS5  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_FS6  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_FS7  
NBCCCommon.h, 1373  
ToneConstants, 738

TONE\_G4  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_G5  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_G6  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_G7  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_GS4  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_GS5  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_GS6  
NBCCCommon.h, 1374  
ToneConstants, 739

TONE\_GS7  
NBCCCommon.h, 1374  
ToneConstants, 739

ToneConstants  
TONE\_A3, 734  
TONE\_A4, 734  
TONE\_A5, 734  
TONE\_A6, 734  
TONE\_A7, 734  
TONE\_AS3, 734  
TONE\_AS4, 734  
TONE\_AS5, 734  
TONE\_AS6, 735  
TONE\_AS7, 735  
TONE\_B3, 735  
TONE\_B4, 735  
TONE\_B5, 735  
TONE\_B6, 735  
TONE\_B7, 735  
TONE\_C4, 735  
TONE\_C5, 735  
TONE\_C6, 736  
TONE\_C7, 736  
TONE\_CS4, 736  
TONE\_CS5, 736  
TONE\_CS6, 736  
TONE\_CS7, 736  
TONE\_D4, 736  
TONE\_D5, 736  
TONE\_D6, 737  
TONE\_D7, 737  
TONE\_DS4, 737  
TONE\_DS5, 737  
TONE\_DS6, 737  
TONE\_DS7, 737  
TONE\_E4, 737  
TONE\_E5, 737  
TONE\_E6, 738  
TONE\_E7, 738  
TONE\_F4, 738  
TONE\_F5, 738  
TONE\_F6, 738  
TONE\_F7, 738  
TONE\_FS4, 738  
TONE\_FS5, 738  
TONE\_FS6, 738  
TONE\_FS7, 738  
TONE\_G4, 739  
TONE\_G5, 739  
TONE\_G6, 739  
TONE\_G7, 739

- TONE\_GS4, [739](#)
- TONE\_GS5, [739](#)
- TONE\_GS6, [739](#)
- TONE\_GS7, [739](#)
- TOPLINE
  - DisplayModuleConstants, [816](#)
  - NBCCCommon.h, [1375](#)
- toupper
  - ctypeAPI, [679](#)
  - NXCDefs.h, [1961](#)
- TRAIN\_CHANNEL\_1
  - IRTrainChannels, [905](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_CHANNEL\_2
  - IRTrainChannels, [905](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_CHANNEL\_3
  - IRTrainChannels, [905](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_CHANNEL\_ALL
  - IRTrainChannels, [905](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_FUNC\_DECR\_SPEED
  - IRTrainFuncs, [904](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_FUNC\_INCR\_SPEED
  - IRTrainFuncs, [904](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_FUNC\_STOP
  - IRTrainFuncs, [904](#)
  - NBCCCommon.h, [1375](#)
- TRAIN\_FUNC\_TOGGLE\_LIGHT
  - IRTrainFuncs, [904](#)
  - NBCCCommon.h, [1376](#)
- TRUE
  - MiscConstants, [262](#)
  - NBCCCommon.h, [1376](#)
- Trunc
  - cmathAPI, [616](#)
  - NXCDefs.h, [1503](#)
- trunc
  - cmathAPI, [632](#)
  - NXCDefs.h, [1961](#)
- TurnRate
  - XGPacketType, [1114](#)
- TurnRatio
  - OutputStateType, [1095](#)
- TurnRatioField
  - NBCCCommon.h, [1376](#)
  - OutputFieldConstants, [789](#)
- Type aliases, [272](#)
- TypeAliases
  - s16, [273](#)
  - s32, [273](#)
  - s8, [273](#)
  - u16, [273](#)
  - u32, [273](#)
  - u8, [273](#)
- TypeField
  - InputFieldConstants, [762](#)
  - NBCCCommon.h, [1376](#)
- u16
  - NXCDefs.h, [1503](#)
  - TypeAliases, [273](#)
- u32
  - NXCDefs.h, [1503](#)
  - TypeAliases, [273](#)
- u8
  - NXCDefs.h, [1503](#)
  - TypeAliases, [273](#)
- UCHAR\_MAX
  - NBCCCommon.h, [1376](#)
  - NXTLimits, [1012](#)
- UF\_PENDING\_UPDATES
  - NBCCCommon.h, [1376](#)
  - OutUFConstants, [776](#)
- UF\_UPDATE\_MODE
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [776](#)
- UF\_UPDATE\_PID\_VALUES
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [776](#)
- UF\_UPDATE\_RESET\_BLOCK\_-  
COUNT
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [776](#)
- UF\_UPDATE\_RESET\_COUNT
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [777](#)
- UF\_UPDATE\_RESET\_ROTATION\_-  
COUNT

- NBCCCommon.h, [1377](#)
- OutUFConstants, [777](#)
- UF\_UPDATE\_SPEED
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [777](#)
- UF\_UPDATE\_TACHO\_LIMIT
  - NBCCCommon.h, [1377](#)
  - OutUFConstants, [777](#)
- Ui module, [56](#)
- Ui module constants, [745](#)
- Ui module functions, [559](#)
- Ui module IOMAP offsets, [753](#)
- Ui module types, [559](#)
- UI\_BT\_CONNECT\_REQUEST
  - NBCCCommon.h, [1377](#)
  - UiBluetoothStateConstants, [751](#)
- UI\_BT\_ERROR\_ATTENTION
  - NBCCCommon.h, [1377](#)
  - UiBluetoothStateConstants, [751](#)
- UI\_BT\_PIN\_REQUEST
  - NBCCCommon.h, [1377](#)
  - UiBluetoothStateConstants, [751](#)
- UI\_BT\_STATE\_CONNECTED
  - NBCCCommon.h, [1378](#)
  - UiBluetoothStateConstants, [751](#)
- UI\_BT\_STATE\_OFF
  - NBCCCommon.h, [1378](#)
  - UiBluetoothStateConstants, [751](#)
- UI\_BT\_STATE\_VISIBLE
  - NBCCCommon.h, [1378](#)
  - UiBluetoothStateConstants, [752](#)
- UI\_BUTTON\_ENTER
  - NBCCCommon.h, [1378](#)
  - UiButtonConstants, [750](#)
- UI\_BUTTON\_EXIT
  - NBCCCommon.h, [1378](#)
  - UiButtonConstants, [750](#)
- UI\_BUTTON\_LEFT
  - NBCCCommon.h, [1378](#)
  - UiButtonConstants, [750](#)
- UI\_BUTTON\_NONE
  - NBCCCommon.h, [1378](#)
  - UiButtonConstants, [750](#)
- UI\_BUTTON\_RIGHT
  - NBCCCommon.h, [1378](#)
  - UiButtonConstants, [750](#)
- UI\_FLAGS\_BUSY
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_DISABLE\_EXIT
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_ENABLE\_STATUS\_UPDATE
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_EXECUTE\_LMS\_FILE
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_REDRAW\_STATUS
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [746](#)
- UI\_FLAGS\_RESET\_SLEEP\_TIMER
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [747](#)
- UI\_FLAGS\_UPDATE
  - NBCCCommon.h, [1379](#)
  - UiFlagsConstants, [747](#)
- UI\_STATE\_BT\_ERROR
  - NBCCCommon.h, [1379](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_CONNECT\_REQUEST
  - NBCCCommon.h, [1380](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_DRAW\_MENU
  - NBCCCommon.h, [1380](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_ENTER\_PRESSED
  - NBCCCommon.h, [1380](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_EXECUTE\_FILE
  - NBCCCommon.h, [1380](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_EXECUTING\_FILE
  - NBCCCommon.h, [1380](#)
  - UiStateConstants, [748](#)
- UI\_STATE\_EXIT\_PRESSED
  - NBCCCommon.h, [1380](#)

- UiStateConstants, 748
- UI\_STATE\_INIT\_DISPLAY
  - NBCCCommon.h, 1380
  - UiStateConstants, 748
- UI\_STATE\_INIT\_INTRO
  - NBCCCommon.h, 1380
  - UiStateConstants, 748
- UI\_STATE\_INIT\_LOW\_BATTERY
  - NBCCCommon.h, 1380
  - UiStateConstants, 748
- UI\_STATE\_INIT\_MENU
  - NBCCCommon.h, 1380
  - UiStateConstants, 749
- UI\_STATE\_INIT\_WAIT
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_STATE\_LEFT\_PRESSED
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_STATE\_LOW\_BATTERY
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_STATE\_NEXT\_MENU
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_STATE\_RIGHT\_PRESSED
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_STATE\_TEST\_BUTTONS
  - NBCCCommon.h, 1381
  - UiStateConstants, 749
- UI\_VM\_IDLE
  - NBCCCommon.h, 1381
  - UiVMRunStateConstants, 752
- UI\_VM\_RESET1
  - NBCCCommon.h, 1381
  - UiVMRunStateConstants, 752
- UI\_VM\_RESET2
  - NBCCCommon.h, 1381
  - UiVMRunStateConstants, 753
- UI\_VM\_RUN\_FREE
  - NBCCCommon.h, 1382
  - UiVMRunStateConstants, 753
- UI\_VM\_RUN\_PAUSE
  - NBCCCommon.h, 1382
  - UiVMRunStateConstants, 753
- UI\_VM\_RUN\_SINGLE
  - NBCCCommon.h, 1382
  - UiVMRunStateConstants, 753
- UiBluetoothStateConstants
  - UI\_BT\_CONNECT\_REQUEST, 751
  - UI\_BT\_ERROR\_ATTENTION, 751
  - UI\_BT\_PIN\_REQUEST, 751
  - UI\_BT\_STATE\_CONNECTED, 751
  - UI\_BT\_STATE\_OFF, 751
  - UI\_BT\_STATE\_VISIBLE, 752
- UIButton
  - NXCDefs.h, 1962
  - UiModuleFunctions, 570
- UIButton constants, 749
- UIButtonConstants
  - UI\_BUTTON\_ENTER, 750
  - UI\_BUTTON\_EXIT, 750
  - UI\_BUTTON\_LEFT, 750
  - UI\_BUTTON\_NONE, 750
  - UI\_BUTTON\_RIGHT, 750
- UiFlagsConstants
  - UI\_FLAGS\_BUSY, 746
  - UI\_FLAGS\_DISABLE\_EXIT, 746
  - UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER, 746
  - UI\_FLAGS\_ENABLE\_STATUS\_UPDATE, 746
  - UI\_FLAGS\_EXECUTE\_LMS\_FILE, 746
  - UI\_FLAGS\_REDRAW\_STATUS, 746
  - UI\_FLAGS\_RESET\_SLEEP\_TIMER, 747
  - UI\_FLAGS\_UPDATE, 747
- UiIOMAP
  - UIOffsetAbortFlag, 754
  - UIOffsetBatteryState, 754
  - UIOffsetBatteryVoltage, 754
  - UIOffsetBluetoothState, 754
  - UIOffsetButton, 754
  - UIOffsetError, 754
  - UIOffsetFlags, 754
  - UIOffsetForceOff, 754
  - UIOffsetLMSfilename, 755
  - UIOffsetOBPPpointer, 755

- UIOffsetPMenu, [755](#)
- UIOffsetRechargeable, [755](#)
- UIOffsetRunState, [755](#)
- UIOffsetSleepTimeout, [755](#)
- UIOffsetSleepTimer, [755](#)
- UIOffsetState, [755](#)
- UIOffsetUsbState, [755](#)
- UIOffsetVolume, [755](#)
- UiModuleFunctions
  - AbortFlag, [561](#)
  - BatteryLevel, [562](#)
  - BatteryState, [562](#)
  - BluetoothState, [562](#)
  - CommandFlags, [563](#)
  - ForceOff, [563](#)
  - LongAbort, [563](#)
  - OnBrickProgramPointer, [564](#)
  - RechargeableBattery, [564](#)
  - SetAbortFlag, [564](#)
  - SetBatteryState, [565](#)
  - SetBluetoothState, [565](#)
  - SetCommandFlags, [565](#)
  - SetLongAbort, [566](#)
  - SetOnBrickProgramPointer, [566](#)
  - SetSleepTime, [566](#)
  - SetSleepTimeout, [567](#)
  - SetSleepTimer, [567](#)
  - SetUIButton, [567](#)
  - SetUIState, [568](#)
  - SetVMRunState, [568](#)
  - SetVolume, [568](#)
  - SleepTime, [569](#)
  - SleepTimeout, [569](#)
  - SleepTimer, [569](#)
  - SysSetSleepTimeout, [570](#)
  - UIButton, [570](#)
  - UIState, [570](#)
  - UsbState, [571](#)
  - VMRunState, [571](#)
  - Volume, [571](#)
- UiModuleID
  - ModuleIDConstants, [260](#)
  - NBCCCommon.h, [1382](#)
- UiModuleName
  - ModuleNameConstants, [258](#)
  - NBCCCommon.h, [1382](#)
- UINT\_MAX
  - NBCCCommon.h, [1382](#)
  - NXTLimits, [1012](#)
- UIOffsetAbortFlag
  - NBCCCommon.h, [1382](#)
  - UiIOMAP, [754](#)
- UIOffsetBatteryState
  - NBCCCommon.h, [1382](#)
  - UiIOMAP, [754](#)
- UIOffsetBatteryVoltage
  - NBCCCommon.h, [1382](#)
  - UiIOMAP, [754](#)
- UIOffsetBluetoothState
  - NBCCCommon.h, [1382](#)
  - UiIOMAP, [754](#)
- UIOffsetButton
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [754](#)
- UIOffsetError
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [754](#)
- UIOffsetFlags
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [754](#)
- UIOffsetForceOff
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [754](#)
- UIOffsetLMSfilename
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetOBPPpointer
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetPMenu
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetRechargeable
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetRunState
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetSleepTimeout
  - NBCCCommon.h, [1383](#)
  - UiIOMAP, [755](#)
- UIOffsetSleepTimer

- NBCCCommon.h, 1384
- UiIOMAP, 755
- UIOffsetState
  - NBCCCommon.h, 1384
  - UiIOMAP, 755
- UIOffsetUsbState
  - NBCCCommon.h, 1384
  - UiIOMAP, 755
- UIOffsetVolume
  - NBCCCommon.h, 1384
  - UiIOMAP, 755
- UIState
  - NXCDefs.h, 1962
  - UiModuleFunctions, 570
- UIState constants, 747
- UIStateConstants
  - UI\_STATE\_BT\_ERROR, 748
  - UI\_STATE\_CONNECT\_-  
REQUEST, 748
  - UI\_STATE\_DRAW\_MENU, 748
  - UI\_STATE\_ENTER\_PRESSED,  
748
  - UI\_STATE\_EXECUTE\_FILE, 748
  - UI\_STATE\_EXECUTING\_FILE,  
748
  - UI\_STATE\_EXIT\_PRESSED, 748
  - UI\_STATE\_INIT\_DISPLAY, 748
  - UI\_STATE\_INIT\_INTRO, 748
  - UI\_STATE\_INIT\_LOW\_-  
BATTERY, 748
  - UI\_STATE\_INIT\_MENU, 749
  - UI\_STATE\_INIT\_WAIT, 749
  - UI\_STATE\_LEFT\_PRESSED, 749
  - UI\_STATE\_LOW\_BATTERY, 749
  - UI\_STATE\_NEXT\_MENU, 749
  - UI\_STATE\_RIGHT\_PRESSED, 749
  - UI\_STATE\_TEST\_BUTTONS, 749
- UiVMRunStateConstants
  - UI\_VM\_IDLE, 752
  - UI\_VM\_RESET1, 752
  - UI\_VM\_RESET2, 753
  - UI\_VM\_RUN\_FREE, 753
  - UI\_VM\_RUN\_PAUSE, 753
  - UI\_VM\_RUN\_SINGLE, 753
- ULONG\_MAX
  - NBCCCommon.h, 1384
- NXTLimits, 1012
- Ultrasonic sensor constants, 803
- UnflattenVar
  - cstringAPI, 672
  - NXCDefs.h, 1962
- UpdateCalibCacheInfo
  - NBCCCommon.h, 1384
  - SysCallConstants, 691
- UpdateCalibCacheInfoType, 1110
  - MaxVal, 1110
  - MinVal, 1110
  - Name, 1111
  - Result, 1111
- UpdateFlagsField
  - NBCCCommon.h, 1384
  - OutputFieldConstants, 789
- US\_CMD\_CONTINUOUS
  - NBCCCommon.h, 1384
  - USI2CConstants, 804
- US\_CMD\_EVENTCAPTURE
  - NBCCCommon.h, 1384
  - USI2CConstants, 804
- US\_CMD\_OFF
  - NBCCCommon.h, 1385
  - USI2CConstants, 804
- US\_CMD\_SINGLESHOT
  - NBCCCommon.h, 1385
  - USI2CConstants, 804
- US\_CMD\_WARMRESET
  - NBCCCommon.h, 1385
  - USI2CConstants, 804
- US\_REG\_ACTUAL\_ZERO
  - NBCCCommon.h, 1385
  - USI2CConstants, 804
- US\_REG\_CM\_INTERVAL
  - NBCCCommon.h, 1385
  - USI2CConstants, 805
- US\_REG\_FACTORY\_ACTUAL\_ZERO
  - NBCCCommon.h, 1385
  - USI2CConstants, 805
- US\_REG\_FACTORY\_SCALE\_-  
DIVISOR
  - NBCCCommon.h, 1385
  - USI2CConstants, 805
- US\_REG\_FACTORY\_SCALE\_FACTOR
  - NBCCCommon.h, 1385

- USI2CConstants, [805](#)
- US\_REG\_MEASUREMENT\_UNITS
  - NBCCCommon.h, [1385](#)
  - USI2CConstants, [805](#)
- US\_REG\_SCALE\_DIVISOR
  - NBCCCommon.h, [1386](#)
  - USI2CConstants, [805](#)
- US\_REG\_SCALE\_FACTOR
  - NBCCCommon.h, [1386](#)
  - USI2CConstants, [805](#)
- USB\_CMD\_READY
  - CommStatusCodesConstants, [856](#)
  - NBCCCommon.h, [1386](#)
- USB\_PROTOCOL\_OVERHEAD
  - CommMiscConstants, [833](#)
  - NBCCCommon.h, [1386](#)
- USBInputBufferInPtr
  - CommModuleFunctions, [514](#)
  - NXCDefs.h, [1963](#)
- USBInputBufferOutPtr
  - CommModuleFunctions, [515](#)
  - NXCDefs.h, [1963](#)
- USBOutputBufferInPtr
  - CommModuleFunctions, [515](#)
  - NXCDefs.h, [1964](#)
- USBOutputBufferOutPtr
  - CommModuleFunctions, [515](#)
  - NXCDefs.h, [1964](#)
- USBPollBufferInPtr
  - CommModuleFunctions, [516](#)
  - NXCDefs.h, [1964](#)
- USBPollBufferOutPtr
  - CommModuleFunctions, [516](#)
  - NXCDefs.h, [1965](#)
- USBState
  - CommModuleFunctions, [516](#)
  - NXCDefs.h, [1965](#)
- UsbState
  - NXCDefs.h, [1965](#)
  - UiModuleFunctions, [571](#)
- UseRS485
  - CommModuleFunctions, [517](#)
  - NXCDefs.h, [1965](#)
- USHRT\_MAX
  - NBCCCommon.h, [1386](#)
  - NXTLimits, [1012](#)
- USI2CConstants
  - US\_CMD\_CONTINUOUS, [804](#)
  - US\_CMD\_EVENTCAPTURE, [804](#)
  - US\_CMD\_OFF, [804](#)
  - US\_CMD\_SINGLESHOT, [804](#)
  - US\_CMD\_WARMRESET, [804](#)
  - US\_REG\_ACTUAL\_ZERO, [804](#)
  - US\_REG\_CM\_INTERVAL, [805](#)
  - US\_REG\_FACTORY\_ACTUAL\_ZERO, [805](#)
  - US\_REG\_FACTORY\_SCALE\_DIVISOR, [805](#)
  - US\_REG\_FACTORY\_SCALE\_FACTOR, [805](#)
  - US\_REG\_MEASUREMENT\_UNITS, [805](#)
  - US\_REG\_SCALE\_DIVISOR, [805](#)
  - US\_REG\_SCALE\_FACTOR, [805](#)
- Valid
  - InputValuesType, [1074](#)
- Variables
  - DrawGraphicArrayType, [1048](#)
  - DrawGraphicType, [1050](#)
- VectorCross
  - cmathAPI, [632](#)
  - NXCDefs.h, [1966](#)
- VectorDot
  - cmathAPI, [632](#)
  - NXCDefs.h, [1966](#)
- VectorNormalize
  - cmathAPI, [633](#)
  - NXCDefs.h, [1966](#)
- VectorType, [1111](#)
  - X, [1112](#)
  - Y, [1112](#)
  - Z, [1112](#)
- VM run state constants, [752](#)
- VM state constants, [705](#)
- VMRunState
  - NXCDefs.h, [1966](#)
  - UiModuleFunctions, [571](#)
- Volume
  - NXCDefs.h, [1967](#)
  - UiModuleFunctions, [571](#)

- Wait
  - CommandModuleFunctions, [447](#)
  - NXCDefs.h, [1967](#)
- Width
  - SizeType, [1104](#)
- Write
  - LoaderModuleFunctions, [593](#)
  - NXCDefs.h, [1968](#)
- WriteBytes
  - LoaderModuleFunctions, [593](#)
  - NXCDefs.h, [1968](#)
- WriteBytesEx
  - LoaderModuleFunctions, [594](#)
  - NXCDefs.h, [1969](#)
- WriteHandle
  - FileResolveHandleType, [1067](#)
- WriteI2CRegister
  - LowSpeedModuleFunctions, [403](#)
  - NXCDefs.h, [1969](#)
- WriteLn
  - LoaderModuleFunctions, [594](#)
  - NXCDefs.h, [1970](#)
- WriteLnString
  - LoaderModuleFunctions, [595](#)
  - NXCDefs.h, [1970](#)
- WriteNRLinkBytes
  - MindSensorsAPI, [229](#)
  - NXCDefs.h, [1971](#)
- WriteSemData
  - NBCCCommon.h, [1386](#)
  - SysCallConstants, [691](#)
- WriteSemDataType, [1112](#)
  - ClearBits, [1113](#)
  - NewVal, [1113](#)
  - Request, [1113](#)
  - SemData, [1113](#)
- WriteString
  - LoaderModuleFunctions, [595](#)
  - NXCDefs.h, [1971](#)
- X
  - LocationType, [1089](#)
  - VectorType, [1112](#)
- X1
  - DisplayExecuteFunctionType, [1041](#)
- X2
  - DisplayExecuteFunctionType, [1041](#)
- XAxis
  - XGPacketType, [1115](#)
- XG1300L\_REG\_2G
  - NBCCCommon.h, [1386](#)
  - XG1300LConstants, [1008](#)
- XG1300L\_REG\_4G
  - NBCCCommon.h, [1386](#)
  - XG1300LConstants, [1008](#)
- XG1300L\_REG\_8G
  - NBCCCommon.h, [1386](#)
  - XG1300LConstants, [1008](#)
- XG1300L\_REG\_ANGLE
  - NBCCCommon.h, [1386](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_REG\_RESET
  - NBCCCommon.h, [1387](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_REG\_TURNRATE
  - NBCCCommon.h, [1387](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_REG\_XAXIS
  - NBCCCommon.h, [1387](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_REG\_YAXIS
  - NBCCCommon.h, [1387](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_REG\_ZAXIS
  - NBCCCommon.h, [1387](#)
  - XG1300LConstants, [1009](#)
- XG1300L\_SCALE\_2G
  - NBCCCommon.h, [1387](#)
  - XG1300LScaleConstants, [1010](#)
- XG1300L\_SCALE\_4G
  - NBCCCommon.h, [1387](#)
  - XG1300LScaleConstants, [1010](#)
- XG1300L\_SCALE\_8G
  - NBCCCommon.h, [1388](#)
  - XG1300LScaleConstants, [1010](#)
- XG1300LConstants
  - MI\_ADDR\_XG1300L, [1008](#)
  - XG1300L\_REG\_2G, [1008](#)
  - XG1300L\_REG\_4G, [1008](#)
  - XG1300L\_REG\_8G, [1008](#)
  - XG1300L\_REG\_ANGLE, [1009](#)
  - XG1300L\_REG\_RESET, [1009](#)

- XG1300L\_REG\_TURNRATE, [1009](#)
- XG1300L\_REG\_XAXIS, [1009](#)
- XG1300L\_REG\_YAXIS, [1009](#)
- XG1300L\_REG\_ZAXIS, [1009](#)
- XG1300LScaleConstants
  - XG1300L\_SCALE\_2G, [1010](#)
  - XG1300L\_SCALE\_4G, [1010](#)
  - XG1300L\_SCALE\_8G, [1010](#)
- XGPacketType, [1114](#)
  - AccAngle, [1114](#)
  - TurnRate, [1114](#)
  - XAxis, [1115](#)
  - YAxis, [1115](#)
  - ZAxis, [1115](#)
- Y
  - LocationType, [1089](#)
  - VectorType, [1112](#)
- Y1
  - DisplayExecuteFunctionType, [1041](#)
- Y2
  - DisplayExecuteFunctionType, [1041](#)
- YAxis
  - XGPacketType, [1115](#)
- Yield
  - CommandModuleFunctions, [448](#)
  - NXCDefs.h, [1972](#)
- Z
  - VectorType, [1112](#)
- ZAxis
  - XGPacketType, [1115](#)