

NXC

Version 1.2.1 r2

Generated by Doxygen 1.6.2

Sun May 23 16:17:23 2010



# Contents

<b>1</b>	<b>NXC Programmer's Guide</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>The NXC Language</b>	<b>5</b>
3.1	Lexical Rules . . . . .	6
3.1.1	Comments . . . . .	6
3.1.2	Whitespace . . . . .	7
3.1.3	Numerical Constants . . . . .	7
3.1.4	String and Character Constants . . . . .	7
3.1.5	Identifiers and Keywords . . . . .	7
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 . . . . .	12
3.2.5	Structures . . . . .	13
3.2.6	Arrays . . . . .	13
3.3	Statements . . . . .	14
3.3.1	Variable Declaration . . . . .	15
3.3.2	Assignment . . . . .	15
3.3.3	Control Structures . . . . .	16
3.3.4	The asm Statement . . . . .	18
3.3.5	Other NXC Statements . . . . .	20
3.4	Expressions . . . . .	20
3.4.1	Conditions . . . . .	21
3.5	The Preprocessor . . . . .	22
3.5.1	#include . . . . .	23
3.5.2	#define . . . . .	23

3.5.3	## (Concatenation)	23
3.5.4	Conditional Compilation	24
3.5.5	#import	24
3.5.6	#download	24
<b>4</b>	<b>Todo List</b>	<b>27</b>
<b>5</b>	<b>Deprecated List</b>	<b>29</b>
<b>6</b>	<b>Module Index</b>	<b>33</b>
6.1	Modules	33
<b>7</b>	<b>Data Structure Index</b>	<b>39</b>
7.1	Data Structures	39
<b>8</b>	<b>File Index</b>	<b>41</b>
8.1	File List	41
<b>9</b>	<b>Module Documentation</b>	<b>43</b>
9.1	NXT Firmware Modules	43
9.1.1	Detailed Description	44
9.2	Input module	45
9.2.1	Detailed Description	45
9.3	Input module constants	46
9.3.1	Detailed Description	47
9.3.2	Define Documentation	47
9.3.2.1	INPUT_CUSTOM9V	47
9.3.2.2	INPUT_CUSTOMACTIVE	47
9.3.2.3	INPUT_CUSTOMINACTIVE	47
9.3.2.4	INPUT_DIGI0	47
9.3.2.5	INPUT_DIGI1	47
9.3.2.6	INPUT_INVALID_DATA	47
9.4	Sensor types and modes	48
9.4.1	Detailed Description	48
9.5	Output module	49
9.5.1	Detailed Description	49
9.6	Output module constants	50
9.6.1	Detailed Description	50
9.7	Command module	51



9.7.1 Detailed Description . . . . .	51
9.8 Command module constants . . . . .	52
9.8.1 Detailed Description . . . . .	52
9.8.2 Define Documentation . . . . .	53
9.8.2.1 NO_ERR . . . . .	53
9.8.2.2 STAT_COMM_PENDING . . . . .	53
9.8.2.3 STAT_MSG_EMPTY_MAILBOX . . . . .	53
9.9 Comm module . . . . .	54
9.9.1 Detailed Description . . . . .	54
9.10 Button module . . . . .	55
9.10.1 Detailed Description . . . . .	55
9.11 IOCtrl module . . . . .	56
9.11.1 Detailed Description . . . . .	56
9.12 Loader module . . . . .	57
9.12.1 Detailed Description . . . . .	57
9.13 Sound module . . . . .	58
9.13.1 Detailed Description . . . . .	58
9.14 Ui module . . . . .	59
9.14.1 Detailed Description . . . . .	59
9.15 Low Speed module . . . . .	60
9.15.1 Detailed Description . . . . .	60
9.16 Display module . . . . .	62
9.16.1 Detailed Description . . . . .	62
9.17 HiTechnic API Functions . . . . .	63
9.17.1 Detailed Description . . . . .	69
9.17.2 Function Documentation . . . . .	70
9.17.2.1 HTIRTrain . . . . .	70
9.17.2.2 HTPFComboDirect . . . . .	70
9.17.2.3 HTPFComboPWM . . . . .	70
9.17.2.4 HTPFRawOutput . . . . .	71
9.17.2.5 HTPFRepeat . . . . .	71
9.17.2.6 HTPFSingleOutputCST . . . . .	72
9.17.2.7 HTPFSingleOutputPWM . . . . .	72
9.17.2.8 HTPFSinglePin . . . . .	73
9.17.2.9 HTPFTrain . . . . .	73
9.17.2.10 HTRCXAddToDatalog . . . . .	74

9.17.2.11 HTRCXBatteryLevel . . . . .	74
9.17.2.12 HTRCXCLEARAllEvents . . . . .	74
9.17.2.13 HTRCXCLEARCounter . . . . .	74
9.17.2.14 HTRCXCLEARMsg . . . . .	75
9.17.2.15 HTRCXCLEARSensor . . . . .	75
9.17.2.16 HTRCXCLEARSound . . . . .	75
9.17.2.17 HTRCXCLEARTimer . . . . .	75
9.17.2.18 HTRCXCreateDatalog . . . . .	75
9.17.2.19 HTRCXDecCounter . . . . .	76
9.17.2.20 HTRCXDeleteSub . . . . .	76
9.17.2.21 HTRCXDeleteSubs . . . . .	76
9.17.2.22 HTRCXDeleteTask . . . . .	76
9.17.2.23 HTRCXDeleteTasks . . . . .	77
9.17.2.24 HTRCXDisableOutput . . . . .	77
9.17.2.25 HTRCXEnableOutput . . . . .	77
9.17.2.26 HTRCXEvent . . . . .	77
9.17.2.27 HTRCXFloat . . . . .	77
9.17.2.28 HTRCXFwd . . . . .	78
9.17.2.29 HTRCXIncCounter . . . . .	78
9.17.2.30 HTRCXInvertOutput . . . . .	78
9.17.2.31 HTRCXMuteSound . . . . .	78
9.17.2.32 HTRCXObvertOutput . . . . .	79
9.17.2.33 HTRCXOff . . . . .	79
9.17.2.34 HTRCXOn . . . . .	79
9.17.2.35 HTRCXOnFor . . . . .	79
9.17.2.36 HTRCXOnFwd . . . . .	80
9.17.2.37 HTRCXOnRev . . . . .	80
9.17.2.38 HTRCXPBTurnOff . . . . .	80
9.17.2.39 HTRCXPing . . . . .	80
9.17.2.40 HTRCXPlaySound . . . . .	80
9.17.2.41 HTRCXPlayTone . . . . .	81
9.17.2.42 HTRCXPlayToneVar . . . . .	81
9.17.2.43 HTRCXPoll . . . . .	81
9.17.2.44 HTRCXPollMemory . . . . .	81
9.17.2.45 HTRCXRemote . . . . .	82
9.17.2.46 HTRCXRev . . . . .	82

9.17.2.47 HTRCXSelectDisplay . . . . .	82
9.17.2.48 HTRCXSelectProgram . . . . .	82
9.17.2.49 HTRCXSendSerial . . . . .	83
9.17.2.50 HTRCXSetDirection . . . . .	83
9.17.2.51 HTRCXSetEvent . . . . .	83
9.17.2.52 HTRCXSetGlobalDirection . . . . .	83
9.17.2.53 HTRCXSetGlobalOutput . . . . .	84
9.17.2.54 HTRCXSetIRLinkPort . . . . .	84
9.17.2.55 HTRCXSetMaxPower . . . . .	84
9.17.2.56 HTRCXSetMessage . . . . .	84
9.17.2.57 HTRCXSetOutput . . . . .	85
9.17.2.58 HTRCXSetPower . . . . .	85
9.17.2.59 HTRCXSetPriority . . . . .	85
9.17.2.60 HTRCXSetSensorMode . . . . .	85
9.17.2.61 HTRCXSetSensorType . . . . .	86
9.17.2.62 HTRCXSetSleepTime . . . . .	86
9.17.2.63 HTRCXSetTxPower . . . . .	86
9.17.2.64 HTRCXSetWatch . . . . .	86
9.17.2.65 HTRCXStartTask . . . . .	87
9.17.2.66 HTRCXStopAllTasks . . . . .	87
9.17.2.67 HTRCXStopTask . . . . .	87
9.17.2.68 HTRCXToggle . . . . .	87
9.17.2.69 HTRCXUnmuteSound . . . . .	87
9.17.2.70 HTScoutCalibrateSensor . . . . .	88
9.17.2.71 HTScoutMuteSound . . . . .	88
9.17.2.72 HTScoutSelectSounds . . . . .	88
9.17.2.73 HTScoutSendVLL . . . . .	88
9.17.2.74 HTScoutSetEventFeedback . . . . .	88
9.17.2.75 HTScoutSetLight . . . . .	89
9.17.2.76 HTScoutSetScoutMode . . . . .	89
9.17.2.77 HTScoutSetSensorClickTime . . . . .	89
9.17.2.78 HTScoutSetSensorHysteresis . . . . .	89
9.17.2.79 HTScoutSetSensorLowerLimit . . . . .	90
9.17.2.80 HTScoutSetSensorUpperLimit . . . . .	90
9.17.2.81 HTScoutUnmuteSound . . . . .	90
9.17.2.82 ReadSensorHTAccel . . . . .	90

9.17.2.83	ReadSensorHTColor	91
9.17.2.84	ReadSensorHTColor2Active	91
9.17.2.85	ReadSensorHTIRReceiver	92
9.17.2.86	ReadSensorHTIRReceiverEx	92
9.17.2.87	ReadSensorHTIRSeeker	92
9.17.2.88	ReadSensorHTIRSeeker2AC	93
9.17.2.89	ReadSensorHTIRSeeker2DC	93
9.17.2.90	ReadSensorHTNormalizedColor	94
9.17.2.91	ReadSensorHTNormalizedColor2Active	94
9.17.2.92	ReadSensorHTRawColor	95
9.17.2.93	ReadSensorHTRawColor2	95
9.17.2.94	ReadSensorHTTouchMultiplexer	96
9.17.2.95	SensorHTColorNum	96
9.17.2.96	SensorHTCompass	96
9.17.2.97	SensorHTEOPD	97
9.17.2.98	SensorHTGyro	97
9.17.2.99	SensorHTIRSeeker2ACDir	97
9.17.2.100	SensorHTIRSeeker2Addr	98
9.17.2.101	SensorHTIRSeeker2DCDir	98
9.17.2.102	SensorHTIRSeekerDir	98
9.17.2.103	SetHTColor2Mode	99
9.17.2.104	SetHTIRSeeker2Mode	99
9.17.2.105	SetSensorHTEOPD	99
9.17.2.106	SetSensorHTGyro	100
9.18	MindSensors API Functions	101
9.18.1	Detailed Description	112
9.18.2	Function Documentation	112
9.18.2.1	DISTNxDistance	112
9.18.2.2	DISTNxDistanceEx	113
9.18.2.3	DISTNxGP2D12	113
9.18.2.4	DISTNxGP2D120	113
9.18.2.5	DISTNxGP2D120Ex	114
9.18.2.6	DISTNxGP2D12Ex	114
9.18.2.7	DISTNxGP2YA02	114
9.18.2.8	DISTNxGP2YA02Ex	115
9.18.2.9	DISTNxGP2YA21	115

9.18.2.10 DISTNxGP2YA21Ex . . . . .	115
9.18.2.11 DISTNxMaxDistance . . . . .	116
9.18.2.12 DISTNxMaxDistanceEx . . . . .	116
9.18.2.13 DISTNxMinDistance . . . . .	116
9.18.2.14 DISTNxMinDistanceEx . . . . .	117
9.18.2.15 DISTNxModuleType . . . . .	117
9.18.2.16 DISTNxModuleTypeEx . . . . .	117
9.18.2.17 DISTNxNumPoints . . . . .	118
9.18.2.18 DISTNxNumPointsEx . . . . .	118
9.18.2.19 DISTNxVoltage . . . . .	118
9.18.2.20 DISTNxVoltageEx . . . . .	119
9.18.2.21 MSADPAOff . . . . .	119
9.18.2.22 MSADPAOffEx . . . . .	119
9.18.2.23 MSADPAOn . . . . .	120
9.18.2.24 MSADPAOnEx . . . . .	120
9.18.2.25 MSDeenergize . . . . .	120
9.18.2.26 MSDeenergizeEx . . . . .	121
9.18.2.27 MSEnergize . . . . .	121
9.18.2.28 MSEnergizeEx . . . . .	121
9.18.2.29 MSIRTrain . . . . .	122
9.18.2.30 MSIRTrainEx . . . . .	122
9.18.2.31 MSPFComboDirect . . . . .	122
9.18.2.32 MSPFComboDirectEx . . . . .	123
9.18.2.33 MSPFComboPWM . . . . .	123
9.18.2.34 MSPFComboPWMEEx . . . . .	124
9.18.2.35 MSPFRawOutput . . . . .	124
9.18.2.36 MSPFRawOutputEx . . . . .	125
9.18.2.37 MSPFRepeat . . . . .	125
9.18.2.38 MSPFRepeatEx . . . . .	126
9.18.2.39 MSPFSingleOutputCST . . . . .	126
9.18.2.40 MSPFSingleOutputCSTEx . . . . .	127
9.18.2.41 MSPFSingleOutputPWM . . . . .	127
9.18.2.42 MSPFSingleOutputPWMEEx . . . . .	128
9.18.2.43 MSPFSinglePin . . . . .	128
9.18.2.44 MSPFSinglePinEx . . . . .	129
9.18.2.45 MSPFTrain . . . . .	129

9.18.2.46 MSPFTrainEx . . . . .	130
9.18.2.47 MSRCXAbsVar . . . . .	130
9.18.2.48 MSRCXAddToDatalog . . . . .	130
9.18.2.49 MSRCXAndVar . . . . .	131
9.18.2.50 MSRCXBatteryLevel . . . . .	131
9.18.2.51 MSRCXBoot . . . . .	131
9.18.2.52 MSRCXCalibrateEvent . . . . .	131
9.18.2.53 MSRCXClearAllEvents . . . . .	132
9.18.2.54 MSRCXClearCounter . . . . .	132
9.18.2.55 MSRCXClearMsg . . . . .	132
9.18.2.56 MSRCXClearSensor . . . . .	132
9.18.2.57 MSRCXClearSound . . . . .	132
9.18.2.58 MSRCXClearTimer . . . . .	133
9.18.2.59 MSRCXCreateDatalog . . . . .	133
9.18.2.60 MSRCXDecCounter . . . . .	133
9.18.2.61 MSRCXDeleteSub . . . . .	133
9.18.2.62 MSRCXDeleteSubs . . . . .	133
9.18.2.63 MSRCXDeleteTask . . . . .	134
9.18.2.64 MSRCXDeleteTasks . . . . .	134
9.18.2.65 MSRCXDisableOutput . . . . .	134
9.18.2.66 MSRCXDivVar . . . . .	134
9.18.2.67 MSRCXEnableOutput . . . . .	135
9.18.2.68 MSRCXEvent . . . . .	135
9.18.2.69 MSRCXFloat . . . . .	135
9.18.2.70 MSRCXFwd . . . . .	135
9.18.2.71 MSRCXIncCounter . . . . .	136
9.18.2.72 MSRCXInvertOutput . . . . .	136
9.18.2.73 MSRCXMulVar . . . . .	136
9.18.2.74 MSRCXMuteSound . . . . .	136
9.18.2.75 MSRCXObvertOutput . . . . .	137
9.18.2.76 MSRCXOff . . . . .	137
9.18.2.77 MSRCXOn . . . . .	137
9.18.2.78 MSRCXOnFor . . . . .	137
9.18.2.79 MSRCXOnFwd . . . . .	138
9.18.2.80 MSRCXOnRev . . . . .	138
9.18.2.81 MSRCXOrVar . . . . .	138

9.18.2.82 MSRCXPBTurnOff . . . . .	138
9.18.2.83 MSRCXPing . . . . .	139
9.18.2.84 MSRCXPlaySound . . . . .	139
9.18.2.85 MSRCXPlayTone . . . . .	139
9.18.2.86 MSRCXPlayToneVar . . . . .	139
9.18.2.87 MSRCXPoll . . . . .	140
9.18.2.88 MSRCXPollMemory . . . . .	140
9.18.2.89 MSRCXRemote . . . . .	140
9.18.2.90 MSRCXReset . . . . .	140
9.18.2.91 MSRCXRev . . . . .	141
9.18.2.92 MSRCXSelectDisplay . . . . .	141
9.18.2.93 MSRCXSelectProgram . . . . .	141
9.18.2.94 MSRCXSendSerial . . . . .	141
9.18.2.95 MSRCXSet . . . . .	142
9.18.2.96 MSRCXSetDirection . . . . .	142
9.18.2.97 MSRCXSetEvent . . . . .	142
9.18.2.98 MSRCXSetGlobalDirection . . . . .	142
9.18.2.99 MSRCXSetGlobalOutput . . . . .	143
9.18.2.100MSRCXSetMaxPower . . . . .	143
9.18.2.101MSRCXSetMessage . . . . .	143
9.18.2.102MSRCXSetNRLinkPort . . . . .	144
9.18.2.103MSRCXSetNRLinkPortEx . . . . .	144
9.18.2.104MSRCXSetOutput . . . . .	144
9.18.2.105MSRCXSetPower . . . . .	144
9.18.2.106MSRCXSetPriority . . . . .	145
9.18.2.107MSRCXSetSensorMode . . . . .	145
9.18.2.108MSRCXSetSensorType . . . . .	145
9.18.2.109MSRCXSetSleepTime . . . . .	146
9.18.2.110MSRCXSetTxPower . . . . .	146
9.18.2.111MSRCXSetUserDisplay . . . . .	146
9.18.2.112MSRCXSetVar . . . . .	146
9.18.2.113MSRCXSetWatch . . . . .	147
9.18.2.114MSRCXSgnVar . . . . .	147
9.18.2.115MSRCXStartTask . . . . .	147
9.18.2.116MSRCXStopAllTasks . . . . .	147
9.18.2.117MSRCXStopTask . . . . .	148

9.18.2.118MSRCXSubVar . . . . .	148
9.18.2.119MSRCXSumVar . . . . .	148
9.18.2.120MSRCXToggle . . . . .	148
9.18.2.121MSRCXUnlock . . . . .	149
9.18.2.122MSRCXUnmuteSound . . . . .	149
9.18.2.123MSReadValue . . . . .	149
9.18.2.124MSReadValueEx . . . . .	149
9.18.2.125MSScoutCalibrateSensor . . . . .	150
9.18.2.126MSScoutMuteSound . . . . .	150
9.18.2.127MSScoutSelectSounds . . . . .	150
9.18.2.128MSScoutSendVLL . . . . .	150
9.18.2.129MSScoutSetCounterLimit . . . . .	151
9.18.2.130MSScoutSetEventFeedback . . . . .	151
9.18.2.131MSScoutSetLight . . . . .	151
9.18.2.132MSScoutSetScoutMode . . . . .	151
9.18.2.133MSScoutSetScoutRules . . . . .	152
9.18.2.134MSScoutSetSensorClickTime . . . . .	152
9.18.2.135MSScoutSetSensorHysteresis . . . . .	152
9.18.2.136MSScoutSetSensorLowerLimit . . . . .	152
9.18.2.137MSScoutSetSensorUpperLimit . . . . .	153
9.18.2.138MSScoutSetTimerLimit . . . . .	153
9.18.2.139MSScoutUnmuteSound . . . . .	153
9.18.2.140NRLink2400 . . . . .	154
9.18.2.141NRLink2400Ex . . . . .	154
9.18.2.142NRLink4800 . . . . .	154
9.18.2.143NRLink4800Ex . . . . .	155
9.18.2.144NRLinkFlush . . . . .	155
9.18.2.145NRLinkFlushEx . . . . .	155
9.18.2.146NRLinkIRLong . . . . .	156
9.18.2.147NRLinkIRLongEx . . . . .	156
9.18.2.148NRLinkIRShort . . . . .	156
9.18.2.149NRLinkIRShortEx . . . . .	157
9.18.2.150NRLinkSetPF . . . . .	157
9.18.2.151NRLinkSetPFEx . . . . .	157
9.18.2.152NRLinkSetRCX . . . . .	158
9.18.2.153NRLinkSetRCXEx . . . . .	158



9.18.2.154	NRLinkSetTrain	158
9.18.2.155	NRLinkSetTrainEx	159
9.18.2.156	NRLinkStatus	159
9.18.2.157	NRLinkStatusEx	159
9.18.2.158	NRLinkTxRaw	160
9.18.2.159	NRLinkTxRawEx	160
9.18.2.160	PSPNxAnalog	160
9.18.2.161	PSPNxAnalogEx	161
9.18.2.162	PSPNxDigital	161
9.18.2.163	PSPNxDigitalEx	161
9.18.2.164	ReadNRLinkBytes	162
9.18.2.165	ReadNRLinkBytesEx	162
9.18.2.166	ReadSensorMSAccel	162
9.18.2.167	ReadSensorMSAccelEx	163
9.18.2.168	ReadSensorMSPlayStation	163
9.18.2.169	ReadSensorMSPlayStationEx	164
9.18.2.170	ReadSensorMSRTClock	164
9.18.2.171	ReadSensorMSTilt	165
9.18.2.172	ReadSensorMSTiltEx	165
9.18.2.173	RunNRLinkMacro	165
9.18.2.174	RunNRLinkMacroEx	166
9.18.2.175	SensorMSCompass	166
9.18.2.176	SensorMSCompassEx	166
9.18.2.177	SensorMSDROD	167
9.18.2.178	SensorMSPressure	167
9.18.2.179	SensorMSPressureRaw	167
9.18.2.180	SetSensorMSDROD	168
9.18.2.181	SetSensorMSPressure	168
9.18.2.182	WriteNRLinkBytes	168
9.18.2.183	WriteNRLinkBytesEx	168
9.19	RIC Macro Wrappers	170
9.19.1	Detailed Description	171
9.19.2	Define Documentation	171
9.19.2.1	RICArg	171
9.19.2.2	RICImgPoint	172
9.19.2.3	RICImgRect	172

9.19.2.4	RICMapArg	172
9.19.2.5	RICMapElement	172
9.19.2.6	RICMapFunction	173
9.19.2.7	RICOpCircle	173
9.19.2.8	RICOpCopyBits	173
9.19.2.9	RICOpDescription	173
9.19.2.10	RICOpEllipse	174
9.19.2.11	RICOpLine	174
9.19.2.12	RICOpNumBox	174
9.19.2.13	RICOpPixel	174
9.19.2.14	RICOpPolygon	175
9.19.2.15	RICOpRect	175
9.19.2.16	RICOpSprite	175
9.19.2.17	RICOpVarMap	175
9.19.2.18	RICPolygonPoints	176
9.19.2.19	RICSetValue	176
9.19.2.20	RICSpriteData	176
9.20	NXT firmware module names	177
9.20.1	Detailed Description	177
9.20.2	Define Documentation	177
9.20.2.1	ButtonModuleName	177
9.20.2.2	CommandModuleName	177
9.20.2.3	CommModuleName	177
9.20.2.4	DisplayModuleName	177
9.20.2.5	InputModuleName	177
9.20.2.6	IOCtrlModuleName	178
9.20.2.7	LoaderModuleName	178
9.20.2.8	LowSpeedModuleName	178
9.20.2.9	OutputModuleName	178
9.20.2.10	SoundModuleName	178
9.20.2.11	UIModuleName	178
9.21	NXT firmware module IDs	179
9.21.1	Detailed Description	179
9.21.2	Define Documentation	179
9.21.2.1	ButtonModuleID	179
9.21.2.2	CommandModuleID	179

9.21.2.3	CommModuleID	179
9.21.2.4	DisplayModuleID	179
9.21.2.5	InputModuleID	179
9.21.2.6	IOCtrlModuleID	180
9.21.2.7	LoaderModuleID	180
9.21.2.8	LowSpeedModuleID	180
9.21.2.9	OutputModuleID	180
9.21.2.10	SoundModuleID	180
9.21.2.11	UIModuleID	180
9.22	Miscellaneous NBC/NXC constants	181
9.22.1	Detailed Description	181
9.22.2	Define Documentation	181
9.22.2.1	DEGREES_PER_RADIAN	181
9.22.2.2	FALSE	181
9.22.2.3	NA	181
9.22.2.4	PI	182
9.22.2.5	RADIANS_PER_DEGREE	182
9.22.2.6	TRUE	182
9.23	Third-party NXT devices	183
9.23.1	Detailed Description	183
9.24	Standard-C API functions	184
9.24.1	Detailed Description	184
9.25	A simple 3D graphics library	185
9.25.1	Detailed Description	186
9.25.2	Function Documentation	186
9.25.2.1	glAddToAngleX	186
9.25.2.2	glAddToAngleY	187
9.25.2.3	glAddToAngleZ	187
9.25.2.4	glAddVertex	187
9.25.2.5	glBegin	187
9.25.2.6	glBeginObject	188
9.25.2.7	glBeginRender	188
9.25.2.8	glBox	188
9.25.2.9	glCallObject	188
9.25.2.10	glCos32768	189
9.25.2.11	glCube	189

9.25.2.12	glEnd	189
9.25.2.13	glEndObject	189
9.25.2.14	glFinishRender	189
9.25.2.15	glInit	190
9.25.2.16	glObjectAction	190
9.25.2.17	glPyramid	190
9.25.2.18	glSet	190
9.25.2.19	glSetAngleX	191
9.25.2.20	glSetAngleY	191
9.25.2.21	glSetAngleZ	191
9.25.2.22	glSin32768	191
9.26	Type aliases	192
9.26.1	Detailed Description	192
9.26.2	Define Documentation	192
9.26.2.1	s16	192
9.26.2.2	s32	192
9.26.2.3	s8	192
9.26.2.4	u16	192
9.26.2.5	u32	192
9.26.2.6	u8	192
9.27	Input port constants	193
9.27.1	Detailed Description	193
9.27.2	Define Documentation	193
9.27.2.1	S1	193
9.27.2.2	S2	194
9.27.2.3	S3	195
9.27.2.4	S4	195
9.28	Sensor type constants	196
9.28.1	Detailed Description	196
9.28.2	Define Documentation	196
9.28.2.1	SENSOR_TYPE_COLORBLUE	196
9.28.2.2	SENSOR_TYPE_COLORFULL	196
9.28.2.3	SENSOR_TYPE_COLORGREEN	196
9.28.2.4	SENSOR_TYPE_COLORNONE	197
9.28.2.5	SENSOR_TYPE_COLORRED	197
9.28.2.6	SENSOR_TYPE_CUSTOM	197

9.28.2.7	SENSOR_TYPE_HIGHSPEED . . . . .	197
9.28.2.8	SENSOR_TYPE_LIGHT . . . . .	197
9.28.2.9	SENSOR_TYPE_LIGHT_ACTIVE . . . . .	197
9.28.2.10	SENSOR_TYPE_LIGHT_INACTIVE . . . . .	197
9.28.2.11	SENSOR_TYPE_LOWSPEED . . . . .	197
9.28.2.12	SENSOR_TYPE_LOWSPEED_9V . . . . .	197
9.28.2.13	SENSOR_TYPE_NONE . . . . .	197
9.28.2.14	SENSOR_TYPE_ROTATION . . . . .	197
9.28.2.15	SENSOR_TYPE_SOUND_DB . . . . .	198
9.28.2.16	SENSOR_TYPE_SOUND_DBA . . . . .	198
9.28.2.17	SENSOR_TYPE_TEMPERATURE . . . . .	198
9.28.2.18	SENSOR_TYPE_TOUCH . . . . .	198
9.29	Sensor mode constants . . . . .	199
9.29.1	Detailed Description . . . . .	199
9.29.2	Define Documentation . . . . .	199
9.29.2.1	SENSOR_MODE_BOOL . . . . .	199
9.29.2.2	SENSOR_MODE_CELSIUS . . . . .	199
9.29.2.3	SENSOR_MODE_EDGE . . . . .	199
9.29.2.4	SENSOR_MODE_FAHRENHEIT . . . . .	199
9.29.2.5	SENSOR_MODE_PERCENT . . . . .	199
9.29.2.6	SENSOR_MODE_PULSE . . . . .	200
9.29.2.7	SENSOR_MODE_RAW . . . . .	200
9.29.2.8	SENSOR_MODE_ROTATION . . . . .	200
9.30	Combined sensor type and mode constants . . . . .	201
9.30.1	Detailed Description . . . . .	201
9.30.2	Define Documentation . . . . .	202
9.30.2.1	_SENSOR_CFG . . . . .	202
9.30.2.2	SENSOR_CELSIUS . . . . .	202
9.30.2.3	SENSOR_COLORBLUE . . . . .	202
9.30.2.4	SENSOR_COLORFULL . . . . .	202
9.30.2.5	SENSOR_COLORGREEN . . . . .	202
9.30.2.6	SENSOR_COLORNONE . . . . .	202
9.30.2.7	SENSOR_COLORRED . . . . .	202
9.30.2.8	SENSOR_EDGE . . . . .	202
9.30.2.9	SENSOR_FAHRENHEIT . . . . .	202
9.30.2.10	SENSOR_LIGHT . . . . .	203

9.30.2.11	SENSOR_LOWSPEED	203
9.30.2.12	SENSOR_LOWSPEED_9V	203
9.30.2.13	SENSOR_NXTLIGHT	203
9.30.2.14	SENSOR_PULSE	203
9.30.2.15	SENSOR_ROTATION	203
9.30.2.16	SENSOR_SOUND	203
9.30.2.17	SENSOR_TOUCH	203
9.31	Input module types	204
9.31.1	Detailed Description	204
9.32	Input module functions	205
9.32.1	Detailed Description	208
9.32.2	Function Documentation	208
9.32.2.1	ClearSensor	208
9.32.2.2	ColorADRaw	208
9.32.2.3	ColorBoolean	209
9.32.2.4	ColorCalibration	209
9.32.2.5	ColorCalibrationState	210
9.32.2.6	ColorCalLimits	210
9.32.2.7	ColorSensorRaw	210
9.32.2.8	ColorSensorValue	211
9.32.2.9	CustomSensorActiveStatus	211
9.32.2.10	CustomSensorPercentFullScale	212
9.32.2.11	CustomSensorZeroOffset	212
9.32.2.12	GetInput	212
9.32.2.13	ReadSensorColorEx	213
9.32.2.14	ReadSensorColorRaw	213
9.32.2.15	ResetSensor	214
9.32.2.16	Sensor	214
9.32.2.17	SensorBoolean	214
9.32.2.18	SensorDigiPinsDirection	215
9.32.2.19	SensorDigiPinsOutputLevel	215
9.32.2.20	SensorDigiPinsStatus	215
9.32.2.21	SensorInvalid	216
9.32.2.22	SensorMode	216
9.32.2.23	SensorNormalized	216
9.32.2.24	SensorRaw	217

9.32.2.25	SensorScaled	217
9.32.2.26	SensorType	217
9.32.2.27	SensorValue	218
9.32.2.28	SensorValueBool	218
9.32.2.29	SensorValueRaw	218
9.32.2.30	SetCustomSensorActiveStatus	219
9.32.2.31	SetCustomSensorPercentFullScale	219
9.32.2.32	SetCustomSensorZeroOffset	219
9.32.2.33	SetInput	219
9.32.2.34	SetSensor	220
9.32.2.35	SetSensorBoolean	220
9.32.2.36	SetSensorColorBlue	220
9.32.2.37	SetSensorColorFull	220
9.32.2.38	SetSensorColorGreen	221
9.32.2.39	SetSensorColorNone	221
9.32.2.40	SetSensorColorRed	221
9.32.2.41	SetSensorDigiPinsDirection	222
9.32.2.42	SetSensorDigiPinsOutputLevel	222
9.32.2.43	SetSensorDigiPinsStatus	222
9.32.2.44	SetSensorLight	222
9.32.2.45	SetSensorLowspeed	223
9.32.2.46	SetSensorMode	223
9.32.2.47	SetSensorSound	223
9.32.2.48	SetSensorTouch	224
9.32.2.49	SetSensorType	224
9.32.2.50	SysColorSensorRead	224
9.33	Basic analog sensor value names	225
9.33.1	Detailed Description	225
9.33.2	Define Documentation	225
9.33.2.1	SENSOR_1	225
9.33.2.2	SENSOR_2	225
9.33.2.3	SENSOR_3	225
9.33.2.4	SENSOR_4	225
9.34	Output module functions	226
9.34.1	Detailed Description	229
9.34.2	Function Documentation	229

9.34.2.1	Coast	229
9.34.2.2	CoastEx	229
9.34.2.3	Float	230
9.34.2.4	GetOutput	230
9.34.2.5	MotorActualSpeed	230
9.34.2.6	MotorBlockTachoCount	230
9.34.2.7	MotorMode	231
9.34.2.8	MotorOverload	231
9.34.2.9	MotorPower	231
9.34.2.10	MotorPwnFreq	231
9.34.2.11	MotorRegDValue	232
9.34.2.12	MotorRegIValue	232
9.34.2.13	MotorRegPValue	232
9.34.2.14	MotorRegulation	232
9.34.2.15	MotorRotationCount	233
9.34.2.16	MotorRunState	233
9.34.2.17	MotorTachoCount	233
9.34.2.18	MotorTachoLimit	233
9.34.2.19	MotorTurnRatio	234
9.34.2.20	Off	234
9.34.2.21	OffEx	234
9.34.2.22	OnFwd	234
9.34.2.23	OnFwdEx	235
9.34.2.24	OnFwdReg	235
9.34.2.25	OnFwdRegEx	235
9.34.2.26	OnFwdRegExPID	236
9.34.2.27	OnFwdRegPID	236
9.34.2.28	OnFwdSync	237
9.34.2.29	OnFwdSyncEx	237
9.34.2.30	OnFwdSyncExPID	237
9.34.2.31	OnFwdSyncPID	238
9.34.2.32	OnRev	238
9.34.2.33	OnRevEx	239
9.34.2.34	OnRevReg	239
9.34.2.35	OnRevRegEx	239
9.34.2.36	OnRevRegExPID	240



9.34.2.37 OnRevRegPID . . . . .	240
9.34.2.38 OnRevSync . . . . .	240
9.34.2.39 OnRevSyncEx . . . . .	241
9.34.2.40 OnRevSyncExPID . . . . .	241
9.34.2.41 OnRevSyncPID . . . . .	242
9.34.2.42 ResetAllTachoCounts . . . . .	242
9.34.2.43 ResetBlockTachoCount . . . . .	242
9.34.2.44 ResetRotationCount . . . . .	243
9.34.2.45 ResetTachoCount . . . . .	243
9.34.2.46 RotateMotor . . . . .	243
9.34.2.47 RotateMotorEx . . . . .	243
9.34.2.48 RotateMotorExPID . . . . .	244
9.34.2.49 RotateMotorPID . . . . .	244
9.34.2.50 SetMotorPwnFreq . . . . .	245
9.34.2.51 SetOutput . . . . .	245
9.35 Display module types . . . . .	246
9.35.1 Detailed Description . . . . .	247
9.36 Display module functions . . . . .	248
9.36.1 Detailed Description . . . . .	251
9.36.2 Function Documentation . . . . .	251
9.36.2.1 CircleOut . . . . .	251
9.36.2.2 ClearLine . . . . .	251
9.36.2.3 ClearScreen . . . . .	251
9.36.2.4 DisplayContrast . . . . .	252
9.36.2.5 DisplayDisplay . . . . .	252
9.36.2.6 DisplayEraseMask . . . . .	252
9.36.2.7 DisplayFlags . . . . .	252
9.36.2.8 DisplayFont . . . . .	253
9.36.2.9 DisplayTextLinesCenterFlags . . . . .	253
9.36.2.10 DisplayUpdateMask . . . . .	253
9.36.2.11 EllipseOut . . . . .	253
9.36.2.12 FontNumOut . . . . .	254
9.36.2.13 FontTextOut . . . . .	255
9.36.2.14 GetDisplayNormal . . . . .	255
9.36.2.15 GetDisplayPopup . . . . .	256
9.36.2.16 GraphicArrayOut . . . . .	256

9.36.2.17 GraphicArrayOutEx . . . . .	256
9.36.2.18 GraphicOut . . . . .	257
9.36.2.19 GraphicOutEx . . . . .	257
9.36.2.20 LineOut . . . . .	258
9.36.2.21 NumOut . . . . .	258
9.36.2.22 PointOut . . . . .	259
9.36.2.23 PolyOut . . . . .	260
9.36.2.24 RectOut . . . . .	260
9.36.2.25 ResetScreen . . . . .	261
9.36.2.26 SetDisplayContrast . . . . .	261
9.36.2.27 SetDisplayDisplay . . . . .	261
9.36.2.28 SetDisplayEraseMask . . . . .	261
9.36.2.29 SetDisplayFlags . . . . .	262
9.36.2.30 SetDisplayFont . . . . .	262
9.36.2.31 SetDisplayNormal . . . . .	262
9.36.2.32 SetDisplayPopup . . . . .	262
9.36.2.33 SetDisplayTextLinesCenterFlags . . . . .	263
9.36.2.34 SetDisplayUpdateMask . . . . .	263
9.36.2.35 SysDisplayExecuteFunction . . . . .	263
9.36.2.36 SysDrawCircle . . . . .	264
9.36.2.37 SysDrawEllipse . . . . .	264
9.36.2.38 SysDrawFont . . . . .	264
9.36.2.39 SysDrawGraphic . . . . .	264
9.36.2.40 SysDrawGraphicArray . . . . .	265
9.36.2.41 SysDrawLine . . . . .	265
9.36.2.42 SysDrawPoint . . . . .	265
9.36.2.43 SysDrawPolygon . . . . .	266
9.36.2.44 SysDrawRect . . . . .	266
9.36.2.45 SysDrawText . . . . .	266
9.36.2.46 SysSetScreenMode . . . . .	266
9.36.2.47 TextOut . . . . .	267
9.37 Sound module types . . . . .	268
9.37.1 Detailed Description . . . . .	268
9.38 Sound module functions . . . . .	269
9.38.1 Detailed Description . . . . .	270
9.38.2 Function Documentation . . . . .	270

9.38.2.1	PlayFile	270
9.38.2.2	PlayFileEx	271
9.38.2.3	PlaySound	271
9.38.2.4	PlayTone	271
9.38.2.5	PlayToneEx	272
9.38.2.6	PlayTones	272
9.38.2.7	SetSoundDuration	272
9.38.2.8	SetSoundFlags	273
9.38.2.9	SetSoundFrequency	273
9.38.2.10	SetSoundMode	273
9.38.2.11	SetSoundModuleState	273
9.38.2.12	SetSoundSampleRate	274
9.38.2.13	SetSoundVolume	274
9.38.2.14	SoundDuration	274
9.38.2.15	SoundFlags	275
9.38.2.16	SoundFrequency	275
9.38.2.17	SoundMode	275
9.38.2.18	SoundSampleRate	275
9.38.2.19	SoundState	276
9.38.2.20	SoundVolume	276
9.38.2.21	StopSound	276
9.38.2.22	SysSoundGetState	277
9.38.2.23	SysSoundPlayFile	277
9.38.2.24	SysSoundPlayTone	277
9.38.2.25	SysSoundSetState	277
9.39	LowSpeed module types	278
9.39.1	Detailed Description	278
9.40	LowSpeed module functions	279
9.40.1	Detailed Description	280
9.40.2	Function Documentation	281
9.40.2.1	I2CBytes	281
9.40.2.2	I2CBytesReady	281
9.40.2.3	I2CCheckStatus	282
9.40.2.4	I2CDeviceId	282
9.40.2.5	I2CDeviceIdEx	283
9.40.2.6	I2CDeviceInfo	283

9.40.2.7	I2CDeviceInfoEx	283
9.40.2.8	I2CRead	284
9.40.2.9	I2CSendCommand	284
9.40.2.10	I2CSendCommandEx	285
9.40.2.11	I2CStatus	285
9.40.2.12	I2CVendorId	286
9.40.2.13	I2CVendorIdEx	286
9.40.2.14	I2CVersion	287
9.40.2.15	I2CVersionEx	287
9.40.2.16	I2CWrite	287
9.40.2.17	LowSpeedBytesReady	288
9.40.2.18	LowSpeedCheckStatus	288
9.40.2.19	LowSpeedRead	289
9.40.2.20	LowSpeedStatus	289
9.40.2.21	LowSpeedWrite	290
9.40.2.22	ReadI2CRegister	290
9.40.2.23	ReadSensorUSEx	291
9.40.2.24	SensorUS	291
9.40.2.25	WriteI2CRegister	292
9.41	Low level LowSpeed module functions	293
9.41.1	Detailed Description	294
9.41.2	Function Documentation	294
9.41.2.1	GetLSInputBuffer	294
9.41.2.2	GetLSOutputBuffer	294
9.41.2.3	LSChannelState	294
9.41.2.4	LSErrorType	295
9.41.2.5	LSInputBufferBytesToRx	295
9.41.2.6	LSInputBufferInPtr	295
9.41.2.7	LSInputBufferOutPtr	296
9.41.2.8	LSMode	296
9.41.2.9	LSNoRestartOnRead	296
9.41.2.10	LSOutputBufferBytesToRx	296
9.41.2.11	LSOutputBufferInPtr	297
9.41.2.12	LSOutputBufferOutPtr	297
9.41.2.13	LSSpeed	297
9.41.2.14	LSState	298

9.42	LowSpeed module system call functions	299
9.42.1	Detailed Description	299
9.42.2	Function Documentation	299
9.42.2.1	SysCommLSCheckStatus	299
9.42.2.2	SysCommLSRead	299
9.42.2.3	SysCommLSWrite	300
9.42.2.4	SysCommLSWriteEx	300
9.43	IOCtrl module types	301
9.44	IOCtrl module functions	302
9.44.1	Detailed Description	302
9.44.2	Function Documentation	302
9.44.2.1	PowerDown	302
9.44.2.2	RebootInFirmwareMode	302
9.44.2.3	SleepNow	302
9.45	Command module types	303
9.45.1	Detailed Description	303
9.46	Command module functions	304
9.46.1	Detailed Description	305
9.46.2	Function Documentation	306
9.46.2.1	Acquire	306
9.46.2.2	CurrentTick	306
9.46.2.3	ExitTo	306
9.46.2.4	FirstTick	306
9.46.2.5	Follows	307
9.46.2.6	Precedes	307
9.46.2.7	Release	307
9.46.2.8	ResetSleepTimer	308
9.46.2.9	StartTask	308
9.46.2.10	Stop	308
9.46.2.11	StopAllTasks	308
9.46.2.12	StopTask	308
9.46.2.13	SysCall	309
9.46.2.14	SysComputeCalibValue	309
9.46.2.15	SysDatalogGetTimes	310
9.46.2.16	SysDatalogWrite	310
9.46.2.17	SysGetStartTick	310

9.46.2.18	SysIOMapRead	311
9.46.2.19	SysIOMapReadByID	311
9.46.2.20	SysIOMapWrite	311
9.46.2.21	SysIOMapWriteByID	311
9.46.2.22	SysKeepAlive	312
9.46.2.23	SysReadSemData	312
9.46.2.24	SysUpdateCalibCacheInfo	312
9.46.2.25	SysWriteSemData	313
9.46.2.26	Wait	313
9.46.2.27	Yield	313
9.47	Array API functions	314
9.47.1	Detailed Description	314
9.47.2	Function Documentation	315
9.47.2.1	ArrayBuild	315
9.47.2.2	ArrayInit	315
9.47.2.3	ArrayLen	315
9.47.2.4	ArrayMax	316
9.47.2.5	ArrayMean	316
9.47.2.6	ArrayMin	316
9.47.2.7	ArrayOp	317
9.47.2.8	ArraySort	317
9.47.2.9	ArrayStd	317
9.47.2.10	ArraySubset	318
9.47.2.11	ArraySum	318
9.47.2.12	ArraySumSqr	319
9.48	Comm module types	320
9.48.1	Detailed Description	320
9.49	Comm module functions	321
9.49.1	Detailed Description	328
9.49.2	Function Documentation	328
9.49.2.1	BluetoothStatus	328
9.49.2.2	BluetoothWrite	328
9.49.2.3	BrickDataBluecoreVersion	329
9.49.2.4	BrickDataBtHardwareStatus	329
9.49.2.5	BrickDataBtStateStatus	329
9.49.2.6	BrickDataName	329

9.49.2.7	BrickDataTimeoutValue	330
9.49.2.8	BTConnectionClass	330
9.49.2.9	BTConnectionHandleNum	330
9.49.2.10	BTConnectionLinkQuality	330
9.49.2.11	BTConnectionName	331
9.49.2.12	BTConnectionPinCode	331
9.49.2.13	BTConnectionStreamStatus	331
9.49.2.14	BTDeviceClass	332
9.49.2.15	BTDeviceCount	332
9.49.2.16	BTDeviceName	332
9.49.2.17	BTDeviceNameCount	333
9.49.2.18	BTDeviceStatus	333
9.49.2.19	BTInputBufferInPtr	333
9.49.2.20	BTInputBufferOutPtr	333
9.49.2.21	BTOutputBufferInPtr	334
9.49.2.22	BTOutputBufferOutPtr	334
9.49.2.23	GetBrickDataAddress	334
9.49.2.24	GetBTConnectionAddress	334
9.49.2.25	GetBTDeviceAddress	335
9.49.2.26	GetBTInputBuffer	335
9.49.2.27	GetBTOutputBuffer	335
9.49.2.28	GetHSInputBuffer	335
9.49.2.29	GetHSOutputBuffer	336
9.49.2.30	GetUSBInputBuffer	336
9.49.2.31	GetUSBOutputBuffer	336
9.49.2.32	GetUSBPollBuffer	337
9.49.2.33	HSFlags	337
9.49.2.34	HSInputBufferInPtr	337
9.49.2.35	HSInputBufferOutPtr	337
9.49.2.36	HSMODE	338
9.49.2.37	HSOutputBufferInPtr	338
9.49.2.38	HSOutputBufferOutPtr	338
9.49.2.39	HSSpeed	338
9.49.2.40	HSSState	339
9.49.2.41	ReceiveMessage	339
9.49.2.42	ReceiveRemoteBool	339

9.49.2.43 ReceiveRemoteMessageEx . . . . .	340
9.49.2.44 ReceiveRemoteNumber . . . . .	340
9.49.2.45 ReceiveRemoteString . . . . .	340
9.49.2.46 RemoteKeepAlive . . . . .	341
9.49.2.47 RemoteMessageRead . . . . .	341
9.49.2.48 RemoteMessageWrite . . . . .	341
9.49.2.49 RemotePlaySoundFile . . . . .	342
9.49.2.50 RemotePlayTone . . . . .	342
9.49.2.51 RemoteResetMotorPosition . . . . .	343
9.49.2.52 RemoteResetScaledValue . . . . .	343
9.49.2.53 RemoteSetInputMode . . . . .	343
9.49.2.54 RemoteSetOutputState . . . . .	344
9.49.2.55 RemoteStartProgram . . . . .	344
9.49.2.56 RemoteStopProgram . . . . .	344
9.49.2.57 RemoteStopSound . . . . .	345
9.49.2.58 RS485Control . . . . .	345
9.49.2.59 RS485DataAvailable . . . . .	346
9.49.2.60 RS485Exit . . . . .	346
9.49.2.61 RS485Init . . . . .	346
9.49.2.62 RS485Read . . . . .	347
9.49.2.63 RS485SendingData . . . . .	347
9.49.2.64 RS485Status . . . . .	347
9.49.2.65 RS485Uart . . . . .	348
9.49.2.66 RS485Write . . . . .	348
9.49.2.67 SendMessage . . . . .	348
9.49.2.68 SendRemoteBool . . . . .	349
9.49.2.69 SendRemoteNumber . . . . .	349
9.49.2.70 SendRemoteString . . . . .	349
9.49.2.71 SendResponseBool . . . . .	350
9.49.2.72 SendResponseNumber . . . . .	350
9.49.2.73 SendResponseString . . . . .	351
9.49.2.74 SendRS485Bool . . . . .	351
9.49.2.75 SendRS485Number . . . . .	351
9.49.2.76 SendRS485String . . . . .	352
9.49.2.77 SetBTInputBuffer . . . . .	352
9.49.2.78 SetBTInputBufferInPtr . . . . .	352



9.49.2.79 SetBTInputBufferOutPtr . . . . .	352
9.49.2.80 SetBTOutputBuffer . . . . .	353
9.49.2.81 SetBTOutputBufferInPtr . . . . .	353
9.49.2.82 SetBTOutputBufferOutPtr . . . . .	353
9.49.2.83 SetHSFlags . . . . .	353
9.49.2.84 SetHSInputBuffer . . . . .	354
9.49.2.85 SetHSInputBufferInPtr . . . . .	354
9.49.2.86 SetHSInputBufferOutPtr . . . . .	354
9.49.2.87 SetHSMode . . . . .	354
9.49.2.88 SetHSOutputBuffer . . . . .	355
9.49.2.89 SetHSOutputBufferInPtr . . . . .	355
9.49.2.90 SetHSOutputBufferOutPtr . . . . .	355
9.49.2.91 SetHSSpeed . . . . .	355
9.49.2.92 SetHSSState . . . . .	356
9.49.2.93 SetUSBInputBuffer . . . . .	356
9.49.2.94 SetUSBInputBufferInPtr . . . . .	356
9.49.2.95 SetUSBInputBufferOutPtr . . . . .	356
9.49.2.96 SetUSBOutputBuffer . . . . .	357
9.49.2.97 SetUSBOutputBufferInPtr . . . . .	357
9.49.2.98 SetUSBOutputBufferOutPtr . . . . .	357
9.49.2.99 SetUSBPollBuffer . . . . .	357
9.49.2.100SetUSBPollBufferInPtr . . . . .	358
9.49.2.101SetUSBPollBufferOutPtr . . . . .	358
9.49.2.102SetUSBState . . . . .	358
9.49.2.103SysCommBTCheckStatus . . . . .	358
9.49.2.104SysCommBTConnection . . . . .	359
9.49.2.105SysCommBTOnOff . . . . .	359
9.49.2.106SysCommBTWrite . . . . .	359
9.49.2.107SysCommExecuteFunction . . . . .	359
9.49.2.108SysCommHSCheckStatus . . . . .	360
9.49.2.109SysCommHSControl . . . . .	360
9.49.2.110SysCommHSRead . . . . .	360
9.49.2.111SysCommHSWrite . . . . .	361
9.49.2.112SysMessageRead . . . . .	361
9.49.2.113SysMessageWrite . . . . .	361
9.49.2.114USBInputBufferInPtr . . . . .	362

9.49.2.115	USBInputBufferOutPtr	362
9.49.2.116	USBOutputBufferInPtr	362
9.49.2.117	USBOutputBufferOutPtr	362
9.49.2.118	USBPollBufferInPtr	363
9.49.2.119	USBPollBufferOutPtr	363
9.49.2.120	USBState	363
9.50	Button module types	364
9.50.1	Detailed Description	364
9.51	Button module functions	365
9.51.1	Detailed Description	366
9.51.2	Function Documentation	366
9.51.2.1	ButtonCount	366
9.51.2.2	ButtonLongPressCount	366
9.51.2.3	ButtonLongReleaseCount	366
9.51.2.4	ButtonPressCount	367
9.51.2.5	ButtonPressed	367
9.51.2.6	ButtonReleaseCount	367
9.51.2.7	ButtonShortReleaseCount	368
9.51.2.8	ButtonState	368
9.51.2.9	ReadButtonEx	368
9.51.2.10	SetButtonLongPressCount	369
9.51.2.11	SetButtonLongReleaseCount	369
9.51.2.12	SetButtonPressCount	369
9.51.2.13	SetButtonReleaseCount	370
9.51.2.14	SetButtonShortReleaseCount	370
9.51.2.15	SetButtonState	370
9.51.2.16	SysReadButton	370
9.52	Ui module types	371
9.52.1	Detailed Description	371
9.53	Ui module functions	372
9.53.1	Detailed Description	374
9.53.2	Function Documentation	374
9.53.2.1	AbortFlag	374
9.53.2.2	BatteryLevel	374
9.53.2.3	BatteryState	374
9.53.2.4	BluetoothState	374

9.53.2.5	CommandFlags	375
9.53.2.6	ForceOff	375
9.53.2.7	LongAbort	375
9.53.2.8	OnBrickProgramPointer	376
9.53.2.9	RechargeableBattery	376
9.53.2.10	SetAbortFlag	376
9.53.2.11	SetBatteryState	376
9.53.2.12	SetBluetoothState	377
9.53.2.13	SetCommandFlags	377
9.53.2.14	SetLongAbort	377
9.53.2.15	SetOnBrickProgramPointer	377
9.53.2.16	SetSleepTime	378
9.53.2.17	SetSleepTimeout	378
9.53.2.18	SetSleepTimer	378
9.53.2.19	SetUIButton	378
9.53.2.20	SetUIState	379
9.53.2.21	SetVMRunState	379
9.53.2.22	SetVolume	379
9.53.2.23	SleepTime	379
9.53.2.24	SleepTimeout	380
9.53.2.25	SleepTimer	380
9.53.2.26	SysSetSleepTimeout	380
9.53.2.27	UIButton	380
9.53.2.28	UIState	381
9.53.2.29	UsbState	381
9.53.2.30	VMRunState	381
9.53.2.31	Volume	381
9.54	Loader module types	382
9.54.1	Detailed Description	382
9.55	Loader module functions	383
9.55.1	Detailed Description	385
9.55.2	Function Documentation	385
9.55.2.1	CloseFile	385
9.55.2.2	CreateFile	386
9.55.2.3	CreateFileLinear	386
9.55.2.4	CreateFileNonLinear	387

9.55.2.5 DeleteFile . . . . .	387
9.55.2.6 FindFirstFile . . . . .	387
9.55.2.7 FindNextFile . . . . .	388
9.55.2.8 FreeMemory . . . . .	388
9.55.2.9 OpenFileAppend . . . . .	388
9.55.2.10 OpenFileRead . . . . .	389
9.55.2.11 OpenFileReadLinear . . . . .	389
9.55.2.12 Read . . . . .	390
9.55.2.13 ReadBytes . . . . .	390
9.55.2.14 ReadLn . . . . .	391
9.55.2.15 ReadLnString . . . . .	391
9.55.2.16 RenameFile . . . . .	391
9.55.2.17 ResizeFile . . . . .	391
9.55.2.18 ResolveHandle . . . . .	392
9.55.2.19 SysFileClose . . . . .	392
9.55.2.20 SysFileDelete . . . . .	392
9.55.2.21 SysFileFindFirst . . . . .	393
9.55.2.22 SysFileFindNext . . . . .	393
9.55.2.23 SysFileOpenAppend . . . . .	393
9.55.2.24 SysFileOpenRead . . . . .	394
9.55.2.25 SysFileOpenReadLinear . . . . .	394
9.55.2.26 SysFileOpenWrite . . . . .	394
9.55.2.27 SysFileOpenWriteLinear . . . . .	394
9.55.2.28 SysFileOpenWriteNonLinear . . . . .	395
9.55.2.29 SysFileRead . . . . .	395
9.55.2.30 SysFileRename . . . . .	395
9.55.2.31 SysFileResize . . . . .	396
9.55.2.32 SysFileResolveHandle . . . . .	396
9.55.2.33 SysFileSeek . . . . .	396
9.55.2.34 SysFileWrite . . . . .	396
9.55.2.35 SysListFiles . . . . .	397
9.55.2.36 SysLoaderExecuteFunction . . . . .	397
9.55.2.37 Write . . . . .	397
9.55.2.38 WriteBytes . . . . .	398
9.55.2.39 WriteBytesEx . . . . .	398
9.55.2.40 WriteLn . . . . .	398

9.55.2.41 WriteLnString . . . . .	399
9.55.2.42 WriteString . . . . .	399
9.56 cmath API . . . . .	400
9.56.1 Detailed Description . . . . .	404
9.56.2 Define Documentation . . . . .	404
9.56.2.1 Acos . . . . .	404
9.56.2.2 AcosD . . . . .	404
9.56.2.3 Asin . . . . .	404
9.56.2.4 AsinD . . . . .	405
9.56.2.5 Atan . . . . .	405
9.56.2.6 Atan2 . . . . .	405
9.56.2.7 Atan2D . . . . .	406
9.56.2.8 AtanD . . . . .	406
9.56.2.9 Ceil . . . . .	406
9.56.2.10 Cos . . . . .	407
9.56.2.11 CosD . . . . .	407
9.56.2.12 Cosh . . . . .	407
9.56.2.13 CoshD . . . . .	408
9.56.2.14 Exp . . . . .	408
9.56.2.15 Floor . . . . .	408
9.56.2.16 Frac . . . . .	409
9.56.2.17 Log . . . . .	409
9.56.2.18 Log10 . . . . .	409
9.56.2.19 MulDiv32 . . . . .	410
9.56.2.20 Pow . . . . .	410
9.56.2.21 Sin . . . . .	410
9.56.2.22 SinD . . . . .	411
9.56.2.23 Sinh . . . . .	411
9.56.2.24 SinhD . . . . .	411
9.56.2.25 Sqrt . . . . .	412
9.56.2.26 Tan . . . . .	412
9.56.2.27 TanD . . . . .	412
9.56.2.28 Tanh . . . . .	413
9.56.2.29 TanhD . . . . .	413
9.56.2.30 Trunc . . . . .	413
9.56.3 Function Documentation . . . . .	414

9.56.3.1	acos	414
9.56.3.2	acosd	414
9.56.3.3	asin	414
9.56.3.4	asind	415
9.56.3.5	atan	415
9.56.3.6	atan2	416
9.56.3.7	atan2d	416
9.56.3.8	atand	416
9.56.3.9	bcd2dec	417
9.56.3.10	ceil	417
9.56.3.11	cos	417
9.56.3.12	cosd	418
9.56.3.13	cosh	418
9.56.3.14	coshd	419
9.56.3.15	exp	419
9.56.3.16	floor	419
9.56.3.17	frac	420
9.56.3.18	isNAN	420
9.56.3.19	log	420
9.56.3.20	log10	421
9.56.3.21	muldiv32	421
9.56.3.22	pow	422
9.56.3.23	sign	422
9.56.3.24	sin	422
9.56.3.25	sind	423
9.56.3.26	sinh	423
9.56.3.27	sinhd	423
9.56.3.28	sqrt	424
9.56.3.29	tan	424
9.56.3.30	tand	424
9.56.3.31	tanh	425
9.56.3.32	tanhd	425
9.56.3.33	trunc	425
9.57	cstdio API	426
9.57.1	Detailed Description	427
9.57.2	Define Documentation	427

9.57.2.1	getc	427
9.57.2.2	putc	428
9.57.3	Function Documentation	428
9.57.3.1	fclose	428
9.57.3.2	feof	428
9.57.3.3	fflush	429
9.57.3.4	fgetc	429
9.57.3.5	fgets	429
9.57.3.6	fopen	430
9.57.3.7	fprintf	430
9.57.3.8	fputc	430
9.57.3.9	fputs	431
9.57.3.10	fseek	431
9.57.3.11	ftell	432
9.57.3.12	getchar	432
9.57.3.13	printf	432
9.57.3.14	remove	433
9.57.3.15	rename	433
9.57.3.16	rewind	433
9.57.3.17	sprintf	433
9.58	fseek origin constants	435
9.58.1	Detailed Description	435
9.58.2	Define Documentation	435
9.58.2.1	SEEK_CUR	435
9.58.2.2	SEEK_END	435
9.58.2.3	SEEK_SET	435
9.59	cstdlib API	436
9.59.1	Detailed Description	437
9.59.2	Function Documentation	437
9.59.2.1	abort	437
9.59.2.2	abs	437
9.59.2.3	atof	437
9.59.2.4	atoi	438
9.59.2.5	atol	438
9.59.2.6	div	439
9.59.2.7	labs	439

9.59.2.8	ldiv	440
9.59.2.9	rand	440
9.59.2.10	Random	440
9.59.2.11	strtod	441
9.59.2.12	strtol	441
9.59.2.13	strtoul	442
9.59.2.14	SysRandomNumber	443
9.60	cstdlib API types	444
9.60.1	Detailed Description	444
9.61	cstring API	445
9.61.1	Detailed Description	447
9.61.2	Function Documentation	447
9.61.2.1	addr	447
9.61.2.2	addrx	447
9.61.2.3	ByteArrayToStr	448
9.61.2.4	ByteArrayToStrEx	448
9.61.2.5	Copy	448
9.61.2.6	Flatten	449
9.61.2.7	FlattenVar	449
9.61.2.8	FormatNum	449
9.61.2.9	LeftStr	450
9.61.2.10	memcmp	450
9.61.2.11	memcpy	450
9.61.2.12	memmove	451
9.61.2.13	MidStr	451
9.61.2.14	NumToStr	451
9.61.2.15	reladdr	452
9.61.2.16	RightStr	452
9.61.2.17	strcat	452
9.61.2.18	StrCat	453
9.61.2.19	strcmp	453
9.61.2.20	strcpy	453
9.61.2.21	StrIndex	454
9.61.2.22	strlen	454
9.61.2.23	StrLen	454
9.61.2.24	strncat	455



9.61.2.25	strncmp	455
9.61.2.26	strncpy	456
9.61.2.27	StrReplace	456
9.61.2.28	StrToByteArray	456
9.61.2.29	StrToNum	457
9.61.2.30	SubStr	457
9.61.2.31	UnflattenVar	457
9.62	ctype API	459
9.62.1	Detailed Description	459
9.62.2	Function Documentation	460
9.62.2.1	isalnum	460
9.62.2.2	isalpha	460
9.62.2.3	isctrl	460
9.62.2.4	isdigit	461
9.62.2.5	isgraph	461
9.62.2.6	islower	461
9.62.2.7	isprint	462
9.62.2.8	ispunct	462
9.62.2.9	isspace	462
9.62.2.10	isupper	462
9.62.2.11	isxdigit	463
9.62.2.12	tolower	463
9.62.2.13	toupper	463
9.63	Array operation constants	465
9.63.1	Detailed Description	465
9.63.2	Define Documentation	465
9.63.2.1	OPARR_MAX	465
9.63.2.2	OPARR_MEAN	465
9.63.2.3	OPARR_MIN	465
9.63.2.4	OPARR_SORT	465
9.63.2.5	OPARR_STD	465
9.63.2.6	OPARR_SUM	465
9.63.2.7	OPARR_SUMSQR	466
9.64	System Call function constants	467
9.64.1	Detailed Description	468
9.64.2	Define Documentation	468

9.64.2.1	ColorSensorRead	468
9.64.2.2	CommBTCheckStatus	468
9.64.2.3	CommBTConnection	468
9.64.2.4	CommBTOnOff	468
9.64.2.5	CommBTRead	469
9.64.2.6	CommBTWrite	469
9.64.2.7	CommExecuteFunction	469
9.64.2.8	CommHSCheckStatus	469
9.64.2.9	CommHSControl	469
9.64.2.10	CommHSRead	469
9.64.2.11	CommHSWrite	469
9.64.2.12	CommLSCheckStatus	469
9.64.2.13	CommLSRead	469
9.64.2.14	CommLSWrite	469
9.64.2.15	CommLSWriteEx	469
9.64.2.16	ComputeCalibValue	470
9.64.2.17	DatalogGetTimes	470
9.64.2.18	DatalogWrite	470
9.64.2.19	DisplayExecuteFunction	470
9.64.2.20	DrawCircle	470
9.64.2.21	DrawEllipse	470
9.64.2.22	DrawFont	470
9.64.2.23	DrawGraphic	470
9.64.2.24	DrawGraphicArray	470
9.64.2.25	DrawLine	470
9.64.2.26	DrawPoint	470
9.64.2.27	DrawPolygon	471
9.64.2.28	DrawRect	471
9.64.2.29	DrawText	471
9.64.2.30	FileClose	471
9.64.2.31	FileDelete	471
9.64.2.32	FileFindFirst	471
9.64.2.33	FileFindNext	471
9.64.2.34	FileOpenAppend	471
9.64.2.35	FileOpenRead	471
9.64.2.36	FileOpenReadLinear	471

9.64.2.37 FileOpenWrite . . . . .	471
9.64.2.38 FileOpenWriteLinear . . . . .	472
9.64.2.39 FileOpenWriteNonLinear . . . . .	472
9.64.2.40 FileRead . . . . .	472
9.64.2.41 FileRename . . . . .	472
9.64.2.42 FileResize . . . . .	472
9.64.2.43 FileResolveHandle . . . . .	472
9.64.2.44 FileSeek . . . . .	472
9.64.2.45 FileWrite . . . . .	472
9.64.2.46 GetStartTick . . . . .	472
9.64.2.47 IOMapRead . . . . .	472
9.64.2.48 IOMapReadByID . . . . .	472
9.64.2.49 IOMapWrite . . . . .	473
9.64.2.50 IOMapWriteByID . . . . .	473
9.64.2.51 KeepAlive . . . . .	473
9.64.2.52 ListFiles . . . . .	473
9.64.2.53 LoaderExecuteFunction . . . . .	473
9.64.2.54 MessageRead . . . . .	473
9.64.2.55 MessageWrite . . . . .	473
9.64.2.56 RandomNumber . . . . .	473
9.64.2.57 ReadButton . . . . .	473
9.64.2.58 ReadSemData . . . . .	473
9.64.2.59 SetScreenMode . . . . .	473
9.64.2.60 SetSleepTimeoutVal . . . . .	474
9.64.2.61 SoundGetState . . . . .	474
9.64.2.62 SoundPlayFile . . . . .	474
9.64.2.63 SoundPlayTone . . . . .	474
9.64.2.64 SoundSetState . . . . .	474
9.64.2.65 UpdateCalibCacheInfo . . . . .	474
9.64.2.66 WriteSemData . . . . .	474
9.65 Line number constants . . . . .	475
9.65.1 Detailed Description . . . . .	475
9.65.2 Define Documentation . . . . .	475
9.65.2.1 LCD_LINE1 . . . . .	475
9.65.2.2 LCD_LINE2 . . . . .	475
9.65.2.3 LCD_LINE3 . . . . .	476

9.65.2.4	LCD_LINE4 . . . . .	476
9.65.2.5	LCD_LINE5 . . . . .	476
9.65.2.6	LCD_LINE6 . . . . .	476
9.65.2.7	LCD_LINE7 . . . . .	477
9.65.2.8	LCD_LINE8 . . . . .	477
9.66	Time constants . . . . .	478
9.66.1	Detailed Description . . . . .	479
9.66.2	Define Documentation . . . . .	479
9.66.2.1	MIN_1 . . . . .	479
9.66.2.2	MS_1 . . . . .	479
9.66.2.3	MS_10 . . . . .	479
9.66.2.4	MS_100 . . . . .	479
9.66.2.5	MS_150 . . . . .	479
9.66.2.6	MS_2 . . . . .	479
9.66.2.7	MS_20 . . . . .	479
9.66.2.8	MS_200 . . . . .	480
9.66.2.9	MS_250 . . . . .	480
9.66.2.10	MS_3 . . . . .	480
9.66.2.11	MS_30 . . . . .	480
9.66.2.12	MS_300 . . . . .	480
9.66.2.13	MS_350 . . . . .	480
9.66.2.14	MS_4 . . . . .	480
9.66.2.15	MS_40 . . . . .	480
9.66.2.16	MS_400 . . . . .	480
9.66.2.17	MS_450 . . . . .	480
9.66.2.18	MS_5 . . . . .	481
9.66.2.19	MS_50 . . . . .	481
9.66.2.20	MS_500 . . . . .	481
9.66.2.21	MS_6 . . . . .	481
9.66.2.22	MS_60 . . . . .	481
9.66.2.23	MS_600 . . . . .	481
9.66.2.24	MS_7 . . . . .	481
9.66.2.25	MS_70 . . . . .	481
9.66.2.26	MS_700 . . . . .	481
9.66.2.27	MS_8 . . . . .	482
9.66.2.28	MS_80 . . . . .	482

9.66.2.29 MS_800 . . . . .	482
9.66.2.30 MS_9 . . . . .	482
9.66.2.31 MS_90 . . . . .	482
9.66.2.32 MS_900 . . . . .	482
9.66.2.33 SEC_1 . . . . .	482
9.66.2.34 SEC_10 . . . . .	482
9.66.2.35 SEC_15 . . . . .	482
9.66.2.36 SEC_2 . . . . .	483
9.66.2.37 SEC_20 . . . . .	483
9.66.2.38 SEC_3 . . . . .	483
9.66.2.39 SEC_30 . . . . .	483
9.66.2.40 SEC_4 . . . . .	483
9.66.2.41 SEC_5 . . . . .	483
9.66.2.42 SEC_6 . . . . .	483
9.66.2.43 SEC_7 . . . . .	484
9.66.2.44 SEC_8 . . . . .	484
9.66.2.45 SEC_9 . . . . .	484
9.67 Mailbox constants . . . . .	485
9.67.1 Detailed Description . . . . .	485
9.67.2 Define Documentation . . . . .	485
9.67.2.1 MAILBOX1 . . . . .	485
9.67.2.2 MAILBOX10 . . . . .	485
9.67.2.3 MAILBOX2 . . . . .	485
9.67.2.4 MAILBOX3 . . . . .	486
9.67.2.5 MAILBOX4 . . . . .	486
9.67.2.6 MAILBOX5 . . . . .	486
9.67.2.7 MAILBOX6 . . . . .	486
9.67.2.8 MAILBOX7 . . . . .	486
9.67.2.9 MAILBOX8 . . . . .	486
9.67.2.10 MAILBOX9 . . . . .	486
9.68 VM state constants . . . . .	487
9.68.1 Detailed Description . . . . .	487
9.68.2 Define Documentation . . . . .	487
9.68.2.1 BREAKOUT_REQ . . . . .	487
9.68.2.2 CLUMP_DONE . . . . .	487
9.68.2.3 CLUMP_SUSPEND . . . . .	487

9.68.2.4	ROTATE_QUEUE . . . . .	487
9.68.2.5	STOP_REQ . . . . .	487
9.68.2.6	TIMES_UP . . . . .	487
9.69	Fatal errors . . . . .	488
9.69.1	Detailed Description . . . . .	488
9.69.2	Define Documentation . . . . .	488
9.69.2.1	ERR_ARG . . . . .	488
9.69.2.2	ERR_BAD_POOL_SIZE . . . . .	488
9.69.2.3	ERR_BAD_PTR . . . . .	488
9.69.2.4	ERR_CLUMP_COUNT . . . . .	488
9.69.2.5	ERR_DEFAULT_OFFSETS . . . . .	489
9.69.2.6	ERR_FILE . . . . .	489
9.69.2.7	ERR_INSANE_OFFSET . . . . .	489
9.69.2.8	ERR_INSTR . . . . .	489
9.69.2.9	ERR_LOADER_ERR . . . . .	489
9.69.2.10	ERR_MEM . . . . .	489
9.69.2.11	ERR_MEMMGR_FAIL . . . . .	489
9.69.2.12	ERR_NO_ACTIVE_CLUMP . . . . .	489
9.69.2.13	ERR_NO_CODE . . . . .	489
9.69.2.14	ERR_NON_FATAL . . . . .	489
9.69.2.15	ERR_SPOTCHECK_FAIL . . . . .	489
9.69.2.16	ERR_VER . . . . .	490
9.70	General errors . . . . .	491
9.70.1	Detailed Description . . . . .	491
9.70.2	Define Documentation . . . . .	491
9.70.2.1	ERR_INVALID_FIELD . . . . .	491
9.70.2.2	ERR_INVALID_PORT . . . . .	491
9.70.2.3	ERR_INVALID_QUEUE . . . . .	491
9.70.2.4	ERR_INVALID_SIZE . . . . .	491
9.70.2.5	ERR_NO_PROG . . . . .	491
9.71	Communications specific errors . . . . .	492
9.71.1	Detailed Description . . . . .	492
9.71.2	Define Documentation . . . . .	492
9.71.2.1	ERR_COMM_BUFFER_FULL . . . . .	492
9.71.2.2	ERR_COMM_BUS_ERR . . . . .	492
9.71.2.3	ERR_COMM_CHAN_INVALID . . . . .	492

9.71.2.4	ERR_COMM_CHAN_NOT_READY	492
9.72	Remote control (direct commands) errors	493
9.72.1	Detailed Description	493
9.72.2	Define Documentation	493
9.72.2.1	ERR_RC_BAD_PACKET	493
9.72.2.2	ERR_RC_FAILED	493
9.72.2.3	ERR_RC_ILLEGAL_VAL	493
9.72.2.4	ERR_RC_UNKNOWN_CMD	493
9.73	Program status constants	494
9.73.1	Detailed Description	494
9.73.2	Define Documentation	494
9.73.2.1	PROG_ABORT	494
9.73.2.2	PROG_ERROR	494
9.73.2.3	PROG_IDLE	494
9.73.2.4	PROG_OK	494
9.73.2.5	PROG_RESET	494
9.73.2.6	PROG_RUNNING	494
9.74	Command module IOMAP offsets	495
9.74.1	Detailed Description	495
9.74.2	Define Documentation	495
9.74.2.1	CommandOffsetActivateFlag	495
9.74.2.2	CommandOffsetAwake	495
9.74.2.3	CommandOffsetDeactivateFlag	495
9.74.2.4	CommandOffsetFileName	495
9.74.2.5	CommandOffsetFormatString	495
9.74.2.6	CommandOffsetMemoryPool	496
9.74.2.7	CommandOffsetOffsetDS	496
9.74.2.8	CommandOffsetOffsetDVA	496
9.74.2.9	CommandOffsetPRCHandler	496
9.74.2.10	CommandOffsetProgStatus	496
9.74.2.11	CommandOffsetSyncTick	496
9.74.2.12	CommandOffsetSyncTime	496
9.74.2.13	CommandOffsetTick	496
9.75	IOCtrl module constants	497
9.75.1	Detailed Description	497
9.76	PowerOn constants	498

9.76.1 Detailed Description . . . . .	498
9.76.2 Define Documentation . . . . .	498
9.76.2.1 IOCTRL_BOOT . . . . .	498
9.76.2.2 IOCTRL_POWERDOWN . . . . .	498
9.77 IOCtrl module IOMAP offsets . . . . .	499
9.77.1 Detailed Description . . . . .	499
9.77.2 Define Documentation . . . . .	499
9.77.2.1 IOCtrlOffsetPowerOn . . . . .	499
9.78 Loader module constants . . . . .	500
9.78.1 Detailed Description . . . . .	500
9.78.2 Define Documentation . . . . .	500
9.78.2.1 EOF . . . . .	500
9.78.2.2 NULL . . . . .	500
9.79 Loader module IOMAP offsets . . . . .	501
9.79.1 Detailed Description . . . . .	501
9.79.2 Define Documentation . . . . .	501
9.79.2.1 LoaderOffsetFreeUserFlash . . . . .	501
9.79.2.2 LoaderOffsetPFunc . . . . .	501
9.80 Loader module error codes . . . . .	502
9.80.1 Detailed Description . . . . .	502
9.80.2 Define Documentation . . . . .	502
9.80.2.1 LDR_APPENDNOTPOSSIBLE . . . . .	502
9.80.2.2 LDR_BTBUSY . . . . .	503
9.80.2.3 LDR_BTCONNECTFAIL . . . . .	503
9.80.2.4 LDR_BTTIMEOUT . . . . .	503
9.80.2.5 LDR_ENDOFFILE . . . . .	503
9.80.2.6 LDR_EOFEXPECTED . . . . .	503
9.80.2.7 LDR_FILEEXISTS . . . . .	503
9.80.2.8 LDR_FILEISBUSY . . . . .	503
9.80.2.9 LDR_FILEISFULL . . . . .	503
9.80.2.10 LDR_FILENOTFOUND . . . . .	503
9.80.2.11 LDR_FILETX_CLOSEERROR . . . . .	503
9.80.2.12 LDR_FILETX_DSTEXISTS . . . . .	503
9.80.2.13 LDR_FILETX_SRCMISSING . . . . .	504
9.80.2.14 LDR_FILETX_STREAMERROR . . . . .	504
9.80.2.15 LDR_FILETX_TIMEOUT . . . . .	504



9.80.2.16 LDR_HANDLEALREADYCLOSED . . . . .	504
9.80.2.17 LDR_ILLEGALFILENAME . . . . .	504
9.80.2.18 LDR_ILLEGALHANDLE . . . . .	504
9.80.2.19 LDR_INPROGRESS . . . . .	504
9.80.2.20 LDR_INVALIDSEEK . . . . .	504
9.80.2.21 LDR_MODULENOTFOUND . . . . .	504
9.80.2.22 LDR_NOLINEARSPACE . . . . .	504
9.80.2.23 LDR_NOMOREFILES . . . . .	504
9.80.2.24 LDR_NOMOREHANDLES . . . . .	505
9.80.2.25 LDR_NOSPACE . . . . .	505
9.80.2.26 LDR_NOTLINEARFILE . . . . .	505
9.80.2.27 LDR_NOWRITEBUFFERS . . . . .	505
9.80.2.28 LDR_OUTOFBOUNDARY . . . . .	505
9.80.2.29 LDR_REQPIN . . . . .	505
9.80.2.30 LDR_SUCCESS . . . . .	505
9.80.2.31 LDR_UNDEFINEDERROR . . . . .	505
9.81 Loader module function constants . . . . .	506
9.81.1 Detailed Description . . . . .	506
9.81.2 Define Documentation . . . . .	506
9.81.2.1 LDR_CMD_BOOTCMD . . . . .	506
9.81.2.2 LDR_CMD_BTFACTORYRESET . . . . .	507
9.81.2.3 LDR_CMD_BTGETADR . . . . .	507
9.81.2.4 LDR_CMD_CLOSE . . . . .	507
9.81.2.5 LDR_CMD_CLOSEMODHANDLE . . . . .	507
9.81.2.6 LDR_CMD_CROPDATAFILE . . . . .	507
9.81.2.7 LDR_CMD_DELETE . . . . .	507
9.81.2.8 LDR_CMD_DELETEUSERFLASH . . . . .	507
9.81.2.9 LDR_CMD_DEVICEINFO . . . . .	507
9.81.2.10 LDR_CMD_FINDFIRST . . . . .	507
9.81.2.11 LDR_CMD_FINDFIRSTMODULE . . . . .	507
9.81.2.12 LDR_CMD_FINDNEXT . . . . .	507
9.81.2.13 LDR_CMD_FINDNEXTMODULE . . . . .	508
9.81.2.14 LDR_CMD_IOMAPREAD . . . . .	508
9.81.2.15 LDR_CMD_IOMAPWRITE . . . . .	508
9.81.2.16 LDR_CMD_OPENAPPENDDATA . . . . .	508
9.81.2.17 LDR_CMD_OPENREAD . . . . .	508

9.81.2.18 LDR_CMD_OPENREADLINEAR . . . . .	508
9.81.2.19 LDR_CMD_OPENWRITE . . . . .	508
9.81.2.20 LDR_CMD_OPENWRITEDATA . . . . .	508
9.81.2.21 LDR_CMD_OPENWRITELINEAR . . . . .	508
9.81.2.22 LDR_CMD_POLLCMD . . . . .	508
9.81.2.23 LDR_CMD_POLLCMDLEN . . . . .	508
9.81.2.24 LDR_CMD_READ . . . . .	509
9.81.2.25 LDR_CMD_RENAMEFILE . . . . .	509
9.81.2.26 LDR_CMD_RESIZEDATAFILE . . . . .	509
9.81.2.27 LDR_CMD_SEEKFROMCURRENT . . . . .	509
9.81.2.28 LDR_CMD_SEEKFROMEND . . . . .	509
9.81.2.29 LDR_CMD_SEEKFROMSTART . . . . .	509
9.81.2.30 LDR_CMD_SETBRICKNAME . . . . .	509
9.81.2.31 LDR_CMD_VERSIONS . . . . .	509
9.81.2.32 LDR_CMD_WRITE . . . . .	509
9.82 Sound module constants . . . . .	510
9.82.1 Detailed Description . . . . .	510
9.83 SoundFlags constants . . . . .	511
9.83.1 Detailed Description . . . . .	511
9.83.2 Define Documentation . . . . .	511
9.83.2.1 SOUND_FLAGS_IDLE . . . . .	511
9.83.2.2 SOUND_FLAGS_RUNNING . . . . .	511
9.83.2.3 SOUND_FLAGS_UPDATE . . . . .	511
9.84 SoundState constants . . . . .	512
9.84.1 Detailed Description . . . . .	512
9.84.2 Define Documentation . . . . .	512
9.84.2.1 SOUND_STATE_FILE . . . . .	512
9.84.2.2 SOUND_STATE_IDLE . . . . .	512
9.84.2.3 SOUND_STATE_STOP . . . . .	512
9.84.2.4 SOUND_STATE_TONE . . . . .	512
9.85 SoundMode constants . . . . .	513
9.85.1 Detailed Description . . . . .	513
9.85.2 Define Documentation . . . . .	513
9.85.2.1 SOUND_MODE_LOOP . . . . .	513
9.85.2.2 SOUND_MODE_ONCE . . . . .	513
9.85.2.3 SOUND_MODE_TONE . . . . .	513

9.86	Sound module IOMAP offsets	514
9.86.1	Detailed Description	514
9.86.2	Define Documentation	514
9.86.2.1	SoundOffsetDuration	514
9.86.2.2	SoundOffsetFlags	514
9.86.2.3	SoundOffsetFreq	514
9.86.2.4	SoundOffsetMode	514
9.86.2.5	SoundOffsetSampleRate	514
9.86.2.6	SoundOffsetSoundFilename	515
9.86.2.7	SoundOffsetState	515
9.86.2.8	SoundOffsetVolume	515
9.87	Sound module miscellaneous constants	516
9.87.1	Detailed Description	516
9.87.2	Define Documentation	516
9.87.2.1	FREQUENCY_MAX	516
9.87.2.2	FREQUENCY_MIN	516
9.87.2.3	SAMPLERATE_DEFAULT	516
9.87.2.4	SAMPLERATE_MAX	516
9.87.2.5	SAMPLERATE_MIN	516
9.88	Tone constants	517
9.88.1	Detailed Description	518
9.88.2	Define Documentation	518
9.88.2.1	TONE_A3	518
9.88.2.2	TONE_A4	518
9.88.2.3	TONE_A5	518
9.88.2.4	TONE_A6	518
9.88.2.5	TONE_A7	518
9.88.2.6	TONE_AS3	518
9.88.2.7	TONE_AS4	519
9.88.2.8	TONE_AS5	519
9.88.2.9	TONE_AS6	519
9.88.2.10	TONE_AS7	519
9.88.2.11	TONE_B3	519
9.88.2.12	TONE_B4	519
9.88.2.13	TONE_B5	519
9.88.2.14	TONE_B6	519

9.88.2.15 TONE_B7 . . . . .	519
9.88.2.16 TONE_C4 . . . . .	519
9.88.2.17 TONE_C5 . . . . .	520
9.88.2.18 TONE_C6 . . . . .	520
9.88.2.19 TONE_C7 . . . . .	520
9.88.2.20 TONE_CS4 . . . . .	520
9.88.2.21 TONE_CS5 . . . . .	520
9.88.2.22 TONE_CS6 . . . . .	520
9.88.2.23 TONE_CS7 . . . . .	520
9.88.2.24 TONE_D4 . . . . .	520
9.88.2.25 TONE_D5 . . . . .	520
9.88.2.26 TONE_D6 . . . . .	520
9.88.2.27 TONE_D7 . . . . .	521
9.88.2.28 TONE_DS4 . . . . .	521
9.88.2.29 TONE_DS5 . . . . .	521
9.88.2.30 TONE_DS6 . . . . .	521
9.88.2.31 TONE_DS7 . . . . .	521
9.88.2.32 TONE_E4 . . . . .	521
9.88.2.33 TONE_E5 . . . . .	521
9.88.2.34 TONE_E6 . . . . .	521
9.88.2.35 TONE_E7 . . . . .	521
9.88.2.36 TONE_F4 . . . . .	521
9.88.2.37 TONE_F5 . . . . .	522
9.88.2.38 TONE_F6 . . . . .	522
9.88.2.39 TONE_F7 . . . . .	522
9.88.2.40 TONE_FS4 . . . . .	522
9.88.2.41 TONE_FS5 . . . . .	522
9.88.2.42 TONE_FS6 . . . . .	522
9.88.2.43 TONE_FS7 . . . . .	522
9.88.2.44 TONE_G4 . . . . .	522
9.88.2.45 TONE_G5 . . . . .	522
9.88.2.46 TONE_G6 . . . . .	522
9.88.2.47 TONE_G7 . . . . .	523
9.88.2.48 TONE_GS4 . . . . .	523
9.88.2.49 TONE_GS5 . . . . .	523
9.88.2.50 TONE_GS6 . . . . .	523

9.88.2.51	TONE_GS7	523
9.89	Button module constants	524
9.89.1	Detailed Description	524
9.90	Button name constants	525
9.90.1	Detailed Description	525
9.90.2	Define Documentation	525
9.90.2.1	BTN1	525
9.90.2.2	BTN2	525
9.90.2.3	BTN3	525
9.90.2.4	BTN4	525
9.90.2.5	BTNCENTER	526
9.90.2.6	BTNEXIT	526
9.90.2.7	BTNLEFT	526
9.90.2.8	BTNRIGHT	526
9.90.2.9	NO_OF_BTNS	526
9.91	ButtonState constants	527
9.91.1	Detailed Description	527
9.91.2	Define Documentation	527
9.91.2.1	BTNSTATE_LONG_PRESSED_EV	527
9.91.2.2	BTNSTATE_LONG_RELEASED_EV	527
9.91.2.3	BTNSTATE_NONE	527
9.91.2.4	BTNSTATE_PRESSED_EV	527
9.91.2.5	BTNSTATE_PRESSED_STATE	528
9.91.2.6	BTNSTATE_SHORT_RELEASED_EV	528
9.92	Button module IOMAP offsets	529
9.92.1	Detailed Description	529
9.92.2	Define Documentation	529
9.92.2.1	ButtonOffsetLongPressCnt	529
9.92.2.2	ButtonOffsetLongRelCnt	529
9.92.2.3	ButtonOffsetPressedCnt	529
9.92.2.4	ButtonOffsetRelCnt	529
9.92.2.5	ButtonOffsetShortRelCnt	529
9.92.2.6	ButtonOffsetState	529
9.93	Ui module constants	530
9.93.1	Detailed Description	530
9.94	CommandFlags constants	531

9.94.1	Detailed Description	531
9.94.2	Define Documentation	531
9.94.2.1	UI_FLAGS_BUSY	531
9.94.2.2	UI_FLAGS_DISABLE_EXIT	531
9.94.2.3	UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER	531
9.94.2.4	UI_FLAGS_ENABLE_STATUS_UPDATE	531
9.94.2.5	UI_FLAGS_EXECUTE_LMS_FILE	531
9.94.2.6	UI_FLAGS_REDRAW_STATUS	532
9.94.2.7	UI_FLAGS_RESET_SLEEP_TIMER	532
9.94.2.8	UI_FLAGS_UPDATE	532
9.95	UIState constants	533
9.95.1	Detailed Description	533
9.95.2	Define Documentation	533
9.95.2.1	UI_STATE_BT_ERROR	533
9.95.2.2	UI_STATE_CONNECT_REQUEST	533
9.95.2.3	UI_STATE_DRAW_MENU	533
9.95.2.4	UI_STATE_ENTER_PRESSED	533
9.95.2.5	UI_STATE_EXECUTE_FILE	534
9.95.2.6	UI_STATE_EXECUTING_FILE	534
9.95.2.7	UI_STATE_EXIT_PRESSED	534
9.95.2.8	UI_STATE_INIT_DISPLAY	534
9.95.2.9	UI_STATE_INIT_INTRO	534
9.95.2.10	UI_STATE_INIT_LOW_BATTERY	534
9.95.2.11	UI_STATE_INIT_MENU	534
9.95.2.12	UI_STATE_INIT_WAIT	534
9.95.2.13	UI_STATE_LEFT_PRESSED	534
9.95.2.14	UI_STATE_LOW_BATTERY	534
9.95.2.15	UI_STATE_NEXT_MENU	534
9.95.2.16	UI_STATE_RIGHT_PRESSED	535
9.95.2.17	UI_STATE_TEST_BUTTONS	535
9.96	UIButton constants	536
9.96.1	Detailed Description	536
9.96.2	Define Documentation	536
9.96.2.1	UI_BUTTON_ENTER	536
9.96.2.2	UI_BUTTON_EXIT	536
9.96.2.3	UI_BUTTON_LEFT	536

9.96.2.4	UI_BUTTON_NONE	536
9.96.2.5	UI_BUTTON_RIGHT	536
9.97	BluetoothState constants	537
9.97.1	Detailed Description	537
9.97.2	Define Documentation	537
9.97.2.1	UI_BT_CONNECT_REQUEST	537
9.97.2.2	UI_BT_ERROR_ATTENTION	537
9.97.2.3	UI_BT_PIN_REQUEST	537
9.97.2.4	UI_BT_STATE_CONNECTED	537
9.97.2.5	UI_BT_STATE_OFF	537
9.97.2.6	UI_BT_STATE_VISIBLE	538
9.98	VM run state constants	539
9.98.1	Detailed Description	539
9.98.2	Define Documentation	539
9.98.2.1	UI_VM_IDLE	539
9.98.2.2	UI_VM_RESET1	539
9.98.2.3	UI_VM_RESET2	539
9.98.2.4	UI_VM_RUN_FREE	539
9.98.2.5	UI_VM_RUN_PAUSE	539
9.98.2.6	UI_VM_RUN_SINGLE	539
9.99	Ui module IOMAP offsets	540
9.99.1	Detailed Description	540
9.99.2	Define Documentation	540
9.99.2.1	UIOffsetAbortFlag	540
9.99.2.2	UIOffsetBatteryState	540
9.99.2.3	UIOffsetBatteryVoltage	540
9.99.2.4	UIOffsetBluetoothState	540
9.99.2.5	UIOffsetButton	541
9.99.2.6	UIOffsetError	541
9.99.2.7	UIOffsetFlags	541
9.99.2.8	UIOffsetForceOff	541
9.99.2.9	UIOffsetLMSfilename	541
9.99.2.10	UIOffsetOBPPointer	541
9.99.2.11	UIOffsetPMenu	541
9.99.2.12	UIOffsetRechargeable	541
9.99.2.13	UIOffsetRunState	541

9.99.2.14 UIOffsetSleepTimeout . . . . .	541
9.99.2.15 UIOffsetSleepTimer . . . . .	541
9.99.2.16 UIOffsetState . . . . .	542
9.99.2.17 UIOffsetUsbState . . . . .	542
9.99.2.18 UIOffsetVolume . . . . .	542
9.100NBC Input port constants . . . . .	543
9.100.1 Detailed Description . . . . .	543
9.100.2 Define Documentation . . . . .	543
9.100.2.1 IN_1 . . . . .	543
9.100.2.2 IN_2 . . . . .	543
9.100.2.3 IN_3 . . . . .	543
9.100.2.4 IN_4 . . . . .	543
9.101NBC sensor type constants . . . . .	544
9.101.1 Detailed Description . . . . .	544
9.101.2 Define Documentation . . . . .	544
9.101.2.1 IN_TYPE_ANGLE . . . . .	544
9.101.2.2 IN_TYPE_COLORBLUE . . . . .	544
9.101.2.3 IN_TYPE_COLOREXIT . . . . .	544
9.101.2.4 IN_TYPE_COLORFULL . . . . .	545
9.101.2.5 IN_TYPE_COLORGREEN . . . . .	545
9.101.2.6 IN_TYPE_COLORNONE . . . . .	545
9.101.2.7 IN_TYPE_COLORRED . . . . .	545
9.101.2.8 IN_TYPE_CUSTOM . . . . .	545
9.101.2.9 IN_TYPE_HISPEED . . . . .	545
9.101.2.10 IN_TYPE_LIGHT_ACTIVE . . . . .	545
9.101.2.11 IN_TYPE_LIGHT_INACTIVE . . . . .	545
9.101.2.12 IN_TYPE_LOWSPEED . . . . .	545
9.101.2.13 IN_TYPE_LOWSPEED_9V . . . . .	545
9.101.2.14 IN_TYPE_NO_SENSOR . . . . .	545
9.101.2.15 IN_TYPE_REFLECTION . . . . .	546
9.101.2.16 IN_TYPE_SOUND_DB . . . . .	546
9.101.2.17 IN_TYPE_SOUND_DBA . . . . .	546
9.101.2.18 IN_TYPE_SWITCH . . . . .	546
9.101.2.19 IN_TYPE_TEMPERATURE . . . . .	546
9.102NBC sensor mode constants . . . . .	547
9.102.1 Detailed Description . . . . .	547



9.102.2 Define Documentation . . . . .	547
9.102.2.1 IN_MODE_ANGLESTEP . . . . .	547
9.102.2.2 IN_MODE_BOOLEAN . . . . .	547
9.102.2.3 IN_MODE_CELSIUS . . . . .	547
9.102.2.4 IN_MODE_FAHRENHEIT . . . . .	547
9.102.2.5 IN_MODE_MODEMASK . . . . .	547
9.102.2.6 IN_MODE_PCTFULLSCALE . . . . .	548
9.102.2.7 IN_MODE_PERIODCOUNTER . . . . .	548
9.102.2.8 IN_MODE_RAW . . . . .	548
9.102.2.9 IN_MODE_SLOPEMASK . . . . .	548
9.102.2.10 IN_MODE_TRANSITIONCNT . . . . .	548
9.103 Input field constants . . . . .	549
9.103.1 Detailed Description . . . . .	549
9.103.2 Define Documentation . . . . .	549
9.103.2.1 InputMode . . . . .	549
9.103.2.2 InvalidData . . . . .	549
9.103.2.3 NormalizedValue . . . . .	549
9.103.2.4 RawValue . . . . .	549
9.103.2.5 ScaledValue . . . . .	549
9.103.2.6 Type . . . . .	549
9.104 Color sensor array indices . . . . .	550
9.104.1 Detailed Description . . . . .	550
9.104.2 Define Documentation . . . . .	550
9.104.2.1 INPUT_BLANK . . . . .	550
9.104.2.2 INPUT_BLUE . . . . .	550
9.104.2.3 INPUT_GREEN . . . . .	550
9.104.2.4 INPUT_NO_OF_COLORS . . . . .	550
9.104.2.5 INPUT_RED . . . . .	550
9.105 Color values . . . . .	551
9.105.1 Detailed Description . . . . .	551
9.105.2 Define Documentation . . . . .	551
9.105.2.1 INPUT_BLACKCOLOR . . . . .	551
9.105.2.2 INPUT_BLUECOLOR . . . . .	551
9.105.2.3 INPUT_GREENCOLOR . . . . .	551
9.105.2.4 INPUT_REDCOLOR . . . . .	551
9.105.2.5 INPUT_WHITECOLOR . . . . .	551

9.105.2.6 INPUT_YELLOWCOLOR . . . . .	551
9.106Color calibration state constants . . . . .	552
9.106.1 Detailed Description . . . . .	552
9.106.2 Define Documentation . . . . .	552
9.106.2.1 INPUT_RESETCAL . . . . .	552
9.106.2.2 INPUT_RUNNINGCAL . . . . .	552
9.106.2.3 INPUT_SENSORCAL . . . . .	552
9.106.2.4 INPUT_SENSOROFF . . . . .	552
9.106.2.5 INPUT_STARTCAL . . . . .	552
9.107Color calibration constants . . . . .	553
9.107.1 Detailed Description . . . . .	553
9.107.2 Define Documentation . . . . .	553
9.107.2.1 INPUT_CAL_POINT_0 . . . . .	553
9.107.2.2 INPUT_CAL_POINT_1 . . . . .	553
9.107.2.3 INPUT_CAL_POINT_2 . . . . .	553
9.107.2.4 INPUT_NO_OF_POINTS . . . . .	553
9.108Input module IOMAP offsets . . . . .	554
9.108.1 Detailed Description . . . . .	554
9.108.2 Define Documentation . . . . .	554
9.108.2.1 InputOffsetADRaw . . . . .	554
9.108.2.2 InputOffsetColorADRaw . . . . .	554
9.108.2.3 InputOffsetColorCalibration . . . . .	554
9.108.2.4 InputOffsetColorCalibrationState . . . . .	554
9.108.2.5 InputOffsetColorCalLimits . . . . .	555
9.108.2.6 InputOffsetColorSensorBoolean . . . . .	555
9.108.2.7 InputOffsetColorSensorRaw . . . . .	555
9.108.2.8 InputOffsetColorSensorValue . . . . .	555
9.108.2.9 InputOffsetCustomActiveStatus . . . . .	555
9.108.2.10InputOffsetCustomPctFullScale . . . . .	555
9.108.2.11InputOffsetCustomZeroOffset . . . . .	555
9.108.2.12InputOffsetDigiPinsDir . . . . .	555
9.108.2.13InputOffsetDigiPinsIn . . . . .	555
9.108.2.14InputOffsetDigiPinsOut . . . . .	555
9.108.2.15InputOffsetInvalidData . . . . .	555
9.108.2.16InputOffsetSensorBoolean . . . . .	556
9.108.2.17InputOffsetSensorMode . . . . .	556

9.108.2.18	InputOffsetSensorRaw	556
9.108.2.19	InputOffsetSensorType	556
9.108.2.20	InputOffsetSensorValue	556
9.109	Output port constants	557
9.109.1	Detailed Description	557
9.109.2	Define Documentation	557
9.109.2.1	OUT_A	557
9.109.2.2	OUT_AB	557
9.109.2.3	OUT_ABC	557
9.109.2.4	OUT_AC	558
9.109.2.5	OUT_B	558
9.109.2.6	OUT_BC	558
9.109.2.7	OUT_C	558
9.110	PID constants	559
9.110.1	Detailed Description	559
9.110.2	Define Documentation	559
9.110.2.1	PID_0	559
9.110.2.2	PID_1	559
9.110.2.3	PID_2	559
9.110.2.4	PID_3	559
9.110.2.5	PID_4	559
9.110.2.6	PID_5	560
9.110.2.7	PID_6	560
9.110.2.8	PID_7	560
9.111	Output port update flag constants	561
9.111.1	Detailed Description	561
9.111.2	Define Documentation	561
9.111.2.1	UF_PENDING_UPDATES	561
9.111.2.2	UF_UPDATE_MODE	561
9.111.2.3	UF_UPDATE_PID_VALUES	561
9.111.2.4	UF_UPDATE_RESET_BLOCK_COUNT	561
9.111.2.5	UF_UPDATE_RESET_COUNT	561
9.111.2.6	UF_UPDATE_RESET_ROTATION_COUNT	562
9.111.2.7	UF_UPDATE_SPEED	562
9.111.2.8	UF_UPDATE_TACHO_LIMIT	562
9.112	Tachometer counter reset flags	563

9.112.1 Detailed Description . . . . .	563
9.112.2 Define Documentation . . . . .	563
9.112.2.1 RESET_ALL . . . . .	563
9.112.2.2 RESET_BLOCK_COUNT . . . . .	563
9.112.2.3 RESET_BLOCKANDTACHO . . . . .	563
9.112.2.4 RESET_COUNT . . . . .	563
9.112.2.5 RESET_NONE . . . . .	563
9.112.2.6 RESET_ROTATION_COUNT . . . . .	564
9.113 Output port mode constants . . . . .	565
9.113.1 Detailed Description . . . . .	565
9.113.2 Define Documentation . . . . .	565
9.113.2.1 OUT_MODE_BRAKE . . . . .	565
9.113.2.2 OUT_MODE_COAST . . . . .	565
9.113.2.3 OUT_MODE_MOTORON . . . . .	565
9.113.2.4 OUT_MODE_REGMETHOD . . . . .	565
9.113.2.5 OUT_MODE_REGULATED . . . . .	565
9.114 Output port option constants . . . . .	566
9.114.1 Detailed Description . . . . .	566
9.114.2 Define Documentation . . . . .	566
9.114.2.1 OUT_OPTION_HOLDATLIMIT . . . . .	566
9.114.2.2 OUT_OPTION_RAMPDOWNTOLIMIT . . . . .	566
9.115 Output port run state constants . . . . .	567
9.115.1 Detailed Description . . . . .	567
9.115.2 Define Documentation . . . . .	567
9.115.2.1 OUT_RUNSTATE_HOLD . . . . .	567
9.115.2.2 OUT_RUNSTATE_IDLE . . . . .	567
9.115.2.3 OUT_RUNSTATE_RAMPDOWN . . . . .	567
9.115.2.4 OUT_RUNSTATE_RAMPUP . . . . .	567
9.115.2.5 OUT_RUNSTATE_RUNNING . . . . .	567
9.116 Output port regulation mode constants . . . . .	568
9.116.1 Detailed Description . . . . .	568
9.116.2 Define Documentation . . . . .	568
9.116.2.1 OUT_REGMODE_IDLE . . . . .	568
9.116.2.2 OUT_REGMODE_SPEED . . . . .	568
9.116.2.3 OUT_REGMODE_SYNC . . . . .	568
9.117 Output field constants . . . . .	569

9.117.1 Detailed Description	570
9.117.2 Define Documentation	570
9.117.2.1 ActualSpeed	570
9.117.2.2 BlockTachoCount	570
9.117.2.3 OutputMode	570
9.117.2.4 OutputOptions	570
9.117.2.5 Overload	571
9.117.2.6 Power	571
9.117.2.7 RegDValue	571
9.117.2.8 RegIValue	571
9.117.2.9 RegMode	571
9.117.2.10RegPValue	571
9.117.2.11RotationCount	572
9.117.2.12RunState	572
9.117.2.13TachoCount	572
9.117.2.14TachoLimit	572
9.117.2.15TurnRatio	572
9.117.2.16UpdateFlags	573
9.118Output module IOMAP offsets	574
9.118.1 Detailed Description	574
9.118.2 Define Documentation	574
9.118.2.1 OutputOffsetActualSpeed	574
9.118.2.2 OutputOffsetBlockTachoCount	574
9.118.2.3 OutputOffsetFlags	574
9.118.2.4 OutputOffsetMode	574
9.118.2.5 OutputOffsetMotorRPM	575
9.118.2.6 OutputOffsetOptions	575
9.118.2.7 OutputOffsetOverloaded	575
9.118.2.8 OutputOffsetPwnFreq	575
9.118.2.9 OutputOffsetRegDParameter	575
9.118.2.10OutputOffsetRegIPParameter	575
9.118.2.11OutputOffsetRegMode	575
9.118.2.12OutputOffsetRegPPParameter	575
9.118.2.13OutputOffsetRotationCount	575
9.118.2.14OutputOffsetRunState	575
9.118.2.15OutputOffsetSpeed	575

9.118.2.16	OutputOffsetSyncTurnParameter	576
9.118.2.17	OutputOffsetTachoCount	576
9.118.2.18	OutputOffsetTachoLimit	576
9.119	LowSpeed module constants	577
9.119.1	Detailed Description	577
9.120	LSSState constants	578
9.120.1	Detailed Description	578
9.120.2	Define Documentation	578
9.120.2.1	COM_CHANNEL_FOUR_ACTIVE	578
9.120.2.2	COM_CHANNEL_NONE_ACTIVE	578
9.120.2.3	COM_CHANNEL_ONE_ACTIVE	578
9.120.2.4	COM_CHANNEL_THREE_ACTIVE	578
9.120.2.5	COM_CHANNEL_TWO_ACTIVE	578
9.121	LSChannelState constants	579
9.121.1	Detailed Description	579
9.121.2	Define Documentation	579
9.121.2.1	LOWSPEED_COMMUNICATING	579
9.121.2.2	LOWSPEED_DONE	579
9.121.2.3	LOWSPEED_ERROR	579
9.121.2.4	LOWSPEED_IDLE	579
9.121.2.5	LOWSPEED_INIT	579
9.121.2.6	LOWSPEED_LOAD_BUFFER	580
9.122	LSMode constants	581
9.122.1	Detailed Description	581
9.122.2	Define Documentation	581
9.122.2.1	LOWSPEED_DATA_RECEIVED	581
9.122.2.2	LOWSPEED_RECEIVING	581
9.122.2.3	LOWSPEED_TRANSMITTING	581
9.123	LSErrorType constants	582
9.123.1	Detailed Description	582
9.123.2	Define Documentation	582
9.123.2.1	LOWSPEED_CH_NOT_READY	582
9.123.2.2	LOWSPEED_NO_ERROR	582
9.123.2.3	LOWSPEED_RX_ERROR	582
9.123.2.4	LOWSPEED_TX_ERROR	582
9.124	Low speed module IOMAP offsets	583

9.124.1 Detailed Description	583
9.124.2 Define Documentation	583
9.124.2.1 LowSpeedOffsetChannelState	583
9.124.2.2 LowSpeedOffsetErrorType	583
9.124.2.3 LowSpeedOffsetInBufBuf	583
9.124.2.4 LowSpeedOffsetInBufBytesToRx	583
9.124.2.5 LowSpeedOffsetInBufInPtr	583
9.124.2.6 LowSpeedOffsetInBufOutPtr	584
9.124.2.7 LowSpeedOffsetMode	584
9.124.2.8 LowSpeedOffsetNoRestartOnRead	584
9.124.2.9 LowSpeedOffsetOutBufBuf	584
9.124.2.10 LowSpeedOffsetOutBufBytesToRx	584
9.124.2.11 LowSpeedOffsetOutBufInPtr	584
9.124.2.12 LowSpeedOffsetOutBufOutPtr	584
9.124.2.13 LowSpeedOffsetSpeed	584
9.124.2.14 LowSpeedOffsetState	584
9.125 LSNNoRestartOnRead constants	585
9.125.1 Detailed Description	585
9.125.2 Define Documentation	585
9.125.2.1 LSREAD_NO_RESTART_1	585
9.125.2.2 LSREAD_NO_RESTART_2	585
9.125.2.3 LSREAD_NO_RESTART_3	585
9.125.2.4 LSREAD_NO_RESTART_4	585
9.125.2.5 LSREAD_NO_RESTART_MASK	585
9.125.2.6 LSREAD_RESTART_ALL	585
9.125.2.7 LSREAD_RESTART_NONE	586
9.126 Standard I2C constants	587
9.126.1 Detailed Description	587
9.126.2 Define Documentation	587
9.126.2.1 I2C_REG_CMD	587
9.126.2.2 I2C_REG_DEVICE_ID	587
9.126.2.3 I2C_REG_VENDOR_ID	587
9.126.2.4 I2C_REG_VERSION	587
9.127 Ultrasonic sensor constants	588
9.127.1 Detailed Description	588
9.127.2 Define Documentation	588

9.127.2.1 US_CMD_CONTINUOUS	588
9.127.2.2 US_CMD_EVENTCAPTURE	588
9.127.2.3 US_CMD_OFF	588
9.127.2.4 US_CMD_SINGLESOT	588
9.127.2.5 US_CMD_WARMRESET	589
9.127.2.6 US_REG_ACTUAL_ZERO	589
9.127.2.7 US_REG_CM_INTERVAL	589
9.127.2.8 US_REG_FACTORY_ACTUAL_ZERO	589
9.127.2.9 US_REG_FACTORY_SCALE_DIVISOR	589
9.127.2.10 US_REG_FACTORY_SCALE_FACTOR	589
9.127.2.11 US_REG_MEASUREMENT_UNITS	589
9.127.2.12 US_REG_SCALE_DIVISOR	589
9.127.2.13 US_REG_SCALE_FACTOR	589
9.128 Temperature sensor constants	590
9.128.1 Detailed Description	590
9.128.2 Define Documentation	590
9.128.2.1 TEMP_FQ_1	590
9.128.2.2 TEMP_FQ_2	590
9.128.2.3 TEMP_FQ_4	590
9.128.2.4 TEMP_FQ_6	590
9.128.2.5 TEMP_I2C_ADDRESS	591
9.128.2.6 TEMP_OS_ONESHOT	591
9.128.2.7 TEMP_POL_HIGH	591
9.128.2.8 TEMP_POL_LOW	591
9.128.2.9 TEMP_REG_CONFIG	591
9.128.2.10 TEMP_REG_TEMP	591
9.128.2.11 TEMP_REG_THIGH	591
9.128.2.12 TEMP_REG_TLOW	591
9.128.2.13 TEMP_RES_10BIT	591
9.128.2.14 TEMP_RES_11BIT	591
9.128.2.15 TEMP_RES_12BIT	591
9.128.2.16 TEMP_RES_9BIT	592
9.128.2.17 TEMP_SD_CONTINUOUS	592
9.128.2.18 TEMP_SD_SHUTDOWN	592
9.128.2.19 TEMP_TM_COMPARATOR	592
9.128.2.20 TEMP_TM_INTERRUPT	592



9.129 Display module constants . . . . .	593
9.129.1 Detailed Description . . . . .	594
9.129.2 Define Documentation . . . . .	594
9.129.2.1 BITMAP_1 . . . . .	594
9.129.2.2 BITMAP_2 . . . . .	594
9.129.2.3 BITMAP_3 . . . . .	594
9.129.2.4 BITMAP_4 . . . . .	594
9.129.2.5 BITMAPS . . . . .	594
9.129.2.6 DISPLAY_HEIGHT . . . . .	595
9.129.2.7 DISPLAY_MENUICONS_X_DIFF . . . . .	595
9.129.2.8 DISPLAY_MENUICONS_X_OFFS . . . . .	595
9.129.2.9 DISPLAY_MENUICONS_Y . . . . .	595
9.129.2.10 DISPLAY_WIDTH . . . . .	595
9.129.2.11 IFRAME_SELECT . . . . .	595
9.129.2.12 MENUICON_CENTER . . . . .	595
9.129.2.13 MENUICON_LEFT . . . . .	595
9.129.2.14 MENUICON_RIGHT . . . . .	595
9.129.2.15 MENUICONS . . . . .	595
9.129.2.16 MENUTEXT . . . . .	595
9.129.2.17 SCREEN_BACKGROUND . . . . .	596
9.129.2.18 SCREEN_LARGE . . . . .	596
9.129.2.19 SCREEN_MODE_CLEAR . . . . .	596
9.129.2.20 SCREEN_MODE_RESTORE . . . . .	596
9.129.2.21 SCREEN_SMALL . . . . .	596
9.129.2.22 SCREENS . . . . .	596
9.129.2.23 SPECIALS . . . . .	596
9.129.2.24 STATUSICON_BATTERY . . . . .	596
9.129.2.25 STATUSICON_BLUETOOTH . . . . .	596
9.129.2.26 STATUSICON_USB . . . . .	596
9.129.2.27 STATUSICON_VM . . . . .	597
9.129.2.28 STATUSICONS . . . . .	597
9.129.2.29 STATUSTEXT . . . . .	597
9.129.2.30 STEPICON_1 . . . . .	597
9.129.2.31 STEPICON_2 . . . . .	597
9.129.2.32 STEPICON_3 . . . . .	597
9.129.2.33 STEPICON_4 . . . . .	597

9.129.2.34STEPICON_5 . . . . .	597
9.129.2.35STEPICONS . . . . .	597
9.129.2.36STEPLINE . . . . .	597
9.129.2.37TOPLINE . . . . .	597
9.130DisplayExecuteFunction constants . . . . .	598
9.130.1 Detailed Description . . . . .	598
9.130.2 Define Documentation . . . . .	598
9.130.2.1 DISPLAY_CHAR . . . . .	598
9.130.2.2 DISPLAY_ERASE_ALL . . . . .	598
9.130.2.3 DISPLAY_ERASE_LINE . . . . .	598
9.130.2.4 DISPLAY_FILL_REGION . . . . .	598
9.130.2.5 DISPLAY_FRAME . . . . .	598
9.130.2.6 DISPLAY_HORIZONTAL_LINE . . . . .	599
9.130.2.7 DISPLAY_PIXEL . . . . .	599
9.130.2.8 DISPLAY_VERTICAL_LINE . . . . .	599
9.131Drawing option constants . . . . .	600
9.131.1 Detailed Description . . . . .	600
9.131.2 Define Documentation . . . . .	600
9.131.2.1 DRAW_OPT_CLEAR . . . . .	600
9.131.2.2 DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN . . . . .	601
9.131.2.3 DRAW_OPT_CLEAR_PIXELS . . . . .	601
9.131.2.4 DRAW_OPT_CLEAR_SCREEN_MODES . . . . .	601
9.131.2.5 DRAW_OPT_CLEAR_WHOLE_SCREEN . . . . .	601
9.131.2.6 DRAW_OPT_FILL_SHAPE . . . . .	601
9.131.2.7 DRAW_OPT_INVERT . . . . .	601
9.131.2.8 DRAW_OPT_LOGICAL_AND . . . . .	601
9.131.2.9 DRAW_OPT_LOGICAL_COPY . . . . .	601
9.131.2.10DRAW_OPT_LOGICAL_OPERATIONS . . . . .	602
9.131.2.11DRAW_OPT_LOGICAL_OR . . . . .	602
9.131.2.12DRAW_OPT_LOGICAL_XOR . . . . .	602
9.131.2.13DRAW_OPT_NORMAL . . . . .	602
9.131.2.14DRAW_OPT_POLYGON_POLYLINE . . . . .	602
9.132Font drawing option constants . . . . .	603
9.132.1 Detailed Description . . . . .	603
9.132.2 Define Documentation . . . . .	603
9.132.2.1 DRAW_OPT_FONT_DIR_B2TL . . . . .	603

9.132.2.2 DRAW_OPT_FONT_DIR_B2TR . . . . .	603
9.132.2.3 DRAW_OPT_FONT_DIR_L2RB . . . . .	603
9.132.2.4 DRAW_OPT_FONT_DIR_L2RT . . . . .	604
9.132.2.5 DRAW_OPT_FONT_DIR_R2LB . . . . .	604
9.132.2.6 DRAW_OPT_FONT_DIR_R2LT . . . . .	604
9.132.2.7 DRAW_OPT_FONT_DIR_T2BL . . . . .	604
9.132.2.8 DRAW_OPT_FONT_DIR_T2BR . . . . .	604
9.132.2.9 DRAW_OPT_FONT_DIRECTIONS . . . . .	604
9.132.2.10 DRAW_OPT_FONT_WRAP . . . . .	604
9.133 Display flags . . . . .	605
9.133.1 Detailed Description . . . . .	605
9.133.2 Define Documentation . . . . .	605
9.133.2.1 DISPLAY_BUSY . . . . .	605
9.133.2.2 DISPLAY_ON . . . . .	605
9.133.2.3 DISPLAY_POPUP . . . . .	605
9.133.2.4 DISPLAY_REFRESH . . . . .	605
9.133.2.5 DISPLAY_REFRESH_DISABLED . . . . .	605
9.134 Display contrast constants . . . . .	606
9.134.1 Detailed Description . . . . .	606
9.134.2 Define Documentation . . . . .	606
9.134.2.1 DISPLAY_CONTRAST_DEFAULT . . . . .	606
9.134.2.2 DISPLAY_CONTRAST_MAX . . . . .	606
9.135 Text line constants . . . . .	607
9.135.1 Detailed Description . . . . .	607
9.135.2 Define Documentation . . . . .	607
9.135.2.1 TEXTLINE_1 . . . . .	607
9.135.2.2 TEXTLINE_2 . . . . .	607
9.135.2.3 TEXTLINE_3 . . . . .	607
9.135.2.4 TEXTLINE_4 . . . . .	607
9.135.2.5 TEXTLINE_5 . . . . .	608
9.135.2.6 TEXTLINE_6 . . . . .	608
9.135.2.7 TEXTLINE_7 . . . . .	608
9.135.2.8 TEXTLINE_8 . . . . .	608
9.135.2.9 TEXTLINES . . . . .	608
9.136 Display module IOMAP offsets . . . . .	609
9.136.1 Detailed Description . . . . .	609

9.136.2 Define Documentation	609
9.136.2.1 DisplayOffsetContrast	609
9.136.2.2 DisplayOffsetDisplay	609
9.136.2.3 DisplayOffsetEraseMask	609
9.136.2.4 DisplayOffsetFlags	609
9.136.2.5 DisplayOffsetNormal	610
9.136.2.6 DisplayOffsetPBitmaps	610
9.136.2.7 DisplayOffsetPFont	610
9.136.2.8 DisplayOffsetPFunc	610
9.136.2.9 DisplayOffsetPMenuIcons	610
9.136.2.10 DisplayOffsetPMenuText	610
9.136.2.11 DisplayOffsetPopup	610
9.136.2.12 DisplayOffsetPScreens	610
9.136.2.13 DisplayOffsetPStatusIcons	610
9.136.2.14 DisplayOffsetPStatusText	610
9.136.2.15 DisplayOffsetPStepIcons	610
9.136.2.16 DisplayOffsetPTextLines	611
9.136.2.17 DisplayOffsetStatusIcons	611
9.136.2.18 DisplayOffsetStepIcons	611
9.136.2.19 DisplayOffsetTextLinesCenterFlags	611
9.136.2.20 DisplayOffsetUpdateMask	611
9.137 Comm module constants	612
9.137.1 Detailed Description	612
9.138 Miscellaneous Comm module constants	613
9.138.1 Detailed Description	613
9.138.2 Define Documentation	613
9.138.2.1 BT_CMD_BYTE	613
9.138.2.2 BT_DEFAULT_INQUIRY_MAX	613
9.138.2.3 BT_DEFAULT_INQUIRY_TIMEOUT_LO	613
9.138.2.4 MAX_BT_MSG_SIZE	613
9.138.2.5 SIZE_OF_BDADDR	613
9.138.2.6 SIZE_OF_BRICK_NAME	614
9.138.2.7 SIZE_OF_BT_CONNECT_TABLE	614
9.138.2.8 SIZE_OF_BT_DEVICE_TABLE	614
9.138.2.9 SIZE_OF_BT_NAME	614
9.138.2.10 SIZE_OF_BT_PINCODE	614

9.138.2.11	SIZE_OF_BTBUF	614
9.138.2.12	SIZE_OF_CLASS_OF_DEVICE	614
9.138.2.13	SIZE_OF_HSBUF	614
9.138.2.14	SIZE_OF_USBBUF	614
9.138.2.15	SIZE_OF_USBDATA	614
9.138.2.16	USB_PROTOCOL_OVERHEAD	614
9.139	Bluetooth State constants	615
9.139.1	Detailed Description	615
9.139.2	Define Documentation	615
9.139.2.1	BT_ARM_CMD_MODE	615
9.139.2.2	BT_ARM_DATA_MODE	615
9.139.2.3	BT_ARM_OFF	615
9.140	Bluetooth state status constants	616
9.140.1	Detailed Description	616
9.140.2	Define Documentation	616
9.140.2.1	BT_BRICK_PORT_OPEN	616
9.140.2.2	BT_BRICK_VISIBILITY	616
9.140.2.3	BT_CONNECTION_0_ENABLE	616
9.140.2.4	BT_CONNECTION_1_ENABLE	616
9.140.2.5	BT_CONNECTION_2_ENABLE	616
9.140.2.6	BT_CONNECTION_3_ENABLE	616
9.141	Bluetooth hardware status constants	617
9.141.1	Detailed Description	617
9.141.2	Define Documentation	617
9.141.2.1	BT_DISABLE	617
9.141.2.2	BT_ENABLE	617
9.142	Hi-speed port constants	618
9.142.1	Detailed Description	618
9.143	Hi-speed port flags constants	619
9.143.1	Detailed Description	619
9.143.2	Define Documentation	619
9.143.2.1	HS_UPDATE	619
9.144	Hi-speed port state constants	620
9.144.1	Detailed Description	620
9.144.2	Define Documentation	620
9.144.2.1	HS_DISABLE	620

9.144.2.2 HS_ENABLE . . . . .	620
9.144.2.3 HS_INIT_RECEIVER . . . . .	620
9.144.2.4 HS_INITIALISE . . . . .	620
9.144.2.5 HS_SEND_DATA . . . . .	620
9.145Hi-speed port SysCommHSControl constants . . . . .	621
9.145.1 Detailed Description . . . . .	621
9.145.2 Define Documentation . . . . .	621
9.145.2.1 HS_CTRL_EXIT . . . . .	621
9.145.2.2 HS_CTRL_INIT . . . . .	621
9.145.2.3 HS_CTRL_UART . . . . .	621
9.146Hi-speed port baud rate constants . . . . .	622
9.146.1 Detailed Description . . . . .	622
9.146.2 Define Documentation . . . . .	622
9.146.2.1 HS_BAUD_115200 . . . . .	622
9.146.2.2 HS_BAUD_1200 . . . . .	622
9.146.2.3 HS_BAUD_14400 . . . . .	622
9.146.2.4 HS_BAUD_19200 . . . . .	622
9.146.2.5 HS_BAUD_230400 . . . . .	623
9.146.2.6 HS_BAUD_2400 . . . . .	623
9.146.2.7 HS_BAUD_28800 . . . . .	623
9.146.2.8 HS_BAUD_3600 . . . . .	623
9.146.2.9 HS_BAUD_38400 . . . . .	623
9.146.2.10HS_BAUD_460800 . . . . .	623
9.146.2.11HS_BAUD_4800 . . . . .	623
9.146.2.12HS_BAUD_57600 . . . . .	623
9.146.2.13HS_BAUD_7200 . . . . .	623
9.146.2.14HS_BAUD_76800 . . . . .	623
9.146.2.15HS_BAUD_921600 . . . . .	623
9.146.2.16HS_BAUD_9600 . . . . .	624
9.147Hi-speed port data bits constants . . . . .	625
9.147.1 Detailed Description . . . . .	625
9.147.2 Define Documentation . . . . .	625
9.147.2.1 HS_MODE_5_DATA . . . . .	625
9.147.2.2 HS_MODE_6_DATA . . . . .	625
9.147.2.3 HS_MODE_7_DATA . . . . .	625
9.147.2.4 HS_MODE_8_DATA . . . . .	625

9.148	Hi-speed port stop bits constants	626
9.148.1	Detailed Description	626
9.148.2	Define Documentation	626
9.148.2.1	HS_MODE_10_STOP	626
9.148.2.2	HS_MODE_15_STOP	626
9.148.2.3	HS_MODE_20_STOP	626
9.149	Hi-speed port parity constants	627
9.149.1	Detailed Description	627
9.149.2	Define Documentation	627
9.149.2.1	HS_MODE_E_PARITY	627
9.149.2.2	HS_MODE_M_PARITY	627
9.149.2.3	HS_MODE_N_PARITY	627
9.149.2.4	HS_MODE_O_PARITY	627
9.149.2.5	HS_MODE_S_PARITY	627
9.150	Hi-speed port combined UART constants	628
9.150.1	Detailed Description	628
9.150.2	Define Documentation	628
9.150.2.1	HS_MODE_7E1	628
9.150.2.2	HS_MODE_8N1	628
9.151	Device status constants	629
9.151.1	Detailed Description	629
9.151.2	Define Documentation	629
9.151.2.1	BT_DEVICE_AWAY	629
9.151.2.2	BT_DEVICE_EMPTY	629
9.151.2.3	BT_DEVICE_KNOWN	629
9.151.2.4	BT_DEVICE_NAME	629
9.151.2.5	BT_DEVICE_UNKNOWN	629
9.152	Comm module interface function constants	630
9.152.1	Detailed Description	630
9.152.2	Define Documentation	630
9.152.2.1	INTF_BTOFF	630
9.152.2.2	INTF_BTON	630
9.152.2.3	INTF_CONNECT	631
9.152.2.4	INTF_CONNECTBYNAME	631
9.152.2.5	INTF_CONNECTREQ	631
9.152.2.6	INTF_DISCONNECT	631

9.152.2.7	INTF_DISCONNECTALL	631
9.152.2.8	INTF_EXTREAD	631
9.152.2.9	INTF_FACTORYRESET	631
9.152.2.10	INTF_OPENSTREAM	631
9.152.2.11	INTF_PINREQ	631
9.152.2.12	INTF_REMOVEDEVICE	631
9.152.2.13	INTF_SEARCH	631
9.152.2.14	INTF_SENDDATA	632
9.152.2.15	INTF_SENDFILE	632
9.152.2.16	INTF_SETBTNAME	632
9.152.2.17	INTF_SETCMDMODE	632
9.152.2.18	INTF_STOPSEARCH	632
9.152.2.19	INTF_VISIBILITY	632
9.153	Comm module status code constants	633
9.153.1	Detailed Description	633
9.153.2	Define Documentation	633
9.153.2.1	BT_CMD_READY	633
9.153.2.2	HS_CMD_READY	633
9.153.2.3	LR_COULD_NOT_SAVE	633
9.153.2.4	LR_ENTRY_REMOVED	633
9.153.2.5	LR_STORE_IS_FULL	633
9.153.2.6	LR_SUCCESS	633
9.153.2.7	LR_UNKNOWN_ADDR	634
9.153.2.8	USB_CMD_READY	634
9.154	Comm module IOMAP offsets	635
9.154.1	Detailed Description	636
9.154.2	Define Documentation	636
9.154.2.1	CommOffsetBrickDataBdAddr	636
9.154.2.2	CommOffsetBrickDataBluecoreVersion	636
9.154.2.3	CommOffsetBrickDataBtHwStatus	636
9.154.2.4	CommOffsetBrickDataBtStateStatus	636
9.154.2.5	CommOffsetBrickDataName	636
9.154.2.6	CommOffsetBrickDataTimeOutValue	636
9.154.2.7	CommOffsetBtConnectTableBdAddr	636
9.154.2.8	CommOffsetBtConnectTableClassOfDevice	636
9.154.2.9	CommOffsetBtConnectTableHandleNr	636



9.154.2.10CommOffsetBtConnectTableLinkQuality . . . . .	636
9.154.2.11CommOffsetBtConnectTableName . . . . .	637
9.154.2.12CommOffsetBtConnectTablePinCode . . . . .	637
9.154.2.13CommOffsetBtConnectTableStreamStatus . . . . .	637
9.154.2.14CommOffsetBtDeviceCnt . . . . .	637
9.154.2.15CommOffsetBtDeviceNameCnt . . . . .	637
9.154.2.16CommOffsetBtDeviceTableBdAddr . . . . .	637
9.154.2.17CommOffsetBtDeviceTableClassOfDevice . . . . .	637
9.154.2.18CommOffsetBtDeviceTableDeviceStatus . . . . .	637
9.154.2.19CommOffsetBtDeviceTableName . . . . .	637
9.154.2.20CommOffsetBtInBufBuf . . . . .	637
9.154.2.21CommOffsetBtInBufInPtr . . . . .	637
9.154.2.22CommOffsetBtInBufOutPtr . . . . .	638
9.154.2.23CommOffsetBtOutBufBuf . . . . .	638
9.154.2.24CommOffsetBtOutBufInPtr . . . . .	638
9.154.2.25CommOffsetBtOutBufOutPtr . . . . .	638
9.154.2.26CommOffsetHsFlags . . . . .	638
9.154.2.27CommOffsetHsInBufBuf . . . . .	638
9.154.2.28CommOffsetHsInBufInPtr . . . . .	638
9.154.2.29CommOffsetHsInBufOutPtr . . . . .	638
9.154.2.30CommOffsetHsMode . . . . .	638
9.154.2.31CommOffsetHsOutBufBuf . . . . .	638
9.154.2.32CommOffsetHsOutBufInPtr . . . . .	638
9.154.2.33CommOffsetHsOutBufOutPtr . . . . .	639
9.154.2.34CommOffsetHsSpeed . . . . .	639
9.154.2.35CommOffsetHsState . . . . .	639
9.154.2.36CommOffsetPFunc . . . . .	639
9.154.2.37CommOffsetPFuncTwo . . . . .	639
9.154.2.38CommOffsetUsbInBufBuf . . . . .	639
9.154.2.39CommOffsetUsbInBufInPtr . . . . .	639
9.154.2.40CommOffsetUsbInBufOutPtr . . . . .	639
9.154.2.41CommOffsetUsbOutBufBuf . . . . .	639
9.154.2.42CommOffsetUsbOutBufInPtr . . . . .	639
9.154.2.43CommOffsetUsbOutBufOutPtr . . . . .	639
9.154.2.44CommOffsetUsbPollBufBuf . . . . .	640
9.154.2.45CommOffsetUsbPollBufInPtr . . . . .	640

9.154.2.46CommOffsetUsbPollBufOutPtr . . . . .	640
9.154.2.47CommOffsetUsbState . . . . .	640
9.155RCX constants . . . . .	641
9.155.1 Detailed Description . . . . .	641
9.156RCX output constants . . . . .	642
9.156.1 Detailed Description . . . . .	642
9.156.2 Define Documentation . . . . .	642
9.156.2.1 RCX_OUT_A . . . . .	642
9.156.2.2 RCX_OUT_AB . . . . .	642
9.156.2.3 RCX_OUT_ABC . . . . .	642
9.156.2.4 RCX_OUT_AC . . . . .	643
9.156.2.5 RCX_OUT_B . . . . .	643
9.156.2.6 RCX_OUT_BC . . . . .	643
9.156.2.7 RCX_OUT_C . . . . .	643
9.157RCX output mode constants . . . . .	644
9.157.1 Detailed Description . . . . .	644
9.157.2 Define Documentation . . . . .	644
9.157.2.1 RCX_OUT_FLOAT . . . . .	644
9.157.2.2 RCX_OUT_OFF . . . . .	644
9.157.2.3 RCX_OUT_ON . . . . .	644
9.158RCX output direction constants . . . . .	645
9.158.1 Detailed Description . . . . .	645
9.158.2 Define Documentation . . . . .	645
9.158.2.1 RCX_OUT_FWD . . . . .	645
9.158.2.2 RCX_OUT_REV . . . . .	645
9.158.2.3 RCX_OUT_TOGGLE . . . . .	645
9.159RCX output power constants . . . . .	646
9.159.1 Detailed Description . . . . .	646
9.159.2 Define Documentation . . . . .	646
9.159.2.1 RCX_OUT_FULL . . . . .	646
9.159.2.2 RCX_OUT_HALF . . . . .	646
9.159.2.3 RCX_OUT_LOW . . . . .	646
9.160RCX IR remote constants . . . . .	647
9.160.1 Detailed Description . . . . .	647
9.160.2 Define Documentation . . . . .	647
9.160.2.1 RCX_RemoteKeysReleased . . . . .	647

9.160.2.2 RCX_RemoteOutABackward . . . . .	647
9.160.2.3 RCX_RemoteOutAForward . . . . .	647
9.160.2.4 RCX_RemoteOutBBackward . . . . .	647
9.160.2.5 RCX_RemoteOutBForward . . . . .	648
9.160.2.6 RCX_RemoteOutCBackward . . . . .	648
9.160.2.7 RCX_RemoteOutCForward . . . . .	648
9.160.2.8 RCX_RemotePBMessage1 . . . . .	648
9.160.2.9 RCX_RemotePBMessage2 . . . . .	648
9.160.2.10 RCX_RemotePBMessage3 . . . . .	648
9.160.2.11 RCX_RemotePlayASound . . . . .	648
9.160.2.12 RCX_RemoteSelProgram1 . . . . .	648
9.160.2.13 RCX_RemoteSelProgram2 . . . . .	648
9.160.2.14 RCX_RemoteSelProgram3 . . . . .	648
9.160.2.15 RCX_RemoteSelProgram4 . . . . .	648
9.160.2.16 RCX_RemoteSelProgram5 . . . . .	649
9.160.2.17 RCX_RemoteStopOutOff . . . . .	649
9.161 RCX and Scout sound constants . . . . .	650
9.161.1 Detailed Description . . . . .	650
9.161.2 Define Documentation . . . . .	650
9.161.2.1 SOUND_CLICK . . . . .	650
9.161.2.2 SOUND_DOUBLE_BEEP . . . . .	650
9.161.2.3 SOUND_DOWN . . . . .	650
9.161.2.4 SOUND_FAST_UP . . . . .	650
9.161.2.5 SOUND_LOW_BEEP . . . . .	651
9.161.2.6 SOUND_UP . . . . .	651
9.162 Scout constants . . . . .	652
9.162.1 Detailed Description . . . . .	652
9.163 Scout light constants . . . . .	653
9.163.1 Detailed Description . . . . .	653
9.163.2 Define Documentation . . . . .	653
9.163.2.1 SCOUT_LIGHT_OFF . . . . .	653
9.163.2.2 SCOUT_LIGHT_ON . . . . .	653
9.164 Scout sound constants . . . . .	654
9.164.1 Detailed Description . . . . .	654
9.164.2 Define Documentation . . . . .	654
9.164.2.1 SCOUT_SOUND_1_BLINK . . . . .	654

9.164.2.2 SCOUT_SOUND_2_BLINK . . . . .	654
9.164.2.3 SCOUT_SOUND_COUNTER1 . . . . .	654
9.164.2.4 SCOUT_SOUND_COUNTER2 . . . . .	655
9.164.2.5 SCOUT_SOUND_ENTER_BRIGHT . . . . .	655
9.164.2.6 SCOUT_SOUND_ENTER_DARK . . . . .	655
9.164.2.7 SCOUT_SOUND_ENTER_NORMAL . . . . .	655
9.164.2.8 SCOUT_SOUND_ENTERSA . . . . .	655
9.164.2.9 SCOUT_SOUND_KEYERROR . . . . .	655
9.164.2.10 SCOUT_SOUND_MAIL_RECEIVED . . . . .	655
9.164.2.11 SCOUT_SOUND_NONE . . . . .	655
9.164.2.12 SCOUT_SOUND_REMOTE . . . . .	655
9.164.2.13 SCOUT_SOUND_SPECIAL1 . . . . .	655
9.164.2.14 SCOUT_SOUND_SPECIAL2 . . . . .	655
9.164.2.15 SCOUT_SOUND_SPECIAL3 . . . . .	656
9.164.2.16 SCOUT_SOUND_TIMER1 . . . . .	656
9.164.2.17 SCOUT_SOUND_TIMER2 . . . . .	656
9.164.2.18 SCOUT_SOUND_TIMER3 . . . . .	656
9.164.2.19 SCOUT_SOUND_TOUCH1_PRES . . . . .	656
9.164.2.20 SCOUT_SOUND_TOUCH1_REL . . . . .	656
9.164.2.21 SCOUT_SOUND_TOUCH2_PRES . . . . .	656
9.164.2.22 SCOUT_SOUND_TOUCH2_REL . . . . .	656
9.165 Scout sound set constants . . . . .	657
9.165.1 Detailed Description . . . . .	657
9.165.2 Define Documentation . . . . .	657
9.165.2.1 SCOUT_SNDSET_ALARM . . . . .	657
9.165.2.2 SCOUT_SNDSET_BASIC . . . . .	657
9.165.2.3 SCOUT_SNDSET_BUG . . . . .	657
9.165.2.4 SCOUT_SNDSET_NONE . . . . .	657
9.165.2.5 SCOUT_SNDSET_RANDOM . . . . .	657
9.165.2.6 SCOUT_SNDSET_SCIENCE . . . . .	657
9.166 Scout mode constants . . . . .	658
9.166.1 Detailed Description . . . . .	658
9.166.2 Define Documentation . . . . .	658
9.166.2.1 SCOUT_MODE_POWER . . . . .	658
9.166.2.2 SCOUT_MODE_STANDALONE . . . . .	658
9.167 Scout motion rule constants . . . . .	659

9.167.1 Detailed Description . . . . .	659
9.167.2 Define Documentation . . . . .	659
9.167.2.1 SCOUT_MR_CIRCLE_LEFT . . . . .	659
9.167.2.2 SCOUT_MR_CIRCLE_RIGHT . . . . .	659
9.167.2.3 SCOUT_MR_FORWARD . . . . .	659
9.167.2.4 SCOUT_MR_LOOP_A . . . . .	659
9.167.2.5 SCOUT_MR_LOOP_AB . . . . .	659
9.167.2.6 SCOUT_MR_LOOP_B . . . . .	659
9.167.2.7 SCOUT_MR_NO_MOTION . . . . .	660
9.167.2.8 SCOUT_MR_ZIGZAG . . . . .	660
9.168 Scout touch rule constants . . . . .	661
9.168.1 Detailed Description . . . . .	661
9.168.2 Define Documentation . . . . .	661
9.168.2.1 SCOUT_TR_AVOID . . . . .	661
9.168.2.2 SCOUT_TR_IGNORE . . . . .	661
9.168.2.3 SCOUT_TR_OFF_WHEN . . . . .	661
9.168.2.4 SCOUT_TR_REVERSE . . . . .	661
9.168.2.5 SCOUT_TR_WAIT_FOR . . . . .	661
9.169 Scout light rule constants . . . . .	662
9.169.1 Detailed Description . . . . .	662
9.169.2 Define Documentation . . . . .	662
9.169.2.1 SCOUT_LR_AVOID . . . . .	662
9.169.2.2 SCOUT_LR_IGNORE . . . . .	662
9.169.2.3 SCOUT_LR_OFF_WHEN . . . . .	662
9.169.2.4 SCOUT_LR_SEEK_DARK . . . . .	662
9.169.2.5 SCOUT_LR_SEEK_LIGHT . . . . .	662
9.169.2.6 SCOUT_LR_WAIT_FOR . . . . .	662
9.170 Scout transmit rule constants . . . . .	663
9.170.1 Detailed Description . . . . .	663
9.170.2 Define Documentation . . . . .	663
9.170.2.1 SCOUT_TGS_LONG . . . . .	663
9.170.2.2 SCOUT_TGS_MEDIUM . . . . .	663
9.170.2.3 SCOUT_TGS_SHORT . . . . .	663
9.171 Scout special effect constants . . . . .	664
9.171.1 Detailed Description . . . . .	664
9.171.2 Define Documentation . . . . .	664

9.171.2.1 SCOUT_FXR_ALARM . . . . .	664
9.171.2.2 SCOUT_FXR_BUG . . . . .	664
9.171.2.3 SCOUT_FXR_NONE . . . . .	664
9.171.2.4 SCOUT_FXR_RANDOM . . . . .	664
9.171.2.5 SCOUT_FXR_SCIENCE . . . . .	664
9.172RCX and Scout source constants . . . . .	665
9.172.1 Detailed Description . . . . .	665
9.172.2 Define Documentation . . . . .	666
9.172.2.1 RCX_BatteryLevelSrc . . . . .	666
9.172.2.2 RCX_ClickCounterSrc . . . . .	666
9.172.2.3 RCX_ConstantSrc . . . . .	666
9.172.2.4 RCX_CounterSrc . . . . .	666
9.172.2.5 RCX_DatalogRawDirectSrc . . . . .	666
9.172.2.6 RCX_DatalogRawIndirectSrc . . . . .	666
9.172.2.7 RCX_DatalogSrcDirectSrc . . . . .	666
9.172.2.8 RCX_DatalogSrcIndirectSrc . . . . .	666
9.172.2.9 RCX_DatalogValueDirectSrc . . . . .	666
9.172.2.10RCX_DatalogValueIndirectSrc . . . . .	667
9.172.2.11RCX_DurationSrc . . . . .	667
9.172.2.12RCX_EventStateSrc . . . . .	667
9.172.2.13RCX_FirmwareVersionSrc . . . . .	667
9.172.2.14RCX_GlobalMotorStatusSrc . . . . .	667
9.172.2.15RCX_HysteresisSrc . . . . .	667
9.172.2.16RCX_IndirectVarSrc . . . . .	667
9.172.2.17RCX_InputBooleanSrc . . . . .	667
9.172.2.18RCX_InputModeSrc . . . . .	667
9.172.2.19RCX_InputRawSrc . . . . .	667
9.172.2.20RCX_InputTypeSrc . . . . .	667
9.172.2.21RCX_InputValueSrc . . . . .	668
9.172.2.22RCX_LowerThresholdSrc . . . . .	668
9.172.2.23RCX_MessageSrc . . . . .	668
9.172.2.24RCX_OutputStatusSrc . . . . .	668
9.172.2.25RCX_ProgramSlotSrc . . . . .	668
9.172.2.26RCX_RandomSrc . . . . .	668
9.172.2.27RCX_ScoutCounterLimitSrc . . . . .	668
9.172.2.28RCX_ScoutEventFBSrc . . . . .	668

9.172.2.29RCX_ScoutLightParamsSrc . . . . .	668
9.172.2.30RCX_ScoutRulesSrc . . . . .	668
9.172.2.31RCX_ScoutTimerLimitSrc . . . . .	669
9.172.2.32RCX_TaskEventsSrc . . . . .	669
9.172.2.33RCX_TenMSTimerSrc . . . . .	669
9.172.2.34RCX_TimerSrc . . . . .	669
9.172.2.35RCX_UARTSetupSrc . . . . .	669
9.172.2.36RCX_UpperThresholdSrc . . . . .	669
9.172.2.37RCX_VariableSrc . . . . .	669
9.172.2.38RCX_WatchSrc . . . . .	669
9.173RCX and Scout opcode constants . . . . .	670
9.173.1 Detailed Description . . . . .	671
9.173.2 Define Documentation . . . . .	671
9.173.2.1 RCX_AbsVarOp . . . . .	671
9.173.2.2 RCX_AndVarOp . . . . .	671
9.173.2.3 RCX_AutoOffOp . . . . .	672
9.173.2.4 RCX_BatteryLevelOp . . . . .	672
9.173.2.5 RCX_BootModeOp . . . . .	672
9.173.2.6 RCX_CalibrateEventOp . . . . .	672
9.173.2.7 RCX_ClearAllEventsOp . . . . .	672
9.173.2.8 RCX_ClearCounterOp . . . . .	672
9.173.2.9 RCX_ClearMsgOp . . . . .	672
9.173.2.10RCX_ClearSensorOp . . . . .	672
9.173.2.11RCX_ClearSoundOp . . . . .	672
9.173.2.12RCX_ClearTimerOp . . . . .	672
9.173.2.13RCX_DatalogOp . . . . .	672
9.173.2.14RCX_DecCounterOp . . . . .	673
9.173.2.15RCX_DeleteSubOp . . . . .	673
9.173.2.16RCX_DeleteSubsOp . . . . .	673
9.173.2.17RCX_DeleteTaskOp . . . . .	673
9.173.2.18RCX_DeleteTasksOp . . . . .	673
9.173.2.19RCX_DirectEventOp . . . . .	673
9.173.2.20RCX_DisplayOp . . . . .	673
9.173.2.21RCX_DivVarOp . . . . .	673
9.173.2.22RCX_GOutputDirOp . . . . .	673
9.173.2.23RCX_GOutputModeOp . . . . .	673

9.173.2.24RCX_GOutputPowerOp . . . . .	673
9.173.2.25RCX_IncCounterOp . . . . .	674
9.173.2.26RCX_InputModeOp . . . . .	674
9.173.2.27RCX_InputTypeOp . . . . .	674
9.173.2.28RCX_IRModeOp . . . . .	674
9.173.2.29RCX_LightOp . . . . .	674
9.173.2.30RCX_LSblinkTimeOp . . . . .	674
9.173.2.31RCX_LSCalibrateOp . . . . .	674
9.173.2.32RCX_LSHysteresisOp . . . . .	674
9.173.2.33RCX_LSLowerThreshOp . . . . .	674
9.173.2.34RCX_LSupperThreshOp . . . . .	674
9.173.2.35RCX_MessageOp . . . . .	674
9.173.2.36RCX_MulVarOp . . . . .	675
9.173.2.37RCX_MuteSoundOp . . . . .	675
9.173.2.38RCX_OnOffFloatOp . . . . .	675
9.173.2.39RCX_OrVarOp . . . . .	675
9.173.2.40RCX_OutputDirOp . . . . .	675
9.173.2.41RCX_OutputPowerOp . . . . .	675
9.173.2.42RCX_PBTurnOffOp . . . . .	675
9.173.2.43RCX_PingOp . . . . .	675
9.173.2.44RCX_PlaySoundOp . . . . .	675
9.173.2.45RCX_PlayToneOp . . . . .	675
9.173.2.46RCX_PlayToneVarOp . . . . .	675
9.173.2.47RCX_PollMemoryOp . . . . .	676
9.173.2.48RCX_PollOp . . . . .	676
9.173.2.49RCX_RemoteOp . . . . .	676
9.173.2.50RCX_ScoutOp . . . . .	676
9.173.2.51RCX_ScoutRulesOp . . . . .	676
9.173.2.52RCX_SelectProgramOp . . . . .	676
9.173.2.53RCX_SendUARTDataOp . . . . .	676
9.173.2.54RCX_SetCounterOp . . . . .	676
9.173.2.55RCX_SetDatalogOp . . . . .	676
9.173.2.56RCX_SetEventOp . . . . .	676
9.173.2.57RCX_SetFeedbackOp . . . . .	676
9.173.2.58RCX_SetPriorityOp . . . . .	677
9.173.2.59RCX_SetSourceValueOp . . . . .	677



9.173.2.60RCX_SetTimerLimitOp . . . . .	677
9.173.2.61RCX_SetVarOp . . . . .	677
9.173.2.62RCX_SetWatchOp . . . . .	677
9.173.2.63RCX_SgnVarOp . . . . .	677
9.173.2.64RCX_SoundOp . . . . .	677
9.173.2.65RCX_StartTaskOp . . . . .	677
9.173.2.66RCX_StopAllTasksOp . . . . .	677
9.173.2.67RCX_StopTaskOp . . . . .	677
9.173.2.68RCX_SubVarOp . . . . .	677
9.173.2.69RCX_SumVarOp . . . . .	678
9.173.2.70RCX_UnlockFirmOp . . . . .	678
9.173.2.71RCX_UnlockOp . . . . .	678
9.173.2.72RCX_UnmuteSoundOp . . . . .	678
9.173.2.73RCX_UploadDatalogOp . . . . .	678
9.173.2.74RCX_ViewSourceValOp . . . . .	678
9.173.2.75RCX_VLLOp . . . . .	678
9.174HiTechnic/mindsensors Power Function/IR Train constants . . . . .	679
9.174.1 Detailed Description . . . . .	679
9.175Power Function command constants . . . . .	680
9.175.1 Detailed Description . . . . .	680
9.175.2 Define Documentation . . . . .	680
9.175.2.1 PF_CMD_BRAKE . . . . .	680
9.175.2.2 PF_CMD_FWD . . . . .	680
9.175.2.3 PF_CMD_REV . . . . .	680
9.175.2.4 PF_CMD_STOP . . . . .	680
9.176Power Function channel constants . . . . .	681
9.176.1 Detailed Description . . . . .	681
9.176.2 Define Documentation . . . . .	681
9.176.2.1 PF_CHANNEL_1 . . . . .	681
9.176.2.2 PF_CHANNEL_2 . . . . .	681
9.176.2.3 PF_CHANNEL_3 . . . . .	681
9.176.2.4 PF_CHANNEL_4 . . . . .	681
9.177Power Function mode constants . . . . .	682
9.177.1 Detailed Description . . . . .	682
9.177.2 Define Documentation . . . . .	682
9.177.2.1 PF_MODE_COMBO_DIRECT . . . . .	682

9.177.2.2 PF_MODE_COMBO_PWM . . . . .	682
9.177.2.3 PF_MODE_SINGLE_OUTPUT_CST . . . . .	682
9.177.2.4 PF_MODE_SINGLE_OUTPUT_PWM . . . . .	682
9.177.2.5 PF_MODE_SINGLE_PIN_CONT . . . . .	682
9.177.2.6 PF_MODE_SINGLE_PIN_TIME . . . . .	682
9.177.2.7 PF_MODE_TRAIN . . . . .	682
9.178PF/IR Train function constants . . . . .	683
9.178.1 Detailed Description . . . . .	683
9.178.2 Define Documentation . . . . .	683
9.178.2.1 TRAIN_FUNC_DECR_SPEED . . . . .	683
9.178.2.2 TRAIN_FUNC_INCR_SPEED . . . . .	683
9.178.2.3 TRAIN_FUNC_STOP . . . . .	683
9.178.2.4 TRAIN_FUNC_TOGGLE_LIGHT . . . . .	683
9.179IR Train channel constants . . . . .	684
9.179.1 Detailed Description . . . . .	684
9.179.2 Define Documentation . . . . .	684
9.179.2.1 TRAIN_CHANNEL_1 . . . . .	684
9.179.2.2 TRAIN_CHANNEL_2 . . . . .	684
9.179.2.3 TRAIN_CHANNEL_3 . . . . .	684
9.179.2.4 TRAIN_CHANNEL_ALL . . . . .	684
9.180Power Function output constants . . . . .	685
9.180.1 Detailed Description . . . . .	685
9.180.2 Define Documentation . . . . .	685
9.180.2.1 PF_OUT_A . . . . .	685
9.180.2.2 PF_OUT_B . . . . .	685
9.181Power Function pin constants . . . . .	686
9.181.1 Detailed Description . . . . .	686
9.181.2 Define Documentation . . . . .	686
9.181.2.1 PF_PIN_C1 . . . . .	686
9.181.2.2 PF_PIN_C2 . . . . .	686
9.182Power Function single pin function constants . . . . .	687
9.182.1 Detailed Description . . . . .	687
9.182.2 Define Documentation . . . . .	687
9.182.2.1 PF_FUNC_CLEAR . . . . .	687
9.182.2.2 PF_FUNC_NOCHANGE . . . . .	687
9.182.2.3 PF_FUNC_SET . . . . .	687

9.182.2.4 PF_FUNC_TOGGLE . . . . .	687
9.183Power Function CST options constants . . . . .	688
9.183.1 Detailed Description . . . . .	688
9.183.2 Define Documentation . . . . .	688
9.183.2.1 PF_CST_CLEAR1_CLEAR2 . . . . .	688
9.183.2.2 PF_CST_CLEAR1_SET2 . . . . .	688
9.183.2.3 PF_CST_DECREMENT_PWM . . . . .	688
9.183.2.4 PF_CST_FULL_FWD . . . . .	688
9.183.2.5 PF_CST_FULL_REV . . . . .	688
9.183.2.6 PF_CST_INCREMENT_PWM . . . . .	688
9.183.2.7 PF_CST_SET1_CLEAR2 . . . . .	689
9.183.2.8 PF_CST_SET1_SET2 . . . . .	689
9.183.2.9 PF_CST_TOGGLE_DIR . . . . .	689
9.184Power Function PWM option constants . . . . .	690
9.184.1 Detailed Description . . . . .	690
9.184.2 Define Documentation . . . . .	690
9.184.2.1 PF_PWM_BRAKE . . . . .	690
9.184.2.2 PF_PWM_FLOAT . . . . .	690
9.184.2.3 PF_PWM_FWD1 . . . . .	690
9.184.2.4 PF_PWM_FWD2 . . . . .	690
9.184.2.5 PF_PWM_FWD3 . . . . .	690
9.184.2.6 PF_PWM_FWD4 . . . . .	691
9.184.2.7 PF_PWM_FWD5 . . . . .	691
9.184.2.8 PF_PWM_FWD6 . . . . .	691
9.184.2.9 PF_PWM_FWD7 . . . . .	691
9.184.2.10PF_PWM_REV1 . . . . .	691
9.184.2.11PF_PWM_REV2 . . . . .	691
9.184.2.12PF_PWM_REV3 . . . . .	691
9.184.2.13PF_PWM_REV4 . . . . .	691
9.184.2.14PF_PWM_REV5 . . . . .	691
9.184.2.15PF_PWM_REV6 . . . . .	692
9.184.2.16PF_PWM_REV7 . . . . .	692
9.185HiTechnic device constants . . . . .	693
9.185.1 Detailed Description . . . . .	693
9.186HiTechnic IRSeeker2 constants . . . . .	694
9.186.1 Detailed Description . . . . .	694

9.186.2 Define Documentation . . . . .	694
9.186.2.1 HTIR2_MODE_1200 . . . . .	694
9.186.2.2 HTIR2_MODE_600 . . . . .	694
9.186.2.3 HTIR2_REG_AC01 . . . . .	694
9.186.2.4 HTIR2_REG_AC02 . . . . .	694
9.186.2.5 HTIR2_REG_AC03 . . . . .	695
9.186.2.6 HTIR2_REG_AC04 . . . . .	695
9.186.2.7 HTIR2_REG_AC05 . . . . .	695
9.186.2.8 HTIR2_REG_ACDIR . . . . .	695
9.186.2.9 HTIR2_REG_DC01 . . . . .	695
9.186.2.10 HTIR2_REG_DC02 . . . . .	695
9.186.2.11 HTIR2_REG_DC03 . . . . .	695
9.186.2.12 HTIR2_REG_DC04 . . . . .	695
9.186.2.13 HTIR2_REG_DC05 . . . . .	695
9.186.2.14 HTIR2_REG_DCAVG . . . . .	695
9.186.2.15 HTIR2_REG_DCDIR . . . . .	695
9.186.2.16 HTIR2_REG_MODE . . . . .	696
9.187 HiTechnic IRReceiver constants . . . . .	697
9.187.1 Detailed Description . . . . .	697
9.187.2 Define Documentation . . . . .	697
9.187.2.1 HT_CH1_A . . . . .	697
9.187.2.2 HT_CH1_B . . . . .	697
9.187.2.3 HT_CH2_A . . . . .	697
9.187.2.4 HT_CH2_B . . . . .	697
9.187.2.5 HT_CH3_A . . . . .	697
9.187.2.6 HT_CH3_B . . . . .	698
9.187.2.7 HT_CH4_A . . . . .	698
9.187.2.8 HT_CH4_B . . . . .	698
9.188 HiTechnic Color2 constants . . . . .	699
9.188.1 Detailed Description . . . . .	699
9.188.2 Define Documentation . . . . .	699
9.188.2.1 HT_CMD_COLOR2_50HZ . . . . .	699
9.188.2.2 HT_CMD_COLOR2_60HZ . . . . .	699
9.188.2.3 HT_CMD_COLOR2_ACTIVE . . . . .	699
9.188.2.4 HT_CMD_COLOR2_BLCAL . . . . .	699
9.188.2.5 HT_CMD_COLOR2_FAR . . . . .	699

9.188.2.6 HT_CMD_COLOR2_LED_HI . . . . .	700
9.188.2.7 HT_CMD_COLOR2_LED_LOW . . . . .	700
9.188.2.8 HT_CMD_COLOR2_NEAR . . . . .	700
9.188.2.9 HT_CMD_COLOR2_PASSIVE . . . . .	700
9.188.2.10 HT_CMD_COLOR2_RAW . . . . .	700
9.188.2.11 HT_CMD_COLOR2_WBCAL . . . . .	700
9.189 MindSensors device constants . . . . .	701
9.189.1 Detailed Description . . . . .	701
9.189.2 Define Documentation . . . . .	701
9.189.2.1 MS_CMD_ADPA_OFF . . . . .	701
9.189.2.2 MS_CMD_ADPA_ON . . . . .	701
9.189.2.3 MS_CMD_DEENERGIZED . . . . .	701
9.189.2.4 MS_CMD_ENERGIZED . . . . .	701
9.190 MindSensors DIST-Nx constants . . . . .	702
9.190.1 Detailed Description . . . . .	702
9.190.2 Define Documentation . . . . .	702
9.190.2.1 DIST_CMD_CUSTOM . . . . .	702
9.190.2.2 DIST_CMD_GP2D12 . . . . .	702
9.190.2.3 DIST_CMD_GP2D120 . . . . .	702
9.190.2.4 DIST_CMD_GP2YA02 . . . . .	702
9.190.2.5 DIST_CMD_GP2YA21 . . . . .	702
9.190.2.6 DIST_REG_DIST . . . . .	703
9.190.2.7 DIST_REG_DIST1 . . . . .	703
9.190.2.8 DIST_REG_DIST_MAX . . . . .	703
9.190.2.9 DIST_REG_DIST_MIN . . . . .	703
9.190.2.10 DIST_REG_MODULE_TYPE . . . . .	703
9.190.2.11 DIST_REG_NUM_POINTS . . . . .	703
9.190.2.12 DIST_REG_VOLT . . . . .	703
9.190.2.13 DIST_REG_VOLT1 . . . . .	703
9.191 MindSensors PSP-Nx constants . . . . .	704
9.191.1 Detailed Description . . . . .	704
9.191.2 Define Documentation . . . . .	704
9.191.2.1 PSP_CMD_ANALOG . . . . .	704
9.191.2.2 PSP_CMD_DIGITAL . . . . .	704
9.191.2.3 PSP_REG_BTNSET1 . . . . .	704
9.191.2.4 PSP_REG_BTNSET2 . . . . .	704

9.191.2.5 PSP_REG_XLEFT . . . . .	705
9.191.2.6 PSP_REG_XRIGHT . . . . .	705
9.191.2.7 PSP_REG_YLEFT . . . . .	705
9.191.2.8 PSP_REG_YRIGHT . . . . .	705
9.192MindSensors PSP-Nx button set 1 constants . . . . .	706
9.192.1 Detailed Description . . . . .	706
9.192.2 Define Documentation . . . . .	706
9.192.2.1 PSP_BTNSET1_DOWN . . . . .	706
9.192.2.2 PSP_BTNSET1_L3 . . . . .	706
9.192.2.3 PSP_BTNSET1_LEFT . . . . .	706
9.192.2.4 PSP_BTNSET1_R3 . . . . .	706
9.192.2.5 PSP_BTNSET1_RIGHT . . . . .	706
9.192.2.6 PSP_BTNSET1_UP . . . . .	706
9.193MindSensors PSP-Nx button set 2 constants . . . . .	707
9.193.1 Detailed Description . . . . .	707
9.193.2 Define Documentation . . . . .	707
9.193.2.1 PSP_BTNSET2_CIRCLE . . . . .	707
9.193.2.2 PSP_BTNSET2_CROSS . . . . .	707
9.193.2.3 PSP_BTNSET2_L1 . . . . .	707
9.193.2.4 PSP_BTNSET2_L2 . . . . .	707
9.193.2.5 PSP_BTNSET2_R1 . . . . .	707
9.193.2.6 PSP_BTNSET2_R2 . . . . .	707
9.193.2.7 PSP_BTNSET2_SQUARE . . . . .	708
9.193.2.8 PSP_BTNSET2_TRIANGLE . . . . .	708
9.194MindSensors nRLink constants . . . . .	709
9.194.1 Detailed Description . . . . .	709
9.194.2 Define Documentation . . . . .	709
9.194.2.1 NRLINK_CMD_2400 . . . . .	709
9.194.2.2 NRLINK_CMD_4800 . . . . .	709
9.194.2.3 NRLINK_CMD_FLUSH . . . . .	709
9.194.2.4 NRLINK_CMD_IR_LONG . . . . .	709
9.194.2.5 NRLINK_CMD_IR_SHORT . . . . .	709
9.194.2.6 NRLINK_CMD_RUN_MACRO . . . . .	710
9.194.2.7 NRLINK_CMD_SET_PF . . . . .	710
9.194.2.8 NRLINK_CMD_SET_RCX . . . . .	710
9.194.2.9 NRLINK_CMD_SET_TRAIN . . . . .	710

9.194.2.10NRLINK_CMD_TX_RAW . . . . .	710
9.194.2.11NRLINK_REG_BYTES . . . . .	710
9.194.2.12NRLINK_REG_DATA . . . . .	710
9.194.2.13NRLINK_REG_EEPROM . . . . .	710
9.195Data type limits . . . . .	711
9.195.1 Detailed Description . . . . .	711
9.195.2 Define Documentation . . . . .	711
9.195.2.1 CHAR_BIT . . . . .	711
9.195.2.2 CHAR_MAX . . . . .	711
9.195.2.3 CHAR_MIN . . . . .	711
9.195.2.4 INT_MAX . . . . .	711
9.195.2.5 INT_MIN . . . . .	711
9.195.2.6 LONG_MAX . . . . .	712
9.195.2.7 LONG_MIN . . . . .	712
9.195.2.8 RAND_MAX . . . . .	712
9.195.2.9 SCHAR_MAX . . . . .	712
9.195.2.10SCHAR_MIN . . . . .	712
9.195.2.11SHRT_MAX . . . . .	712
9.195.2.12SHRT_MIN . . . . .	712
9.195.2.13UCHAR_MAX . . . . .	712
9.195.2.14UINT_MAX . . . . .	712
9.195.2.15ULONG_MAX . . . . .	712
9.195.2.16USHRT_MAX . . . . .	712
9.196Graphics library begin modes . . . . .	713
9.196.1 Detailed Description . . . . .	713
9.196.2 Define Documentation . . . . .	713
9.196.2.1 GL_CIRCLE . . . . .	713
9.196.2.2 GL_LINE . . . . .	713
9.196.2.3 GL_POINT . . . . .	713
9.196.2.4 GL_POLYGON . . . . .	713
9.197Graphics library actions . . . . .	714
9.197.1 Detailed Description . . . . .	714
9.197.2 Define Documentation . . . . .	714
9.197.2.1 GL_ROTATE_X . . . . .	714
9.197.2.2 GL_ROTATE_Y . . . . .	714
9.197.2.3 GL_ROTATE_Z . . . . .	714

9.197.2.4 GL_SCALE_X . . . . .	714
9.197.2.5 GL_SCALE_Y . . . . .	715
9.197.2.6 GL_SCALE_Z . . . . .	715
9.197.2.7 GL_TRANSLATE_X . . . . .	715
9.197.2.8 GL_TRANSLATE_Y . . . . .	715
9.197.2.9 GL_TRANSLATE_Z . . . . .	715
9.198 Graphics library settings . . . . .	716
9.198.1 Detailed Description . . . . .	716
9.198.2 Define Documentation . . . . .	716
9.198.2.1 GL_CAMERA_DEPTH . . . . .	716
9.198.2.2 GL_CIRCLE_SIZE . . . . .	716
9.198.2.3 GL_CULL_MODE . . . . .	716
9.198.2.4 GL_ZOOM_FACTOR . . . . .	716
9.199 Graphics library cull mode . . . . .	717
9.199.1 Detailed Description . . . . .	717
9.199.2 Define Documentation . . . . .	717
9.199.2.1 GL_CULL_BACK . . . . .	717
9.199.2.2 GL_CULL_FRONT . . . . .	717
9.199.2.3 GL_CULL_NONE . . . . .	717
<b>10 Data Structure Documentation</b>	<b>719</b>
10.1 ColorSensorReadType Struct Reference . . . . .	719
10.1.1 Detailed Description . . . . .	719
10.1.2 Field Documentation . . . . .	719
10.1.2.1 ColorValue . . . . .	719
10.1.2.2 Invalid . . . . .	720
10.1.2.3 NormalizedArray . . . . .	720
10.1.2.4 Port . . . . .	720
10.1.2.5 RawArray . . . . .	720
10.1.2.6 Result . . . . .	720
10.1.2.7 ScaledArray . . . . .	720
10.2 CommBTCheckStatusType Struct Reference . . . . .	721
10.2.1 Detailed Description . . . . .	721
10.2.2 Field Documentation . . . . .	721
10.2.2.1 Connection . . . . .	721
10.2.2.2 Result . . . . .	721
10.3 CommBTConnectionType Struct Reference . . . . .	722



10.3.1 Detailed Description . . . . .	722
10.3.2 Field Documentation . . . . .	722
10.3.2.1 Action . . . . .	722
10.3.2.2 ConnectionSlot . . . . .	722
10.3.2.3 Name . . . . .	722
10.3.2.4 Result . . . . .	723
10.4 CommBTOffType Struct Reference . . . . .	724
10.4.1 Detailed Description . . . . .	724
10.4.2 Field Documentation . . . . .	724
10.4.2.1 PowerState . . . . .	724
10.4.2.2 Result . . . . .	724
10.5 CommBTWriteType Struct Reference . . . . .	725
10.5.1 Detailed Description . . . . .	725
10.5.2 Field Documentation . . . . .	725
10.5.2.1 Buffer . . . . .	725
10.5.2.2 Connection . . . . .	725
10.5.2.3 Result . . . . .	725
10.6 CommExecuteFunctionType Struct Reference . . . . .	726
10.6.1 Detailed Description . . . . .	726
10.6.2 Field Documentation . . . . .	727
10.6.2.1 Cmd . . . . .	727
10.6.2.2 Name . . . . .	727
10.6.2.3 Param1 . . . . .	727
10.6.2.4 Param2 . . . . .	727
10.6.2.5 Param3 . . . . .	727
10.6.2.6 Result . . . . .	727
10.6.2.7 RetVal . . . . .	727
10.7 CommHSCheckStatusType Struct Reference . . . . .	728
10.7.1 Detailed Description . . . . .	728
10.7.2 Field Documentation . . . . .	728
10.7.2.1 DataAvailable . . . . .	728
10.7.2.2 SendingData . . . . .	728
10.8 CommHSControlType Struct Reference . . . . .	729
10.8.1 Detailed Description . . . . .	729
10.8.2 Field Documentation . . . . .	729
10.8.2.1 BaudRate . . . . .	729

10.8.2.2 Command . . . . .	729
10.8.2.3 Mode . . . . .	729
10.8.2.4 Result . . . . .	729
10.9 CommHSReadWriteType Struct Reference . . . . .	731
10.9.1 Detailed Description . . . . .	731
10.9.2 Field Documentation . . . . .	731
10.9.2.1 Buffer . . . . .	731
10.9.2.2 Status . . . . .	731
10.10CommLSCheckStatusType Struct Reference . . . . .	732
10.10.1 Detailed Description . . . . .	732
10.10.2 Field Documentation . . . . .	732
10.10.2.1 BytesReady . . . . .	732
10.10.2.2 Port . . . . .	732
10.10.2.3 Result . . . . .	732
10.11CommLSReadType Struct Reference . . . . .	733
10.11.1 Detailed Description . . . . .	733
10.11.2 Field Documentation . . . . .	733
10.11.2.1 Buffer . . . . .	733
10.11.2.2 BufferLen . . . . .	733
10.11.2.3 Port . . . . .	733
10.11.2.4 Result . . . . .	734
10.12CommLSWriteExType Struct Reference . . . . .	735
10.12.1 Detailed Description . . . . .	735
10.12.2 Field Documentation . . . . .	735
10.12.2.1 Buffer . . . . .	735
10.12.2.2 NoRestartOnRead . . . . .	735
10.12.2.3 Port . . . . .	735
10.12.2.4 Result . . . . .	736
10.12.2.5 ReturnLen . . . . .	736
10.13CommLSWriteType Struct Reference . . . . .	737
10.13.1 Detailed Description . . . . .	737
10.13.2 Field Documentation . . . . .	737
10.13.2.1 Buffer . . . . .	737
10.13.2.2 Port . . . . .	737
10.13.2.3 Result . . . . .	737
10.13.2.4 ReturnLen . . . . .	738

10.14ComputeCalibValueType Struct Reference . . . . .	739
10.14.1 Detailed Description . . . . .	739
10.14.2 Field Documentation . . . . .	739
10.14.2.1 Name . . . . .	739
10.14.2.2 RawVal . . . . .	739
10.14.2.3 Result . . . . .	740
10.15DatalogGetTimesType Struct Reference . . . . .	741
10.15.1 Detailed Description . . . . .	741
10.15.2 Field Documentation . . . . .	741
10.15.2.1 SyncTick . . . . .	741
10.15.2.2 SyncTime . . . . .	741
10.16DatalogWriteType Struct Reference . . . . .	742
10.16.1 Detailed Description . . . . .	742
10.16.2 Field Documentation . . . . .	742
10.16.2.1 Message . . . . .	742
10.16.2.2 Result . . . . .	742
10.17DisplayExecuteFunctionType Struct Reference . . . . .	743
10.17.1 Detailed Description . . . . .	743
10.17.2 Field Documentation . . . . .	743
10.17.2.1 Cmd . . . . .	743
10.17.2.2 On . . . . .	744
10.17.2.3 Status . . . . .	744
10.17.2.4 X1 . . . . .	744
10.17.2.5 X2 . . . . .	744
10.17.2.6 Y1 . . . . .	744
10.17.2.7 Y2 . . . . .	744
10.18div_t Struct Reference . . . . .	745
10.18.1 Detailed Description . . . . .	745
10.18.2 Field Documentation . . . . .	745
10.18.2.1 quot . . . . .	745
10.18.2.2 rem . . . . .	745
10.19DrawCircleType Struct Reference . . . . .	746
10.19.1 Detailed Description . . . . .	746
10.19.2 Field Documentation . . . . .	746
10.19.2.1 Center . . . . .	746
10.19.2.2 Options . . . . .	746

10.19.2.3 Result . . . . .	746
10.19.2.4 Size . . . . .	747
10.20DrawEllipseType Struct Reference . . . . .	748
10.20.1 Detailed Description . . . . .	748
10.20.2 Field Documentation . . . . .	748
10.20.2.1 Center . . . . .	748
10.20.2.2 Options . . . . .	748
10.20.2.3 Result . . . . .	748
10.20.2.4 SizeX . . . . .	749
10.20.2.5 SizeY . . . . .	749
10.21DrawFontType Struct Reference . . . . .	750
10.21.1 Detailed Description . . . . .	750
10.21.2 Field Documentation . . . . .	750
10.21.2.1 Filename . . . . .	750
10.21.2.2 Location . . . . .	750
10.21.2.3 Options . . . . .	750
10.21.2.4 Result . . . . .	751
10.21.2.5 Text . . . . .	751
10.22DrawGraphicArrayType Struct Reference . . . . .	752
10.22.1 Detailed Description . . . . .	752
10.22.2 Field Documentation . . . . .	752
10.22.2.1 Data . . . . .	752
10.22.2.2 Location . . . . .	752
10.22.2.3 Options . . . . .	753
10.22.2.4 Result . . . . .	753
10.22.2.5 Variables . . . . .	753
10.23DrawGraphicType Struct Reference . . . . .	754
10.23.1 Detailed Description . . . . .	754
10.23.2 Field Documentation . . . . .	754
10.23.2.1 Filename . . . . .	754
10.23.2.2 Location . . . . .	754
10.23.2.3 Options . . . . .	755
10.23.2.4 Result . . . . .	755
10.23.2.5 Variables . . . . .	755
10.24DrawLineType Struct Reference . . . . .	756
10.24.1 Detailed Description . . . . .	756

10.24.2 Field Documentation . . . . .	756
10.24.2.1 EndLoc . . . . .	756
10.24.2.2 Options . . . . .	756
10.24.2.3 Result . . . . .	756
10.24.2.4 StartLoc . . . . .	757
10.25DrawPointType Struct Reference . . . . .	758
10.25.1 Detailed Description . . . . .	758
10.25.2 Field Documentation . . . . .	758
10.25.2.1 Location . . . . .	758
10.25.2.2 Options . . . . .	758
10.25.2.3 Result . . . . .	758
10.26DrawPolygonType Struct Reference . . . . .	759
10.26.1 Detailed Description . . . . .	759
10.26.2 Field Documentation . . . . .	759
10.26.2.1 Options . . . . .	759
10.26.2.2 Points . . . . .	759
10.26.2.3 Result . . . . .	759
10.27DrawRectType Struct Reference . . . . .	760
10.27.1 Detailed Description . . . . .	760
10.27.2 Field Documentation . . . . .	760
10.27.2.1 Location . . . . .	760
10.27.2.2 Options . . . . .	760
10.27.2.3 Result . . . . .	760
10.27.2.4 Size . . . . .	761
10.28DrawTextType Struct Reference . . . . .	762
10.28.1 Detailed Description . . . . .	762
10.28.2 Field Documentation . . . . .	762
10.28.2.1 Location . . . . .	762
10.28.2.2 Options . . . . .	762
10.28.2.3 Result . . . . .	762
10.28.2.4 Text . . . . .	763
10.29FileCloseType Struct Reference . . . . .	764
10.29.1 Detailed Description . . . . .	764
10.29.2 Field Documentation . . . . .	764
10.29.2.1 FileHandle . . . . .	764
10.29.2.2 Result . . . . .	764

10.30FileDeleteType Struct Reference . . . . .	765
10.30.1 Detailed Description . . . . .	765
10.30.2 Field Documentation . . . . .	765
10.30.2.1 Filename . . . . .	765
10.30.2.2 Result . . . . .	765
10.31FileFindType Struct Reference . . . . .	766
10.31.1 Detailed Description . . . . .	766
10.31.2 Field Documentation . . . . .	766
10.31.2.1 FileHandle . . . . .	766
10.31.2.2 Filename . . . . .	766
10.31.2.3 Length . . . . .	766
10.31.2.4 Result . . . . .	767
10.32FileOpenType Struct Reference . . . . .	768
10.32.1 Detailed Description . . . . .	768
10.32.2 Field Documentation . . . . .	768
10.32.2.1 FileHandle . . . . .	768
10.32.2.2 Filename . . . . .	768
10.32.2.3 Length . . . . .	768
10.32.2.4 Result . . . . .	769
10.33FileReadWriteType Struct Reference . . . . .	770
10.33.1 Detailed Description . . . . .	770
10.33.2 Field Documentation . . . . .	770
10.33.2.1 Buffer . . . . .	770
10.33.2.2 FileHandle . . . . .	770
10.33.2.3 Length . . . . .	770
10.33.2.4 Result . . . . .	771
10.34FileRenameType Struct Reference . . . . .	772
10.34.1 Detailed Description . . . . .	772
10.34.2 Field Documentation . . . . .	772
10.34.2.1 NewFilename . . . . .	772
10.34.2.2 OldFilename . . . . .	772
10.34.2.3 Result . . . . .	772
10.35FileResizeType Struct Reference . . . . .	774
10.35.1 Detailed Description . . . . .	774
10.35.2 Field Documentation . . . . .	774
10.35.2.1 FileHandle . . . . .	774

10.35.2.2 NewSize . . . . .	774
10.35.2.3 Result . . . . .	774
10.36FileResolveHandleType Struct Reference . . . . .	775
10.36.1 Detailed Description . . . . .	775
10.36.2 Field Documentation . . . . .	775
10.36.2.1 FileHandle . . . . .	775
10.36.2.2 Filename . . . . .	775
10.36.2.3 Result . . . . .	775
10.36.2.4 WriteHandle . . . . .	776
10.37FileSeekType Struct Reference . . . . .	777
10.37.1 Detailed Description . . . . .	777
10.37.2 Field Documentation . . . . .	777
10.37.2.1 FileHandle . . . . .	777
10.37.2.2 Length . . . . .	777
10.37.2.3 Origin . . . . .	777
10.37.2.4 Result . . . . .	778
10.38GetStartTickType Struct Reference . . . . .	779
10.38.1 Detailed Description . . . . .	779
10.38.2 Field Documentation . . . . .	779
10.38.2.1 Result . . . . .	779
10.39IOMapReadByIDType Struct Reference . . . . .	780
10.39.1 Detailed Description . . . . .	780
10.39.2 Field Documentation . . . . .	780
10.39.2.1 Buffer . . . . .	780
10.39.2.2 Count . . . . .	780
10.39.2.3 ModuleID . . . . .	780
10.39.2.4 Offset . . . . .	781
10.39.2.5 Result . . . . .	781
10.40IOMapReadType Struct Reference . . . . .	782
10.40.1 Detailed Description . . . . .	782
10.40.2 Field Documentation . . . . .	782
10.40.2.1 Buffer . . . . .	782
10.40.2.2 Count . . . . .	782
10.40.2.3 ModuleName . . . . .	782
10.40.2.4 Offset . . . . .	783
10.40.2.5 Result . . . . .	783

10.41IOMapWriteByIDType Struct Reference . . . . .	784
10.41.1 Detailed Description . . . . .	784
10.41.2 Field Documentation . . . . .	784
10.41.2.1 Buffer . . . . .	784
10.41.2.2 ModuleID . . . . .	784
10.41.2.3 Offset . . . . .	784
10.41.2.4 Result . . . . .	785
10.42IOMapWriteType Struct Reference . . . . .	786
10.42.1 Detailed Description . . . . .	786
10.42.2 Field Documentation . . . . .	786
10.42.2.1 Buffer . . . . .	786
10.42.2.2 ModuleName . . . . .	786
10.42.2.3 Offset . . . . .	786
10.42.2.4 Result . . . . .	787
10.43KeepAliveType Struct Reference . . . . .	788
10.43.1 Detailed Description . . . . .	788
10.43.2 Field Documentation . . . . .	788
10.43.2.1 Result . . . . .	788
10.44ldiv_t Struct Reference . . . . .	789
10.44.1 Detailed Description . . . . .	789
10.44.2 Field Documentation . . . . .	789
10.44.2.1 quot . . . . .	789
10.44.2.2 rem . . . . .	789
10.45ListFilesType Struct Reference . . . . .	790
10.45.1 Detailed Description . . . . .	790
10.45.2 Field Documentation . . . . .	790
10.45.2.1 FileList . . . . .	790
10.45.2.2 Pattern . . . . .	790
10.45.2.3 Result . . . . .	790
10.46LoaderExecuteFunctionType Struct Reference . . . . .	792
10.46.1 Detailed Description . . . . .	792
10.46.2 Field Documentation . . . . .	793
10.46.2.1 Buffer . . . . .	793
10.46.2.2 Cmd . . . . .	793
10.46.2.3 Filename . . . . .	793
10.46.2.4 Length . . . . .	793



10.46.2.5 Result . . . . .	793
10.47LocationType Struct Reference . . . . .	794
10.47.1 Detailed Description . . . . .	794
10.47.2 Field Documentation . . . . .	794
10.47.2.1 X . . . . .	794
10.47.2.2 Y . . . . .	794
10.48MessageReadType Struct Reference . . . . .	795
10.48.1 Detailed Description . . . . .	795
10.48.2 Field Documentation . . . . .	795
10.48.2.1 Message . . . . .	795
10.48.2.2 QueueID . . . . .	795
10.48.2.3 Remove . . . . .	795
10.48.2.4 Result . . . . .	796
10.49MessageWriteType Struct Reference . . . . .	797
10.49.1 Detailed Description . . . . .	797
10.49.2 Field Documentation . . . . .	797
10.49.2.1 Message . . . . .	797
10.49.2.2 QueueID . . . . .	797
10.49.2.3 Result . . . . .	797
10.50RandomNumberType Struct Reference . . . . .	798
10.50.1 Detailed Description . . . . .	798
10.50.2 Field Documentation . . . . .	798
10.50.2.1 Result . . . . .	798
10.51ReadButtonType Struct Reference . . . . .	799
10.51.1 Detailed Description . . . . .	799
10.51.2 Field Documentation . . . . .	799
10.51.2.1 Count . . . . .	799
10.51.2.2 Index . . . . .	799
10.51.2.3 Pressed . . . . .	799
10.51.2.4 Reset . . . . .	800
10.51.2.5 Result . . . . .	800
10.52ReadSemDataType Struct Reference . . . . .	801
10.52.1 Detailed Description . . . . .	801
10.52.2 Field Documentation . . . . .	801
10.52.2.1 Request . . . . .	801
10.52.2.2 SemData . . . . .	801

10.53SetScreenModeType Struct Reference . . . . .	802
10.53.1 Detailed Description . . . . .	802
10.53.2 Field Documentation . . . . .	802
10.53.2.1 Result . . . . .	802
10.53.2.2 ScreenMode . . . . .	802
10.54SetSleepTimeoutType Struct Reference . . . . .	803
10.54.1 Detailed Description . . . . .	803
10.54.2 Field Documentation . . . . .	803
10.54.2.1 Result . . . . .	803
10.54.2.2 TheSleepTimeoutMS . . . . .	803
10.55SizeType Struct Reference . . . . .	804
10.55.1 Detailed Description . . . . .	804
10.55.2 Field Documentation . . . . .	804
10.55.2.1 Height . . . . .	804
10.55.2.2 Width . . . . .	804
10.56SoundGetStateType Struct Reference . . . . .	805
10.56.1 Detailed Description . . . . .	805
10.56.2 Field Documentation . . . . .	805
10.56.2.1 Flags . . . . .	805
10.56.2.2 State . . . . .	805
10.57SoundPlayFileType Struct Reference . . . . .	806
10.57.1 Detailed Description . . . . .	806
10.57.2 Field Documentation . . . . .	806
10.57.2.1 Filename . . . . .	806
10.57.2.2 Loop . . . . .	806
10.57.2.3 Result . . . . .	806
10.57.2.4 SoundLevel . . . . .	807
10.58SoundPlayToneType Struct Reference . . . . .	808
10.58.1 Detailed Description . . . . .	808
10.58.2 Field Documentation . . . . .	808
10.58.2.1 Duration . . . . .	808
10.58.2.2 Frequency . . . . .	808
10.58.2.3 Loop . . . . .	808
10.58.2.4 Result . . . . .	809
10.58.2.5 SoundLevel . . . . .	809
10.59SoundSetStateType Struct Reference . . . . .	810

10.59.1 Detailed Description . . . . .	810
10.59.2 Field Documentation . . . . .	810
10.59.2.1 Flags . . . . .	810
10.59.2.2 Result . . . . .	810
10.59.2.3 State . . . . .	810
10.60Tone Struct Reference . . . . .	811
10.60.1 Detailed Description . . . . .	811
10.60.2 Field Documentation . . . . .	811
10.60.2.1 Duration . . . . .	811
10.60.2.2 Frequency . . . . .	811
10.61UpdateCalibCacheInfoType Struct Reference . . . . .	812
10.61.1 Detailed Description . . . . .	812
10.61.2 Field Documentation . . . . .	812
10.61.2.1 MaxVal . . . . .	812
10.61.2.2 MinVal . . . . .	812
10.61.2.3 Name . . . . .	812
10.61.2.4 Result . . . . .	813
10.62WriteSemDataType Struct Reference . . . . .	814
10.62.1 Detailed Description . . . . .	814
10.62.2 Field Documentation . . . . .	814
10.62.2.1 ClearBits . . . . .	814
10.62.2.2 NewVal . . . . .	814
10.62.2.3 Request . . . . .	814
10.62.2.4 SemData . . . . .	815
<b>11 File Documentation</b>	<b>817</b>
11.1 NBCCCommon.h File Reference . . . . .	817
11.1.1 Detailed Description . . . . .	846
11.1.2 Define Documentation . . . . .	846
11.1.2.1 ActualSpeed . . . . .	846
11.1.2.2 BITMAP_1 . . . . .	846
11.1.2.3 BITMAP_2 . . . . .	846
11.1.2.4 BITMAP_3 . . . . .	847
11.1.2.5 BITMAP_4 . . . . .	847
11.1.2.6 BITMAPS . . . . .	847
11.1.2.7 BlockTachoCount . . . . .	847
11.1.2.8 BREAKOUT_REQ . . . . .	847

11.1.2.9 BT_ARM_CMD_MODE . . . . .	847
11.1.2.10 BT_ARM_DATA_MODE . . . . .	847
11.1.2.11 BT_ARM_OFF . . . . .	847
11.1.2.12 BT_BRICK_PORT_OPEN . . . . .	847
11.1.2.13 BT_BRICK_VISIBILITY . . . . .	847
11.1.2.14 BT_CMD_BYTE . . . . .	848
11.1.2.15 BT_CMD_READY . . . . .	848
11.1.2.16 BT_CONNECTION_0_ENABLE . . . . .	848
11.1.2.17 BT_CONNECTION_1_ENABLE . . . . .	848
11.1.2.18 BT_CONNECTION_2_ENABLE . . . . .	848
11.1.2.19 BT_CONNECTION_3_ENABLE . . . . .	848
11.1.2.20 BT_DEFAULT_INQUIRY_MAX . . . . .	848
11.1.2.21 BT_DEFAULT_INQUIRY_TIMEOUT_LO . . . . .	848
11.1.2.22 BT_DEVICE_AWAY . . . . .	848
11.1.2.23 BT_DEVICE_EMPTY . . . . .	848
11.1.2.24 BT_DEVICE_KNOWN . . . . .	848
11.1.2.25 BT_DEVICE_NAME . . . . .	849
11.1.2.26 BT_DEVICE_UNKNOWN . . . . .	849
11.1.2.27 BT_DISABLE . . . . .	849
11.1.2.28 BT_ENABLE . . . . .	849
11.1.2.29 BTN1 . . . . .	849
11.1.2.30 BTN2 . . . . .	849
11.1.2.31 BTN3 . . . . .	849
11.1.2.32 BTN4 . . . . .	849
11.1.2.33 BTNCENTER . . . . .	849
11.1.2.34 BTNEXIT . . . . .	850
11.1.2.35 BTNLEFT . . . . .	850
11.1.2.36 BTRIGHT . . . . .	850
11.1.2.37 BTNSTATE_LONG_PRESSED_EV . . . . .	850
11.1.2.38 BTNSTATE_LONG_RELEASED_EV . . . . .	850
11.1.2.39 BTNSTATE_NONE . . . . .	850
11.1.2.40 BTNSTATE_PRESSED_EV . . . . .	850
11.1.2.41 BTNSTATE_PRESSED_STATE . . . . .	851
11.1.2.42 BTNSTATE_SHORT_RELEASED_EV . . . . .	851
11.1.2.43 ButtonModuleID . . . . .	851
11.1.2.44 ButtonModuleName . . . . .	851

11.1.2.45 ButtonOffsetLongPressCnt . . . . .	851
11.1.2.46 ButtonOffsetLongRelCnt . . . . .	851
11.1.2.47 ButtonOffsetPressedCnt . . . . .	851
11.1.2.48 ButtonOffsetRelCnt . . . . .	851
11.1.2.49 ButtonOffsetShortRelCnt . . . . .	851
11.1.2.50 ButtonOffsetState . . . . .	851
11.1.2.51 CHAR_BIT . . . . .	851
11.1.2.52 CHAR_MAX . . . . .	852
11.1.2.53 CHAR_MIN . . . . .	852
11.1.2.54 CLUMP_DONE . . . . .	852
11.1.2.55 CLUMP_SUSPEND . . . . .	852
11.1.2.56 ColorSensorRead . . . . .	852
11.1.2.57 COM_CHANNEL_FOUR_ACTIVE . . . . .	852
11.1.2.58 COM_CHANNEL_NONE_ACTIVE . . . . .	852
11.1.2.59 COM_CHANNEL_ONE_ACTIVE . . . . .	852
11.1.2.60 COM_CHANNEL_THREE_ACTIVE . . . . .	852
11.1.2.61 COM_CHANNEL_TWO_ACTIVE . . . . .	852
11.1.2.62 CommandModuleID . . . . .	852
11.1.2.63 CommandModuleName . . . . .	853
11.1.2.64 CommandOffsetActivateFlag . . . . .	853
11.1.2.65 CommandOffsetAwake . . . . .	853
11.1.2.66 CommandOffsetDeactivateFlag . . . . .	853
11.1.2.67 CommandOffsetFileName . . . . .	853
11.1.2.68 CommandOffsetFormatString . . . . .	853
11.1.2.69 CommandOffsetMemoryPool . . . . .	853
11.1.2.70 CommandOffsetOffsetDS . . . . .	853
11.1.2.71 CommandOffsetOffsetDVA . . . . .	853
11.1.2.72 CommandOffsetPRCHandler . . . . .	853
11.1.2.73 CommandOffsetProgStatus . . . . .	854
11.1.2.74 CommandOffsetSyncTick . . . . .	854
11.1.2.75 CommandOffsetSyncTime . . . . .	854
11.1.2.76 CommandOffsetTick . . . . .	854
11.1.2.77 CommBTCheckStatus . . . . .	854
11.1.2.78 CommBTConnection . . . . .	854
11.1.2.79 CommBTOnOff . . . . .	854
11.1.2.80 CommBTRead . . . . .	854

11.1.2.81 CommBTWrite . . . . .	854
11.1.2.82 CommExecuteFunction . . . . .	854
11.1.2.83 CommHSCheckStatus . . . . .	854
11.1.2.84 CommHSControl . . . . .	855
11.1.2.85 CommHSRead . . . . .	855
11.1.2.86 CommHSWrite . . . . .	855
11.1.2.87 CommLSCheckStatus . . . . .	855
11.1.2.88 CommLSRead . . . . .	855
11.1.2.89 CommLSWrite . . . . .	855
11.1.2.90 CommLSWriteEx . . . . .	855
11.1.2.91 CommModuleID . . . . .	855
11.1.2.92 CommModuleName . . . . .	855
11.1.2.93 CommOffsetBrickDataBdAddr . . . . .	855
11.1.2.94 CommOffsetBrickDataBluecoreVersion . . . . .	855
11.1.2.95 CommOffsetBrickDataBtHwStatus . . . . .	856
11.1.2.96 CommOffsetBrickDataBtStateStatus . . . . .	856
11.1.2.97 CommOffsetBrickDataName . . . . .	856
11.1.2.98 CommOffsetBrickDataTimeOutValue . . . . .	856
11.1.2.99 CommOffsetBtConnectTableBdAddr . . . . .	856
11.1.2.100CommOffsetBtConnectTableClassOfDevice . . . . .	856
11.1.2.101CommOffsetBtConnectTableHandleNr . . . . .	856
11.1.2.102CommOffsetBtConnectTableLinkQuality . . . . .	856
11.1.2.103CommOffsetBtConnectTableName . . . . .	856
11.1.2.104CommOffsetBtConnectTablePinCode . . . . .	856
11.1.2.105CommOffsetBtConnectTableStreamStatus . . . . .	856
11.1.2.106CommOffsetBtDeviceCnt . . . . .	857
11.1.2.107CommOffsetBtDeviceNameCnt . . . . .	857
11.1.2.108CommOffsetBtDeviceTableBdAddr . . . . .	857
11.1.2.109CommOffsetBtDeviceTableClassOfDevice . . . . .	857
11.1.2.110CommOffsetBtDeviceTableDeviceStatus . . . . .	857
11.1.2.111CommOffsetBtDeviceTableName . . . . .	857
11.1.2.112CommOffsetBtInBufBuf . . . . .	857
11.1.2.113CommOffsetBtInBufInPtr . . . . .	857
11.1.2.114CommOffsetBtInBufOutPtr . . . . .	857
11.1.2.115CommOffsetBtOutBufBuf . . . . .	857
11.1.2.116CommOffsetBtOutBufInPtr . . . . .	857

11.1.2.117CommOffsetBtOutBufOutPtr . . . . .	858
11.1.2.118CommOffsetHsFlags . . . . .	858
11.1.2.119CommOffsetHsInBufBuf . . . . .	858
11.1.2.120CommOffsetHsInBufInPtr . . . . .	858
11.1.2.121CommOffsetHsInBufOutPtr . . . . .	858
11.1.2.122CommOffsetHsMode . . . . .	858
11.1.2.123CommOffsetHsOutBufBuf . . . . .	858
11.1.2.124CommOffsetHsOutBufInPtr . . . . .	858
11.1.2.125CommOffsetHsOutBufOutPtr . . . . .	858
11.1.2.126CommOffsetHsSpeed . . . . .	858
11.1.2.127CommOffsetHsState . . . . .	858
11.1.2.128CommOffsetPFunc . . . . .	859
11.1.2.129CommOffsetPFuncTwo . . . . .	859
11.1.2.130CommOffsetUsbInBufBuf . . . . .	859
11.1.2.131CommOffsetUsbInBufInPtr . . . . .	859
11.1.2.132CommOffsetUsbInBufOutPtr . . . . .	859
11.1.2.133CommOffsetUsbOutBufBuf . . . . .	859
11.1.2.134CommOffsetUsbOutBufInPtr . . . . .	859
11.1.2.135CommOffsetUsbOutBufOutPtr . . . . .	859
11.1.2.136CommOffsetUsbPollBufBuf . . . . .	859
11.1.2.137CommOffsetUsbPollBufInPtr . . . . .	859
11.1.2.138CommOffsetUsbPollBufOutPtr . . . . .	859
11.1.2.139CommOffsetUsbState . . . . .	860
11.1.2.140ComputeCalibValue . . . . .	860
11.1.2.141DatalogGetTimes . . . . .	860
11.1.2.142DatalogWrite . . . . .	860
11.1.2.143DEGREES_PER_RADIAN . . . . .	860
11.1.2.144DISPLAY_BUSY . . . . .	860
11.1.2.145DISPLAY_CHAR . . . . .	860
11.1.2.146DISPLAY_CONTRAST_DEFAULT . . . . .	860
11.1.2.147DISPLAY_CONTRAST_MAX . . . . .	860
11.1.2.148DISPLAY_ERASE_ALL . . . . .	861
11.1.2.149DISPLAY_ERASE_LINE . . . . .	861
11.1.2.150DISPLAY_FILL_REGION . . . . .	861
11.1.2.151DISPLAY_FRAME . . . . .	861
11.1.2.152DISPLAY_HEIGHT . . . . .	861

11.1.2.153	DISPLAY_HORIZONTAL_LINE	861
11.1.2.154	DISPLAY_MENUICONS_X_DIFF	861
11.1.2.155	DISPLAY_MENUICONS_X_OFFS	861
11.1.2.156	DISPLAY_MENUICONS_Y	861
11.1.2.157	DISPLAY_ON	861
11.1.2.158	DISPLAY_PIXEL	861
11.1.2.159	DISPLAY_POPUP	862
11.1.2.160	DISPLAY_REFRESH	862
11.1.2.161	DISPLAY_REFRESH_DISABLED	862
11.1.2.162	DISPLAY_VERTICAL_LINE	862
11.1.2.163	DISPLAY_WIDTH	862
11.1.2.164	DisplayExecuteFunction	862
11.1.2.165	DisplayModuleID	862
11.1.2.166	DisplayModuleName	862
11.1.2.167	DisplayOffsetContrast	862
11.1.2.168	DisplayOffsetDisplay	862
11.1.2.169	DisplayOffsetEraseMask	863
11.1.2.170	DisplayOffsetFlags	863
11.1.2.171	DisplayOffsetNormal	863
11.1.2.172	DisplayOffsetPBitmaps	863
11.1.2.173	DisplayOffsetPFont	863
11.1.2.174	DisplayOffsetPFunc	863
11.1.2.175	DisplayOffsetPMenuIcons	863
11.1.2.176	DisplayOffsetPMenuText	863
11.1.2.177	DisplayOffsetPopup	863
11.1.2.178	DisplayOffsetPScreens	863
11.1.2.179	DisplayOffsetPStatusIcons	863
11.1.2.180	DisplayOffsetPStatusText	864
11.1.2.181	DisplayOffsetPStepIcons	864
11.1.2.182	DisplayOffsetPTextLines	864
11.1.2.183	DisplayOffsetStatusIcons	864
11.1.2.184	DisplayOffsetStepIcons	864
11.1.2.185	DisplayOffsetTextLinesCenterFlags	864
11.1.2.186	DisplayOffsetUpdateMask	864
11.1.2.187	DIST_CMD_CUSTOM	864
11.1.2.188	DIST_CMD_GP2D12	864



11.1.2.189DIST_CMD_GP2D120 . . . . .	864
11.1.2.190DIST_CMD_GP2YA02 . . . . .	864
11.1.2.191DIST_CMD_GP2YA21 . . . . .	865
11.1.2.192DIST_REG_DIST . . . . .	865
11.1.2.193DIST_REG_DIST1 . . . . .	865
11.1.2.194DIST_REG_DIST_MAX . . . . .	865
11.1.2.195DIST_REG_DIST_MIN . . . . .	865
11.1.2.196DIST_REG_MODULE_TYPE . . . . .	865
11.1.2.197DIST_REG_NUM_POINTS . . . . .	865
11.1.2.198DIST_REG_VOLT . . . . .	865
11.1.2.199DIST_REG_VOLT1 . . . . .	865
11.1.2.200DRAW_OPT_CLEAR . . . . .	865
11.1.2.201DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN . . . . .	865
11.1.2.202DRAW_OPT_CLEAR_PIXELS . . . . .	866
11.1.2.203DRAW_OPT_CLEAR_SCREEN_MODES . . . . .	866
11.1.2.204DRAW_OPT_CLEAR_WHOLE_SCREEN . . . . .	866
11.1.2.205DRAW_OPT_FILL_SHAPE . . . . .	866
11.1.2.206DRAW_OPT_FONT_DIR_B2TL . . . . .	866
11.1.2.207DRAW_OPT_FONT_DIR_B2TR . . . . .	866
11.1.2.208DRAW_OPT_FONT_DIR_L2RB . . . . .	866
11.1.2.209DRAW_OPT_FONT_DIR_L2RT . . . . .	866
11.1.2.210DRAW_OPT_FONT_DIR_R2LB . . . . .	867
11.1.2.211DRAW_OPT_FONT_DIR_R2LT . . . . .	867
11.1.2.212DRAW_OPT_FONT_DIR_T2BL . . . . .	867
11.1.2.213DRAW_OPT_FONT_DIR_T2BR . . . . .	867
11.1.2.214DRAW_OPT_FONT_DIRECTIONS . . . . .	867
11.1.2.215DRAW_OPT_FONT_WRAP . . . . .	867
11.1.2.216DRAW_OPT_INVERT . . . . .	867
11.1.2.217DRAW_OPT_LOGICAL_AND . . . . .	867
11.1.2.218DRAW_OPT_LOGICAL_COPY . . . . .	868
11.1.2.219DRAW_OPT_LOGICAL_OPERATIONS . . . . .	868
11.1.2.220DRAW_OPT_LOGICAL_OR . . . . .	868
11.1.2.221DRAW_OPT_LOGICAL_XOR . . . . .	868
11.1.2.222DRAW_OPT_NORMAL . . . . .	868
11.1.2.223DRAW_OPT_POLYGON_POLYLINE . . . . .	868
11.1.2.224DrawCircle . . . . .	868

11.1.2.225	DrawEllipse	868
11.1.2.226	DrawFont	868
11.1.2.227	DrawGraphic	869
11.1.2.228	DrawGraphicArray	869
11.1.2.229	DrawLine	869
11.1.2.230	DrawPoint	869
11.1.2.231	IDrawPolygon	869
11.1.2.232	DrawRect	869
11.1.2.233	DrawText	869
11.1.2.234	EOF	869
11.1.2.235	ERR_ARG	869
11.1.2.236	ERR_BAD_POOL_SIZE	869
11.1.2.237	ERR_BAD_PTR	870
11.1.2.238	ERR_CLUMP_COUNT	870
11.1.2.239	ERR_COMM_BUFFER_FULL	870
11.1.2.240	ERR_COMM_BUS_ERR	870
11.1.2.241	ERR_COMM_CHAN_INVALID	870
11.1.2.242	ERR_COMM_CHAN_NOT_READY	870
11.1.2.243	ERR_DEFAULT_OFFSETS	870
11.1.2.244	ERR_FILE	870
11.1.2.245	ERR_INSANE_OFFSET	870
11.1.2.246	ERR_INSTR	870
11.1.2.247	ERR_INVALID_FIELD	870
11.1.2.248	ERR_INVALID_PORT	871
11.1.2.249	ERR_INVALID_QUEUE	871
11.1.2.250	ERR_INVALID_SIZE	871
11.1.2.251	ERR_LOADER_ERR	871
11.1.2.252	ERR_MEM	871
11.1.2.253	ERR_MEMMGR_FAIL	871
11.1.2.254	ERR_NO_ACTIVE_CLUMP	871
11.1.2.255	ERR_NO_CODE	871
11.1.2.256	ERR_NO_PROG	871
11.1.2.257	ERR_NON_FATAL	871
11.1.2.258	ERR_RC_BAD_PACKET	871
11.1.2.259	ERR_RC_FAILED	872
11.1.2.260	ERR_RC_ILLEGAL_VAL	872

11.1.2.26	IERR_RC_UNKNOWN_CMD	872
11.1.2.26	2ERR_SPOTCHECK_FAIL	872
11.1.2.26	3ERR_VER	872
11.1.2.26	4FALSE	872
11.1.2.26	5FileClose	872
11.1.2.26	6FileDelete	872
11.1.2.26	7FileFindFirst	872
11.1.2.26	8FileFindNext	872
11.1.2.26	9FileOpenAppend	872
11.1.2.27	0FileOpenRead	873
11.1.2.27	1FileOpenReadLinear	873
11.1.2.27	2FileOpenWrite	873
11.1.2.27	3FileOpenWriteLinear	873
11.1.2.27	4FileOpenWriteNonLinear	873
11.1.2.27	5FileRead	873
11.1.2.27	6FileRename	873
11.1.2.27	7FileResize	873
11.1.2.27	8FileResolveHandle	873
11.1.2.27	9FileSeek	873
11.1.2.28	0FileWrite	873
11.1.2.28	1FRAME_SELECT	874
11.1.2.28	2FREQUENCY_MAX	874
11.1.2.28	3FREQUENCY_MIN	874
11.1.2.28	4GetStartTick	874
11.1.2.28	5GL_CAMERA_DEPTH	874
11.1.2.28	6GL_CIRCLE	874
11.1.2.28	7GL_CIRCLE_SIZE	874
11.1.2.28	8GL_CULL_BACK	874
11.1.2.28	9GL_CULL_FRONT	874
11.1.2.29	0GL_CULL_MODE	874
11.1.2.29	1GL_CULL_NONE	875
11.1.2.29	2GL_LINE	875
11.1.2.29	3GL_POINT	875
11.1.2.29	4GL_POLYGON	875
11.1.2.29	5GL_ROTATE_X	875
11.1.2.29	6GL_ROTATE_Y	875

11.1.2.297GL_ROTATE_Z . . . . .	875
11.1.2.298GL_SCALE_X . . . . .	876
11.1.2.299GL_SCALE_Y . . . . .	876
11.1.2.300GL_SCALE_Z . . . . .	876
11.1.2.301GL_TRANSLATE_X . . . . .	876
11.1.2.302GL_TRANSLATE_Y . . . . .	876
11.1.2.303GL_TRANSLATE_Z . . . . .	876
11.1.2.304GL_ZOOM_FACTOR . . . . .	876
11.1.2.305HS_BAUD_115200 . . . . .	876
11.1.2.306HS_BAUD_1200 . . . . .	877
11.1.2.307HS_BAUD_14400 . . . . .	877
11.1.2.308HS_BAUD_19200 . . . . .	877
11.1.2.309HS_BAUD_230400 . . . . .	877
11.1.2.310HS_BAUD_2400 . . . . .	877
11.1.2.311HS_BAUD_28800 . . . . .	877
11.1.2.312HS_BAUD_3600 . . . . .	877
11.1.2.313HS_BAUD_38400 . . . . .	877
11.1.2.314HS_BAUD_460800 . . . . .	877
11.1.2.315HS_BAUD_4800 . . . . .	877
11.1.2.316HS_BAUD_57600 . . . . .	877
11.1.2.317HS_BAUD_7200 . . . . .	878
11.1.2.318HS_BAUD_76800 . . . . .	878
11.1.2.319HS_BAUD_921600 . . . . .	878
11.1.2.320HS_BAUD_9600 . . . . .	878
11.1.2.321HS_CMD_READY . . . . .	878
11.1.2.322HS_CTRL_EXIT . . . . .	878
11.1.2.323HS_CTRL_INIT . . . . .	878
11.1.2.324HS_CTRL_UART . . . . .	878
11.1.2.325HS_DISABLE . . . . .	878
11.1.2.326HS_ENABLE . . . . .	879
11.1.2.327HS_INIT_RECEIVER . . . . .	879
11.1.2.328HS_INITIALISE . . . . .	879
11.1.2.329HS_MODE_10_STOP . . . . .	879
11.1.2.330HS_MODE_15_STOP . . . . .	879
11.1.2.331HS_MODE_20_STOP . . . . .	879
11.1.2.332HS_MODE_5_DATA . . . . .	879

11.1.2.333HS_MODE_6_DATA . . . . .	879
11.1.2.334HS_MODE_7_DATA . . . . .	879
11.1.2.335HS_MODE_7E1 . . . . .	879
11.1.2.336HS_MODE_8_DATA . . . . .	879
11.1.2.337HS_MODE_8N1 . . . . .	880
11.1.2.338HS_MODE_E_PARITY . . . . .	880
11.1.2.339HS_MODE_M_PARITY . . . . .	880
11.1.2.340HS_MODE_N_PARITY . . . . .	880
11.1.2.341HS_MODE_O_PARITY . . . . .	880
11.1.2.342HS_MODE_S_PARITY . . . . .	880
11.1.2.343HS_SEND_DATA . . . . .	880
11.1.2.344HS_UPDATE . . . . .	880
11.1.2.345HT_CH1_A . . . . .	880
11.1.2.346HT_CH1_B . . . . .	880
11.1.2.347HT_CH2_A . . . . .	881
11.1.2.348HT_CH2_B . . . . .	881
11.1.2.349HT_CH3_A . . . . .	881
11.1.2.350HT_CH3_B . . . . .	881
11.1.2.351HT_CH4_A . . . . .	881
11.1.2.352HT_CH4_B . . . . .	881
11.1.2.353HT_CMD_COLOR2_50HZ . . . . .	881
11.1.2.354HT_CMD_COLOR2_60HZ . . . . .	881
11.1.2.355HT_CMD_COLOR2_ACTIVE . . . . .	881
11.1.2.356HT_CMD_COLOR2_BLCAL . . . . .	881
11.1.2.357HT_CMD_COLOR2_FAR . . . . .	881
11.1.2.358HT_CMD_COLOR2_LED_HI . . . . .	882
11.1.2.359HT_CMD_COLOR2_LED_LOW . . . . .	882
11.1.2.360HT_CMD_COLOR2_NEAR . . . . .	882
11.1.2.361HT_CMD_COLOR2_PASSIVE . . . . .	882
11.1.2.362HT_CMD_COLOR2_RAW . . . . .	882
11.1.2.363HT_CMD_COLOR2_WBCAL . . . . .	882
11.1.2.364HTIR2_MODE_1200 . . . . .	882
11.1.2.365HTIR2_MODE_600 . . . . .	882
11.1.2.366HTIR2_REG_AC01 . . . . .	882
11.1.2.367HTIR2_REG_AC02 . . . . .	882
11.1.2.368HTIR2_REG_AC03 . . . . .	882

11.1.2.369HTIR2_REG_AC04 . . . . .	883
11.1.2.370HTIR2_REG_AC05 . . . . .	883
11.1.2.371HTIR2_REG_ACDIR . . . . .	883
11.1.2.372HTIR2_REG_DC01 . . . . .	883
11.1.2.373HTIR2_REG_DC02 . . . . .	883
11.1.2.374HTIR2_REG_DC03 . . . . .	883
11.1.2.375HTIR2_REG_DC04 . . . . .	883
11.1.2.376HTIR2_REG_DC05 . . . . .	883
11.1.2.377HTIR2_REG_DCAVG . . . . .	883
11.1.2.378HTIR2_REG_DCDIR . . . . .	883
11.1.2.379HTIR2_REG_MODE . . . . .	883
11.1.2.380I2C_REG_CMD . . . . .	884
11.1.2.381I2C_REG_DEVICE_ID . . . . .	884
11.1.2.382I2C_REG_VENDOR_ID . . . . .	884
11.1.2.383I2C_REG_VERSION . . . . .	884
11.1.2.384IN_1 . . . . .	884
11.1.2.385IN_2 . . . . .	884
11.1.2.386IN_3 . . . . .	884
11.1.2.387IN_4 . . . . .	884
11.1.2.388IN_MODE_ANGLESTEP . . . . .	885
11.1.2.389IN_MODE_BOOLEAN . . . . .	885
11.1.2.390IN_MODE_CELSIUS . . . . .	885
11.1.2.391IN_MODE_FAHRENHEIT . . . . .	885
11.1.2.392IN_MODE_MODEMASK . . . . .	885
11.1.2.393IN_MODE_PCTFULLSCALE . . . . .	885
11.1.2.394IN_MODE_PERIODCOUNTER . . . . .	885
11.1.2.395IN_MODE_RAW . . . . .	885
11.1.2.396IN_MODE_SLOPEMASK . . . . .	885
11.1.2.397IN_MODE_TRANSITIONCNT . . . . .	885
11.1.2.398IN_TYPE_ANGLE . . . . .	885
11.1.2.399IN_TYPE_COLORBLUE . . . . .	886
11.1.2.400IN_TYPE_COLOREXIT . . . . .	886
11.1.2.401IN_TYPE_COLORFULL . . . . .	886
11.1.2.402IN_TYPE_COLORGREEN . . . . .	886
11.1.2.403IN_TYPE_COLORNONE . . . . .	886
11.1.2.404IN_TYPE_COLORRED . . . . .	886

11.1.2.405IN_TYPE_CUSTOM . . . . .	886
11.1.2.406IN_TYPE_HISPEED . . . . .	886
11.1.2.407IN_TYPE_LIGHT_ACTIVE . . . . .	886
11.1.2.408IN_TYPE_LIGHT_INACTIVE . . . . .	886
11.1.2.409IN_TYPE_LOWSPEED . . . . .	886
11.1.2.410IN_TYPE_LOWSPEED_9V . . . . .	887
11.1.2.411IN_TYPE_NO_SENSOR . . . . .	887
11.1.2.412IN_TYPE_REFLECTION . . . . .	887
11.1.2.413IN_TYPE_SOUND_DB . . . . .	887
11.1.2.414IN_TYPE_SOUND_DBA . . . . .	887
11.1.2.415IN_TYPE_SWITCH . . . . .	887
11.1.2.416IN_TYPE_TEMPERATURE . . . . .	887
11.1.2.417INPUT_BLACKCOLOR . . . . .	887
11.1.2.418INPUT_BLANK . . . . .	887
11.1.2.419INPUT_BLUE . . . . .	887
11.1.2.420INPUT_BLUECOLOR . . . . .	887
11.1.2.421INPUT_CAL_POINT_0 . . . . .	888
11.1.2.422INPUT_CAL_POINT_1 . . . . .	888
11.1.2.423INPUT_CAL_POINT_2 . . . . .	888
11.1.2.424INPUT_CUSTOM9V . . . . .	888
11.1.2.425INPUT_CUSTOMACTIVE . . . . .	888
11.1.2.426INPUT_CUSTOMINACTIVE . . . . .	888
11.1.2.427INPUT_DIGI0 . . . . .	888
11.1.2.428INPUT_DIGI1 . . . . .	888
11.1.2.429INPUT_GREEN . . . . .	888
11.1.2.430INPUT_GREENCOLOR . . . . .	888
11.1.2.431INPUT_INVALID_DATA . . . . .	888
11.1.2.432INPUT_NO_OF_COLORS . . . . .	889
11.1.2.433INPUT_NO_OF_POINTS . . . . .	889
11.1.2.434INPUT_RED . . . . .	889
11.1.2.435INPUT_REDCOLOR . . . . .	889
11.1.2.436INPUT_RESETCAL . . . . .	889
11.1.2.437INPUT_RUNNINGCAL . . . . .	889
11.1.2.438INPUT_SENSORCAL . . . . .	889
11.1.2.439INPUT_SENSOROFF . . . . .	889
11.1.2.440INPUT_STARTCAL . . . . .	889

11.1.2.44	INPUT_WHITECOLOR	889
11.1.2.44	INPUT_YELLOWCOLOR	890
11.1.2.44	InputMode	890
11.1.2.44	InputModuleID	890
11.1.2.44	InputModuleName	890
11.1.2.44	InputOffsetADRaw	890
11.1.2.44	InputOffsetColorADRaw	890
11.1.2.44	InputOffsetColorCalibration	890
11.1.2.44	InputOffsetColorCalibrationState	890
11.1.2.45	InputOffsetColorCalLimits	890
11.1.2.45	InputOffsetColorSensorBoolean	890
11.1.2.45	InputOffsetColorSensorRaw	890
11.1.2.45	InputOffsetColorSensorValue	891
11.1.2.45	InputOffsetCustomActiveStatus	891
11.1.2.45	InputOffsetCustomPctFullScale	891
11.1.2.45	InputOffsetCustomZeroOffset	891
11.1.2.45	InputOffsetDigiPinsDir	891
11.1.2.45	InputOffsetDigiPinsIn	891
11.1.2.45	InputOffsetDigiPinsOut	891
11.1.2.46	InputOffsetInvalidData	891
11.1.2.46	InputOffsetSensorBoolean	891
11.1.2.46	InputOffsetSensorMode	891
11.1.2.46	InputOffsetSensorRaw	891
11.1.2.46	InputOffsetSensorType	892
11.1.2.46	InputOffsetSensorValue	892
11.1.2.46	INT_MAX	892
11.1.2.46	INT_MIN	892
11.1.2.46	INTF_BTOFF	892
11.1.2.46	INTF_BTON	892
11.1.2.47	INTF_CONNECT	892
11.1.2.47	INTF_CONNECTBYNAME	892
11.1.2.47	INTF_CONNECTREQ	892
11.1.2.47	INTF_DISCONNECT	892
11.1.2.47	INTF_DISCONNECTALL	892
11.1.2.47	INTF_EXTREAD	893
11.1.2.47	INTF_FACTORYRESET	893



11.1.2.477	INTF_OPENSTREAM	893
11.1.2.478	INTF_PINREQ	893
11.1.2.479	INTF_REMOVEDEVICE	893
11.1.2.480	INTF_SEARCH	893
11.1.2.481	INTF_SENDDATA	893
11.1.2.482	INTF_SENDFILE	893
11.1.2.483	INTF_SETBTNAME	893
11.1.2.484	INTF_SETCMDMODE	893
11.1.2.485	INTF_STOPSEARCH	893
11.1.2.486	INTF_VISIBILITY	894
11.1.2.487	InvalidData	894
11.1.2.488	IOCTRL_BOOT	894
11.1.2.489	IOCTRL_POWERDOWN	894
11.1.2.490	IOCtrlModuleID	894
11.1.2.491	IOCtrlModuleName	894
11.1.2.492	IOCtrlOffsetPowerOn	894
11.1.2.493	IOMapRead	894
11.1.2.494	IOMapReadByID	894
11.1.2.495	IOMapWrite	894
11.1.2.496	IOMapWriteByID	894
11.1.2.497	KeepAlive	895
11.1.2.498	LCD_LINE1	895
11.1.2.499	LCD_LINE2	895
11.1.2.500	LCD_LINE3	895
11.1.2.501	LCD_LINE4	896
11.1.2.502	LCD_LINE5	896
11.1.2.503	LCD_LINE6	896
11.1.2.504	LCD_LINE7	896
11.1.2.505	LCD_LINE8	896
11.1.2.506	LDR_APPENDNOTPOSSIBLE	896
11.1.2.507	LDR_BTBUSY	896
11.1.2.508	LDR_BTCONNECTFAIL	897
11.1.2.509	LDR_BTTIMEOUT	897
11.1.2.510	LDR_CMD_BOOTCMD	897
11.1.2.511	LDR_CMD_BTFACTORYRESET	897
11.1.2.512	LDR_CMD_BTGETADR	897

11.1.2.513LDR_CMD_CLOSE . . . . .	897
11.1.2.514LDR_CMD_CLOSEMODHANDLE . . . . .	897
11.1.2.515LDR_CMD_CROPDATAFILE . . . . .	897
11.1.2.516LDR_CMD_DELETE . . . . .	897
11.1.2.517LDR_CMD_DELETEUSERFLASH . . . . .	897
11.1.2.518LDR_CMD_DEVICEINFO . . . . .	897
11.1.2.519LDR_CMD_FINDFIRST . . . . .	898
11.1.2.520LDR_CMD_FINDFIRSTMODULE . . . . .	898
11.1.2.521LDR_CMD_FINDNEXT . . . . .	898
11.1.2.522LDR_CMD_FINDNEXTMODULE . . . . .	898
11.1.2.523LDR_CMD_IOMAPREAD . . . . .	898
11.1.2.524LDR_CMD_IOMAPWRITE . . . . .	898
11.1.2.525LDR_CMD_OPENAPPENDDATA . . . . .	898
11.1.2.526LDR_CMD_OPENREAD . . . . .	898
11.1.2.527LDR_CMD_OPENREADLINEAR . . . . .	898
11.1.2.528LDR_CMD_OPENWRITE . . . . .	898
11.1.2.529LDR_CMD_OPENWRITEDATA . . . . .	898
11.1.2.530LDR_CMD_OPENWRITELINEAR . . . . .	899
11.1.2.531LDR_CMD_POLLCMD . . . . .	899
11.1.2.532LDR_CMD_POLLCMDLEN . . . . .	899
11.1.2.533LDR_CMD_READ . . . . .	899
11.1.2.534LDR_CMD_RENAMEFILE . . . . .	899
11.1.2.535LDR_CMD_RESIZEDATAFILE . . . . .	899
11.1.2.536LDR_CMD_SEEKFROMCURRENT . . . . .	899
11.1.2.537LDR_CMD_SEEKFROMEND . . . . .	899
11.1.2.538LDR_CMD_SEEKFROMSTART . . . . .	899
11.1.2.539LDR_CMD_SETBRICKNAME . . . . .	899
11.1.2.540LDR_CMD_VERSIONS . . . . .	899
11.1.2.541LDR_CMD_WRITE . . . . .	900
11.1.2.542LDR_ENDOFFILE . . . . .	900
11.1.2.543LDR_EOFEXPECTED . . . . .	900
11.1.2.544LDR_FILEEXISTS . . . . .	900
11.1.2.545LDR_FILEISBUSY . . . . .	900
11.1.2.546LDR_FILEISFULL . . . . .	900
11.1.2.547LDR_FILENOTFOUND . . . . .	900
11.1.2.548LDR_FILETX_CLOSEERROR . . . . .	900

11.1.2.54	LDR_FILETX_DSTEXISTS	900
11.1.2.55	LDR_FILETX_SRCMISSING	900
11.1.2.55	LDR_FILETX_STREAMERROR	900
11.1.2.55	LDR_FILETX_TIMEOUT	901
11.1.2.55	LDR_HANDLEALREADYCLOSED	901
11.1.2.55	LDR_ILLEGALFILENAME	901
11.1.2.55	LDR_ILLEGALHANDLE	901
11.1.2.55	LDR_INPROGRESS	901
11.1.2.55	LDR_INVALIDSEEK	901
11.1.2.55	LDR_MODULENOTFOUND	901
11.1.2.55	LDR_NOLINEARSPACE	901
11.1.2.56	LDR_NOMOREFILES	901
11.1.2.56	LDR_NOMOREHANDLES	901
11.1.2.56	LDR_NOSPACE	901
11.1.2.56	LDR_NOTLINEARFILE	902
11.1.2.56	LDR_NOWRITEBUFFERS	902
11.1.2.56	LDR_OUTOFBOUNDARY	902
11.1.2.56	LDR_REQPIN	902
11.1.2.56	LDR_SUCCESS	902
11.1.2.56	LDR_UNDEFINEDERROR	902
11.1.2.56	ListFiles	902
11.1.2.57	LoaderExecuteFunction	902
11.1.2.57	LoaderModuleID	902
11.1.2.57	LoaderModuleName	902
11.1.2.57	LoaderOffsetFreeUserFlash	903
11.1.2.57	LoaderOffsetPFunc	903
11.1.2.57	LONG_MAX	903
11.1.2.57	LONG_MIN	903
11.1.2.57	LOWSPEED_CH_NOT_READY	903
11.1.2.57	LOWSPEED_COMMUNICATING	903
11.1.2.57	LOWSPEED_DATA_RECEIVED	903
11.1.2.58	LOWSPEED_DONE	903
11.1.2.58	LOWSPEED_ERROR	903
11.1.2.58	LOWSPEED_IDLE	903
11.1.2.58	LOWSPEED_INIT	903
11.1.2.58	LOWSPEED_LOAD_BUFFER	904

11.1.2.585	LOWSPEED_NO_ERROR	904
11.1.2.586	LOWSPEED_RECEIVING	904
11.1.2.587	LOWSPEED_RX_ERROR	904
11.1.2.588	LOWSPEED_TRANSMITTING	904
11.1.2.589	LOWSPEED_TX_ERROR	904
11.1.2.590	LowSpeedModuleID	904
11.1.2.591	LowSpeedModuleName	904
11.1.2.592	LowSpeedOffsetChannelState	904
11.1.2.593	LowSpeedOffsetErrorType	904
11.1.2.594	LowSpeedOffsetInBufBuf	904
11.1.2.595	LowSpeedOffsetInBufBytesToRx	905
11.1.2.596	LowSpeedOffsetInBufInPtr	905
11.1.2.597	LowSpeedOffsetInBufOutPtr	905
11.1.2.598	LowSpeedOffsetMode	905
11.1.2.599	LowSpeedOffsetNoRestartOnRead	905
11.1.2.600	LowSpeedOffsetOutBufBuf	905
11.1.2.601	LowSpeedOffsetOutBufBytesToRx	905
11.1.2.602	LowSpeedOffsetOutBufInPtr	905
11.1.2.603	LowSpeedOffsetOutBufOutPtr	905
11.1.2.604	LowSpeedOffsetSpeed	905
11.1.2.605	LowSpeedOffsetState	905
11.1.2.606	LR_COULD_NOT_SAVE	906
11.1.2.607	LR_ENTRY_REMOVED	906
11.1.2.608	LR_STORE_IS_FULL	906
11.1.2.609	LR_SUCCESS	906
11.1.2.610	LR_UNKNOWN_ADDR	906
11.1.2.611	LSREAD_NO_RESTART_1	906
11.1.2.612	LSREAD_NO_RESTART_2	906
11.1.2.613	LSREAD_NO_RESTART_3	906
11.1.2.614	LSREAD_NO_RESTART_4	906
11.1.2.615	LSREAD_NO_RESTART_MASK	906
11.1.2.616	LSREAD_RESTART_ALL	906
11.1.2.617	LSREAD_RESTART_NONE	907
11.1.2.618	MAILBOX1	907
11.1.2.619	MAILBOX10	907
11.1.2.620	MAILBOX2	907

11.1.2.621MAILBOX3 . . . . .	907
11.1.2.622MAILBOX4 . . . . .	907
11.1.2.623MAILBOX5 . . . . .	907
11.1.2.624MAILBOX6 . . . . .	907
11.1.2.625MAILBOX7 . . . . .	907
11.1.2.626MAILBOX8 . . . . .	907
11.1.2.627MAILBOX9 . . . . .	908
11.1.2.628MAX_BT_MSG_SIZE . . . . .	908
11.1.2.629MENUICON_CENTER . . . . .	908
11.1.2.630MENUICON_LEFT . . . . .	908
11.1.2.631MENUICON_RIGHT . . . . .	908
11.1.2.632MENUICONS . . . . .	908
11.1.2.633MENUTEXT . . . . .	908
11.1.2.634MessageRead . . . . .	908
11.1.2.635MessageWrite . . . . .	908
11.1.2.636MIN_1 . . . . .	908
11.1.2.637MS_1 . . . . .	908
11.1.2.638MS_10 . . . . .	909
11.1.2.639MS_100 . . . . .	909
11.1.2.640MS_150 . . . . .	909
11.1.2.641MS_2 . . . . .	909
11.1.2.642MS_20 . . . . .	909
11.1.2.643MS_200 . . . . .	909
11.1.2.644MS_250 . . . . .	909
11.1.2.645MS_3 . . . . .	909
11.1.2.646MS_30 . . . . .	909
11.1.2.647MS_300 . . . . .	910
11.1.2.648MS_350 . . . . .	910
11.1.2.649MS_4 . . . . .	910
11.1.2.650MS_40 . . . . .	910
11.1.2.651MS_400 . . . . .	910
11.1.2.652MS_450 . . . . .	910
11.1.2.653MS_5 . . . . .	910
11.1.2.654MS_50 . . . . .	910
11.1.2.655MS_500 . . . . .	910
11.1.2.656MS_6 . . . . .	911

11.1.2.657MS_60 . . . . .	911
11.1.2.658MS_600 . . . . .	911
11.1.2.659MS_7 . . . . .	911
11.1.2.660MS_70 . . . . .	911
11.1.2.661MS_700 . . . . .	911
11.1.2.662MS_8 . . . . .	911
11.1.2.663MS_80 . . . . .	911
11.1.2.664MS_800 . . . . .	911
11.1.2.665MS_9 . . . . .	911
11.1.2.666MS_90 . . . . .	911
11.1.2.667MS_900 . . . . .	912
11.1.2.668MS_CMD_ADPA_OFF . . . . .	912
11.1.2.669MS_CMD_ADPA_ON . . . . .	912
11.1.2.670MS_CMD_DEENERGIZED . . . . .	912
11.1.2.671MS_CMD_ENERGIZED . . . . .	912
11.1.2.672NA . . . . .	912
11.1.2.673NO_ERR . . . . .	912
11.1.2.674NO_OF_BTNS . . . . .	912
11.1.2.675NormalizedValue . . . . .	913
11.1.2.676NRLINK_CMD_2400 . . . . .	913
11.1.2.677NRLINK_CMD_4800 . . . . .	913
11.1.2.678NRLINK_CMD_FLUSH . . . . .	913
11.1.2.679NRLINK_CMD_IR_LONG . . . . .	913
11.1.2.680NRLINK_CMD_IR_SHORT . . . . .	913
11.1.2.681NRLINK_CMD_RUN_MACRO . . . . .	913
11.1.2.682NRLINK_CMD_SET_PF . . . . .	913
11.1.2.683NRLINK_CMD_SET_RCX . . . . .	913
11.1.2.684NRLINK_CMD_SET_TRAIN . . . . .	913
11.1.2.685NRLINK_CMD_TX_RAW . . . . .	913
11.1.2.686NRLINK_REG_BYTES . . . . .	914
11.1.2.687NRLINK_REG_DATA . . . . .	914
11.1.2.688NRLINK_REG_EEPROM . . . . .	914
11.1.2.689NULL . . . . .	914
11.1.2.690OPARR_MAX . . . . .	914
11.1.2.691OPARR_MEAN . . . . .	914
11.1.2.692OPARR_MIN . . . . .	914

11.1.2.693OPARR_SORT	914
11.1.2.694OPARR_STD	914
11.1.2.695OPARR_SUM	914
11.1.2.696OPARR_SUMSQR	914
11.1.2.697OUT_A	915
11.1.2.698OUT_AB	915
11.1.2.699OUT_ABC	915
11.1.2.700OUT_AC	915
11.1.2.701OUT_B	915
11.1.2.702OUT_BC	915
11.1.2.703OUT_C	915
11.1.2.704OUT_MODE_BRAKE	916
11.1.2.705OUT_MODE_COAST	916
11.1.2.706OUT_MODE_MOTORON	916
11.1.2.707OUT_MODE_REGMETHOD	916
11.1.2.708OUT_MODE_REGULATED	916
11.1.2.709OUT_OPTION_HOLDATLIMIT	916
11.1.2.710OUT_OPTION_RAMPDOWNTOLIMIT	916
11.1.2.711OUT_REGMODE_IDLE	916
11.1.2.712OUT_REGMODE_SPEED	916
11.1.2.713OUT_REGMODE_SYNC	917
11.1.2.714OUT_RUNSTATE_HOLD	917
11.1.2.715OUT_RUNSTATE_IDLE	917
11.1.2.716OUT_RUNSTATE_RAMPDOWN	917
11.1.2.717OUT_RUNSTATE_RAMPUP	917
11.1.2.718OUT_RUNSTATE_RUNNING	917
11.1.2.719OutputMode	917
11.1.2.720OutputModuleID	917
11.1.2.721OutputModuleName	917
11.1.2.722OutputOffsetActualSpeed	917
11.1.2.723OutputOffsetBlockTachoCount	918
11.1.2.724OutputOffsetFlags	918
11.1.2.725OutputOffsetMode	918
11.1.2.726OutputOffsetMotorRPM	918
11.1.2.727OutputOffsetOptions	918
11.1.2.728OutputOffsetOverloaded	918

11.1.2.729OutputOffsetPwnFreq . . . . .	918
11.1.2.730OutputOffsetRegDParameter . . . . .	918
11.1.2.731OutputOffsetRegIParameter . . . . .	918
11.1.2.732OutputOffsetRegMode . . . . .	918
11.1.2.733OutputOffsetRegPParameter . . . . .	918
11.1.2.734OutputOffsetRotationCount . . . . .	919
11.1.2.735OutputOffsetRunState . . . . .	919
11.1.2.736OutputOffsetSpeed . . . . .	919
11.1.2.737OutputOffsetSyncTurnParameter . . . . .	919
11.1.2.738OutputOffsetTachoCount . . . . .	919
11.1.2.739OutputOffsetTachoLimit . . . . .	919
11.1.2.740OutputOptions . . . . .	919
11.1.2.741Overload . . . . .	919
11.1.2.742PF_CHANNEL_1 . . . . .	919
11.1.2.743PF_CHANNEL_2 . . . . .	920
11.1.2.744PF_CHANNEL_3 . . . . .	920
11.1.2.745PF_CHANNEL_4 . . . . .	920
11.1.2.746PF_CMD_BRAKE . . . . .	920
11.1.2.747PF_CMD_FWD . . . . .	920
11.1.2.748PF_CMD_REV . . . . .	920
11.1.2.749PF_CMD_STOP . . . . .	920
11.1.2.750PF_CST_CLEAR1_CLEAR2 . . . . .	920
11.1.2.751PF_CST_CLEAR1_SET2 . . . . .	921
11.1.2.752PF_CST_DECREMENT_PWM . . . . .	921
11.1.2.753PF_CST_FULL_FWD . . . . .	921
11.1.2.754PF_CST_FULL_REV . . . . .	921
11.1.2.755PF_CST_INCREMENT_PWM . . . . .	921
11.1.2.756PF_CST_SET1_CLEAR2 . . . . .	921
11.1.2.757PF_CST_SET1_SET2 . . . . .	921
11.1.2.758PF_CST_TOGGLE_DIR . . . . .	921
11.1.2.759PF_FUNC_CLEAR . . . . .	921
11.1.2.760PF_FUNC_NOCHANGE . . . . .	921
11.1.2.761PF_FUNC_SET . . . . .	922
11.1.2.762PF_FUNC_TOGGLE . . . . .	922
11.1.2.763PF_MODE_COMBO_DIRECT . . . . .	922
11.1.2.764PF_MODE_COMBO_PWM . . . . .	922



11.1.2.765PF_MODE_SINGLE_OUTPUT_CST . . . . .	922
11.1.2.766PF_MODE_SINGLE_OUTPUT_PWM . . . . .	922
11.1.2.767PF_MODE_SINGLE_PIN_CONT . . . . .	922
11.1.2.768PF_MODE_SINGLE_PIN_TIME . . . . .	922
11.1.2.769PF_MODE_TRAIN . . . . .	922
11.1.2.770PF_OUT_A . . . . .	922
11.1.2.771PF_OUT_B . . . . .	923
11.1.2.772PF_PIN_C1 . . . . .	923
11.1.2.773PF_PIN_C2 . . . . .	923
11.1.2.774PF_PWM_BRAKE . . . . .	923
11.1.2.775PF_PWM_FLOAT . . . . .	923
11.1.2.776PF_PWM_FWD1 . . . . .	923
11.1.2.777PF_PWM_FWD2 . . . . .	923
11.1.2.778PF_PWM_FWD3 . . . . .	923
11.1.2.779PF_PWM_FWD4 . . . . .	923
11.1.2.780PF_PWM_FWD5 . . . . .	924
11.1.2.781PF_PWM_FWD6 . . . . .	924
11.1.2.782PF_PWM_FWD7 . . . . .	924
11.1.2.783PF_PWM_REV1 . . . . .	924
11.1.2.784PF_PWM_REV2 . . . . .	924
11.1.2.785PF_PWM_REV3 . . . . .	924
11.1.2.786PF_PWM_REV4 . . . . .	924
11.1.2.787PF_PWM_REV5 . . . . .	924
11.1.2.788PF_PWM_REV6 . . . . .	924
11.1.2.789PF_PWM_REV7 . . . . .	925
11.1.2.790PI . . . . .	925
11.1.2.791PID_0 . . . . .	925
11.1.2.792PID_1 . . . . .	925
11.1.2.793PID_2 . . . . .	925
11.1.2.794PID_3 . . . . .	925
11.1.2.795PID_4 . . . . .	925
11.1.2.796PID_5 . . . . .	925
11.1.2.797PID_6 . . . . .	925
11.1.2.798PID_7 . . . . .	925
11.1.2.799Power . . . . .	926
11.1.2.800PROG_ABORT . . . . .	926

11.1.2.80	IPROG_ERROR	926
11.1.2.80	2PROG_IDLE	926
11.1.2.80	3PROG_OK	926
11.1.2.80	4PROG_RESET	926
11.1.2.80	5PROG_RUNNING	926
11.1.2.80	6PSP_BTNSET1_DOWN	926
11.1.2.80	7PSP_BTNSET1_L3	926
11.1.2.80	8PSP_BTNSET1_LEFT	926
11.1.2.80	9PSP_BTNSET1_R3	926
11.1.2.81	0PSP_BTNSET1_RIGHT	927
11.1.2.81	1PSP_BTNSET1_UP	927
11.1.2.81	2PSP_BTNSET2_CIRCLE	927
11.1.2.81	3PSP_BTNSET2_CROSS	927
11.1.2.81	4PSP_BTNSET2_L1	927
11.1.2.81	5PSP_BTNSET2_L2	927
11.1.2.81	6PSP_BTNSET2_R1	927
11.1.2.81	7PSP_BTNSET2_R2	927
11.1.2.81	8PSP_BTNSET2_SQUARE	927
11.1.2.81	9PSP_BTNSET2_TRIANGLE	927
11.1.2.82	0PSP_CMD_ANALOG	927
11.1.2.82	1PSP_CMD_DIGITAL	928
11.1.2.82	2PSP_REG_BTNSET1	928
11.1.2.82	3PSP_REG_BTNSET2	928
11.1.2.82	4PSP_REG_XLEFT	928
11.1.2.82	5PSP_REG_XRIGHT	928
11.1.2.82	6PSP_REG_YLEFT	928
11.1.2.82	7PSP_REG_YRIGHT	928
11.1.2.82	8RADIANS_PER_DEGREE	928
11.1.2.82	9RAND_MAX	928
11.1.2.83	0RandomNumber	928
11.1.2.83	1RawValue	928
11.1.2.83	2RCX_AbsVarOp	929
11.1.2.83	3RCX_AndVarOp	929
11.1.2.83	4RCX_AutoOffOp	929
11.1.2.83	5RCX_BatteryLevelOp	929
11.1.2.83	6RCX_BatteryLevelSrc	929

11.1.2.837RCX_BootModeOp . . . . .	929
11.1.2.838RCX_CalibrateEventOp . . . . .	929
11.1.2.839RCX_ClearAllEventsOp . . . . .	929
11.1.2.840RCX_ClearCounterOp . . . . .	929
11.1.2.841RCX_ClearMsgOp . . . . .	929
11.1.2.842RCX_ClearSensorOp . . . . .	929
11.1.2.843RCX_ClearSoundOp . . . . .	930
11.1.2.844RCX_ClearTimerOp . . . . .	930
11.1.2.845RCX_ClickCounterSrc . . . . .	930
11.1.2.846RCX_ConstantSrc . . . . .	930
11.1.2.847RCX_CounterSrc . . . . .	930
11.1.2.848RCX_DatalogOp . . . . .	930
11.1.2.849RCX_DatalogRawDirectSrc . . . . .	930
11.1.2.850RCX_DatalogRawIndirectSrc . . . . .	930
11.1.2.851RCX_DatalogSrcDirectSrc . . . . .	930
11.1.2.852RCX_DatalogSrcIndirectSrc . . . . .	931
11.1.2.853RCX_DatalogValueDirectSrc . . . . .	931
11.1.2.854RCX_DatalogValueIndirectSrc . . . . .	931
11.1.2.855RCX_DecCounterOp . . . . .	931
11.1.2.856RCX_DeleteSubOp . . . . .	931
11.1.2.857RCX_DeleteSubsOp . . . . .	931
11.1.2.858RCX_DeleteTaskOp . . . . .	931
11.1.2.859RCX_DeleteTasksOp . . . . .	931
11.1.2.860RCX_DirectEventOp . . . . .	931
11.1.2.861RCX_DisplayOp . . . . .	931
11.1.2.862RCX_DivVarOp . . . . .	931
11.1.2.863RCX_DurationSrc . . . . .	932
11.1.2.864RCX_EventStateSrc . . . . .	932
11.1.2.865RCX_FirmwareVersionSrc . . . . .	932
11.1.2.866RCX_GlobalMotorStatusSrc . . . . .	932
11.1.2.867RCX_GOutputDirOp . . . . .	932
11.1.2.868RCX_GOutputModeOp . . . . .	932
11.1.2.869RCX_GOutputPowerOp . . . . .	932
11.1.2.870RCX_HysteresisSrc . . . . .	932
11.1.2.871RCX_IncCounterOp . . . . .	932
11.1.2.872RCX_IndirectVarSrc . . . . .	932

11.1.2.873RCX_InputBooleanSrc . . . . .	932
11.1.2.874RCX_InputModeOp . . . . .	933
11.1.2.875RCX_InputModeSrc . . . . .	933
11.1.2.876RCX_InputRawSrc . . . . .	933
11.1.2.877RCX_InputTypeOp . . . . .	933
11.1.2.878RCX_InputTypeSrc . . . . .	933
11.1.2.879RCX_InputValueSrc . . . . .	933
11.1.2.880RCX_IRModeOp . . . . .	933
11.1.2.881RCX_LightOp . . . . .	933
11.1.2.882RCX_LowerThresholdSrc . . . . .	933
11.1.2.883RCX_LSblinkTimeOp . . . . .	933
11.1.2.884RCX_LSCalibrateOp . . . . .	933
11.1.2.885RCX_LSHysteresisOp . . . . .	934
11.1.2.886RCX_LSLowerThreshOp . . . . .	934
11.1.2.887RCX_LSupperThreshOp . . . . .	934
11.1.2.888RCX_MessageOp . . . . .	934
11.1.2.889RCX_MessageSrc . . . . .	934
11.1.2.890RCX_MulVarOp . . . . .	934
11.1.2.891RCX_MuteSoundOp . . . . .	934
11.1.2.892RCX_OnOffFloatOp . . . . .	934
11.1.2.893RCX_OrVarOp . . . . .	934
11.1.2.894RCX_OUT_A . . . . .	934
11.1.2.895RCX_OUT_AB . . . . .	935
11.1.2.896RCX_OUT_ABC . . . . .	935
11.1.2.897RCX_OUT_AC . . . . .	935
11.1.2.898RCX_OUT_B . . . . .	935
11.1.2.899RCX_OUT_BC . . . . .	935
11.1.2.900RCX_OUT_C . . . . .	935
11.1.2.901RCX_OUT_FLOAT . . . . .	935
11.1.2.902RCX_OUT_FULL . . . . .	935
11.1.2.903RCX_OUT_FWD . . . . .	936
11.1.2.904RCX_OUT_HALF . . . . .	936
11.1.2.905RCX_OUT_LOW . . . . .	936
11.1.2.906RCX_OUT_OFF . . . . .	936
11.1.2.907RCX_OUT_ON . . . . .	936
11.1.2.908RCX_OUT_REV . . . . .	936

11.1.2.90RCX_OUT_TOGGLE . . . . .	936
11.1.2.910RCX_OutputDirOp . . . . .	936
11.1.2.911RCX_OutputPowerOp . . . . .	936
11.1.2.912RCX_OutputStatusSrc . . . . .	937
11.1.2.913RCX_PBTurnOffOp . . . . .	937
11.1.2.914RCX_PingOp . . . . .	937
11.1.2.915RCX_PlaySoundOp . . . . .	937
11.1.2.916RCX_PlayToneOp . . . . .	937
11.1.2.917RCX_PlayToneVarOp . . . . .	937
11.1.2.918RCX_PollMemoryOp . . . . .	937
11.1.2.919RCX_PollOp . . . . .	937
11.1.2.920RCX_ProgramSlotSrc . . . . .	937
11.1.2.921RCX_RandomSrc . . . . .	937
11.1.2.922RCX_RemoteKeysReleased . . . . .	937
11.1.2.923RCX_RemoteOp . . . . .	938
11.1.2.924RCX_RemoteOutABackward . . . . .	938
11.1.2.925RCX_RemoteOutAForward . . . . .	938
11.1.2.926RCX_RemoteOutBBackward . . . . .	938
11.1.2.927RCX_RemoteOutBForward . . . . .	938
11.1.2.928RCX_RemoteOutCBackward . . . . .	938
11.1.2.929RCX_RemoteOutCForward . . . . .	938
11.1.2.930RCX_RemotePBMessage1 . . . . .	938
11.1.2.931RCX_RemotePBMessage2 . . . . .	938
11.1.2.932RCX_RemotePBMessage3 . . . . .	938
11.1.2.933RCX_RemotePlayASound . . . . .	938
11.1.2.934RCX_RemoteSelProgram1 . . . . .	939
11.1.2.935RCX_RemoteSelProgram2 . . . . .	939
11.1.2.936RCX_RemoteSelProgram3 . . . . .	939
11.1.2.937RCX_RemoteSelProgram4 . . . . .	939
11.1.2.938RCX_RemoteSelProgram5 . . . . .	939
11.1.2.939RCX_RemoteStopOutOff . . . . .	939
11.1.2.940RCX_ScoutCounterLimitSrc . . . . .	939
11.1.2.941RCX_ScoutEventFBsrc . . . . .	939
11.1.2.942RCX_ScoutLightParamsSrc . . . . .	939
11.1.2.943RCX_ScoutOp . . . . .	939
11.1.2.944RCX_ScoutRulesOp . . . . .	939

11.1.2.945RCX_ScoutRulesSrc . . . . .	940
11.1.2.946RCX_ScoutTimerLimitSrc . . . . .	940
11.1.2.947RCX_SelectProgramOp . . . . .	940
11.1.2.948RCX_SendUARTDataOp . . . . .	940
11.1.2.949RCX_SetCounterOp . . . . .	940
11.1.2.950RCX_SetDatalogOp . . . . .	940
11.1.2.951RCX_SetEventOp . . . . .	940
11.1.2.952RCX_SetFeedbackOp . . . . .	940
11.1.2.953RCX_SetPriorityOp . . . . .	940
11.1.2.954RCX_SetSourceValueOp . . . . .	940
11.1.2.955RCX_SetTimerLimitOp . . . . .	940
11.1.2.956RCX_SetVarOp . . . . .	941
11.1.2.957RCX_SetWatchOp . . . . .	941
11.1.2.958RCX_SgnVarOp . . . . .	941
11.1.2.959RCX_SoundOp . . . . .	941
11.1.2.960RCX_StartTaskOp . . . . .	941
11.1.2.961RCX_StopAllTasksOp . . . . .	941
11.1.2.962RCX_StopTaskOp . . . . .	941
11.1.2.963RCX_SubVarOp . . . . .	941
11.1.2.964RCX_SumVarOp . . . . .	941
11.1.2.965RCX_TaskEventsSrc . . . . .	941
11.1.2.966RCX_TenMSTimerSrc . . . . .	941
11.1.2.967RCX_TimerSrc . . . . .	942
11.1.2.968RCX_UARTSetupSrc . . . . .	942
11.1.2.969RCX_UnlockFirmOp . . . . .	942
11.1.2.970RCX_UnlockOp . . . . .	942
11.1.2.971RCX_UnmuteSoundOp . . . . .	942
11.1.2.972RCX_UploadDatalogOp . . . . .	942
11.1.2.973RCX_UpperThresholdSrc . . . . .	942
11.1.2.974RCX_VariableSrc . . . . .	942
11.1.2.975RCX_ViewSourceValOp . . . . .	942
11.1.2.976RCX_VLLOp . . . . .	942
11.1.2.977RCX_WatchSrc . . . . .	943
11.1.2.978ReadButton . . . . .	943
11.1.2.979ReadSemData . . . . .	943
11.1.2.980RegDValue . . . . .	943

11.1.2.981RegIValue . . . . .	943
11.1.2.982RegMode . . . . .	943
11.1.2.983RegPValue . . . . .	943
11.1.2.984RESET_ALL . . . . .	944
11.1.2.985RESET_BLOCK_COUNT . . . . .	944
11.1.2.986RESET_BLOCKANDTACHO . . . . .	944
11.1.2.987RESET_COUNT . . . . .	944
11.1.2.988RESET_NONE . . . . .	944
11.1.2.989RESET_ROTATION_COUNT . . . . .	944
11.1.2.990RICArg . . . . .	944
11.1.2.991RICImgPoint . . . . .	944
11.1.2.992RICImgRect . . . . .	945
11.1.2.993RICMapArg . . . . .	945
11.1.2.994RICMapElement . . . . .	945
11.1.2.995RICMapFunction . . . . .	945
11.1.2.996RICOpCircle . . . . .	946
11.1.2.997RICOpCopyBits . . . . .	946
11.1.2.998RICOpDescription . . . . .	946
11.1.2.999RICOpEllipse . . . . .	946
11.1.2.1000RICOpLine . . . . .	947
11.1.2.1001RICOpNumBox . . . . .	947
11.1.2.1002RICOpPixel . . . . .	947
11.1.2.1003RICOpPolygon . . . . .	947
11.1.2.1004RICOpRect . . . . .	948
11.1.2.1005RICOpSprite . . . . .	948
11.1.2.1006RICOpVarMap . . . . .	948
11.1.2.1007RICPolygonPoints . . . . .	948
11.1.2.1008RICSpriteData . . . . .	949
11.1.2.1009ROTATE_QUEUE . . . . .	949
11.1.2.1010RotationCount . . . . .	949
11.1.2.1011RunState . . . . .	949
11.1.2.1012SAMPLERATE_DEFAULT . . . . .	949
11.1.2.1013SAMPLERATE_MAX . . . . .	949
11.1.2.1014SAMPLERATE_MIN . . . . .	949
11.1.2.1015ScaledValue . . . . .	950
11.1.2.1016SCHAR_MAX . . . . .	950

11.1.2.101	SCOUT_FXR_MIN	950
11.1.2.101	SCOUT_FXR_ALARM	950
11.1.2.101	SCOUT_FXR_BUG	950
11.1.2.102	SCOUT_FXR_NONE	950
11.1.2.102	SCOUT_FXR_RANDOM	950
11.1.2.102	SCOUT_FXR_SCIENCE	950
11.1.2.102	SCOUT_LIGHT_OFF	950
11.1.2.102	SCOUT_LIGHT_ON	950
11.1.2.102	SCOUT_LR_AVOID	951
11.1.2.102	SCOUT_LR_IGNORE	951
11.1.2.102	SCOUT_LR_OFF_WHEN	951
11.1.2.102	SCOUT_LR_SEEK_DARK	951
11.1.2.102	SCOUT_LR_SEEK_LIGHT	951
11.1.2.103	SCOUT_LR_WAIT_FOR	951
11.1.2.103	SCOUT_MODE_POWER	951
11.1.2.103	SCOUT_MODE_STANDALONE	951
11.1.2.103	SCOUT_MR_CIRCLE_LEFT	951
11.1.2.103	SCOUT_MR_CIRCLE_RIGHT	951
11.1.2.103	SCOUT_MR_FORWARD	952
11.1.2.103	SCOUT_MR_LOOP_A	952
11.1.2.103	SCOUT_MR_LOOP_AB	952
11.1.2.103	SCOUT_MR_LOOP_B	952
11.1.2.103	SCOUT_MR_NO_MOTION	952
11.1.2.104	SCOUT_MR_ZIGZAG	952
11.1.2.104	SCOUT_SNDSET_ALARM	952
11.1.2.104	SCOUT_SNDSET_BASIC	952
11.1.2.104	SCOUT_SNDSET_BUG	952
11.1.2.104	SCOUT_SNDSET_NONE	952
11.1.2.104	SCOUT_SNDSET_RANDOM	952
11.1.2.104	SCOUT_SNDSET_SCIENCE	953
11.1.2.104	SCOUT_SOUND_1_BLINK	953
11.1.2.104	SCOUT_SOUND_2_BLINK	953
11.1.2.104	SCOUT_SOUND_COUNTER1	953
11.1.2.105	SCOUT_SOUND_COUNTER2	953
11.1.2.105	SCOUT_SOUND_ENTER_BRIGHT	953
11.1.2.105	SCOUT_SOUND_ENTER_DARK	953



11.1.2.1055	SCOUT_SOUND_ENTER_NORMAL	953
11.1.2.1055	SCOUT_SOUND_ENTERSA	953
11.1.2.1055	SCOUT_SOUND_KEYERROR	953
11.1.2.1055	SCOUT_SOUND_MAIL_RECEIVED	953
11.1.2.1055	SCOUT_SOUND_NONE	954
11.1.2.1055	SCOUT_SOUND_REMOTE	954
11.1.2.1055	SCOUT_SOUND_SPECIAL1	954
11.1.2.1065	SCOUT_SOUND_SPECIAL2	954
11.1.2.1065	SCOUT_SOUND_SPECIAL3	954
11.1.2.1065	SCOUT_SOUND_TIMER1	954
11.1.2.1065	SCOUT_SOUND_TIMER2	954
11.1.2.1065	SCOUT_SOUND_TIMER3	954
11.1.2.1065	SCOUT_SOUND_TOUCH1_PRES	954
11.1.2.1065	SCOUT_SOUND_TOUCH1_REL	954
11.1.2.1065	SCOUT_SOUND_TOUCH2_PRES	954
11.1.2.1065	SCOUT_SOUND_TOUCH2_REL	955
11.1.2.1065	SCOUT_TGS_LONG	955
11.1.2.1075	SCOUT_TGS_MEDIUM	955
11.1.2.1075	SCOUT_TGS_SHORT	955
11.1.2.1075	SCOUT_TR_AVOID	955
11.1.2.1075	SCOUT_TR_IGNORE	955
11.1.2.1075	SCOUT_TR_OFF_WHEN	955
11.1.2.1075	SCOUT_TR_REVERSE	955
11.1.2.1075	SCOUT_TR_WAIT_FOR	955
11.1.2.1075	SCREEN_BACKGROUND	955
11.1.2.1075	SCREEN_LARGE	956
11.1.2.1075	SCREEN_MODE_CLEAR	956
11.1.2.1085	SCREEN_MODE_RESTORE	956
11.1.2.1085	SCREEN_SMALL	956
11.1.2.1085	SCREENS	956
11.1.2.1085	SEC_1	956
11.1.2.1085	SEC_10	956
11.1.2.1085	SEC_15	957
11.1.2.1085	SEC_2	957
11.1.2.1085	SEC_20	957
11.1.2.1085	SEC_3	957

11.1.2.1089	SEC_30	957
11.1.2.1090	SEC_4	957
11.1.2.1091	SEC_5	957
11.1.2.1092	SEC_6	958
11.1.2.1093	SEC_7	958
11.1.2.1094	SEC_8	958
11.1.2.1095	SEC_9	958
11.1.2.1096	SetScreenMode	958
11.1.2.1097	SetSleepTimeoutVal	958
11.1.2.1098	SHRT_MAX	958
11.1.2.1099	SHRT_MIN	958
11.1.2.1100	SIZE_OF_BDADDR	958
11.1.2.1101	SIZE_OF_BRICK_NAME	958
11.1.2.1102	SIZE_OF_BT_CONNECT_TABLE	959
11.1.2.1103	SIZE_OF_BT_DEVICE_TABLE	959
11.1.2.1104	SIZE_OF_BT_NAME	959
11.1.2.1105	SIZE_OF_BT_PINCODE	959
11.1.2.1106	SIZE_OF_BTBUF	959
11.1.2.1107	SIZE_OF_CLASS_OF_DEVICE	959
11.1.2.1108	SIZE_OF_HSBUF	959
11.1.2.1109	SIZE_OF_USBBUF	959
11.1.2.1110	SIZE_OF_USBDATA	959
11.1.2.1111	SOUND_CLICK	959
11.1.2.1112	SOUND_DOUBLE_BEEP	959
11.1.2.1113	SOUND_DOWN	960
11.1.2.1114	SOUND_FAST_UP	960
11.1.2.1115	SOUND_FLAGS_IDLE	960
11.1.2.1116	SOUND_FLAGS_RUNNING	960
11.1.2.1117	SOUND_FLAGS_UPDATE	960
11.1.2.1118	SOUND_LOW_BEEP	960
11.1.2.1119	SOUND_MODE_LOOP	960
11.1.2.1120	SOUND_MODE_ONCE	960
11.1.2.1121	SOUND_MODE_TONE	961
11.1.2.1122	SOUND_STATE_FILE	961
11.1.2.1123	SOUND_STATE_IDLE	961
11.1.2.1124	SOUND_STATE_STOP	961

11.1.2.112	SOUND_STATE_TONE	961
11.1.2.112	SOUND_UP	961
11.1.2.112	SoundGetState	961
11.1.2.112	SoundModuleID	961
11.1.2.112	SoundModuleName	962
11.1.2.113	SoundOffsetDuration	962
11.1.2.113	SoundOffsetFlags	962
11.1.2.113	SoundOffsetFreq	962
11.1.2.113	SoundOffsetMode	962
11.1.2.113	SoundOffsetSampleRate	962
11.1.2.113	SoundOffsetSoundFilename	962
11.1.2.113	SoundOffsetState	962
11.1.2.113	SoundOffsetVolume	962
11.1.2.113	SoundPlayFile	962
11.1.2.113	SoundPlayTone	963
11.1.2.114	SoundSetState	963
11.1.2.114	SPECIALS	963
11.1.2.114	STAT_COMM_PENDING	963
11.1.2.114	STAT_MSG_EMPTY_MAILBOX	963
11.1.2.114	STATUSICON_BATTERY	963
11.1.2.114	STATUSICON_BLUETOOTH	963
11.1.2.114	STATUSICON_USB	963
11.1.2.114	STATUSICON_VM	963
11.1.2.114	STATUSICONS	963
11.1.2.114	STATUSTEXT	963
11.1.2.115	STEPICON_1	964
11.1.2.115	STEPICON_2	964
11.1.2.115	STEPICON_3	964
11.1.2.115	STEPICON_4	964
11.1.2.115	STEPICON_5	964
11.1.2.115	STEPICONS	964
11.1.2.115	STEPLINE	964
11.1.2.115	STOP_REQ	964
11.1.2.115	TachoCount	964
11.1.2.115	TachoLimit	964
11.1.2.116	TEMP_FQ_1	964

11.1.2.116TEMP_FQ_2	965
11.1.2.116TEMP_FQ_4	965
11.1.2.116TEMP_FQ_6	965
11.1.2.116TEMP_I2C_ADDRESS	965
11.1.2.116TEMP_OS_ONESHOT	965
11.1.2.116TEMP_POL_HIGH	965
11.1.2.116TEMP_POL_LOW	965
11.1.2.116TEMP_REG_CONFIG	965
11.1.2.116TEMP_REG_TEMP	965
11.1.2.117TEMP_REG_THIGH	965
11.1.2.117TEMP_REG_TLOW	965
11.1.2.117TEMP_RES_10BIT	966
11.1.2.117TEMP_RES_11BIT	966
11.1.2.117TEMP_RES_12BIT	966
11.1.2.117TEMP_RES_9BIT	966
11.1.2.117TEMP_SD_CONTINUOUS	966
11.1.2.117TEMP_SD_SHUTDOWN	966
11.1.2.117TEMP_TM_COMPARATOR	966
11.1.2.117TEMP_TM_INTERRUPT	966
11.1.2.118EXTLINE_1	966
11.1.2.118EXTLINE_2	966
11.1.2.118EXTLINE_3	967
11.1.2.118EXTLINE_4	967
11.1.2.118EXTLINE_5	967
11.1.2.118EXTLINE_6	967
11.1.2.118EXTLINE_7	967
11.1.2.118EXTLINE_8	967
11.1.2.118EXTLINES	967
11.1.2.118TIMES_UP	967
11.1.2.119TONE_A3	967
11.1.2.119TONE_A4	967
11.1.2.119TONE_A5	967
11.1.2.119TONE_A6	968
11.1.2.119TONE_A7	968
11.1.2.119TONE_AS3	968
11.1.2.119TONE_AS4	968

11.1.2.119TONE_AS5 . . . . .	968
11.1.2.119TONE_AS6 . . . . .	968
11.1.2.119TONE_AS7 . . . . .	968
11.1.2.120TONE_B3 . . . . .	968
11.1.2.120TONE_B4 . . . . .	968
11.1.2.120TONE_B5 . . . . .	968
11.1.2.120TONE_B6 . . . . .	968
11.1.2.120TONE_B7 . . . . .	969
11.1.2.120TONE_C4 . . . . .	969
11.1.2.120TONE_C5 . . . . .	969
11.1.2.120TONE_C6 . . . . .	969
11.1.2.120TONE_C7 . . . . .	969
11.1.2.120TONE_CS4 . . . . .	969
11.1.2.121TONE_CS5 . . . . .	969
11.1.2.121TONE_CS6 . . . . .	969
11.1.2.121TONE_CS7 . . . . .	969
11.1.2.121TONE_D4 . . . . .	970
11.1.2.121TONE_D5 . . . . .	970
11.1.2.121TONE_D6 . . . . .	970
11.1.2.121TONE_D7 . . . . .	970
11.1.2.121TONE_DS4 . . . . .	970
11.1.2.121TONE_DS5 . . . . .	970
11.1.2.121TONE_DS6 . . . . .	970
11.1.2.122TONE_DS7 . . . . .	970
11.1.2.122TONE_E4 . . . . .	970
11.1.2.122TONE_E5 . . . . .	970
11.1.2.122TONE_E6 . . . . .	971
11.1.2.122TONE_E7 . . . . .	971
11.1.2.122TONE_F4 . . . . .	971
11.1.2.122TONE_F5 . . . . .	971
11.1.2.122TONE_F6 . . . . .	971
11.1.2.122TONE_F7 . . . . .	971
11.1.2.122TONE_FS4 . . . . .	971
11.1.2.123TONE_FS5 . . . . .	971
11.1.2.123TONE_FS6 . . . . .	971
11.1.2.123TONE_FS7 . . . . .	971

11.1.2.123 <del>B</del> ONE_G4 . . . . .	971
11.1.2.123 <del>B</del> ONE_G5 . . . . .	972
11.1.2.123 <del>B</del> ONE_G6 . . . . .	972
11.1.2.123 <del>B</del> ONE_G7 . . . . .	972
11.1.2.123 <del>T</del> ONE_GS4 . . . . .	972
11.1.2.123 <del>B</del> ONE_GS5 . . . . .	972
11.1.2.123 <del>B</del> ONE_GS6 . . . . .	972
11.1.2.124 <del>O</del> ONE_GS7 . . . . .	972
11.1.2.124 <del>T</del> OPLINE . . . . .	972
11.1.2.124 <del>B</del> RAIN_CHANNEL_1 . . . . .	972
11.1.2.124 <del>B</del> RAIN_CHANNEL_2 . . . . .	972
11.1.2.124 <del>B</del> RAIN_CHANNEL_3 . . . . .	973
11.1.2.124 <del>B</del> RAIN_CHANNEL_ALL . . . . .	973
11.1.2.124 <del>B</del> RAIN_FUNC_DECR_SPEED . . . . .	973
11.1.2.124 <del>B</del> RAIN_FUNC_INCR_SPEED . . . . .	973
11.1.2.124 <del>B</del> RAIN_FUNC_STOP . . . . .	973
11.1.2.124 <del>B</del> RAIN_FUNC_TOGGLE_LIGHT . . . . .	973
11.1.2.125 <del>T</del> URUE . . . . .	973
11.1.2.125 <del>T</del> urnRatio . . . . .	973
11.1.2.125 <del>T</del> ype . . . . .	974
11.1.2.125 <del>B</del> CHAR_MAX . . . . .	974
11.1.2.125 <del>B</del> F_PENDING_UPDATES . . . . .	974
11.1.2.125 <del>B</del> F_UPDATE_MODE . . . . .	974
11.1.2.125 <del>B</del> F_UPDATE_PID_VALUES . . . . .	974
11.1.2.125 <del>B</del> F_UPDATE_RESET_BLOCK_COUNT . . . . .	974
11.1.2.125 <del>B</del> F_UPDATE_RESET_COUNT . . . . .	974
11.1.2.125 <del>B</del> F_UPDATE_RESET_ROTATION_COUNT . . . . .	974
11.1.2.126 <del>B</del> F_UPDATE_SPEED . . . . .	974
11.1.2.126 <del>B</del> F_UPDATE_TACHO_LIMIT . . . . .	974
11.1.2.126 <del>B</del> I_BT_CONNECT_REQUEST . . . . .	974
11.1.2.126 <del>B</del> I_BT_ERROR_ATTENTION . . . . .	975
11.1.2.126 <del>B</del> I_BT_PIN_REQUEST . . . . .	975
11.1.2.126 <del>B</del> I_BT_STATE_CONNECTED . . . . .	975
11.1.2.126 <del>B</del> I_BT_STATE_OFF . . . . .	975
11.1.2.126 <del>B</del> I_BT_STATE_VISIBLE . . . . .	975
11.1.2.126 <del>B</del> I_BUTTON_ENTER . . . . .	975

11.1.2.1269I_BUTTON_EXIT . . . . .	975
11.1.2.1270I_BUTTON_LEFT . . . . .	975
11.1.2.1271I_BUTTON_NONE . . . . .	975
11.1.2.1272I_BUTTON_RIGHT . . . . .	975
11.1.2.1273I_FLAGS_BUSY . . . . .	976
11.1.2.1274I_FLAGS_DISABLE_EXIT . . . . .	976
11.1.2.1275I_FLAGS_DISABLE_LEFT_RIGHT_ENTER . . . . .	976
11.1.2.1276I_FLAGS_ENABLE_STATUS_UPDATE . . . . .	976
11.1.2.1277I_FLAGS_EXECUTE_LMS_FILE . . . . .	976
11.1.2.1278I_FLAGS_REDRAW_STATUS . . . . .	976
11.1.2.1279I_FLAGS_RESET_SLEEP_TIMER . . . . .	976
11.1.2.1280I_FLAGS_UPDATE . . . . .	976
11.1.2.1281I_STATE_BT_ERROR . . . . .	976
11.1.2.1282I_STATE_CONNECT_REQUEST . . . . .	976
11.1.2.1283I_STATE_DRAW_MENU . . . . .	976
11.1.2.1284I_STATE_ENTER_PRESSED . . . . .	977
11.1.2.1285I_STATE_EXECUTE_FILE . . . . .	977
11.1.2.1286I_STATE_EXECUTING_FILE . . . . .	977
11.1.2.1287I_STATE_EXIT_PRESSED . . . . .	977
11.1.2.1288I_STATE_INIT_DISPLAY . . . . .	977
11.1.2.1289I_STATE_INIT_INTRO . . . . .	977
11.1.2.1290I_STATE_INIT_LOW_BATTERY . . . . .	977
11.1.2.1291I_STATE_INIT_MENU . . . . .	977
11.1.2.1292I_STATE_INIT_WAIT . . . . .	977
11.1.2.1293I_STATE_LEFT_PRESSED . . . . .	977
11.1.2.1294I_STATE_LOW_BATTERY . . . . .	977
11.1.2.1295I_STATE_NEXT_MENU . . . . .	978
11.1.2.1296I_STATE_RIGHT_PRESSED . . . . .	978
11.1.2.1297I_STATE_TEST_BUTTONS . . . . .	978
11.1.2.1298I_VM_IDLE . . . . .	978
11.1.2.1299I_VM_RESET1 . . . . .	978
11.1.2.1300I_VM_RESET2 . . . . .	978
11.1.2.1301I_VM_RUN_FREE . . . . .	978
11.1.2.1302I_VM_RUN_PAUSE . . . . .	978
11.1.2.1303I_VM_RUN_SINGLE . . . . .	978
11.1.2.1304I_ModuleID . . . . .	978

11.1.2.1305	IModuleName	978
11.1.2.1306	INT_MAX	979
11.1.2.1307	IOffsetAbortFlag	979
11.1.2.1308	IOffsetBatteryState	979
11.1.2.1309	IOffsetBatteryVoltage	979
11.1.2.1310	IOffsetBluetoothState	979
11.1.2.1311	IOffsetButton	979
11.1.2.1312	IOffsetError	979
11.1.2.1313	IOffsetFlags	979
11.1.2.1314	IOffsetForceOff	979
11.1.2.1315	IOffsetLMSfilename	979
11.1.2.1316	IOffsetOBPPointer	979
11.1.2.1317	IOffsetPMenu	980
11.1.2.1318	IOffsetRechargeable	980
11.1.2.1319	IOffsetRunState	980
11.1.2.1320	IOffsetSleepTimeout	980
11.1.2.1321	IOffsetSleepTimer	980
11.1.2.1322	IOffsetState	980
11.1.2.1323	IOffsetUsbState	980
11.1.2.1324	IOffsetVolume	980
11.1.2.1325	LONG_MAX	980
11.1.2.1326	pdateCalibCacheInfo	980
11.1.2.1327	pdateFlags	980
11.1.2.1328	S_CMD_CONTINUOUS	981
11.1.2.1329	S_CMD_EVENTCAPTURE	981
11.1.2.1330	S_CMD_OFF	981
11.1.2.1331	S_CMD_SINGLESHOT	981
11.1.2.1332	S_CMD_WARMRESET	981
11.1.2.1333	S_REG_ACTUAL_ZERO	981
11.1.2.1334	S_REG_CM_INTERVAL	981
11.1.2.1335	S_REG_FACTORY_ACTUAL_ZERO	981
11.1.2.1336	S_REG_FACTORY_SCALE_DIVISOR	981
11.1.2.1337	S_REG_FACTORY_SCALE_FACTOR	981
11.1.2.1338	S_REG_MEASUREMENT_UNITS	981
11.1.2.1339	S_REG_SCALE_DIVISOR	982
11.1.2.1340	S_REG_SCALE_FACTOR	982



11.1.2.134	USB_CMD_READY	982
11.1.2.134	USB_PROTOCOL_OVERHEAD	982
11.1.2.134	USHRT_MAX	982
11.1.2.134	WriteSemData	982
11.2	NXCAPIDocs.h File Reference	983
11.2.1	Detailed Description	983
11.3	NXCDefs.h File Reference	984
11.3.1	Detailed Description	1045
11.3.2	Define Documentation	1045
11.3.2.1	_SENSOR_CFG	1045
11.3.2.2	Acos	1046
11.3.2.3	AcosD	1046
11.3.2.4	Asin	1046
11.3.2.5	AsinD	1047
11.3.2.6	Atan	1047
11.3.2.7	Atan2	1047
11.3.2.8	Atan2D	1048
11.3.2.9	AtanD	1048
11.3.2.10	Ceil	1048
11.3.2.11	Cos	1049
11.3.2.12	CosD	1049
11.3.2.13	Cosh	1049
11.3.2.14	CoshD	1050
11.3.2.15	Exp	1050
11.3.2.16	Floor	1050
11.3.2.17	Frac	1051
11.3.2.18	getc	1051
11.3.2.19	Log	1051
11.3.2.20	Log10	1052
11.3.2.21	MulDiv32	1052
11.3.2.22	Pow	1052
11.3.2.23	putc	1053
11.3.2.24	RICSetValue	1053
11.3.2.25	S1	1053
11.3.2.26	s16	1055
11.3.2.27	S2	1055

11.3.2.28 S3	1055
11.3.2.29 s32	1055
11.3.2.30 S4	1055
11.3.2.31 s8	1055
11.3.2.32 SEEK_CUR	1055
11.3.2.33 SEEK_END	1055
11.3.2.34 SEEK_SET	1056
11.3.2.35 SENSOR_1	1056
11.3.2.36 SENSOR_2	1056
11.3.2.37 SENSOR_3	1056
11.3.2.38 SENSOR_4	1056
11.3.2.39 SENSOR_CELSIUS	1056
11.3.2.40 SENSOR_COLORBLUE	1056
11.3.2.41 SENSOR_COLORFULL	1056
11.3.2.42 SENSOR_COLORGREEN	1056
11.3.2.43 SENSOR_COLORNONE	1056
11.3.2.44 SENSOR_COLORRED	1057
11.3.2.45 SENSOR_EDGE	1057
11.3.2.46 SENSOR_FAHRENHEIT	1057
11.3.2.47 SENSOR_LIGHT	1057
11.3.2.48 SENSOR_LOWSPEED	1057
11.3.2.49 SENSOR_LOWSPEED_9V	1057
11.3.2.50 SENSOR_MODE_BOOL	1057
11.3.2.51 SENSOR_MODE_CELSIUS	1057
11.3.2.52 SENSOR_MODE_EDGE	1057
11.3.2.53 SENSOR_MODE_FAHRENHEIT	1058
11.3.2.54 SENSOR_MODE_PERCENT	1058
11.3.2.55 SENSOR_MODE_PULSE	1058
11.3.2.56 SENSOR_MODE_RAW	1058
11.3.2.57 SENSOR_MODE_ROTATION	1058
11.3.2.58 SENSOR_NXTLIGHT	1058
11.3.2.59 SENSOR_PULSE	1058
11.3.2.60 SENSOR_ROTATION	1058
11.3.2.61 SENSOR_SOUND	1058
11.3.2.62 SENSOR_TOUCH	1059
11.3.2.63 SENSOR_TYPE_COLORBLUE	1059

11.3.2.64	SENSOR_TYPE_COLORFULL	1059
11.3.2.65	SENSOR_TYPE_COLORGREEN	1059
11.3.2.66	SENSOR_TYPE_COLORNONE	1059
11.3.2.67	SENSOR_TYPE_COLORRED	1059
11.3.2.68	SENSOR_TYPE_CUSTOM	1059
11.3.2.69	SENSOR_TYPE_HIGHSPEED	1059
11.3.2.70	SENSOR_TYPE_LIGHT	1059
11.3.2.71	SENSOR_TYPE_LIGHT_ACTIVE	1059
11.3.2.72	SENSOR_TYPE_LIGHT_INACTIVE	1060
11.3.2.73	SENSOR_TYPE_LOWSPEED	1060
11.3.2.74	SENSOR_TYPE_LOWSPEED_9V	1060
11.3.2.75	SENSOR_TYPE_NONE	1060
11.3.2.76	SENSOR_TYPE_ROTATION	1060
11.3.2.77	SENSOR_TYPE_SOUND_DB	1060
11.3.2.78	SENSOR_TYPE_SOUND_DBA	1060
11.3.2.79	SENSOR_TYPE_TEMPERATURE	1060
11.3.2.80	SENSOR_TYPE_TOUCH	1060
11.3.2.81	Sin	1061
11.3.2.82	SinD	1061
11.3.2.83	Sinh	1061
11.3.2.84	SinhD	1062
11.3.2.85	Sqrt	1062
11.3.2.86	Tan	1062
11.3.2.87	TanD	1063
11.3.2.88	Tanh	1063
11.3.2.89	TanhD	1063
11.3.2.90	Trunc	1064
11.3.2.91	u16	1064
11.3.2.92	u32	1064
11.3.2.93	u8	1064
11.3.3	Function Documentation	1064
11.3.3.1	abort	1064
11.3.3.2	AbortFlag	1064
11.3.3.3	abs	1065
11.3.3.4	acos	1065
11.3.3.5	acosd	1065

11.3.3.6	Acquire	1066
11.3.3.7	addr	1066
11.3.3.8	address	1066
11.3.3.9	ArrayBuild	1067
11.3.3.10	ArrayInit	1067
11.3.3.11	ArrayLen	1068
11.3.3.12	ArrayMax	1068
11.3.3.13	ArrayMean	1068
11.3.3.14	ArrayMin	1069
11.3.3.15	ArrayOp	1069
11.3.3.16	ArraySort	1069
11.3.3.17	ArrayStd	1070
11.3.3.18	ArraySubset	1070
11.3.3.19	ArraySum	1071
11.3.3.20	ArraySumSqr	1071
11.3.3.21	asin	1071
11.3.3.22	asind	1072
11.3.3.23	atan	1072
11.3.3.24	atan2	1073
11.3.3.25	atan2d	1073
11.3.3.26	atand	1073
11.3.3.27	atof	1074
11.3.3.28	atoi	1075
11.3.3.29	atol	1075
11.3.3.30	BatteryLevel	1076
11.3.3.31	BatteryState	1076
11.3.3.32	bcd2dec	1076
11.3.3.33	BluetoothState	1076
11.3.3.34	BluetoothStatus	1077
11.3.3.35	BluetoothWrite	1077
11.3.3.36	BrickDataBluecoreVersion	1077
11.3.3.37	BrickDataBtHardwareStatus	1077
11.3.3.38	BrickDataBtStateStatus	1078
11.3.3.39	BrickDataName	1078
11.3.3.40	BrickDataTimeoutValue	1078
11.3.3.41	BTConnectionClass	1078

11.3.3.42 BTConnectionHandleNum . . . . .	1079
11.3.3.43 BTConnectionLinkQuality . . . . .	1079
11.3.3.44 BTConnectionName . . . . .	1079
11.3.3.45 BTConnectionPinCode . . . . .	1080
11.3.3.46 BTConnectionStreamStatus . . . . .	1080
11.3.3.47 BTDeviceClass . . . . .	1080
11.3.3.48 BTDeviceCount . . . . .	1081
11.3.3.49 BTDeviceName . . . . .	1081
11.3.3.50 BTDeviceNameCount . . . . .	1081
11.3.3.51 BTDeviceStatus . . . . .	1081
11.3.3.52 BTInputBufferInPtr . . . . .	1082
11.3.3.53 BTInputBufferOutPtr . . . . .	1082
11.3.3.54 BTOutputBufferInPtr . . . . .	1082
11.3.3.55 BTOutputBufferOutPtr . . . . .	1082
11.3.3.56 ButtonCount . . . . .	1083
11.3.3.57 ButtonLongPressCount . . . . .	1083
11.3.3.58 ButtonLongReleaseCount . . . . .	1083
11.3.3.59 ButtonPressCount . . . . .	1084
11.3.3.60 ButtonPressed . . . . .	1084
11.3.3.61 ButtonReleaseCount . . . . .	1084
11.3.3.62 ButtonShortReleaseCount . . . . .	1085
11.3.3.63 ButtonState . . . . .	1085
11.3.3.64 ByteArrayToStr . . . . .	1085
11.3.3.65 ByteArrayToStrEx . . . . .	1086
11.3.3.66 ceil . . . . .	1086
11.3.3.67 CircleOut . . . . .	1086
11.3.3.68 ClearLine . . . . .	1087
11.3.3.69 ClearScreen . . . . .	1087
11.3.3.70 ClearSensor . . . . .	1087
11.3.3.71 CloseFile . . . . .	1087
11.3.3.72 Coast . . . . .	1088
11.3.3.73 CoastEx . . . . .	1088
11.3.3.74 ColorADRaw . . . . .	1088
11.3.3.75 ColorBoolean . . . . .	1089
11.3.3.76 ColorCalibration . . . . .	1089
11.3.3.77 ColorCalibrationState . . . . .	1090

11.3.3.78 ColorCalLimits . . . . .	1090
11.3.3.79 ColorSensorRaw . . . . .	1090
11.3.3.80 ColorSensorValue . . . . .	1091
11.3.3.81 CommandFlags . . . . .	1091
11.3.3.82 Copy . . . . .	1091
11.3.3.83 cos . . . . .	1092
11.3.3.84 cosd . . . . .	1092
11.3.3.85 cosh . . . . .	1093
11.3.3.86 coshd . . . . .	1093
11.3.3.87 CreateFile . . . . .	1093
11.3.3.88 CreateFileLinear . . . . .	1094
11.3.3.89 CreateFileNonLinear . . . . .	1094
11.3.3.90 CurrentTick . . . . .	1095
11.3.3.91 CustomSensorActiveStatus . . . . .	1095
11.3.3.92 CustomSensorPercentFullScale . . . . .	1095
11.3.3.93 CustomSensorZeroOffset . . . . .	1095
11.3.3.94 DeleteFile . . . . .	1096
11.3.3.95 DisplayContrast . . . . .	1096
11.3.3.96 DisplayDisplay . . . . .	1096
11.3.3.97 DisplayEraseMask . . . . .	1097
11.3.3.98 DisplayFlags . . . . .	1097
11.3.3.99 DisplayFont . . . . .	1097
11.3.3.100 DisplayTextLinesCenterFlags . . . . .	1097
11.3.3.101 DisplayUpdateMask . . . . .	1098
11.3.3.102 DISTNxDistance . . . . .	1098
11.3.3.103 DISTNxDistanceEx . . . . .	1098
11.3.3.104 DISTNxGP2D12 . . . . .	1098
11.3.3.105 DISTNxGP2D120 . . . . .	1099
11.3.3.106 DISTNxGP2D120Ex . . . . .	1099
11.3.3.107 DISTNxGP2D12Ex . . . . .	1099
11.3.3.108 DISTNxGP2YA02 . . . . .	1100
11.3.3.109 DISTNxGP2YA02Ex . . . . .	1100
11.3.3.110 DISTNxGP2YA21 . . . . .	1100
11.3.3.111 DISTNxGP2YA21Ex . . . . .	1101
11.3.3.112 DISTNxMaxDistance . . . . .	1101
11.3.3.113 DISTNxMaxDistanceEx . . . . .	1101

11.3.3.114DISTNxMinDistance . . . . .	1102
11.3.3.115DISTNxMinDistanceEx . . . . .	1102
11.3.3.116DISTNxModuleType . . . . .	1102
11.3.3.117DISTNxModuleTypeEx . . . . .	1103
11.3.3.118DISTNxNumPoints . . . . .	1103
11.3.3.119DISTNxNumPointsEx . . . . .	1103
11.3.3.120DISTNxVoltage . . . . .	1104
11.3.3.121DISTNxVoltageEx . . . . .	1104
11.3.3.122div . . . . .	1104
11.3.3.123EllipseOut . . . . .	1105
11.3.3.124ExitTo . . . . .	1105
11.3.3.125exp . . . . .	1106
11.3.3.126close . . . . .	1106
11.3.3.127feof . . . . .	1106
11.3.3.128flush . . . . .	1107
11.3.3.129getc . . . . .	1107
11.3.3.130gets . . . . .	1107
11.3.3.131FindFirstFile . . . . .	1108
11.3.3.132FindNextFile . . . . .	1108
11.3.3.133FirstTick . . . . .	1108
11.3.3.134Flatten . . . . .	1109
11.3.3.135FlattenVar . . . . .	1109
11.3.3.136Float . . . . .	1109
11.3.3.137floor . . . . .	1110
11.3.3.138Follows . . . . .	1110
11.3.3.139FontNumOut . . . . .	1110
11.3.3.140FontTextOut . . . . .	1111
11.3.3.141fopen . . . . .	1112
11.3.3.142ForceOff . . . . .	1112
11.3.3.143FormatNum . . . . .	1112
11.3.3.144fprintf . . . . .	1113
11.3.3.145fputc . . . . .	1113
11.3.3.146puts . . . . .	1113
11.3.3.147frac . . . . .	1114
11.3.3.148FreeMemory . . . . .	1114
11.3.3.149fseek . . . . .	1114

11.3.3.150	tell . . . . .	1115
11.3.3.151	GetBrickDataAddress . . . . .	1115
11.3.3.152	GetBTConnectionAddress . . . . .	1115
11.3.3.153	GetBTDeviceAddress . . . . .	1115
11.3.3.154	GetBTInputBuffer . . . . .	1116
11.3.3.155	GetBTOutputBuffer . . . . .	1116
11.3.3.156	getchar . . . . .	1116
11.3.3.157	GetDisplayNormal . . . . .	1116
11.3.3.158	GetDisplayPopup . . . . .	1117
11.3.3.159	GetHSInputBuffer . . . . .	1117
11.3.3.160	GetHSOutputBuffer . . . . .	1118
11.3.3.161	GetInput . . . . .	1118
11.3.3.162	GetLSInputBuffer . . . . .	1118
11.3.3.163	GetLSOutputBuffer . . . . .	1119
11.3.3.164	GetOutput . . . . .	1119
11.3.3.165	GetUSBInputBuffer . . . . .	1119
11.3.3.166	GetUSBOutputBuffer . . . . .	1120
11.3.3.167	GetUSBPollBuffer . . . . .	1120
11.3.3.168	glAddToAngleX . . . . .	1120
11.3.3.169	glAddToAngleY . . . . .	1120
11.3.3.170	glAddToAngleZ . . . . .	1121
11.3.3.171	glAddVertex . . . . .	1121
11.3.3.172	glBegin . . . . .	1121
11.3.3.173	glBeginObject . . . . .	1121
11.3.3.174	glBeginRender . . . . .	1121
11.3.3.175	glBox . . . . .	1122
11.3.3.176	glCallObject . . . . .	1122
11.3.3.177	glCos32768 . . . . .	1122
11.3.3.178	glCube . . . . .	1122
11.3.3.179	glEnd . . . . .	1123
11.3.3.180	glEndObject . . . . .	1123
11.3.3.181	glFinishRender . . . . .	1123
11.3.3.182	glInit . . . . .	1123
11.3.3.183	glObjectAction . . . . .	1123
11.3.3.184	glPyramid . . . . .	1124
11.3.3.185	glSet . . . . .	1124



11.3.3.186glSetAngleX . . . . .	1124
11.3.3.187glSetAngleY . . . . .	1124
11.3.3.188glSetAngleZ . . . . .	1125
11.3.3.189glSin32768 . . . . .	1125
11.3.3.190GraphicArrayOut . . . . .	1125
11.3.3.191GraphicArrayOutEx . . . . .	1126
11.3.3.192GraphicOut . . . . .	1126
11.3.3.193GraphicOutEx . . . . .	1127
11.3.3.194HSFlags . . . . .	1127
11.3.3.195HSInputBufferInPtr . . . . .	1127
11.3.3.196HSInputBufferOutPtr . . . . .	1128
11.3.3.197HSMode . . . . .	1128
11.3.3.198HSOutputBufferInPtr . . . . .	1128
11.3.3.199HSOutputBufferOutPtr . . . . .	1128
11.3.3.200HSSpeed . . . . .	1129
11.3.3.201HSSState . . . . .	1129
11.3.3.202HTIRTrain . . . . .	1129
11.3.3.203HTPFCComboDirect . . . . .	1129
11.3.3.204HTPFCComboPWM . . . . .	1130
11.3.3.205HTPFRawOutput . . . . .	1130
11.3.3.206HTPFRepeat . . . . .	1131
11.3.3.207HTPFSingleOutputCST . . . . .	1131
11.3.3.208HTPFSingleOutputPWM . . . . .	1132
11.3.3.209HTPFSinglePin . . . . .	1132
11.3.3.210HTPFTrain . . . . .	1133
11.3.3.211HTRCXAddToDatalog . . . . .	1133
11.3.3.212HTRCXBatteryLevel . . . . .	1133
11.3.3.213HTRCXCLEARAllEvents . . . . .	1134
11.3.3.214HTRCXCLEARCounter . . . . .	1134
11.3.3.215HTRCXCLEARMsg . . . . .	1134
11.3.3.216HTRCXCLEARSensor . . . . .	1134
11.3.3.217HTRCXCLEARSound . . . . .	1134
11.3.3.218HTRCXCLEARTimer . . . . .	1135
11.3.3.219HTRCXCreateDatalog . . . . .	1135
11.3.3.220HTRCXDecCounter . . . . .	1135
11.3.3.221HTRCXDeleteSub . . . . .	1135

11.3.3.22	HTRCXDeleteSubs	1135
11.3.3.23	HTRCXDeleteTask	1136
11.3.3.24	HTRCXDeleteTasks	1136
11.3.3.25	HTRCXDisableOutput	1136
11.3.3.26	HTRCXEnableOutput	1136
11.3.3.27	HTRCXEvent	1136
11.3.3.28	HTRCXFloat	1137
11.3.3.29	HTRCXFwd	1137
11.3.3.30	HTRCXIncCounter	1137
11.3.3.31	HTRCXInvertOutput	1137
11.3.3.32	HTRCXMuteSound	1138
11.3.3.33	HTRCXObvertOutput	1138
11.3.3.34	HTRCXOff	1138
11.3.3.35	HTRCXOn	1138
11.3.3.36	HTRCXOnFor	1138
11.3.3.37	HTRCXOnFwd	1139
11.3.3.38	HTRCXOnRev	1139
11.3.3.39	HTRCXPBTurnOff	1139
11.3.3.40	HTRCXPing	1139
11.3.3.41	HTRCXPlaySound	1140
11.3.3.42	HTRCXPlayTone	1140
11.3.3.43	HTRCXPlayToneVar	1140
11.3.3.44	HTRCXPoll	1140
11.3.3.45	HTRCXPollMemory	1141
11.3.3.46	HTRCXRemote	1141
11.3.3.47	HTRCXRev	1141
11.3.3.48	HTRCXSelectDisplay	1141
11.3.3.49	HTRCXSelectProgram	1142
11.3.3.50	HTRCXSendSerial	1142
11.3.3.51	HTRCXSetDirection	1142
11.3.3.52	HTRCXSetEvent	1142
11.3.3.53	HTRCXSetGlobalDirection	1143
11.3.3.54	HTRCXSetGlobalOutput	1143
11.3.3.55	HTRCXSetIRLinkPort	1143
11.3.3.56	HTRCXSetMaxPower	1143
11.3.3.57	HTRCXSetMessage	1144

11.3.3.258HTRCXSetOutput . . . . .	1144
11.3.3.259HTRCXSetPower . . . . .	1144
11.3.3.260HTRCXSetPriority . . . . .	1144
11.3.3.261HTRCXSetSensorMode . . . . .	1145
11.3.3.262HTRCXSetSensorType . . . . .	1145
11.3.3.263HTRCXSetSleepTime . . . . .	1145
11.3.3.264HTRCXSetTxPower . . . . .	1145
11.3.3.265HTRCXSetWatch . . . . .	1146
11.3.3.266HTRCXStartTask . . . . .	1146
11.3.3.267HTRCXStopAllTasks . . . . .	1146
11.3.3.268HTRCXStopTask . . . . .	1146
11.3.3.269HTRCXToggle . . . . .	1146
11.3.3.270HTRCXUnmuteSound . . . . .	1147
11.3.3.271HTScoutCalibrateSensor . . . . .	1147
11.3.3.272HTScoutMuteSound . . . . .	1147
11.3.3.273HTScoutSelectSounds . . . . .	1147
11.3.3.274HTScoutSendVLL . . . . .	1147
11.3.3.275HTScoutSetEventFeedback . . . . .	1148
11.3.3.276HTScoutSetLight . . . . .	1148
11.3.3.277HTScoutSetScoutMode . . . . .	1148
11.3.3.278HTScoutSetSensorClickTime . . . . .	1148
11.3.3.279HTScoutSetSensorHysteresis . . . . .	1149
11.3.3.280HTScoutSetSensorLowerLimit . . . . .	1149
11.3.3.281HTScoutSetSensorUpperLimit . . . . .	1149
11.3.3.282HTScoutUnmuteSound . . . . .	1149
11.3.3.283I2CBytes . . . . .	1150
11.3.3.284I2CBytesReady . . . . .	1150
11.3.3.285I2CCheckStatus . . . . .	1151
11.3.3.286I2CDeviceId . . . . .	1151
11.3.3.287I2CDeviceIdEx . . . . .	1152
11.3.3.288I2CDeviceInfo . . . . .	1152
11.3.3.289I2CDeviceInfoEx . . . . .	1152
11.3.3.290I2CRead . . . . .	1153
11.3.3.291I2CSendCommand . . . . .	1153
11.3.3.292I2CSendCommandEx . . . . .	1154
11.3.3.293I2CStatus . . . . .	1154

11.3.3.294	2CVendorId	1155
11.3.3.295	52CVendorIdEx	1155
11.3.3.296	62CVersion	1156
11.3.3.297	72CVersionEx	1156
11.3.3.298	82CWrite	1156
11.3.3.299	9salnum	1157
11.3.3.300	0salpha	1157
11.3.3.301	1iscntrl	1158
11.3.3.302	2sdigit	1158
11.3.3.303	3sgraph	1158
11.3.3.304	4slower	1159
11.3.3.305	5sNAN	1159
11.3.3.306	6sprint	1159
11.3.3.307	7spunct	1159
11.3.3.308	8sspace	1160
11.3.3.309	9supper	1160
11.3.3.310	0sxdigit	1160
11.3.3.311	1labs	1161
11.3.3.312	2div	1161
11.3.3.313	3LeftStr	1161
11.3.3.314	4LineOut	1162
11.3.3.315	5log	1162
11.3.3.316	6og10	1163
11.3.3.317	7LongAbort	1163
11.3.3.318	8LowspeedBytesReady	1163
11.3.3.319	9LowspeedCheckStatus	1164
11.3.3.320	0LowspeedRead	1164
11.3.3.321	1LowspeedStatus	1165
11.3.3.322	2LowspeedWrite	1165
11.3.3.323	3LSChannelState	1166
11.3.3.324	4LSErrorType	1166
11.3.3.325	5LSInputBufferBytesToRx	1167
11.3.3.326	6LSInputBufferInPtr	1167
11.3.3.327	7LSInputBufferOutPtr	1167
11.3.3.328	8LSMode	1168
11.3.3.329	9LSNoRestartOnRead	1168

11.3.3.330LSOutputBufferBytesToRx . . . . .	1168
11.3.3.331LSOutputBufferInPtr . . . . .	1168
11.3.3.332LSOutputBufferOutPtr . . . . .	1169
11.3.3.333LSSpeed . . . . .	1169
11.3.3.334LSState . . . . .	1169
11.3.3.335memcmp . . . . .	1170
11.3.3.336memcpy . . . . .	1170
11.3.3.337memmove . . . . .	1170
11.3.3.338MidStr . . . . .	1170
11.3.3.339MotorActualSpeed . . . . .	1171
11.3.3.340MotorBlockTachoCount . . . . .	1171
11.3.3.341MotorMode . . . . .	1171
11.3.3.342MotorOverload . . . . .	1172
11.3.3.343MotorPower . . . . .	1172
11.3.3.344MotorPwnFreq . . . . .	1172
11.3.3.345MotorRegDValue . . . . .	1172
11.3.3.346MotorRegIValue . . . . .	1173
11.3.3.347MotorRegPValue . . . . .	1173
11.3.3.348MotorRegulation . . . . .	1173
11.3.3.349MotorRotationCount . . . . .	1173
11.3.3.350MotorRunState . . . . .	1174
11.3.3.351MotorTachoCount . . . . .	1174
11.3.3.352MotorTachoLimit . . . . .	1174
11.3.3.353MotorTurnRatio . . . . .	1174
11.3.3.354MSADPAOff . . . . .	1175
11.3.3.355MSADPAOffEx . . . . .	1175
11.3.3.356MSADPAOn . . . . .	1175
11.3.3.357MSADPAOnEx . . . . .	1176
11.3.3.358MSDeenergize . . . . .	1176
11.3.3.359MSDeenergizeEx . . . . .	1176
11.3.3.360MSEnergize . . . . .	1177
11.3.3.361MSEnergizeEx . . . . .	1177
11.3.3.362MSIRTrain . . . . .	1177
11.3.3.363MSIRTrainEx . . . . .	1178
11.3.3.364MSPFComboDirect . . . . .	1178
11.3.3.365MSPFComboDirectEx . . . . .	1178

11.3.3.366MSPFComboPWM	1179
11.3.3.367MSPFComboPWMLEx	1179
11.3.3.368MSPFRawOutput	1180
11.3.3.369MSPFRawOutputEx	1180
11.3.3.370MSPFRepeat	1181
11.3.3.371MSPFRepeatEx	1181
11.3.3.372MSPFSingleOutputCST	1182
11.3.3.373MSPFSingleOutputCSTEx	1182
11.3.3.374MSPFSingleOutputPWM	1183
11.3.3.375MSPFSingleOutputPWMLEx	1183
11.3.3.376MSPFSinglePin	1184
11.3.3.377MSPFSinglePinEx	1184
11.3.3.378MSPFTrain	1185
11.3.3.379MSPFTrainEx	1185
11.3.3.380MSRCXAbsVar	1185
11.3.3.381MSRCXAddToDatalog	1186
11.3.3.382MSRCXAndVar	1186
11.3.3.383MSRCXBatteryLevel	1186
11.3.3.384MSRCXBoot	1187
11.3.3.385MSRCXCalibrateEvent	1187
11.3.3.386MSRCXClearAllEvents	1187
11.3.3.387MSRCXClearCounter	1187
11.3.3.388MSRCXClearMsg	1187
11.3.3.389MSRCXClearSensor	1188
11.3.3.390MSRCXClearSound	1188
11.3.3.391MSRCXClearTimer	1188
11.3.3.392MSRCXCreateDatalog	1188
11.3.3.393MSRCXDecCounter	1188
11.3.3.394MSRCXDeleteSub	1189
11.3.3.395MSRCXDeleteSubs	1189
11.3.3.396MSRCXDeleteTask	1189
11.3.3.397MSRCXDeleteTasks	1189
11.3.3.398MSRCXDisableOutput	1189
11.3.3.399MSRCXDivVar	1190
11.3.3.400MSRCXEnableOutput	1190
11.3.3.401MSRCXEvent	1190

11.3.3.402MSRCXFloat . . . . .	1190
11.3.3.403MSRCXFwd . . . . .	1191
11.3.3.404MSRCXIncCounter . . . . .	1191
11.3.3.405MSRCXInvertOutput . . . . .	1191
11.3.3.406MSRCXMulVar . . . . .	1191
11.3.3.407MSRCXMuteSound . . . . .	1192
11.3.3.408MSRCXObvertOutput . . . . .	1192
11.3.3.409MSRCXOff . . . . .	1192
11.3.3.410MSRCXOn . . . . .	1192
11.3.3.411MSRCXOnFor . . . . .	1192
11.3.3.412MSRCXOnFwd . . . . .	1193
11.3.3.413MSRCXOnRev . . . . .	1193
11.3.3.414MSRCXOrVar . . . . .	1193
11.3.3.415MSRCXPBTurnOff . . . . .	1193
11.3.3.416MSRCXPing . . . . .	1194
11.3.3.417MSRCXPlaySound . . . . .	1194
11.3.3.418MSRCXPlayTone . . . . .	1194
11.3.3.419MSRCXPlayToneVar . . . . .	1194
11.3.3.420MSRCXPoll . . . . .	1195
11.3.3.421MSRCXPollMemory . . . . .	1195
11.3.3.422MSRCXRemote . . . . .	1195
11.3.3.423MSRCXReset . . . . .	1195
11.3.3.424MSRCXRev . . . . .	1196
11.3.3.425MSRCXSelectDisplay . . . . .	1196
11.3.3.426MSRCXSelectProgram . . . . .	1196
11.3.3.427MSRCXSendSerial . . . . .	1196
11.3.3.428MSRCXSet . . . . .	1197
11.3.3.429MSRCXSetDirection . . . . .	1197
11.3.3.430MSRCXSetEvent . . . . .	1197
11.3.3.431MSRCXSetGlobalDirection . . . . .	1197
11.3.3.432MSRCXSetGlobalOutput . . . . .	1198
11.3.3.433MSRCXSetMaxPower . . . . .	1198
11.3.3.434MSRCXSetMessage . . . . .	1198
11.3.3.435MSRCXSetNRLinkPort . . . . .	1199
11.3.3.436MSRCXSetNRLinkPortEx . . . . .	1199
11.3.3.437MSRCXSetOutput . . . . .	1199

11.3.3.438MSRCXSetPower . . . . .	1199
11.3.3.439MSRCXSetPriority . . . . .	1200
11.3.3.440MSRCXSetSensorMode . . . . .	1200
11.3.3.441MSRCXSetSensorType . . . . .	1200
11.3.3.442MSRCXSetSleepTime . . . . .	1201
11.3.3.443MSRCXSetTxPower . . . . .	1201
11.3.3.444MSRCXSetUserDisplay . . . . .	1201
11.3.3.445MSRCXSetVar . . . . .	1201
11.3.3.446MSRCXSetWatch . . . . .	1202
11.3.3.447MSRCXSgnVar . . . . .	1202
11.3.3.448MSRCXStartTask . . . . .	1202
11.3.3.449MSRCXStopAllTasks . . . . .	1202
11.3.3.450MSRCXStopTask . . . . .	1203
11.3.3.451MSRCXSubVar . . . . .	1203
11.3.3.452MSRCXSumVar . . . . .	1203
11.3.3.453MSRCXToggle . . . . .	1203
11.3.3.454MSRCXUnlock . . . . .	1204
11.3.3.455MSRCXUnmuteSound . . . . .	1204
11.3.3.456MSReadValue . . . . .	1204
11.3.3.457MSReadValueEx . . . . .	1204
11.3.3.458MSScoutCalibrateSensor . . . . .	1205
11.3.3.459MSScoutMuteSound . . . . .	1205
11.3.3.460MSScoutSelectSounds . . . . .	1205
11.3.3.461MSScoutSendVLL . . . . .	1205
11.3.3.462MSScoutSetCounterLimit . . . . .	1206
11.3.3.463MSScoutSetEventFeedback . . . . .	1206
11.3.3.464MSScoutSetLight . . . . .	1206
11.3.3.465MSScoutSetScoutMode . . . . .	1206
11.3.3.466MSScoutSetScoutRules . . . . .	1207
11.3.3.467MSScoutSetSensorClickTime . . . . .	1207
11.3.3.468MSScoutSetSensorHysteresis . . . . .	1207
11.3.3.469MSScoutSetSensorLowerLimit . . . . .	1207
11.3.3.470MSScoutSetSensorUpperLimit . . . . .	1208
11.3.3.471MSScoutSetTimerLimit . . . . .	1208
11.3.3.472MSScoutUnmuteSound . . . . .	1208
11.3.3.473muldiv32 . . . . .	1209



11.3.3.474NRLink2400 . . . . .	1209
11.3.3.475NRLink2400Ex . . . . .	1209
11.3.3.476NRLink4800 . . . . .	1210
11.3.3.477NRLink4800Ex . . . . .	1210
11.3.3.478NRLinkFlush . . . . .	1210
11.3.3.479NRLinkFlushEx . . . . .	1211
11.3.3.480NRLinkIRLong . . . . .	1211
11.3.3.481NRLinkIRLongEx . . . . .	1211
11.3.3.482NRLinkIRShort . . . . .	1212
11.3.3.483NRLinkIRShortEx . . . . .	1212
11.3.3.484NRLinkSetPF . . . . .	1212
11.3.3.485NRLinkSetPFEx . . . . .	1213
11.3.3.486NRLinkSetRCX . . . . .	1213
11.3.3.487NRLinkSetRCXEx . . . . .	1213
11.3.3.488NRLinkSetTrain . . . . .	1214
11.3.3.489NRLinkSetTrainEx . . . . .	1214
11.3.3.490NRLinkStatus . . . . .	1214
11.3.3.491NRLinkStatusEx . . . . .	1215
11.3.3.492NRLinkTxRaw . . . . .	1215
11.3.3.493NRLinkTxRawEx . . . . .	1215
11.3.3.494NumOut . . . . .	1216
11.3.3.495NumToStr . . . . .	1216
11.3.3.496Off . . . . .	1217
11.3.3.497OffEx . . . . .	1217
11.3.3.498OnBrickProgramPointer . . . . .	1217
11.3.3.499OnFwd . . . . .	1217
11.3.3.500OnFwdEx . . . . .	1218
11.3.3.501OnFwdReg . . . . .	1218
11.3.3.502OnFwdRegEx . . . . .	1218
11.3.3.503OnFwdRegExPID . . . . .	1219
11.3.3.504OnFwdRegPID . . . . .	1219
11.3.3.505OnFwdSync . . . . .	1219
11.3.3.506OnFwdSyncEx . . . . .	1220
11.3.3.507OnFwdSyncExPID . . . . .	1220
11.3.3.508OnFwdSyncPID . . . . .	1221
11.3.3.509OnRev . . . . .	1221

11.3.3.510OnRevEx . . . . .	1221
11.3.3.511OnRevReg . . . . .	1222
11.3.3.512OnRevRegEx . . . . .	1222
11.3.3.513OnRevRegExPID . . . . .	1222
11.3.3.514OnRevRegPID . . . . .	1223
11.3.3.515OnRevSync . . . . .	1223
11.3.3.516OnRevSyncEx . . . . .	1224
11.3.3.517OnRevSyncExPID . . . . .	1224
11.3.3.518OnRevSyncPID . . . . .	1224
11.3.3.519OpenFileAppend . . . . .	1225
11.3.3.520OpenFileRead . . . . .	1225
11.3.3.521OpenFileReadLinear . . . . .	1226
11.3.3.522PlayFile . . . . .	1226
11.3.3.523PlayFileEx . . . . .	1227
11.3.3.524PlaySound . . . . .	1227
11.3.3.525PlayTone . . . . .	1227
11.3.3.526PlayToneEx . . . . .	1228
11.3.3.527PlayTones . . . . .	1228
11.3.3.528PointOut . . . . .	1228
11.3.3.529PolyOut . . . . .	1229
11.3.3.530pow . . . . .	1229
11.3.3.531PowerDown . . . . .	1229
11.3.3.532Precedes . . . . .	1230
11.3.3.533printf . . . . .	1230
11.3.3.534PSPNxAnalog . . . . .	1230
11.3.3.535PSPNxAnalogEx . . . . .	1231
11.3.3.536PSPNxDigital . . . . .	1231
11.3.3.537PSPNxDigitalEx . . . . .	1231
11.3.3.538rand . . . . .	1232
11.3.3.539Random . . . . .	1232
11.3.3.540Read . . . . .	1232
11.3.3.541ReadButtonEx . . . . .	1233
11.3.3.542ReadBytes . . . . .	1233
11.3.3.543ReadI2CRegister . . . . .	1233
11.3.3.544ReadLn . . . . .	1234
11.3.3.545ReadLnString . . . . .	1234

11.3.3.546ReadNRLinkBytes . . . . .	1234
11.3.3.547ReadNRLinkBytesEx . . . . .	1235
11.3.3.548ReadSensorColorEx . . . . .	1235
11.3.3.549ReadSensorColorRaw . . . . .	1236
11.3.3.550ReadSensorHTAccel . . . . .	1236
11.3.3.551ReadSensorHTColor . . . . .	1237
11.3.3.552ReadSensorHTColor2Active . . . . .	1237
11.3.3.553ReadSensorHTIRReceiver . . . . .	1238
11.3.3.554ReadSensorHTIRReceiverEx . . . . .	1238
11.3.3.555ReadSensorHTIRSeeker . . . . .	1238
11.3.3.556ReadSensorHTIRSeeker2AC . . . . .	1239
11.3.3.557ReadSensorHTIRSeeker2DC . . . . .	1239
11.3.3.558ReadSensorHTNormalizedColor . . . . .	1240
11.3.3.559ReadSensorHTNormalizedColor2Active . . . . .	1240
11.3.3.560ReadSensorHTRawColor . . . . .	1241
11.3.3.561ReadSensorHTRawColor2 . . . . .	1241
11.3.3.562ReadSensorHTTouchMultiplexer . . . . .	1242
11.3.3.563ReadSensorMSAccel . . . . .	1242
11.3.3.564ReadSensorMSAccelEx . . . . .	1242
11.3.3.565ReadSensorMSPlayStation . . . . .	1243
11.3.3.566ReadSensorMSPlayStationEx . . . . .	1243
11.3.3.567ReadSensorMSRTClock . . . . .	1244
11.3.3.568ReadSensorMSTilt . . . . .	1244
11.3.3.569ReadSensorMSTiltEx . . . . .	1245
11.3.3.570ReadSensorUSEx . . . . .	1245
11.3.3.571RebootInFirmwareMode . . . . .	1246
11.3.3.572ReceiveMessage . . . . .	1246
11.3.3.573ReceiveRemoteBool . . . . .	1246
11.3.3.574ReceiveRemoteMessageEx . . . . .	1246
11.3.3.575ReceiveRemoteNumber . . . . .	1247
11.3.3.576ReceiveRemoteString . . . . .	1247
11.3.3.577RechargeableBattery . . . . .	1248
11.3.3.578RectOut . . . . .	1248
11.3.3.579reladdr . . . . .	1248
11.3.3.580Release . . . . .	1249
11.3.3.581RemoteKeepAlive . . . . .	1249

11.3.3.582RemoteMessageRead . . . . .	1249
11.3.3.583RemoteMessageWrite . . . . .	1250
11.3.3.584RemotePlaySoundFile . . . . .	1250
11.3.3.585RemotePlayTone . . . . .	1250
11.3.3.586RemoteResetMotorPosition . . . . .	1251
11.3.3.587RemoteResetScaledValue . . . . .	1251
11.3.3.588RemoteSetInputMode . . . . .	1251
11.3.3.589RemoteSetOutputState . . . . .	1252
11.3.3.590RemoteStartProgram . . . . .	1252
11.3.3.591RemoteStopProgram . . . . .	1253
11.3.3.592RemoteStopSound . . . . .	1253
11.3.3.593remove . . . . .	1253
11.3.3.594rename . . . . .	1253
11.3.3.595RenameFile . . . . .	1254
11.3.3.596ResetAllTachoCounts . . . . .	1254
11.3.3.597ResetBlockTachoCount . . . . .	1254
11.3.3.598ResetRotationCount . . . . .	1255
11.3.3.599ResetScreen . . . . .	1255
11.3.3.600ResetSensor . . . . .	1255
11.3.3.601ResetSleepTimer . . . . .	1255
11.3.3.602ResetTachoCount . . . . .	1256
11.3.3.603ResizeFile . . . . .	1256
11.3.3.604ResolveHandle . . . . .	1256
11.3.3.605rewind . . . . .	1257
11.3.3.606RightStr . . . . .	1257
11.3.3.607RotateMotor . . . . .	1257
11.3.3.608RotateMotorEx . . . . .	1258
11.3.3.609RotateMotorExPID . . . . .	1258
11.3.3.610RotateMotorPID . . . . .	1259
11.3.3.611RS485Control . . . . .	1259
11.3.3.612RS485DataAvailable . . . . .	1259
11.3.3.613RS485Exit . . . . .	1260
11.3.3.614RS485Init . . . . .	1260
11.3.3.615RS485Read . . . . .	1260
11.3.3.616RS485SendingData . . . . .	1261
11.3.3.617RS485Status . . . . .	1261

11.3.3.618RS485Uart . . . . .	1261
11.3.3.619RS485Write . . . . .	1262
11.3.3.620RunNRLinkMacro . . . . .	1262
11.3.3.621RunNRLinkMacroEx . . . . .	1263
11.3.3.622SendMessage . . . . .	1263
11.3.3.623SendRemoteBool . . . . .	1263
11.3.3.624SendRemoteNumber . . . . .	1264
11.3.3.625SendRemoteString . . . . .	1264
11.3.3.626SendResponseBool . . . . .	1264
11.3.3.627SendResponseNumber . . . . .	1265
11.3.3.628SendResponseString . . . . .	1265
11.3.3.629SendRS485Bool . . . . .	1265
11.3.3.630SendRS485Number . . . . .	1266
11.3.3.631SendRS485String . . . . .	1266
11.3.3.632Sensor . . . . .	1266
11.3.3.633SensorBoolean . . . . .	1267
11.3.3.634SensorDigiPinsDirection . . . . .	1267
11.3.3.635SensorDigiPinsOutputLevel . . . . .	1267
11.3.3.636SensorDigiPinsStatus . . . . .	1268
11.3.3.637SensorHTColorNum . . . . .	1268
11.3.3.638SensorHTCompass . . . . .	1268
11.3.3.639SensorHTEOPD . . . . .	1269
11.3.3.640SensorHTGyro . . . . .	1269
11.3.3.641SensorHTIRSeeker2ACDir . . . . .	1269
11.3.3.642SensorHTIRSeeker2Addr . . . . .	1270
11.3.3.643SensorHTIRSeeker2DCDir . . . . .	1270
11.3.3.644SensorHTIRSeekerDir . . . . .	1270
11.3.3.645SensorInvalid . . . . .	1271
11.3.3.646SensorMode . . . . .	1271
11.3.3.647SensorMSCompass . . . . .	1271
11.3.3.648SensorMSCompassEx . . . . .	1272
11.3.3.649SensorMSDROD . . . . .	1272
11.3.3.650SensorMSPressure . . . . .	1272
11.3.3.651SensorMSPressureRaw . . . . .	1273
11.3.3.652SensorNormalized . . . . .	1273
11.3.3.653SensorRaw . . . . .	1273

11.3.3.654SensorScaled . . . . .	1274
11.3.3.655SensorType . . . . .	1274
11.3.3.656SensorUS . . . . .	1274
11.3.3.657SensorValue . . . . .	1275
11.3.3.658SensorValueBool . . . . .	1275
11.3.3.659SensorValueRaw . . . . .	1275
11.3.3.660SetAbortFlag . . . . .	1276
11.3.3.661SetBatteryState . . . . .	1276
11.3.3.662SetBluetoothState . . . . .	1276
11.3.3.663SetBTInputBuffer . . . . .	1276
11.3.3.664SetBTInputBufferInPtr . . . . .	1277
11.3.3.665SetBTInputBufferOutPtr . . . . .	1277
11.3.3.666SetBTOutputBuffer . . . . .	1277
11.3.3.667SetBTOutputBufferInPtr . . . . .	1277
11.3.3.668SetBTOutputBufferOutPtr . . . . .	1278
11.3.3.669SetButtonLongPressCount . . . . .	1278
11.3.3.670SetButtonLongReleaseCount . . . . .	1278
11.3.3.671SetButtonPressCount . . . . .	1278
11.3.3.672SetButtonReleaseCount . . . . .	1279
11.3.3.673SetButtonShortReleaseCount . . . . .	1279
11.3.3.674SetButtonState . . . . .	1279
11.3.3.675SetCommandFlags . . . . .	1279
11.3.3.676SetCustomSensorActiveStatus . . . . .	1280
11.3.3.677SetCustomSensorPercentFullScale . . . . .	1280
11.3.3.678SetCustomSensorZeroOffset . . . . .	1280
11.3.3.679SetDisplayContrast . . . . .	1280
11.3.3.680SetDisplayDisplay . . . . .	1281
11.3.3.681SetDisplayEraseMask . . . . .	1281
11.3.3.682SetDisplayFlags . . . . .	1281
11.3.3.683SetDisplayFont . . . . .	1281
11.3.3.684SetDisplayNormal . . . . .	1282
11.3.3.685SetDisplayPopup . . . . .	1282
11.3.3.686SetDisplayTextLinesCenterFlags . . . . .	1282
11.3.3.687SetDisplayUpdateMask . . . . .	1283
11.3.3.688SetHSFlags . . . . .	1283
11.3.3.689SetHSInputBuffer . . . . .	1283

11.3.3.690SetHSInputBufferInPtr . . . . .	1283
11.3.3.691SetHSInputBufferOutPtr . . . . .	1284
11.3.3.692SetHSMode . . . . .	1284
11.3.3.693SetHSOutputBuffer . . . . .	1284
11.3.3.694SetHSOutputBufferInPtr . . . . .	1284
11.3.3.695SetHSOutputBufferOutPtr . . . . .	1285
11.3.3.696SetHSSpeed . . . . .	1285
11.3.3.697SetHSState . . . . .	1285
11.3.3.698SetHTColor2Mode . . . . .	1285
11.3.3.699SetHTIRSeeker2Mode . . . . .	1286
11.3.3.700SetInput . . . . .	1286
11.3.3.701SetLongAbort . . . . .	1286
11.3.3.702SetMotorPwnFreq . . . . .	1287
11.3.3.703SetOnBrickProgramPointer . . . . .	1287
11.3.3.704SetOutput . . . . .	1287
11.3.3.705SetSensor . . . . .	1287
11.3.3.706SetSensorBoolean . . . . .	1288
11.3.3.707SetSensorColorBlue . . . . .	1288
11.3.3.708SetSensorColorFull . . . . .	1288
11.3.3.709SetSensorColorGreen . . . . .	1289
11.3.3.710SetSensorColorNone . . . . .	1289
11.3.3.711SetSensorColorRed . . . . .	1289
11.3.3.712SetSensorDigiPinsDirection . . . . .	1290
11.3.3.713SetSensorDigiPinsOutputLevel . . . . .	1290
11.3.3.714SetSensorDigiPinsStatus . . . . .	1290
11.3.3.715SetSensorHTEOPD . . . . .	1290
11.3.3.716SetSensorHTGyro . . . . .	1291
11.3.3.717SetSensorLight . . . . .	1291
11.3.3.718SetSensorLowspeed . . . . .	1291
11.3.3.719SetSensorMode . . . . .	1291
11.3.3.720SetSensorMSDROD . . . . .	1292
11.3.3.721SetSensorMSPressure . . . . .	1292
11.3.3.722SetSensorSound . . . . .	1292
11.3.3.723SetSensorTouch . . . . .	1293
11.3.3.724SetSensorType . . . . .	1293
11.3.3.725SetSleepTime . . . . .	1293

11.3.3.726SetSleepTimeout . . . . .	1293
11.3.3.727SetSleepTimer . . . . .	1294
11.3.3.728SetSoundDuration . . . . .	1294
11.3.3.729SetSoundFlags . . . . .	1294
11.3.3.730SetSoundFrequency . . . . .	1295
11.3.3.731SetSoundMode . . . . .	1295
11.3.3.732SetSoundModuleState . . . . .	1295
11.3.3.733SetSoundSampleRate . . . . .	1295
11.3.3.734SetSoundVolume . . . . .	1296
11.3.3.735SetUIButton . . . . .	1296
11.3.3.736SetUIState . . . . .	1296
11.3.3.737SetUSBInputBuffer . . . . .	1297
11.3.3.738SetUSBInputBufferInPtr . . . . .	1297
11.3.3.739SetUSBInputBufferOutPtr . . . . .	1297
11.3.3.740SetUSBOutputBuffer . . . . .	1297
11.3.3.741SetUSBOutputBufferInPtr . . . . .	1298
11.3.3.742SetUSBOutputBufferOutPtr . . . . .	1298
11.3.3.743SetUSBPollBuffer . . . . .	1298
11.3.3.744SetUSBPollBufferInPtr . . . . .	1298
11.3.3.745SetUSBPollBufferOutPtr . . . . .	1299
11.3.3.746SetUSBState . . . . .	1299
11.3.3.747SetVMRunState . . . . .	1299
11.3.3.748SetVolume . . . . .	1299
11.3.3.749sign . . . . .	1300
11.3.3.750sin . . . . .	1300
11.3.3.751sind . . . . .	1300
11.3.3.752sinh . . . . .	1301
11.3.3.753sinhd . . . . .	1301
11.3.3.754SleepNow . . . . .	1301
11.3.3.755SleepTime . . . . .	1301
11.3.3.756SleepTimeout . . . . .	1302
11.3.3.757SleepTimer . . . . .	1302
11.3.3.758SoundDuration . . . . .	1302
11.3.3.759SoundFlags . . . . .	1303
11.3.3.760SoundFrequency . . . . .	1303
11.3.3.761SoundMode . . . . .	1303



11.3.3.762SoundSampleRate . . . . .	1303
11.3.3.763SoundState . . . . .	1304
11.3.3.764SoundVolume . . . . .	1304
11.3.3.765sprintf . . . . .	1304
11.3.3.766sqrt . . . . .	1305
11.3.3.767StartTask . . . . .	1305
11.3.3.768Stop . . . . .	1305
11.3.3.769StopAllTasks . . . . .	1306
11.3.3.770StopSound . . . . .	1306
11.3.3.771StopTask . . . . .	1306
11.3.3.772strcat . . . . .	1306
11.3.3.773StrCat . . . . .	1307
11.3.3.774strcmp . . . . .	1307
11.3.3.775strcpy . . . . .	1307
11.3.3.776StrIndex . . . . .	1308
11.3.3.777strlen . . . . .	1308
11.3.3.778StrLen . . . . .	1309
11.3.3.779strncat . . . . .	1309
11.3.3.780strncmp . . . . .	1309
11.3.3.781strncpy . . . . .	1310
11.3.3.782StrReplace . . . . .	1310
11.3.3.783StrToByteArray . . . . .	1310
11.3.3.784strtod . . . . .	1311
11.3.3.785strtol . . . . .	1311
11.3.3.786StrToNum . . . . .	1312
11.3.3.787strtoul . . . . .	1312
11.3.3.788SubStr . . . . .	1313
11.3.3.789SysCall . . . . .	1313
11.3.3.790SysColorSensorRead . . . . .	1314
11.3.3.791SysCommBTCheckStatus . . . . .	1314
11.3.3.792SysCommBTConnection . . . . .	1314
11.3.3.793SysCommBTOnOff . . . . .	1315
11.3.3.794SysCommBTWrite . . . . .	1315
11.3.3.795SysCommExecuteFunction . . . . .	1315
11.3.3.796SysCommHSCheckStatus . . . . .	1315
11.3.3.797SysCommHSControl . . . . .	1316

11.3.3.798SysCommHSRead . . . . .	1316
11.3.3.799SysCommHSWrite . . . . .	1316
11.3.3.800SysCommLSCheckStatus . . . . .	1317
11.3.3.801SysCommLSRead . . . . .	1317
11.3.3.802SysCommLSWrite . . . . .	1317
11.3.3.803SysCommLSWriteEx . . . . .	1318
11.3.3.804SysComputeCalibValue . . . . .	1318
11.3.3.805SysDatalogGetTimes . . . . .	1318
11.3.3.806SysDatalogWrite . . . . .	1319
11.3.3.807SysDisplayExecuteFunction . . . . .	1319
11.3.3.808SysDrawCircle . . . . .	1319
11.3.3.809SysDrawEllipse . . . . .	1319
11.3.3.810SysDrawFont . . . . .	1320
11.3.3.811SysDrawGraphic . . . . .	1320
11.3.3.812SysDrawGraphicArray . . . . .	1320
11.3.3.813SysDrawLine . . . . .	1321
11.3.3.814SysDrawPoint . . . . .	1321
11.3.3.815SysDrawPolygon . . . . .	1321
11.3.3.816SysDrawRect . . . . .	1322
11.3.3.817SysDrawText . . . . .	1322
11.3.3.818SysFileClose . . . . .	1322
11.3.3.819SysFileDelete . . . . .	1322
11.3.3.820SysFileFindFirst . . . . .	1323
11.3.3.821SysFileFindNext . . . . .	1323
11.3.3.822SysFileOpenAppend . . . . .	1323
11.3.3.823SysFileOpenRead . . . . .	1323
11.3.3.824SysFileOpenReadLinear . . . . .	1324
11.3.3.825SysFileOpenWrite . . . . .	1324
11.3.3.826SysFileOpenWriteLinear . . . . .	1324
11.3.3.827SysFileOpenWriteNonLinear . . . . .	1325
11.3.3.828SysFileRead . . . . .	1325
11.3.3.829SysFileRename . . . . .	1325
11.3.3.830SysFileResize . . . . .	1325
11.3.3.831SysFileResolveHandle . . . . .	1326
11.3.3.832SysFileSeek . . . . .	1326
11.3.3.833SysFileWrite . . . . .	1326

11.3.3.834	SysGetStartTick	1327
11.3.3.835	SysIOMapRead	1327
11.3.3.836	SysIOMapReadByID	1327
11.3.3.837	SysIOMapWrite	1327
11.3.3.838	SysIOMapWriteByID	1328
11.3.3.839	SysKeepAlive	1328
11.3.3.840	SysListFiles	1328
11.3.3.841	SysLoaderExecuteFunction	1328
11.3.3.842	SysMessageRead	1329
11.3.3.843	SysMessageWrite	1329
11.3.3.844	SysRandomNumber	1329
11.3.3.845	SysReadButton	1330
11.3.3.846	SysReadSemData	1330
11.3.3.847	SysSetScreenMode	1330
11.3.3.848	SysSetSleepTimeout	1330
11.3.3.849	SysSoundGetState	1331
11.3.3.850	SysSoundPlayFile	1331
11.3.3.851	SysSoundPlayTone	1331
11.3.3.852	SysSoundSetState	1332
11.3.3.853	SysUpdateCalibCacheInfo	1332
11.3.3.854	SysWriteSemData	1332
11.3.3.855	tan	1333
11.3.3.856	tand	1333
11.3.3.857	tanh	1333
11.3.3.858	tanhd	1334
11.3.3.859	TextOut	1334
11.3.3.860	tolower	1335
11.3.3.861	toupper	1335
11.3.3.862	trunc	1335
11.3.3.863	UIButton	1336
11.3.3.864	UIState	1336
11.3.3.865	UnflattenVar	1336
11.3.3.866	USBInputBufferInPtr	1336
11.3.3.867	USBInputBufferOutPtr	1337
11.3.3.868	USBOutputBufferInPtr	1337
11.3.3.869	USBOutputBufferOutPtr	1337

11.3.3.870USBPollBufferInPtr . . . . .	1337
11.3.3.871USBPollBufferOutPtr . . . . .	1338
11.3.3.872UsbState . . . . .	1338
11.3.3.873USBState . . . . .	1338
11.3.3.874VMRunState . . . . .	1338
11.3.3.875Volume . . . . .	1338
11.3.3.876Wait . . . . .	1339
11.3.3.877Write . . . . .	1339
11.3.3.878WriteBytes . . . . .	1339
11.3.3.879WriteBytesEx . . . . .	1340
11.3.3.880WriteI2CRegister . . . . .	1340
11.3.3.881WriteLn . . . . .	1341
11.3.3.882WriteLnString . . . . .	1341
11.3.3.883WriteNRLinkBytes . . . . .	1341
11.3.3.884WriteNRLinkBytesEx . . . . .	1342
11.3.3.885WriteString . . . . .	1342
11.3.3.886Yield . . . . .	1342
<b>12 Example Documentation</b>	<b>1343</b>
12.1 alternating_tasks.nxc . . . . .	1343
12.2 ex_abort.nxc . . . . .	1344
12.3 ex_AbortFlag.nxc . . . . .	1345
12.4 ex_abs.nxc . . . . .	1346
12.5 ex_acos.nxc . . . . .	1347
12.6 ex_acosd.nxc . . . . .	1348
12.7 ex_Acquire.nxc . . . . .	1349
12.8 ex_addr.nxc . . . . .	1350
12.9 ex_addrex.nxc . . . . .	1351
12.10ex_arraybuild.nxc . . . . .	1352
12.11ex_arrayinit.nxc . . . . .	1353
12.12ex_arraylen.nxc . . . . .	1354
12.13ex_ArrayMax.nxc . . . . .	1355
12.14ex_ArrayMean.nxc . . . . .	1356
12.15ex_ArrayMin.nxc . . . . .	1357
12.16ex_ArrayOp.nxc . . . . .	1358
12.17ex_ArraySort.nxc . . . . .	1359
12.18ex_ArrayStd.nxc . . . . .	1360

12.19ex_arraysubset.nxc . . . . .	1361
12.20ex_ArraySum.nxc . . . . .	1362
12.21ex_ArraySumSqr.nxc . . . . .	1363
12.22ex_asin.nxc . . . . .	1364
12.23ex_asind.nxc . . . . .	1365
12.24ex_atan.nxc . . . . .	1366
12.25ex_atan2.nxc . . . . .	1367
12.26ex_atan2d.nxc . . . . .	1368
12.27ex_atand.nxc . . . . .	1369
12.28ex_atof.nxc . . . . .	1370
12.29ex_atoi.nxc . . . . .	1371
12.30ex_atol.nxc . . . . .	1372
12.31ex_BatteryState.nxc . . . . .	1373
12.32ex_bcd2dec.nxc . . . . .	1374
12.33ex_BluetoothState.nxc . . . . .	1375
12.34ex_bluetoothstatus.nxc . . . . .	1376
12.35ex_bluetoothwrite.nxc . . . . .	1377
12.36ex_BrickDataBluecoreVersion.nxc . . . . .	1378
12.37ex_BrickDataBtHardwareStatus.nxc . . . . .	1379
12.38ex_BrickDataBtStateStatus.nxc . . . . .	1380
12.39ex_BrickDataName.nxc . . . . .	1381
12.40ex_BrickDataTimeoutValue.nxc . . . . .	1382
12.41ex_BTConnectionClass.nxc . . . . .	1383
12.42ex_BTConnectionHandleNum.nxc . . . . .	1384
12.43ex_BTConnectionLinkQuality.nxc . . . . .	1385
12.44ex_BTConnectionName.nxc . . . . .	1386
12.45ex_BTConnectionPinCode.nxc . . . . .	1387
12.46ex_BTConnectionStreamStatus.nxc . . . . .	1388
12.47ex_BTDeviceClass.nxc . . . . .	1389
12.48ex_BTDeviceCount.nxc . . . . .	1390
12.49ex_BTDeviceName.nxc . . . . .	1391
12.50ex_BTDeviceNameCount.nxc . . . . .	1392
12.51ex_BTDeviceStatus.nxc . . . . .	1393
12.52ex_BTInputBufferInPtr.nxc . . . . .	1394
12.53ex_BTInputBufferOutPtr.nxc . . . . .	1395
12.54ex_BTOutputBufferInPtr.nxc . . . . .	1396

12.55ex_BTOutputBufferOutPtr.nxc . . . . .	1397
12.56ex_ButtonCount.nxc . . . . .	1398
12.57ex_ButtonLongPressCount.nxc . . . . .	1399
12.58ex_ButtonLongReleaseCount.nxc . . . . .	1400
12.59ex_ButtonPressCount.nxc . . . . .	1401
12.60ex_ButtonPressed.nxc . . . . .	1402
12.61ex_ButtonReleaseCount.nxc . . . . .	1403
12.62ex_ButtonShortReleaseCount.nxc . . . . .	1404
12.63ex_ButtonState.nxc . . . . .	1405
12.64ex_ByteArrayToStr.nxc . . . . .	1406
12.65ex_ByteArrayToStrEx.nxc . . . . .	1407
12.66ex_ceil.nxc . . . . .	1408
12.67ex_CircleOut.nxc . . . . .	1409
12.68ex_ClearLine.nxc . . . . .	1410
12.69ex_ClearScreen.nxc . . . . .	1411
12.70ex_ClearSensor.nxc . . . . .	1412
12.71ex_CloseFile.nxc . . . . .	1413
12.72ex_coast.nxc . . . . .	1414
12.73ex_coastex.nxc . . . . .	1415
12.74ex_ColorADRaw.nxc . . . . .	1416
12.75ex_ColorBoolean.nxc . . . . .	1417
12.76ex_ColorCalibration.nxc . . . . .	1418
12.77ex_ColorCalibrationState.nxc . . . . .	1419
12.78ex_ColorCalLimits.nxc . . . . .	1420
12.79ex_ColorSensorRaw.nxc . . . . .	1421
12.80ex_ColorSensorValue.nxc . . . . .	1422
12.81ex_CommandFlags.nxc . . . . .	1423
12.82ex_contrast.nxc . . . . .	1424
12.83ex_Copy.nxc . . . . .	1425
12.84ex_cos.nxc . . . . .	1426
12.85ex_cosd.nxc . . . . .	1427
12.86ex_cosh.nxc . . . . .	1428
12.87ex_CreateFile.nxc . . . . .	1429
12.88ex_CreateFileLinear.nxc . . . . .	1430
12.89ex_CreateFileNonLinear.nxc . . . . .	1431
12.90ex_cstdio.nxc . . . . .	1432

12.91ex_cstring.nxc . . . . .	1433
12.92ex_ctype.nxc . . . . .	1434
12.93ex_CurrentTick.nxc . . . . .	1435
12.94ex_CustomSensorActiveStatus.nxc . . . . .	1436
12.95ex_CustomSensorPercentFullScale.nxc . . . . .	1437
12.96ex_CustomSensorZeroOffset.nxc . . . . .	1438
12.97ex_DeleteFile.nxc . . . . .	1439
12.98ex_dispfnout.nxc . . . . .	1440
12.99ex_dispfnout.nxc . . . . .	1441
12.100x_dispfnc.nxc . . . . .	1442
12.101x_dispgaout.nxc . . . . .	1443
12.102x_dispgaoutex.nxc . . . . .	1444
12.103x_dispgout.nxc . . . . .	1449
12.104x_dispgoutex.nxc . . . . .	1450
12.105x_DisplayDisplay.nxc . . . . .	1451
12.106x_DisplayEraseMask.nxc . . . . .	1452
12.107x_DisplayFlags.nxc . . . . .	1453
12.108x_DisplayFont.nxc . . . . .	1454
12.109x_DisplayTextLinesCenterFlags.nxc . . . . .	1455
12.110x_DisplayUpdateMask.nxc . . . . .	1456
12.111x_dispmisc.nxc . . . . .	1457
12.112x_DISTNxDistance.nxc . . . . .	1458
12.113x_DISTNxDistanceEx.nxc . . . . .	1459
12.114x_DISTNxGP2D12.nxc . . . . .	1460
12.115x_DISTNxGP2D120.nxc . . . . .	1461
12.116x_DISTNxGP2D120Ex.nxc . . . . .	1462
12.117x_DISTNxGP2D12Ex.nxc . . . . .	1463
12.118x_DISTNxGP2YA02.nxc . . . . .	1464
12.119x_DISTNxGP2YA02Ex.nxc . . . . .	1465
12.120x_DISTNxGP2YA21.nxc . . . . .	1466
12.121x_DISTNxGP2YA21Ex.nxc . . . . .	1467
12.122x_DISTNxMaxDistance.nxc . . . . .	1468
12.123x_DISTNxMaxDistanceEx.nxc . . . . .	1469
12.124x_DISTNxMinDistance.nxc . . . . .	1470
12.125x_DISTNxMinDistanceEx.nxc . . . . .	1471
12.126x_DISTNxModuleType.nxc . . . . .	1472

12.127x_DISTNxModuleTypeEx.nxc . . . . .	1473
12.128x_DISTNxNumPoints.nxc . . . . .	1474
12.129x_DISTNxNumPointsEx.nxc . . . . .	1475
12.130x_DISTNxVoltage.nxc . . . . .	1476
12.131x_DISTNxVoltageEx.nxc . . . . .	1477
12.132x_div.nxc . . . . .	1478
12.133x_EllipseOut.nxc . . . . .	1479
12.134x_exp.nxc . . . . .	1480
12.135x_fclose.nxc . . . . .	1481
12.136x_feof.nxc . . . . .	1482
12.137x_fflush.nxc . . . . .	1483
12.138x_fgetc.nxc . . . . .	1484
12.139x_fgets.nxc . . . . .	1485
12.140x_FindFirstFile.nxc . . . . .	1486
12.141x_FindNextFile.nxc . . . . .	1487
12.142x_FirstTick.nxc . . . . .	1488
12.143x_Flatten.nxc . . . . .	1489
12.144x_FlattenVar.nxc . . . . .	1490
12.145x_float.nxc . . . . .	1491
12.146x_floor.nxc . . . . .	1492
12.147x_Follows.nxc . . . . .	1493
12.148x_fopen.nxc . . . . .	1494
12.149x_ForceOff.nxc . . . . .	1495
12.150x_FormatNum.nxc . . . . .	1496
12.151x_fprintf.nxc . . . . .	1497
12.152x_fputc.nxc . . . . .	1498
12.153x_fputs.nxc . . . . .	1499
12.154x_frac.nxc . . . . .	1500
12.155x_FreeMemory.nxc . . . . .	1501
12.156x_fseek.nxc . . . . .	1502
12.157x_ftell.nxc . . . . .	1503
12.158x_GetBrickDataAddress.nxc . . . . .	1504
12.159x_GetBTConnectionAddress.nxc . . . . .	1505
12.160x_GetBTDeviceAddress.nxc . . . . .	1506
12.161x_GetBTInputBuffer.nxc . . . . .	1507
12.162x_GetBTOutputBuffer.nxc . . . . .	1508



12.163x_getc.nxc	1509
12.164x_getchar.nxc	1510
12.165x_GetDisplayNormal.nxc	1511
12.166x_GetDisplayPopup.nxc	1512
12.167x_GetHSInputBuffer.nxc	1513
12.168x_GetHSOutputBuffer.nxc	1514
12.169x_GetInput.nxc	1515
12.170x_GetLSInputBuffer.nxc	1516
12.171x_GetLSOutputBuffer.nxc	1517
12.172x_getoutput.nxc	1518
12.173x_GetUSBInputBuffer.nxc	1519
12.174x_GetUSBOutputBuffer.nxc	1520
12.175x_GetUSBPollBuffer.nxc	1521
12.176x_GraphicOut.nxc	1522
12.177x_GraphicOutEx.nxc	1523
12.178x_HSFlags.nxc	1524
12.179x_HSInputBufferInPtr.nxc	1525
12.180x_HSInputBufferOutPtr.nxc	1526
12.181x_HSMODE.nxc	1527
12.182x_HSOutputBufferInPtr.nxc	1528
12.183x_HSOutputBufferOutPtr.nxc	1529
12.184x_HSSpeed.nxc	1530
12.185x_HSState.nxc	1531
12.186x_HTIRTrain.nxc	1532
12.187x_HTPFComboDirect.nxc	1533
12.188x_HTPFComboPWM.nxc	1534
12.189x_HTPFRawOutput.nxc	1535
12.190x_HTPFRepeat.nxc	1536
12.191x_HTPFSingleOutputCST.nxc	1537
12.192x_HTPFSingleOutputPWM.nxc	1538
12.193x_HTPFSinglePin.nxc	1539
12.194x_HTPFTrain.nxc	1540
12.195x_HTRCXAddToDatalog.nxc	1541
12.196x_HTRCXBatteryLevel.nxc	1542
12.197x_HTRCXCLEARAllEvents.nxc	1543
12.198x_HTRCXCLEARCounter.nxc	1544

12.199x_HTRCXClearMsg.nxc	1545
12.200x_HTRCXClearSensor.nxc	1546
12.201x_HTRCXClearSound.nxc	1547
12.202x_HTRCXClearTimer.nxc	1548
12.203x_HTRCXCreateDatalog.nxc	1549
12.204x_HTRCXDecCounter.nxc	1550
12.205x_HTRCXDeleteSub.nxc	1551
12.206x_HTRCXDeleteSubs.nxc	1552
12.207x_HTRCXDeleteTask.nxc	1553
12.208x_HTRCXDeleteTasks.nxc	1554
12.209x_HTRCXDisableOutput.nxc	1555
12.210x_HTRCXEnableOutput.nxc	1556
12.211x_HTRCXEvent.nxc	1557
12.212x_HTRCXFloat.nxc	1558
12.213x_HTRCXFwd.nxc	1559
12.214x_HTRCXIncCounter.nxc	1560
12.215x_HTRCXInvertOutput.nxc	1561
12.216x_HTRCXMuteSound.nxc	1562
12.217x_HTRCXObvertOutput.nxc	1563
12.218x_HTRCXOff.nxc	1564
12.219x_HTRCXOn.nxc	1565
12.220x_HTRCXOnFor.nxc	1566
12.221x_HTRCXOnFwd.nxc	1567
12.222x_HTRCXOnRev.nxc	1568
12.223x_HTRCXPBTurnOff.nxc	1569
12.224x_HTRCXPing.nxc	1570
12.225x_HTRCXPlaySound.nxc	1571
12.226x_HTRCXPlayTone.nxc	1572
12.227x_HTRCXPlayToneVar.nxc	1573
12.228x_HTRCXPoll.nxc	1574
12.229x_HTRCXPollMemory.nxc	1575
12.230x_HTRCXRemote.nxc	1576
12.231x_HTRCXRev.nxc	1577
12.232x_HTRCXSelectDisplay.nxc	1578
12.233x_HTRCXSelectProgram.nxc	1579
12.234x_HTRCXSendSerial.nxc	1580

12.235x_HTRCXSetDirection.nxc	1581
12.236x_HTRCXSetEvent.nxc	1582
12.237x_HTRCXSetGlobalDirection.nxc	1583
12.238x_HTRCXSetGlobalOutput.nxc	1584
12.239x_HTRCXSetIRLinkPort.nxc	1585
12.240x_HTRCXSetMaxPower.nxc	1586
12.241x_HTRCXSetMessage.nxc	1587
12.242x_HTRCXSetOutput.nxc	1588
12.243x_HTRCXSetPower.nxc	1589
12.244x_HTRCXSetPriority.nxc	1590
12.245x_HTRCXSetSensorMode.nxc	1591
12.246x_HTRCXSetSensorType.nxc	1592
12.247x_HTRCXSetSleepTime.nxc	1593
12.248x_HTRCXSetTxPower.nxc	1594
12.249x_HTRCXSetWatch.nxc	1595
12.250x_HTRCXStartTask.nxc	1596
12.251x_HTRCXStopAllTasks.nxc	1597
12.252x_HTRCXStopTask.nxc	1598
12.253x_HTRCXToggle.nxc	1599
12.254x_HTRCXUnmuteSound.nxc	1600
12.255x_HTScoutCalibrateSensor.nxc	1601
12.256x_HTScoutMuteSound.nxc	1602
12.257x_HTScoutSelectSounds.nxc	1603
12.258x_HTScoutSendVLL.nxc	1604
12.259x_HTScoutSetEventFeedback.nxc	1605
12.260x_HTScoutSetLight.nxc	1606
12.261x_HTScoutSetScoutMode.nxc	1607
12.262x_HTScoutSetSensorClickTime.nxc	1608
12.263x_HTScoutSetSensorHysteresis.nxc	1609
12.264x_HTScoutSetSensorLowerLimit.nxc	1610
12.265x_HTScoutSetSensorUpperLimit.nxc	1611
12.266x_HTScoutUnmuteSound.nxc	1612
12.267x_I2CBytes.nxc	1613
12.268x_I2CBytesReady.nxc	1614
12.269x_I2CCheckStatus.nxc	1615
12.270x_I2CDeviceId.nxc	1616

12.271x_I2CDeviceIdEx.nxc	1617
12.272x_I2CDeviceInfo.nxc	1618
12.273x_I2CDeviceInfoEx.nxc	1619
12.274x_I2CRead.nxc	1620
12.275x_I2CSendCommand.nxc	1621
12.276x_I2CSendCommandEx.nxc	1622
12.277x_I2CStatus.nxc	1623
12.278x_I2CVendorId.nxc	1624
12.279x_I2CVendorIdEx.nxc	1625
12.280x_I2CVersion.nxc	1626
12.281x_I2CVersionEx.nxc	1627
12.282x_i2cwrite.nxc	1628
12.283x_isalnum.nxc	1629
12.284x_isalpha.nxc	1630
12.285x_iscentrl.nxc	1631
12.286x_isdigit.nxc	1632
12.287x_isgraph.nxc	1633
12.288x_islower.nxc	1634
12.289x_isNAN.nxc	1635
12.290x_isprint.nxc	1636
12.291x_ispunct.nxc	1637
12.292x_isspace.nxc	1638
12.293x_isupper.nxc	1639
12.294x_isxdigit.nxc	1640
12.295x_labs.nxc	1641
12.296x_ldiv.nxc	1642
12.297x_leftstr.nxc	1643
12.298x_LineOut.nxc	1644
12.299x_log.nxc	1645
12.300x_log10.nxc	1646
12.301x_LongAbort.nxc	1647
12.302x_LowspeedBytesReady.nxc	1648
12.303x_LowspeedCheckStatus.nxc	1649
12.304x_LowspeedRead.nxc	1650
12.305x_lowspeedstatus.nxc	1651
12.306x_LowspeedWrite.nxc	1652

12.307x_LSChannelState.nxc . . . . .	1653
12.308x_LSErrorType.nxc . . . . .	1654
12.309x_LSInputBufferBytesToRx.nxc . . . . .	1655
12.310x_LSInputBufferInPtr.nxc . . . . .	1656
12.311x_LSInputBufferOutPtr.nxc . . . . .	1657
12.312x_LSMODE.nxc . . . . .	1658
12.313x_LSNorestartOnRead.nxc . . . . .	1659
12.314x_LSOutputBufferBytesToRx.nxc . . . . .	1660
12.315x_LSOutputBufferInPtr.nxc . . . . .	1661
12.316x_LSOutputBufferOutPtr.nxc . . . . .	1662
12.317x_LSSpeed.nxc . . . . .	1663
12.318x_LSSState.nxc . . . . .	1664
12.319x_memcmp.nxc . . . . .	1665
12.320x_memcpy.nxc . . . . .	1666
12.321x_memmove.nxc . . . . .	1667
12.322x_midstr.nxc . . . . .	1668
12.323x_motoractualspeed.nxc . . . . .	1669
12.324x_motorblocktachocount.nxc . . . . .	1670
12.325x_motormode.nxc . . . . .	1671
12.326x_motoroverload.nxc . . . . .	1672
12.327x_motorpower.nxc . . . . .	1673
12.328x_motorpwnfreq.nxc . . . . .	1674
12.329x_motorregdvalue.nxc . . . . .	1675
12.330x_motorregivalue.nxc . . . . .	1676
12.331x_motorregpvalue.nxc . . . . .	1677
12.332x_motorregulation.nxc . . . . .	1678
12.333x_motorrotationcount.nxc . . . . .	1679
12.334x_motorrunstate.nxc . . . . .	1680
12.335x_motortachocount.nxc . . . . .	1681
12.336x_motortacholimit.nxc . . . . .	1682
12.337x_motorturnratio.nxc . . . . .	1683
12.338x_MSADPAOff.nxc . . . . .	1684
12.339x_MSADPAOffEx.nxc . . . . .	1685
12.340x_MSADPAOn.nxc . . . . .	1686
12.341x_MSADPAOnEx.nxc . . . . .	1687
12.342x_MSDeenergize.nxc . . . . .	1688

12.343x_MSDeenergizeEx.nxc . . . . .	1689
12.344x_MSEnergize.nxc . . . . .	1690
12.345x_MSEnergizeEx.nxc . . . . .	1691
12.346x_MSIRTrain.nxc . . . . .	1692
12.347x_MSIRTrainEx.nxc . . . . .	1693
12.348x_MSPFComboDirect.nxc . . . . .	1694
12.349x_MSPFComboDirectEx.nxc . . . . .	1695
12.350x_MSPFComboPWM.nxc . . . . .	1696
12.351x_MSPFComboPWMLEx.nxc . . . . .	1697
12.352x_MSPFRawOutput.nxc . . . . .	1698
12.353x_MSPFRawOutputEx.nxc . . . . .	1699
12.354x_MSPFRepeat.nxc . . . . .	1700
12.355x_MSPFRepeatEx.nxc . . . . .	1701
12.356x_MSPFSingleOutputCST.nxc . . . . .	1702
12.357x_MSPFSingleOutputCSTEx.nxc . . . . .	1703
12.358x_MSPFSingleOutputPWM.nxc . . . . .	1704
12.359x_MSPFSingleOutputPWMLEx.nxc . . . . .	1705
12.360x_MSPFSinglePin.nxc . . . . .	1706
12.361x_MSPFSinglePinEx.nxc . . . . .	1707
12.362x_MSPFTrain.nxc . . . . .	1708
12.363x_MSPFTrainEx.nxc . . . . .	1709
12.364x_MSRCXAbsVar.nxc . . . . .	1710
12.365x_MSRCXAddToDatalog.nxc . . . . .	1711
12.366x_MSRCXAndVar.nxc . . . . .	1712
12.367x_MSRCXBatteryLevel.nxc . . . . .	1713
12.368x_MSRCXBoot.nxc . . . . .	1714
12.369x_MSRCXCalibrateEvent.nxc . . . . .	1715
12.370x_MSRCXClearAllEvents.nxc . . . . .	1716
12.371x_MSRCXClearCounter.nxc . . . . .	1717
12.372x_MSRCXClearMsg.nxc . . . . .	1718
12.373x_MSRCXClearSensor.nxc . . . . .	1719
12.374x_MSRCXClearSound.nxc . . . . .	1720
12.375x_MSRCXClearTimer.nxc . . . . .	1721
12.376x_MSRCXCreateDatalog.nxc . . . . .	1722
12.377x_MSRCXDecCounter.nxc . . . . .	1723
12.378x_MSRCXDeleteSub.nxc . . . . .	1724

12.379x_MSRCXDeleteSubs.nxc . . . . .	1725
12.380x_MSRCXDeleteTask.nxc . . . . .	1726
12.381x_MSRCXDeleteTasks.nxc . . . . .	1727
12.382x_MSRCXDisableOutput.nxc . . . . .	1728
12.383x_MSRCXDivVar.nxc . . . . .	1729
12.384x_MSRCXEnableOutput.nxc . . . . .	1730
12.385x_MSRCXEvent.nxc . . . . .	1731
12.386x_MSRCXFloat.nxc . . . . .	1732
12.387x_MSRCXFwd.nxc . . . . .	1733
12.388x_MSRCXIncCounter.nxc . . . . .	1734
12.389x_MSRCXInvertOutput.nxc . . . . .	1735
12.390x_MSRCXMulVar.nxc . . . . .	1736
12.391x_MSRCXMuteSound.nxc . . . . .	1737
12.392x_MSRCXObvertOutput.nxc . . . . .	1738
12.393x_MSRCXOff.nxc . . . . .	1739
12.394x_MSRCXOn.nxc . . . . .	1740
12.395x_MSRCXOnFor.nxc . . . . .	1741
12.396x_MSRCXOnFwd.nxc . . . . .	1742
12.397x_MSRCXOnRev.nxc . . . . .	1743
12.398x_MSRCXOrVar.nxc . . . . .	1744
12.399x_MSRCXPBTurnOff.nxc . . . . .	1745
12.400x_MSRCXPing.nxc . . . . .	1746
12.401x_MSRCXPlaySound.nxc . . . . .	1747
12.402x_MSRCXPlayTone.nxc . . . . .	1748
12.403x_MSRCXPlayToneVar.nxc . . . . .	1749
12.404x_MSRCXPoll.nxc . . . . .	1750
12.405x_MSRCXPollMemory.nxc . . . . .	1751
12.406x_MSRCXRemote.nxc . . . . .	1752
12.407x_MSRCXReset.nxc . . . . .	1753
12.408x_MSRCXRev.nxc . . . . .	1754
12.409x_MSRCXSelectDisplay.nxc . . . . .	1755
12.410x_MSRCXSelectProgram.nxc . . . . .	1756
12.411x_MSRCXSendSerial.nxc . . . . .	1757
12.412x_MSRCXSet.nxc . . . . .	1758
12.413x_MSRCXSetDirection.nxc . . . . .	1759
12.414x_MSRCXSetEvent.nxc . . . . .	1760

12.415x_MSRCXSetGlobalDirection.nxc	1761
12.416x_MSRCXSetGlobalOutput.nxc	1762
12.417x_MSRCXSetMaxPower.nxc	1763
12.418x_MSRCXSetMessage.nxc	1764
12.419x_MSRCXSetNRLinkPort.nxc	1765
12.420x_MSRCXSetNRLinkPortEx.nxc	1766
12.421x_MSRCXSetOutput.nxc	1767
12.422x_MSRCXSetPower.nxc	1768
12.423x_MSRCXSetPriority.nxc	1769
12.424x_MSRCXSetSensorMode.nxc	1770
12.425x_MSRCXSetSensorType.nxc	1771
12.426x_MSRCXSetSleepTime.nxc	1772
12.427x_MSRCXSetTxPower.nxc	1773
12.428x_MSRCXSetUserDisplay.nxc	1774
12.429x_MSRCXSetVar.nxc	1775
12.430x_MSRCXSetWatch.nxc	1776
12.431x_MSRCXSgnVar.nxc	1777
12.432x_MSRCXStartTask.nxc	1778
12.433x_MSRCXStopAllTasks.nxc	1779
12.434x_MSRCXStopTask.nxc	1780
12.435x_MSRCXSubVar.nxc	1781
12.436x_MSRCXSumVar.nxc	1782
12.437x_MSRCXToggle.nxc	1783
12.438x_MSRCXUnlock.nxc	1784
12.439x_MSRCXUnmuteSound.nxc	1785
12.440x_MSReadValue.nxc	1786
12.441x_MSReadValueEx.nxc	1787
12.442x_MSScoutCalibrateSensor.nxc	1788
12.443x_MSScoutMuteSound.nxc	1789
12.444x_MSScoutSelectSounds.nxc	1790
12.445x_MSScoutSendVLL.nxc	1791
12.446x_MSScoutSetCounterLimit.nxc	1792
12.447x_MSScoutSetEventFeedback.nxc	1793
12.448x_MSScoutSetLight.nxc	1794
12.449x_MSScoutSetScoutMode.nxc	1795
12.450x_MSScoutSetScoutRules.nxc	1796



12.451x_MSScoutSetSensorClickTime.nxc . . . . .	1797
12.452x_MSScoutSetSensorHysteresis.nxc . . . . .	1798
12.453x_MSScoutSetSensorLowerLimit.nxc . . . . .	1799
12.454x_MSScoutSetSensorUpperLimit.nxc . . . . .	1800
12.455x_MSScoutSetTimerLimit.nxc . . . . .	1801
12.456x_MSScoutUnmuteSound.nxc . . . . .	1802
12.457x_muldiv32.nxc . . . . .	1803
12.458x_NRLink2400.nxc . . . . .	1804
12.459x_NRLink2400Ex.nxc . . . . .	1805
12.460x_NRLink4800.nxc . . . . .	1806
12.461x_NRLink4800Ex.nxc . . . . .	1807
12.462x_NRLinkFlush.nxc . . . . .	1808
12.463x_NRLinkFlushEx.nxc . . . . .	1809
12.464x_NRLinkIRLong.nxc . . . . .	1810
12.465x_NRLinkIRLongEx.nxc . . . . .	1811
12.466x_NRLinkIRShort.nxc . . . . .	1812
12.467x_NRLinkIRShortEx.nxc . . . . .	1813
12.468x_NRLinkSetPF.nxc . . . . .	1814
12.469x_NRLinkSetPFEx.nxc . . . . .	1815
12.470x_NRLinkSetRCX.nxc . . . . .	1816
12.471x_NRLinkSetRCXEx.nxc . . . . .	1817
12.472x_NRLinkSetTrain.nxc . . . . .	1818
12.473x_NRLinkSetTrainEx.nxc . . . . .	1819
12.474x_NRLinkStatus.nxc . . . . .	1820
12.475x_NRLinkStatusEx.nxc . . . . .	1821
12.476x_NRLinkTxRaw.nxc . . . . .	1822
12.477x_NRLinkTxRawEx.nxc . . . . .	1823
12.478x_NumOut.nxc . . . . .	1824
12.479x_NumToStr.nxc . . . . .	1825
12.480x_off.nxc . . . . .	1826
12.481x_offex.nxc . . . . .	1827
12.482x_OnBrickProgramPointer.nxc . . . . .	1828
12.483x_onfwd.nxc . . . . .	1829
12.484x_onfwdex.nxc . . . . .	1830
12.485x_onfwdreg.nxc . . . . .	1831
12.486x_onfwdregex.nxc . . . . .	1832

12.487x_onfwdregexpid.nxc	1833
12.488x_onfwdregpid.nxc	1834
12.489x_onfwdsync.nxc	1835
12.490x_onfwdsyncex.nxc	1836
12.491x_onfwdsyncexpid.nxc	1837
12.492x_onfwdsyncpid.nxc	1838
12.493x_onrev.nxc	1839
12.494x_onrevex.nxc	1840
12.495x_onrevreg.nxc	1841
12.496x_onrevregex.nxc	1842
12.497x_onrevregexpid.nxc	1843
12.498x_onrevregpid.nxc	1844
12.499x_onrevsync.nxc	1845
12.500x_onrevsyncex.nxc	1846
12.501x_onrevsyncexpid.nxc	1847
12.502x_onrevsyncpid.nxc	1848
12.503x_OpenFileAppend.nxc	1849
12.504x_OpenFileRead.nxc	1850
12.505x_OpenFileReadLinear.nxc	1851
12.506x_PlayFile.nxc	1852
12.507x_PlayFileEx.nxc	1853
12.508x_playsound.nxc	1854
12.509x_PlayTone.nxc	1855
12.510x_PlayToneEx.nxc	1856
12.511x_playtones.nxc	1857
12.512x_PointOut.nxc	1858
12.513x_PolyOut.nxc	1859
12.514x_pow.nxc	1860
12.515x_powerdown.nxc	1861
12.516x_Precedes.nxc	1862
12.517x_printf.nxc	1863
12.518x_PSPNxAnalog.nxc	1864
12.519x_PSPNxAnalogEx.nxc	1865
12.520x_PSPNxDigital.nxc	1866
12.521x_PSPNxDigitalEx.nxc	1867
12.522x_putc.nxc	1868

12.523x_rand.nxc . . . . .	1869
12.524x_Random.nxc . . . . .	1870
12.525x_Read.nxc . . . . .	1871
12.526x_ReadButtonEx.nxc . . . . .	1872
12.527x_ReadBytes.nxc . . . . .	1873
12.528x_readi2cregister.nxc . . . . .	1874
12.529x_ReadLn.nxc . . . . .	1875
12.530x_ReadNRLinkBytes.nxc . . . . .	1876
12.531x_ReadNRLinkBytesEx.nxc . . . . .	1877
12.532x_ReadSensorColorEx.nxc . . . . .	1878
12.533x_ReadSensorColorRaw.nxc . . . . .	1879
12.534x_ReadSensorHTAccel.nxc . . . . .	1880
12.535x_ReadSensorHTColor.nxc . . . . .	1881
12.536x_ReadSensorHTColor2Active.nxc . . . . .	1882
12.537x_ReadSensorHTIRReceiver.nxc . . . . .	1883
12.538x_ReadSensorHTIRReceiverEx.nxc . . . . .	1884
12.539x_ReadSensorHTIRSeeker.nxc . . . . .	1885
12.540x_ReadSensorHTIRSeeker2AC.nxc . . . . .	1886
12.541x_ReadSensorHTIRSeeker2DC.nxc . . . . .	1887
12.542x_ReadSensorHTNormalizedColor.nxc . . . . .	1888
12.543x_ReadSensorHTNormalizedColor2Active.nxc . . . . .	1889
12.544x_ReadSensorHTRawColor.nxc . . . . .	1890
12.545x_ReadSensorHTRawColor2.nxc . . . . .	1891
12.546x_ReadSensorHTTouchMultiplexer.nxc . . . . .	1892
12.547x_ReadSensorMSAccel.nxc . . . . .	1893
12.548x_ReadSensorMSAccelEx.nxc . . . . .	1894
12.549x_ReadSensorMSPlayStation.nxc . . . . .	1895
12.550x_ReadSensorMSPlayStationEx.nxc . . . . .	1896
12.551x_ReadSensorMSRTClock.nxc . . . . .	1897
12.552x_ReadSensorMSTilt.nxc . . . . .	1898
12.553x_ReadSensorMSTiltEx.nxc . . . . .	1899
12.554x_readsensorsex.nxc . . . . .	1900
12.555x_rebootinfirmwaremode.nxc . . . . .	1901
12.556x_receivemessage.nxc . . . . .	1902
12.557x_ReceiveRemoteBool.nxc . . . . .	1903
12.558x_ReceiveRemoteMessageEx.nxc . . . . .	1904

12.550x_ReceiveRemoteNumber.nxc . . . . .	1905
12.560x_ReceiveRemoteString.nxc . . . . .	1906
12.561x_RechargeableBattery.nxc . . . . .	1907
12.562x_RectOut.nxc . . . . .	1908
12.563x_reladdr.nxc . . . . .	1909
12.564x_Release.nxc . . . . .	1910
12.565x_RemoteKeepAlive.nxc . . . . .	1911
12.566x_RemoteMessageRead.nxc . . . . .	1912
12.567x_RemoteMessageWrite.nxc . . . . .	1913
12.568x_RemotePlaySoundFile.nxc . . . . .	1914
12.569x_RemotePlayTone.nxc . . . . .	1915
12.570x_RemoteResetMotorPosition.nxc . . . . .	1916
12.571x_RemoteResetScaledValue.nxc . . . . .	1917
12.572x_RemoteSetInputMode.nxc . . . . .	1918
12.573x_RemoteSetOutputState.nxc . . . . .	1919
12.574x_RemoteStartProgram.nxc . . . . .	1920
12.575x_RemoteStopProgram.nxc . . . . .	1921
12.576x_RemoteStopSound.nxc . . . . .	1922
12.577x_remove.nxc . . . . .	1923
12.578x_rename.nxc . . . . .	1924
12.579x_RenameFile.nxc . . . . .	1925
12.580x_resetalltachocounts.nxc . . . . .	1926
12.581x_resetblocktachocount.nxc . . . . .	1927
12.582x_resetrotationcount.nxc . . . . .	1928
12.583x_ResetScreen.nxc . . . . .	1929
12.584x_ResetSensor.nxc . . . . .	1930
12.585x_ResetSleepTimer.nxc . . . . .	1931
12.586x_resettachocount.nxc . . . . .	1932
12.587x_ResizeFile.nxc . . . . .	1933
12.588x_ResolveHandle.nxc . . . . .	1934
12.589x_rewind.nxc . . . . .	1935
12.590x_rightstr.nxc . . . . .	1936
12.591x_rotatemotor.nxc . . . . .	1937
12.592x_rotatemotorex.nxc . . . . .	1938
12.593x_rotatemotorexpid.nxc . . . . .	1939
12.594x_rotatemotorpid.nxc . . . . .	1940

12.595x_RS485Control.nxc . . . . .	1941
12.596x_RS485DataAvailable.nxc . . . . .	1942
12.597x_RS485Exit.nxc . . . . .	1943
12.598x_RS485Init.nxc . . . . .	1944
12.599x_RS485Read.nxc . . . . .	1945
12.600x_RS485SendingData.nxc . . . . .	1946
12.601x_RS485Status.nxc . . . . .	1947
12.602x_RS485Uart.nxc . . . . .	1948
12.603x_RS485Write.nxc . . . . .	1949
12.604x_RunNRLinkMacro.nxc . . . . .	1950
12.605x_RunNRLinkMacroEx.nxc . . . . .	1951
12.606x_sendmessage.nxc . . . . .	1952
12.607x_sendremotebool.nxc . . . . .	1953
12.608x_SendRemoteNumber.nxc . . . . .	1954
12.609x_SendRemoteString.nxc . . . . .	1955
12.610x_SendResponseBool.nxc . . . . .	1956
12.611x_SendResponseNumber.nxc . . . . .	1957
12.612x_SendResponseString.nxc . . . . .	1958
12.613x_SendRS485Bool.nxc . . . . .	1959
12.614x_SendRS485Number.nxc . . . . .	1960
12.615x_SendRS485String.nxc . . . . .	1961
12.616x_Sensor.nxc . . . . .	1962
12.617x_SensorBoolean.nxc . . . . .	1963
12.618x_SensorDigiPinsDirection.nxc . . . . .	1964
12.619x_SensorDigiPinsOutputLevel.nxc . . . . .	1965
12.620x_SensorDigiPinsStatus.nxc . . . . .	1966
12.621x_SensorHTColorNum.nxc . . . . .	1967
12.622x_SensorHTCompass.nxc . . . . .	1968
12.623x_SensorHTEOPD.nxc . . . . .	1969
12.624x_SensorHTGyro.nxc . . . . .	1970
12.625x_SensorHTIRSeeker2ACDir.nxc . . . . .	1971
12.626x_SensorHTIRSeeker2Addr.nxc . . . . .	1972
12.627x_SensorHTIRSeeker2DCDir.nxc . . . . .	1973
12.628x_SensorHTIRSeekerDir.nxc . . . . .	1974
12.629x_SensorInvalid.nxc . . . . .	1975
12.630x_SensorMode.nxc . . . . .	1976

12.631x_SensorMSCompass.nxc . . . . .	1977
12.632x_SensorMSCompassEx.nxc . . . . .	1978
12.633x_SensorMSDROD.nxc . . . . .	1979
12.634x_SensorMSPressure.nxc . . . . .	1980
12.635x_SensorMSPressureRaw.nxc . . . . .	1981
12.636x_SensorNormalized.nxc . . . . .	1982
12.637x_SensorRaw.nxc . . . . .	1983
12.638x_SensorScaled.nxc . . . . .	1984
12.639x_SensorType.nxc . . . . .	1985
12.640x_sensorus.nxc . . . . .	1986
12.641x_SensorValue.nxc . . . . .	1987
12.642x_SensorValueBool.nxc . . . . .	1988
12.643x_SensorValueRaw.nxc . . . . .	1989
12.644x_SetAbortFlag.nxc . . . . .	1990
12.645x_SetBatteryState.nxc . . . . .	1991
12.646x_SetBluetoothState.nxc . . . . .	1992
12.647x_SetBTInputBuffer.nxc . . . . .	1993
12.648x_SetBTInputBufferInPtr.nxc . . . . .	1994
12.649x_SetBTInputBufferOutPtr.nxc . . . . .	1995
12.650x_SetBTOutputBuffer.nxc . . . . .	1996
12.651x_SetBTOutputBufferInPtr.nxc . . . . .	1997
12.652x_SetBTOutputBufferOutPtr.nxc . . . . .	1998
12.653x_SetButtonLongPressCount.nxc . . . . .	1999
12.654x_SetButtonLongReleaseCount.nxc . . . . .	2000
12.655x_SetButtonPressCount.nxc . . . . .	2001
12.656x_SetButtonReleaseCount.nxc . . . . .	2002
12.657x_SetButtonShortReleaseCount.nxc . . . . .	2003
12.658x_SetButtonState.nxc . . . . .	2004
12.659x_SetCommandFlags.nxc . . . . .	2005
12.660x_SetCustomSensorActiveStatus.nxc . . . . .	2006
12.661x_SetCustomSensorPercentFullScale.nxc . . . . .	2007
12.662x_SetCustomSensorZeroOffset.nxc . . . . .	2008
12.663x_SetDisplayContrast.nxc . . . . .	2009
12.664x_SetDisplayDisplay.nxc . . . . .	2010
12.665x_SetDisplayEraseMask.nxc . . . . .	2011
12.666x_SetDisplayFlags.nxc . . . . .	2012

12.667x_SetDisplayFont.nxc	2013
12.668x_SetDisplayNormal.nxc	2014
12.669x_SetDisplayPopup.nxc	2015
12.670x_SetDisplayTextLinesCenterFlags.nxc	2016
12.671x_SetDisplayUpdateMask.nxc	2017
12.672x_SetHSFlags.nxc	2018
12.673x_SetHSInputBuffer.nxc	2019
12.674x_SetHSInputBufferInPtr.nxc	2020
12.675x_SetHSInputBufferOutPtr.nxc	2021
12.676x_SetHSMODE.nxc	2022
12.677x_SetHSOutputBuffer.nxc	2023
12.678x_SetHSOutputBufferInPtr.nxc	2024
12.679x_SetHSOutputBufferOutPtr.nxc	2025
12.680x_SetHSSpeed.nxc	2026
12.681x_SetHSSState.nxc	2027
12.682x_SetHTColor2Mode.nxc	2028
12.683x_SetHTIRSeeker2Mode.nxc	2029
12.684x_SetInput.nxc	2030
12.685x_SetLongAbort.nxc	2031
12.686x_setmotorpwnfreq.nxc	2032
12.687x_SetOnBrickProgramPointer.nxc	2033
12.688x_setoutput.nxc	2034
12.689x_SetSensor.nxc	2035
12.690x_SetSensorBoolean.nxc	2036
12.691x_SetSensorColorBlue.nxc	2037
12.692x_SetSensorColorFull.nxc	2038
12.693x_SetSensorColorGreen.nxc	2039
12.694x_SetSensorColorNone.nxc	2040
12.695x_SetSensorColorRed.nxc	2041
12.696x_SetSensorDigiPinsDirection.nxc	2042
12.697x_SetSensorDigiPinsOutputLevel.nxc	2043
12.698x_SetSensorDigiPinsStatus.nxc	2044
12.699x_SetSensorHTEOPD.nxc	2045
12.700x_SetSensorHTGyro.nxc	2046
12.701x_SetSensorLight.nxc	2047
12.702x_SetSensorLowSpeed.nxc	2048

12.703x_SetSensorMode.nxc . . . . .	2049
12.704x_SetSensorMSDROD.nxc . . . . .	2050
12.705x_SetSensorMSPressure.nxc . . . . .	2051
12.706x_SetSensorSound.nxc . . . . .	2052
12.707x_SetSensorTouch.nxc . . . . .	2053
12.708x_SetSensorType.nxc . . . . .	2054
12.709x_SetSleepTime.nxc . . . . .	2055
12.710x_SetSleepTimeout.nxc . . . . .	2056
12.711x_SetSleepTimer.nxc . . . . .	2057
12.712x_SetSoundDuration.nxc . . . . .	2058
12.713x_SetSoundFlags.nxc . . . . .	2059
12.714x_SetSoundFrequency.nxc . . . . .	2060
12.715x_SetSoundMode.nxc . . . . .	2061
12.716x_SetSoundModuleState.nxc . . . . .	2062
12.717x_SetSoundSampleRate.nxc . . . . .	2063
12.718x_SetSoundVolume.nxc . . . . .	2064
12.719x_SetUIButton.nxc . . . . .	2065
12.720x_SetUIState.nxc . . . . .	2066
12.721x_SetUSBInputBuffer.nxc . . . . .	2067
12.722x_SetUSBInputBufferInPtr.nxc . . . . .	2068
12.723x_SetUSBInputBufferOutPtr.nxc . . . . .	2069
12.724x_SetUSBOutputBuffer.nxc . . . . .	2070
12.725x_SetUSBOutputBufferInPtr.nxc . . . . .	2071
12.726x_SetUSBOutputBufferOutPtr.nxc . . . . .	2072
12.727x_SetUSBPollBuffer.nxc . . . . .	2073
12.728x_SetUSBPollBufferInPtr.nxc . . . . .	2074
12.729x_SetUSBPollBufferOutPtr.nxc . . . . .	2075
12.730x_SetUSBState.nxc . . . . .	2076
12.731x_SetVMRunState.nxc . . . . .	2077
12.732x_SetVolume.nxc . . . . .	2078
12.733x_sign.nxc . . . . .	2079
12.734x_sin.nxc . . . . .	2080
12.735x_sind.nxc . . . . .	2081
12.736x_sinh.nxc . . . . .	2082
12.737x_sleepnow.nxc . . . . .	2083
12.738x_SleepTime.nxc . . . . .	2084



12.730x_SleepTimeout.nxc . . . . .	2085
12.740x_SleepTimer.nxc . . . . .	2086
12.741x_SoundDuration.nxc . . . . .	2087
12.742x_SoundFlags.nxc . . . . .	2088
12.743x_SoundFrequency.nxc . . . . .	2089
12.744x_SoundMode.nxc . . . . .	2090
12.745x_SoundSampleRate.nxc . . . . .	2091
12.746x_SoundState.nxc . . . . .	2092
12.747x_SoundVolume.nxc . . . . .	2093
12.748x_sprintf.nxc . . . . .	2094
12.749x_sqrt.nxc . . . . .	2095
12.750x_starttask.nxc . . . . .	2096
12.751x_stop.nxc . . . . .	2097
12.752x_stopalltasks.nxc . . . . .	2098
12.753x_StopSound.nxc . . . . .	2099
12.754x_stoptask.nxc . . . . .	2100
12.755x_strcat.nxc . . . . .	2101
12.756x_StrCatOld.nxc . . . . .	2102
12.757x_strcmp.nxc . . . . .	2103
12.758x_strcpy.nxc . . . . .	2104
12.759x_StrIndex.nxc . . . . .	2105
12.760x_string.nxc . . . . .	2106
12.761x_strlen.nxc . . . . .	2107
12.762x_StrLenOld.nxc . . . . .	2108
12.763x_strncat.nxc . . . . .	2109
12.764x_strncmp.nxc . . . . .	2110
12.765x_strncpy.nxc . . . . .	2111
12.766x_StrReplace.nxc . . . . .	2112
12.767x_StrToByteArray.nxc . . . . .	2113
12.768x_strtod.nxc . . . . .	2114
12.769x_strtol.nxc . . . . .	2115
12.770x_StrToNum.nxc . . . . .	2116
12.771x_strtoul.nxc . . . . .	2117
12.772x_SubStr.nxc . . . . .	2118
12.773x_syscall.nxc . . . . .	2119
12.774x_SysColorSensorRead.nxc . . . . .	2120

12.775x_syscommmbtcheckstatus.nxc . . . . .	2121
12.776x_SysCommBTConnection.nxc . . . . .	2122
12.777x_SysCommBTOnOff.nxc . . . . .	2123
12.778x_syscommmbtwrite.nxc . . . . .	2124
12.779x_syscommmexecutefunction.nxc . . . . .	2125
12.780x_SysCommHSCheckStatus.nxc . . . . .	2126
12.781x_SysCommHSControl.nxc . . . . .	2127
12.782x_SysCommHSRead.nxc . . . . .	2128
12.783x_SysCommHSWrite.nxc . . . . .	2129
12.784x_syscommmlscheckstatus.nxc . . . . .	2130
12.785x_syscommmlsread.nxc . . . . .	2131
12.786x_syscommmlswrite.nxc . . . . .	2132
12.787x_syscommmlswriteex.nxc . . . . .	2133
12.788x_SysComputeCalibValue.nxc . . . . .	2134
12.789x_SysDatalogGetTimes.nxc . . . . .	2135
12.790x_SysDatalogWrite.nxc . . . . .	2136
12.791x_sysdisplayexecutefunction.nxc . . . . .	2137
12.792x_sysdrawcircle.nxc . . . . .	2138
12.793x_sysdrawellipse.nxc . . . . .	2139
12.794x_sysdrawfont.nxc . . . . .	2140
12.795x_sysdrawgraphic.nxc . . . . .	2141
12.796x_sysdrawgraphicarray.nxc . . . . .	2142
12.797x_sysdrawline.nxc . . . . .	2143
12.798x_sysdrawpoint.nxc . . . . .	2144
12.799x_sysdrawpolygon.nxc . . . . .	2145
12.800x_sysdrawrect.nxc . . . . .	2146
12.801x_sysdrawtext.nxc . . . . .	2147
12.802x_sysfileclose.nxc . . . . .	2148
12.803x_sysfiledelete.nxc . . . . .	2149
12.804x_sysfilefindfirst.nxc . . . . .	2150
12.805x_sysfilefindnext.nxc . . . . .	2151
12.806x_sysfileopenappend.nxc . . . . .	2152
12.807x_sysfileopenread.nxc . . . . .	2153
12.808x_sysfileopenreadlinear.nxc . . . . .	2154
12.809x_sysfileopenwrite.nxc . . . . .	2155
12.810x_sysfileopenwritelinear.nxc . . . . .	2156

12.811x_sysfileopenwritenonlinear.nxc . . . . .	2157
12.812x_sysfileread.nxc . . . . .	2158
12.813x_sysfilerename.nxc . . . . .	2159
12.814x_sysfileresize.nxc . . . . .	2160
12.815x_sysfileresolvehandle.nxc . . . . .	2161
12.816x_sysfileseek.nxc . . . . .	2162
12.817x_sysfilewrite.nxc . . . . .	2163
12.818x_sysgetstarttick.nxc . . . . .	2164
12.819x_sysiomapread.nxc . . . . .	2165
12.820x_sysiomapreadbyid.nxc . . . . .	2166
12.821x_sysiomapwrite.nxc . . . . .	2167
12.822x_sysiomapwritebyid.nxc . . . . .	2168
12.823x_syskeepalive.nxc . . . . .	2169
12.824x_syslistfiles.nxc . . . . .	2170
12.825x_sysloaderexecutefunction.nxc . . . . .	2171
12.826x_sysmessageread.nxc . . . . .	2172
12.827x_sysmessagewrite.nxc . . . . .	2173
12.828x_sysrandomnumber.nxc . . . . .	2174
12.829x_sysreadbutton.nxc . . . . .	2175
12.830x_SysReadSemData.nxc . . . . .	2176
12.831x_syssetscreenmode.nxc . . . . .	2177
12.832x_SysSetSleepTimeout.nxc . . . . .	2178
12.833x_syssoundgetstate.nxc . . . . .	2179
12.834x_syssoundplayfile.nxc . . . . .	2180
12.835x_syssoundplaytone.nxc . . . . .	2181
12.836x_syssoundsetstate.nxc . . . . .	2182
12.837x_SysUpdateCalibCacheInfo.nxc . . . . .	2183
12.838x_SysWriteSemData.nxc . . . . .	2184
12.839x_tan.nxc . . . . .	2185
12.840x_tand.nxc . . . . .	2186
12.841x_tanh.nxc . . . . .	2187
12.842x_TextOut.nxc . . . . .	2188
12.843x_tolower.nxc . . . . .	2189
12.844x_toupper.nxc . . . . .	2190
12.845x_trunc.nxc . . . . .	2191
12.846x_UIButton.nxc . . . . .	2192

12.847x_UIState.nxc . . . . .	2193
12.848x_UiUsbState.nxc . . . . .	2194
12.849x_UnflattenVar.nxc . . . . .	2195
12.850x_USBInputBufferInPtr.nxc . . . . .	2196
12.851x_USBInputBufferOutPtr.nxc . . . . .	2197
12.852x_USBOutputBufferInPtr.nxc . . . . .	2198
12.853x_USBOutputBufferOutPtr.nxc . . . . .	2199
12.854x_USBPollBufferInPtr.nxc . . . . .	2200
12.855x_USBPollBufferOutPtr.nxc . . . . .	2201
12.856x_USBState.nxc . . . . .	2202
12.857x_VMRunState.nxc . . . . .	2203
12.858x_Volume.nxc . . . . .	2204
12.859x_wait.nxc . . . . .	2205
12.860x_Write.nxc . . . . .	2206
12.861x_WriteBytes.nxc . . . . .	2207
12.862x_WriteBytesEx.nxc . . . . .	2208
12.863x_writei2cregister.nxc . . . . .	2209
12.864x_WriteLn.nxc . . . . .	2210
12.865x_WriteLnString.nxc . . . . .	2211
12.866x_WriteNRLinkBytes.nxc . . . . .	2212
12.867x_WriteNRLinkBytesEx.nxc . . . . .	2213
12.868x_WriteString.nxc . . . . .	2214
12.869x_yield.nxc . . . . .	2215
12.870IBoxDemo.nxc . . . . .	2216
12.871ICircleDemo.nxc . . . . .	2217
12.872IRotateDemo.nxc . . . . .	2218
12.873IScaleDemo.nxc . . . . .	2219
12.874ITranslateDemo.nxc . . . . .	2220
12.875til_battery_1.nxc . . . . .	2222
12.876til_battery_2.nxc . . . . .	2223
12.877til_rpm.nxc . . . . .	2224

# **Chapter 1**

# **NXC Programmer's Guide**

**May 22, 2010**

**by John Hansen**

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



## **Chapter 2**

### **Introduction**

NXC stands for Not eXactly C.

It is a simple language for programming the LEGO MINDSTORMS NXT product. The NXT has a byte-code 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 Bricx 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/>.



## **Chapter 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 and 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 and Character 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");
```

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
```

### 3.1.5 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:

asm, bool, break, byte, case, char, const, continue, default, do, else, enum, false, float, for, goto, if, inline, int, long, mutex, priority, repeat, return, safecall, short, start, stop, string, struct, sub, switch, task, true, typedef, unsigned, until, void, while

## 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 safecall 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.

The argument list 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, strings, 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!
}

```

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.

Another 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.

### 3.2.4 Variables

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

Type Name	Information
bool	8 bit unsigned
byte, unsigned char	8 bit unsigned
char	8 bit signed
unsigned int	16 bit unsigned
short, int	16 bit signed
unsigned long	32 bit unsigned
long	32 bit signed
float	32 bit floating point value (IEEE-754)
mutex	Special type used for mutually exclusive code access
string	Array of byte with a null terminator
Structure (struct)	User-defined structure types
Array	Arrays of any type

Table 2. Variable Types

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
```

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
}
```

### 3.2.5 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 nine 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

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.

The if statement evaluates a condition. If the condition is true, it executes one statement (the consequence). An optional second statement (the alternative), preceded by the keyword else, can be 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 two syntaxes for an if statement are shown below.

```
if (condition) consequence
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.

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

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;
}
```

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.

Another kind of loop is the for loop. 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;
}
```

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.

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
```

The case and default labels of a switch statement are not statements in themselves - they are labels that precede statements. Multiple labels can precede the same statement. These labels have the syntax shown below.

```
case constant_expression :
default :
```

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.

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. For example, here is 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.

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:

```
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 slong
        s10005 slong
        bGTrue byte
    dseg ends
    mov     s10000, 0x0
    mov     s10005, s10000
    mov     s10000, 0x1
    cmp     GT, bGTrue, s10005, s10000
    set bGTrue, FALSE
    brtst   EQ, __lb100D5, bGTrue
    __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
__RETURN__, __RETURNS__	Used to return a signed value other than __RETVAL__ or __STRRETVAL__
__RETURNU__	Used to return an unsigned value.
__RETURNF__	Used to return a floating point value.
__RETVAL__	Writing to this 4-byte signed value returns it to the calling program
__GENRETVAL__	Writing to this generic value returns it to the calling program
__URETVAL__	Writing to this 4-byte unsigned value returns it to the calling program
__STRRETVAL__	Writing to this string value returns it to the calling program
__FLTRETVAL__	Writing to this 4-byte floating point value returns it to the calling program
__STRBUFFER__	This is primary string buffer which can be used to store intermediate string values.
__STRTMPBUFFER__	This is a secondary string buffer.
__TMPBYTE__	Use this temporary variable to write and return single byte signed values
__TMPWORD__	Use this temporary variable to write and return 2-byte signed values
__TMPLONG__	Use this temporary variable to write and return 4-byte signed values
__TMPULONG__	Use this temporary variable to write and return 4-byte unsigned values
__TMPFLOAT__	Use this temporary variable to write and return 4-byte floating point values
__I__	A local counter variable
__J__	A second local counter variable
__IncI__	Increment the local counter variable named I
__IncJ__	Increment the local counter variable named J
__DecI__	Decrement the local counter variable named I
__DecJ__	Decrement the local counter variable named J
__ResetI__	Reset the local counter variable named I to zero
__ResetJ__	Reset the local counter variable named J to zero
__THREADNAME__	The current thread name
__LINE__	The current line number
__FILE__	The current file name
__VER__	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

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.

You can start or stop tasks with the following statements. The stop statement is only supported if you are running the enhanced NBC/NXC firmware on your NXT.

```
start task_name;
stop task_name;
```

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;
```

Within loops (such as a while loop) you can use the break statement to exit the loop immediately. You can also use the continue statement to skip to the top of the next iteration of the loop.

```
break;
continue;
```

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.

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];
```

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.

## 3.4 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
<a href="#">abs()</a>	Absolute value	n/a		abs(x)
<a href="#">sign()</a>	Sign of operand	n/a		sign(x)
++, --	Pre/post increment, Pre/post decrement	right/left	variables only	++x, x++
-	Unary minus	right		-x
~	Bitwise negation (unary)	right		~123
!	Logical negation	right		!x
*, /, %	Multiplication, division, modulo	left		x * y
+, -	Addition, subtraction	left		x + y
<<, >>	Left and right shift	left		x << 4
<, >, <=, >=	relational operators	left		x < y
==, !=	equal to, not equal to	left		x == 1
&	Bitwise AND	left		x & y
^	Bitwise XOR	left		x ^ y
	Bitwise OR	left		x   y
&&	Logical AND	left		x && y
	Logical OR	left		x    y
? :	conditional value	n/a		x==1 ? y : z

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.

There are also two constant conditions - true and false - that always evaluate to true or false respectively. 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
true	always true
false	always false
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

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

## 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"
```



## **Chapter 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` ?.



## **Chapter 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.



# Chapter 6

## Module Index

### 6.1 Modules

Here is a list of all modules:

A simple 3D graphics library . . . . .	185
Graphics library actions . . . . .	714
Graphics library begin modes . . . . .	713
Graphics library cull mode . . . . .	717
Graphics library settings . . . . .	716
Miscellaneous NBC/NXC constants . . . . .	181
Data type limits . . . . .	711
Type aliases . . . . .	192
NXT Firmware Modules . . . . .	43
Button module . . . . .	55
Button module constants . . . . .	524
Button module IOMAP offsets . . . . .	529
Button name constants . . . . .	525
ButtonState constants . . . . .	527
Button module functions . . . . .	365
Button module types . . . . .	364
Comm module . . . . .	54
Comm module constants . . . . .	612
Bluetooth State constants . . . . .	615
Bluetooth hardware status constants . . . . .	617
Bluetooth state status constants . . . . .	616
Comm module IOMAP offsets . . . . .	635
Comm module interface function constants . . . . .	630
Comm module status code constants . . . . .	633
Device status constants . . . . .	629
Hi-speed port constants . . . . .	618
Hi-speed port SysCommHSControl constants . . . . .	621
Hi-speed port baud rate constants . . . . .	622
Hi-speed port combined UART constants . . . . .	628
Hi-speed port data bits constants . . . . .	625
Hi-speed port flags constants . . . . .	619
Hi-speed port parity constants . . . . .	627

Hi-speed port state constants . . . . .	620
Hi-speed port stop bits constants . . . . .	626
Mailbox constants . . . . .	485
Miscellaneous Comm module constants . . . . .	613
Comm module functions . . . . .	321
Comm module types . . . . .	320
Command module . . . . .	51
Command module constants . . . . .	52
Array operation constants . . . . .	465
Command module IOMAP offsets . . . . .	495
Communications specific errors . . . . .	492
Fatal errors . . . . .	488
General errors . . . . .	491
Program status constants . . . . .	494
Remote control (direct commands) errors . . . . .	493
System Call function constants . . . . .	467
Time constants . . . . .	478
VM state constants . . . . .	487
Command module functions . . . . .	304
Array API functions . . . . .	314
Command module types . . . . .	303
Display module . . . . .	62
Display module constants . . . . .	593
Display contrast constants . . . . .	606
Display flags . . . . .	605
Display module IOMAP offsets . . . . .	609
DisplayExecuteFunction constants . . . . .	598
Drawing option constants . . . . .	600
Font drawing option constants . . . . .	603
Line number constants . . . . .	475
Text line constants . . . . .	607
Display module functions . . . . .	248
Display module types . . . . .	246
IOCtrl module . . . . .	56
IOCtrl module constants . . . . .	497
IOCtrl module IOMAP offsets . . . . .	499
PowerOn constants . . . . .	498
IOCtrl module functions . . . . .	302
IOCtrl module types . . . . .	301
Input module . . . . .	45
Input module constants . . . . .	46
Color calibration constants . . . . .	553
Color calibration state constants . . . . .	552
Color sensor array indices . . . . .	550
Color values . . . . .	551
Input field constants . . . . .	549
Input module IOMAP offsets . . . . .	554
Input port constants . . . . .	193
NBC Input port constants . . . . .	543
NBC sensor mode constants . . . . .	547
NBC sensor type constants . . . . .	544
Sensor types and modes . . . . .	48

Combined sensor type and mode constants . . . . .	201
Sensor mode constants . . . . .	199
Sensor type constants . . . . .	196
Input module functions . . . . .	205
Basic analog sensor value names . . . . .	225
Input module types . . . . .	204
Loader module . . . . .	57
Loader module constants . . . . .	500
Loader module IOMAP offsets . . . . .	501
Loader module error codes . . . . .	502
Loader module function constants . . . . .	506
Loader module functions . . . . .	383
Loader module types . . . . .	382
Low Speed module . . . . .	60
LowSpeed module constants . . . . .	577
LSChannelState constants . . . . .	579
LSErrorType constants . . . . .	582
LSMode constants . . . . .	581
LSNoRestartOnRead constants . . . . .	585
LSState constants . . . . .	578
Low speed module IOMAP offsets . . . . .	583
Standard I2C constants . . . . .	587
Temperature sensor constants . . . . .	590
Ultrasonic sensor constants . . . . .	588
LowSpeed module functions . . . . .	279
Low level LowSpeed module functions . . . . .	293
LowSpeed module system call functions . . . . .	299
LowSpeed module types . . . . .	278
NXT firmware module IDs . . . . .	179
NXT firmware module names . . . . .	177
Output module . . . . .	49
Output module constants . . . . .	50
Output field constants . . . . .	569
Output module IOMAP offsets . . . . .	574
Output port constants . . . . .	557
Output port mode constants . . . . .	565
Output port option constants . . . . .	566
Output port regulation mode constants . . . . .	568
Output port run state constants . . . . .	567
Output port update flag constants . . . . .	561
PID constants . . . . .	559
Tachometer counter reset flags . . . . .	563
Output module functions . . . . .	226
Sound module . . . . .	58
Sound module constants . . . . .	510
Sound module IOMAP offsets . . . . .	514
Sound module miscellaneous constants . . . . .	516
SoundFlags constants . . . . .	511
SoundMode constants . . . . .	513
SoundState constants . . . . .	512
Tone constants . . . . .	517
Sound module functions . . . . .	269

Sound module types . . . . .	268
Ui module . . . . .	59
Ui module constants . . . . .	530
BluetoothState constants . . . . .	537
CommandFlags constants . . . . .	531
UIButton constants . . . . .	536
UIState constants . . . . .	533
Ui module IOMAP offsets . . . . .	540
VM run state constants . . . . .	539
Ui module functions . . . . .	372
Ui module types . . . . .	371
RIC Macro Wrappers . . . . .	170
Standard-C API functions . . . . .	184
cmath API . . . . .	400
cstdio API . . . . .	426
fseek origin constants . . . . .	435
cstdlib API . . . . .	436
cstdlib API types . . . . .	444
cstring API . . . . .	445
ctype API . . . . .	459
Third-party NXT devices . . . . .	183
HiTechnic API Functions . . . . .	63
HiTechnic device constants . . . . .	693
HiTechnic Color2 constants . . . . .	699
HiTechnic IRReceiver constants . . . . .	697
HiTechnic IRSeeker2 constants . . . . .	694
HiTechnic/mindsensors Power Function/IR Train constants . . . . .	679
IR Train channel constants . . . . .	684
PF/IR Train function constants . . . . .	683
Power Function CST options constants . . . . .	688
Power Function PWM option constants . . . . .	690
Power Function channel constants . . . . .	681
Power Function command constants . . . . .	680
Power Function mode constants . . . . .	682
Power Function output constants . . . . .	685
Power Function pin constants . . . . .	686
Power Function single pin function constants . . . . .	687
MindSensors API Functions . . . . .	101
MindSensors device constants . . . . .	701
MindSensors DIST-Nx constants . . . . .	702
MindSensors PSP-Nx constants . . . . .	704
MindSensors PSP-Nx button set 1 constants . . . . .	706
MindSensors PSP-Nx button set 2 constants . . . . .	707
MindSensors nRLink constants . . . . .	709
RCX constants . . . . .	641
RCX IR remote constants . . . . .	647
RCX and Scout opcode constants . . . . .	670
RCX and Scout sound constants . . . . .	650
RCX and Scout source constants . . . . .	665
RCX output constants . . . . .	642
RCX output direction constants . . . . .	645
RCX output mode constants . . . . .	644



---

RCX output power constants . . . . .	646
Scout constants . . . . .	652
Scout light constants . . . . .	653
Scout light rule constants . . . . .	662
Scout mode constants . . . . .	658
Scout motion rule constants . . . . .	659
Scout sound constants . . . . .	654
Scout sound set constants . . . . .	657
Scout special effect constants . . . . .	664
Scout touch rule constants . . . . .	661
Scout transmit rule constants . . . . .	663



## Chapter 7

# Data Structure Index

### 7.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ColorSensorReadType</a> (Parameters for the ColorSensorRead system call ) . . . . .	719
<a href="#">CommBTCheckStatusType</a> (Parameters for the CommBTCheckStatus system call ) . . . . .	721
<a href="#">CommBTConnectionType</a> (Parameters for the CommBTConnection system call ) . . . . .	722
<a href="#">CommBTOnOffType</a> (Parameters for the CommBTOnOff system call ) . . . . .	724
<a href="#">CommBTWriteType</a> (Parameters for the CommBTWrite system call ) . . . . .	725
<a href="#">CommExecuteFunctionType</a> (Parameters for the CommExecuteFunction system call ) . . . . .	726
<a href="#">CommHSCheckStatusType</a> (Parameters for the CommHSCheckStatus system call ) . . . . .	728
<a href="#">CommHSControlType</a> (Parameters for the CommHSControl system call ) . . . . .	729
<a href="#">CommHSReadWriteType</a> (Parameters for the CommHSReadWrite system call ) . . . . .	731
<a href="#">CommLSCheckStatusType</a> (Parameters for the CommLSCheckStatus system call ) . . . . .	732
<a href="#">CommLSReadType</a> (Parameters for the CommLSRead system call ) . . . . .	733
<a href="#">CommLSWriteExType</a> (Parameters for the CommLSWriteEx system call ) . . . . .	735
<a href="#">CommLSWriteType</a> (Parameters for the CommLSWrite system call ) . . . . .	737
<a href="#">ComputeCalibValueType</a> (Parameters for the ComputeCalibValue system call ) . . . . .	739
<a href="#">DatalogGetTimesType</a> (Parameters for the DatalogGetTimes system call ) . . . . .	741
<a href="#">DatalogWriteType</a> (Parameters for the DatalogWrite system call ) . . . . .	742
<a href="#">DisplayExecuteFunctionType</a> (Parameters for the DisplayExecuteFunction system call ) . . . . .	743
<a href="#">div_t</a> (Output type of the div function ) . . . . .	745
<a href="#">DrawCircleType</a> (Parameters for the DrawCircle system call ) . . . . .	746
<a href="#">DrawEllipseType</a> (Parameters for the DrawEllipse system call ) . . . . .	748
<a href="#">DrawFontType</a> (Parameters for the DrawFont system call ) . . . . .	750
<a href="#">DrawGraphicArrayType</a> (Parameters for the DrawGraphicArray system call ) . . . . .	752
<a href="#">DrawGraphicType</a> (Parameters for the DrawGraphic system call ) . . . . .	754
<a href="#">DrawLineType</a> (Parameters for the DrawLine system call ) . . . . .	756
<a href="#">DrawPointType</a> (Parameters for the DrawPoint system call ) . . . . .	758
<a href="#">DrawPolygonType</a> (Parameters for the DrawPolygon system call ) . . . . .	759
<a href="#">DrawRectType</a> (Parameters for the DrawRect system call ) . . . . .	760
<a href="#">DrawTextType</a> (Parameters for the DrawText system call ) . . . . .	762
<a href="#">FileCloseType</a> (Parameters for the FileClose system call ) . . . . .	764
<a href="#">FileDeleteType</a> (Parameters for the FileDelete system call ) . . . . .	765
<a href="#">FileFindType</a> (Parameters for the FileFind system call ) . . . . .	766
<a href="#">FileOpenType</a> (Parameters for the FileOpen system call ) . . . . .	768
<a href="#">FileReadWriteType</a> (Parameters for the FileReadWrite system call ) . . . . .	770

<a href="#">FileRenameType</a> (Parameters for the FileRename system call ) . . . . .	772
<a href="#">FileResizeType</a> (Parameters for the FileResize system call ) . . . . .	774
<a href="#">FileResolveHandleType</a> (Parameters for the FileResolveHandle system call ) . . . . .	775
<a href="#">FileSeekType</a> (Parameters for the FileSeek system call ) . . . . .	777
<a href="#">GetStartTickType</a> (Parameters for the GetStartTick system call ) . . . . .	779
<a href="#">IOMapReadByIDType</a> (Parameters for the IOMapReadByID system call ) . . . . .	780
<a href="#">IOMapReadType</a> (Parameters for the IOMapRead system call ) . . . . .	782
<a href="#">IOMapWriteByIDType</a> (Parameters for the IOMapWriteByID system call ) . . . . .	784
<a href="#">IOMapWriteType</a> (Parameters for the IOMapWrite system call ) . . . . .	786
<a href="#">KeepAliveType</a> (Parameters for the KeepAlive system call ) . . . . .	788
<a href="#">ldiv_t</a> (Output type of the ldiv function ) . . . . .	789
<a href="#">ListFilesType</a> (Parameters for the ListFiles system call ) . . . . .	790
<a href="#">LoaderExecuteFunctionType</a> (Parameters for the LoaderExecuteFunction system call ) . . . . .	792
<a href="#">LocationType</a> (A point on the NXT LCD screen ) . . . . .	794
<a href="#">MessageReadType</a> (Parameters for the MessageRead system call ) . . . . .	795
<a href="#">MessageWriteType</a> (Parameters for the MessageWrite system call ) . . . . .	797
<a href="#">RandomNumberType</a> (Parameters for the RandomNumber system call ) . . . . .	798
<a href="#">ReadButtonType</a> (Parameters for the ReadButton system call ) . . . . .	799
<a href="#">ReadSemDataType</a> (Parameters for the ReadSemData system call ) . . . . .	801
<a href="#">SetScreenModeType</a> (Parameters for the SetScreenMode system call ) . . . . .	802
<a href="#">SetSleepTimeoutType</a> (Parameters for the SetSleepTimeout system call ) . . . . .	803
<a href="#">SizeType</a> (Width and height dimensions for the DrawRect system call ) . . . . .	804
<a href="#">SoundGetStateType</a> (Parameters for the SoundGetState system call ) . . . . .	805
<a href="#">SoundPlayFileType</a> (Parameters for the SoundPlayFile system call ) . . . . .	806
<a href="#">SoundPlayToneType</a> (Parameters for the SoundPlayTone system call ) . . . . .	808
<a href="#">SoundSetStateType</a> (Parameters for the SoundSetState system call ) . . . . .	810
<a href="#">Tone</a> (Type used with the PlayTones API function ) . . . . .	811
<a href="#">UpdateCalibCacheInfoType</a> (Parameters for the UpdateCalibCacheInfo system call ) . . . . .	812
<a href="#">WriteSemDataType</a> (Parameters for the WriteSemData system call ) . . . . .	814

# Chapter 8

## File Index

### 8.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">NBCCommon.h</a> (Constants and macros common to both NBC and NXC ) . . . . .	817
<a href="#">NXCAPIDocs.h</a> (Additional documentation for the NXC API ) . . . . .	983
<a href="#">NXCDefs.h</a> (Constants, macros, and API functions for NXC ) . . . . .	984



# Chapter 9

## Module Documentation

### 9.1 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.*
- [IOCtrl module](#)  
*Constants and functions related to the IOCtrl module.*
- [Command module](#)  
*Constants and functions related to the Command 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.*

### 9.1.1 Detailed Description

Documentation common to all NXT firmware modules.



## 9.2 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.*

### 9.2.1 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
```

## 9.3 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.*

- [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.*

- [Input field constants](#)

*Constants for use with [SetInput\(\)](#) and [GetInput\(\)](#).*

- [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.*

- [Sensor types and modes](#)

*Constants that are used for defining sensor types and modes.*

### Defines

- `#define INPUT_DIGI0 1`
- `#define INPUT_DIGI1 2`
- `#define INPUT_CUSTOMINACTIVE 0x00`
- `#define INPUT_CUSTOM9V 0x01`
- `#define INPUT_CUSTOMACTIVE 0x02`
- `#define INPUT_INVALID_DATA 0x01`

### 9.3.1 Detailed Description

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

### 9.3.2 Define Documentation

#### 9.3.2.1 `#define INPUT_CUSTOM9V 0x01`

Custom sensor 9V

#### 9.3.2.2 `#define INPUT_CUSTOMACTIVE 0x02`

Custom sensor active

#### 9.3.2.3 `#define INPUT_CUSTOMINACTIVE 0x00`

Custom sensor inactive

#### 9.3.2.4 `#define INPUT_DIGI0 1`

Digital pin 0

#### 9.3.2.5 `#define INPUT_DIGI1 2`

Digital pin 1

#### 9.3.2.6 `#define INPUT_INVALID_DATA 0x01`

Invalid data flag

## 9.4 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.*

### 9.4.1 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.

## 9.5 Output module

Constants and functions related to the Output module.

### Modules

- [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.*

### 9.5.1 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.

[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.

## 9.6 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 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, or multi-motor synchronization.*

- [Output field constants](#)

*Constants for use with [SetOutput\(\)](#) and [GetOutput\(\)](#).*

- [Output module IOMAP offsets](#)

*Constant offsets into the Output module IOMAP structure.*

### 9.6.1 Detailed Description

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

## 9.7 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.*

### 9.7.1 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.

## 9.8 Command module constants

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

### Modules

- [Array operation constants](#)

*Constants for use with the [ArrayOp](#) function.*

- [System Call function constants](#)

*Constants for use in the [SysCall\(\)](#) function.*

- [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` [NO\\_ERR](#) 0

### 9.8.1 Detailed Description

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



## 9.8.2 Define Documentation

### 9.8.2.1 #define NO\_ERR 0

Successful execution of the specified command

**Examples:**

[ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysCommBTOnOff.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](#), and [ex\\_sysmessageread.nxc](#).

### 9.8.2.2 #define STAT\_COMM\_PENDING 32

Pending setup operation in progress

### 9.8.2.3 #define STAT\_MSG\_EMPTY\_MAILBOX 64

Specified mailbox contains no new messages

## 9.9 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.*

### 9.9.1 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.

## 9.10 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.*

### 9.10.1 Detailed Description

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

## 9.11 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.*

### 9.11.1 Detailed Description

Constants and functions related to the IOCtrl module. The NXT ioctl 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.

## 9.12 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.*

### 9.12.1 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.

## 9.13 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.*

### 9.13.1 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 [NBCCommon.h](#) for the complete list.

## 9.14 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.*

### 9.14.1 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.

## 9.15 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.*

### 9.15.1 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 `InvalidData` property to true after setting a new `Type`, and then wait in a loop for the NXT firmware to set `InvalidData` 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 [InvalidData](#) 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.

## 9.16 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.*

### 9.16.1 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.

## 9.17 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 [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.*
- 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 addr)  
*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.*

- 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)

*HTRCXCleatSensor function.*

- void [HTRCXCleatSound](#) (void)  
*HTRCXCleatSound function.*
- void [HTRCXCleatTimer](#) (const byte timer)  
*HTRCXCleatTimer 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.*

### 9.17.1 Detailed Description

Functions for accessing and modifying HiTechnic devices.

## 9.17.2 Function Documentation

### 9.17.2.1 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 Lowspeed 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](#).

### 9.17.2.2 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](#).

### 9.17.2.3 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](#).

#### 9.17.2.4 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](#).

#### 9.17.2.5 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\\_HTTPFRepeat.nxc](#).

#### 9.17.2.6 char HTTPFSingleOutputCST (const byte *port*, const byte *channel*, const byte *out*, const byte *func*) [inline]

HTTPFSingleOutputCST 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\\_HTTPFSingleOutputCST.nxc](#).

#### 9.17.2.7 char HTTPFSingleOutputPWM (const byte *port*, const byte *channel*, const byte *out*, const byte *func*) [inline]

HTTPFSingleOutputPWM 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 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 PWM function. See [Power Function PWM option constants](#).

**Returns:**

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

**Examples:**

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

**9.17.2.8 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 (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](#).

**9.17.2.9 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](#).

**9.17.2.10 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](#).

**9.17.2.11 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](#).

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

HTRCXCLEARAllEvents function. Send the CLEARAllEvents command to an RCX.

**Examples:**

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

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

HTRCXCLEARCounter function. Send the CLEARCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

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

**9.17.2.14 void HTRCXClearMsg (void) [inline]**

HTRCXClearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

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

**9.17.2.15 void HTRCXClearSensor (const byte *port*) [inline]**

HTRCXClearSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

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

**9.17.2.16 void HTRCXClearSound (void) [inline]**

HTRCXClearSound function. Send the ClearSound command to an RCX.

**Examples:**

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

**9.17.2.17 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](#).

**9.17.2.18 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](#).

**9.17.2.19 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](#).

**9.17.2.20 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](#).

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

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

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

**9.17.2.22 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](#).



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

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

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

**9.17.2.24 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](#).

**9.17.2.25 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](#).

**9.17.2.26 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](#).

**9.17.2.27 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](#).

**9.17.2.28 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](#).

**9.17.2.29 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](#).

**9.17.2.30 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](#).

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

HTRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

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

**9.17.2.32 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](#).

**9.17.2.33 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](#).

**9.17.2.34 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](#).

**9.17.2.35 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](#).

**9.17.2.36 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](#).

**9.17.2.37 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](#).

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

HTRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

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

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

HTRCXPing function. Send the Ping command to an RCX.

**Examples:**

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

**9.17.2.40 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](#).

**9.17.2.41 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](#).

**9.17.2.42 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](#).

**9.17.2.43 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](#).

**9.17.2.44 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](#).

**9.17.2.45 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](#).

**9.17.2.46 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](#).

**9.17.2.47 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](#).

**9.17.2.48 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](#).

**9.17.2.49 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](#).

**9.17.2.50 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](#).

**9.17.2.51 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](#).

**9.17.2.52 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](#).

**9.17.2.53 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](#).

**9.17.2.54 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](#).

**9.17.2.55 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](#).

**9.17.2.56 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](#).



**9.17.2.57 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](#).

**9.17.2.58 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](#).

**9.17.2.59 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](#).

**9.17.2.60 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](#).

**9.17.2.61 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](#).

**9.17.2.62 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](#).

**9.17.2.63 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](#).

**9.17.2.64 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](#).

**9.17.2.65 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](#).

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

HTRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

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

**9.17.2.67 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](#).

**9.17.2.68 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](#).

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

HTRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

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

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

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

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

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

HTScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

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

**9.17.2.72 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](#).

**9.17.2.73 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](#).

**9.17.2.74 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](#).

**9.17.2.75 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](#).

**9.17.2.76 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](#).

**9.17.2.77 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](#).

**9.17.2.78 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](#).

### 9.17.2.79 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](#).

### 9.17.2.80 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](#).

### 9.17.2.81 void HTScoutUnmuteSound (void) [inline]

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

#### Examples:

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

### 9.17.2.82 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](#).

**9.17.2.83 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](#).

**9.17.2.84 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](#).

**9.17.2.85 bool ReadSensorHTIRReceiver (const byte *port*, char & *pfdata*[ ])** **[inline]**

Read HiTechnic IRReceiver Power Function bytes. Read Power Function bytes from the HiTechnic IR-Receiver 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](#).

**9.17.2.86 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 IR-Receiver 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](#).

**9.17.2.87 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](#).

**9.17.2.88 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](#).

**9.17.2.89 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](#).

**9.17.2.90 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](#).

**9.17.2.91 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](#).

**9.17.2.92 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](#).

**9.17.2.93 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](#).

#### 9.17.2.94 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](#).

#### 9.17.2.95 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](#).

#### 9.17.2.96 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](#).

**9.17.2.97 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](#).

**9.17.2.98 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\\_SensorHTGyro.nxc](#).

**9.17.2.99 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](#).

**9.17.2.100 int SensorHTIRSeeker2Addr (const byte & *port*, const byte *addr*) [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](#).

*addr* The register address. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The IRSeeker2 register value.

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**9.17.2.101 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](#).

**9.17.2.102 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](#).

**9.17.2.103 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](#).

**9.17.2.104 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](#).

**9.17.2.105 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](#).

**9.17.2.106 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\\_SensorHTGyro.nxc](#), and [ex\\_SetSensorHTGyro.nxc](#).



## 9.18 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.*
- int [SensorMSPressure](#) (const byte &port)  
*Read mindsensors pressure sensor.*
- int [SensorMSCompass](#) (const byte &port)  
*Read mindsensors compass value.*
- int [SensorMSCompassEx](#) (const byte &port, const byte addr)  
*Read mindsensors compass value.*
- int [SensorMSDROD](#) (const byte &port)  
*Read mindsensors DROD value.*
- int [SensorMSPressureRaw](#) (const byte &port)  
*Read mindsensors raw pressure value.*
- bool [ReadSensorMSAccel](#) (const byte port, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSAccelEx](#) (const byte port, const byte addr, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSPlayStation](#) (const byte port, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)  
*Read mindsensors playstation controller values.*
- bool [ReadSensorMSPlayStationEx](#) (const byte port, const byte addr, 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, byte &x, byte &y, byte &z)  
*Read mindsensors tilt values.*
- bool [ReadSensorMSTiltEx](#) (const byte port, const byte addr, byte &x, byte &y, byte &z)  
*Read mindsensors tilt values.*
- int [MSReadValue](#) (const byte port, const byte reg, const byte numbytes)  
*Read a mindsensors device value.*
- int [MSReadValueEx](#) (const byte port, const byte addr, const byte reg, const byte numbytes)  
*Read a mindsensors device value.*
- char [MSEnergize](#) (const byte port)  
*Turn on power to device.*
- char [MSEnergizeEx](#) (const byte port, const byte addr)  
*Turn on power to device.*
- char [MSDeenergize](#) (const byte port)  
*Turn off power to device.*
- char [MSDeenergizeEx](#) (const byte port, const byte addr)  
*Turn off power to device.*
- char [MSADPAOn](#) (const byte port)  
*Turn on mindsensors ADPA mode.*
- char [MSADPAOnEx](#) (const byte port, const byte addr)  
*Turn on mindsensors ADPA mode.*
- char [MSADPAOff](#) (const byte port)  
*Turn off mindsensors ADPA mode.*
- char [MSADPAOffEx](#) (const byte port, const byte addr)  
*Turn off mindsensors ADPA mode.*
- char [DISTNxGP2D12](#) (const byte port)  
*Configure DISTNx as GP2D12.*
- char [DISTNxGP2D12Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2D12.*
- char [DISTNxGP2D120](#) (const byte port)  
*Configure DISTNx as GP2D120.*
- char [DISTNxGP2D120Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2D120.*

- char [DISTNxGP2YA02](#) (const byte port)  
*Configure DISTNx as GP2YA02.*
- char [DISTNxGP2YA02Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2YA02.*
- char [DISTNxGP2YA21](#) (const byte port)  
*Configure DISTNx as GP2YA21.*
- char [DISTNxGP2YA21Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2YA21.*
- int [DISTNxDistance](#) (const byte port)  
*Read DISTNx distance value.*
- int [DISTNxDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx distance value.*
- int [DISTNxMaxDistance](#) (const byte port)  
*Read DISTNx maximum distance value.*
- int [DISTNxMaxDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx maximum distance value.*
- int [DISTNxMinDistance](#) (const byte port)  
*Read DISTNx minimum distance value.*
- int [DISTNxMinDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx minimum distance value.*
- byte [DISTNxModuleType](#) (const byte port)  
*Read DISTNx module type value.*
- byte [DISTNxModuleTypeEx](#) (const byte port, const byte addr)  
*Read DISTNx module type value.*
- byte [DISTNxNumPoints](#) (const byte port)  
*Read DISTNx num points value.*
- byte [DISTNxNumPointsEx](#) (const byte port, const byte addr)  
*Read DISTNx num points value.*
- int [DISTNxVoltage](#) (const byte port)  
*Read DISTNx voltage value.*
- int [DISTNxVoltageEx](#) (const byte port, const byte addr)  
*Read DISTNx voltage value.*
- char [PSPNxDigital](#) (const byte &port)  
*Configure PSPNx in digital mode.*

- char [PSPNxDigitalEx](#) (const byte &port, const byte &addr)  
*Configure PSPNx in digital mode.*
- char [PSPNxAnalog](#) (const byte &port)  
*Configure PSPNx in analog mode.*
- char [PSPNxAnalogEx](#) (const byte &port, const byte &addr)  
*Configure PSPNx in analog mode.*
- char [NRLink2400](#) (const byte port)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink2400Ex](#) (const byte port, const byte addr)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink4800](#) (const byte port)  
*Configure NRLink in 4800 baud mode.*
- char [NRLink4800Ex](#) (const byte port, const byte addr)  
*Configure NRLink in 4800 baud mode.*
- char [NRLinkFlush](#) (const byte port)  
*Flush NRLink buffers.*
- char [NRLinkFlushEx](#) (const byte port, const byte addr)  
*Flush NRLink buffers.*
- char [NRLinkIRLong](#) (const byte port)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRLongEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRShort](#) (const byte port)  
*Configure NRLink in IR short mode.*
- char [NRLinkIRShortEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR short mode.*
- char [NRLinkSetPF](#) (const byte port)  
*Configure NRLink in power function mode.*
- char [NRLinkSetPFEx](#) (const byte port, const byte addr)  
*Configure NRLink in power function mode.*
- char [NRLinkSetRCX](#) (const byte port)  
*Configure NRLink in RCX mode.*
- char [NRLinkSetRCXEx](#) (const byte port, const byte addr)

*Configure NRLink in RCX mode.*

- char [NRLinkSetTrain](#) (const byte port)  
*Configure NRLink in IR train mode.*
- char [NRLinkSetTrainEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR train mode.*
- char [NRLinkTxRaw](#) (const byte port)  
*Configure NRLink in raw IR transmit mode.*
- char [NRLinkTxRawEx](#) (const byte port, const byte addr)  
*Configure NRLink in raw IR transmit mode.*
- byte [NRLinkStatus](#) (const byte port)  
*Read NRLink status.*
- byte [NRLinkStatusEx](#) (const byte port, const byte addr)  
*Read NRLink status.*
- char [RunNRLinkMacro](#) (const byte port, const byte macro)  
*Run NRLink macro.*
- char [RunNRLinkMacroEx](#) (const byte port, const byte addr, const byte macro)  
*Run NRLink macro.*
- char [WriteNRLinkBytes](#) (const byte port, const byte data[ ])  
*Write data to NRLink.*
- char [WriteNRLinkBytesEx](#) (const byte port, const byte addr, const byte data[ ])  
*Write data to NRLink.*
- bool [ReadNRLinkBytes](#) (const byte port, byte &data[ ])  
*Read data from NRLink.*
- bool [ReadNRLinkBytesEx](#) (const byte port, const byte addr, byte &data[ ])  
*Read data from NRLink.*
- char [MSIRTrain](#) (const byte port, const byte channel, const byte func)  
*MSIRTrain function.*
- char [MSIRTrainEx](#) (const byte port, const byte addr, const byte channel, const byte func)  
*MSIRTrainEx function.*
- char [MSPFComboDirect](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirect function.*
- char [MSPFComboDirectEx](#) (const byte port, const byte addr, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirectEx function.*

- char [MSPFComboPWM](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWM function.*
- char [MSPFComboPWMEEx](#) (const byte port, const byte addr, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWMEEx function.*
- char [MSPFRawOutput](#) (const byte port, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutput function.*
- char [MSPFRawOutputEx](#) (const byte port, const byte addr, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutputEx function.*
- char [MSPFRepeat](#) (const byte port, const byte count, const unsigned int delay)  
*MSPFRepeat function.*
- char [MSPFRepeatEx](#) (const byte port, const byte addr, const byte count, const unsigned int delay)  
*MSPFRepeatEx function.*
- char [MSPFSingleOutputCST](#) (const byte port, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCST function.*
- char [MSPFSingleOutputCSTEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCSTEx function.*
- char [MSPFSingleOutputPWM](#) (const byte port, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWM function.*
- char [MSPFSingleOutputPWMEEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWMEEx function.*
- char [MSPFSinglePin](#) (const byte port, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePin function.*
- char [MSPFSinglePinEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePinEx function.*
- char [MSPFTrain](#) (const byte port, const byte channel, const byte func)  
*MSPFTrain function.*
- char [MSPFTrainEx](#) (const byte port, const byte addr, const byte channel, const byte func)  
*MSPFTrainEx function.*
- void [MSRCXSetNRLinkPort](#) (const byte port)

*MSRCXSetIRLinkPort function.*

- void [MSRCXSetNRLinkPortEx](#) (const byte port, const byte addr)  
*MSRCXSetIRLinkPortEx 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 pwrsrc, 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 pwrsrc, 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.*

### 9.18.1 Detailed Description

Functions for accessing and modifying MindSensors devices.

### 9.18.2 Function Documentation

#### 9.18.2.1 int DISTNxDistance (const byte *port*) [**inline**]

Read DISTNx distance value. Read the mindsensors DISTNx distance 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 distance value.

##### Examples:

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

**9.18.2.2 int DISTNxDistanceEx (const byte *port*, const byte *addr*) [inline]**

Read DISTNx distance value. Read the mindsensors DISTNx sensor's distance value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

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

**9.18.2.3 char DISTNxGP2D12 (const byte *port*) [inline]**

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. 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 function call result.

**Examples:**

[ex\\_DISTNxGP2D12.nxc](#).

**9.18.2.4 char DISTNxGP2D120 (const byte *port*) [inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. 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 function call result.

**Examples:**

[ex\\_DISTNxGP2D120.nxc](#).

#### 9.18.2.5 `char DISTNxGP2D120Ex (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

##### Returns:

The function call result.

##### Examples:

[ex\\_DISTNxGP2D120Ex.nxc](#).

#### 9.18.2.6 `char DISTNxGP2D12Ex (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

##### Returns:

The function call result.

##### Examples:

[ex\\_DISTNxGP2D12Ex.nxc](#).

#### 9.18.2.7 `char DISTNxGP2YA02 (const byte port) [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](#).

##### Returns:

The function call result.

##### Examples:

[ex\\_DISTNxGP2YA02.nxc](#).

**9.18.2.8 char DISTNxGP2YA02Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA02Ex.nxc](#).

**9.18.2.9 char DISTNxGP2YA21 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA21.nxc](#).

**9.18.2.10 char DISTNxGP2YA21Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA21Ex.nxc](#).

**9.18.2.11 int DISTNxMaxDistance (const byte *port*) [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](#).

**Returns:**

The maximum distance value.

**Examples:**

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

**9.18.2.12 int DISTNxMaxDistanceEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The maximum distance value.

**Examples:**

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

**9.18.2.13 int DISTNxMinDistance (const byte *port*) [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](#).

**Returns:**

The distance value.

**Examples:**

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



**9.18.2.14 int DISTNxMinDistanceEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

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

**9.18.2.15 byte DISTNxModuleType (const byte *port*) [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](#).

**Returns:**

The module type value.

**Examples:**

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

**9.18.2.16 byte DISTNxModuleTypeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The module type value.

**Examples:**

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

### 9.18.2.17 byte DISTNxNumPoints (const byte *port*) [inline]

Read DISTNx num points value. Read the mindsensors DISTNx sensor's num points 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 num points value.

#### Examples:

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

### 9.18.2.18 byte DISTNxNumPointsEx (const byte *port*, const byte *addr*) [inline]

Read DISTNx num points value. Read the mindsensors DISTNx sensor's num points value. The port must be configured as a Low-speed port before using this function.

#### Parameters:

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

*addr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The num points value.

#### Examples:

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

### 9.18.2.19 int DISTNxVoltage (const byte *port*) [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](#).

#### Returns:

The voltage value.

#### Examples:

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

**9.18.2.20 int DISTNxVoltageEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The voltage value.

**Examples:**

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

**9.18.2.21 char MSADPAOff (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.22 char MSADPAOffEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

### 9.18.2.23 char MSADPAOn (const byte *port*) [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](#).

#### Returns:

The function call result.

#### Examples:

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

### 9.18.2.24 char MSADPAOnEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

#### Returns:

The function call result.

#### Examples:

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

### 9.18.2.25 char MSDeenergize (const byte *port*) [inline]

Turn off power to device. Turn power 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](#).

#### Returns:

The function call result.

#### Examples:

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

**9.18.2.26 char MSDeenergizeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.27 char MSEnergize (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.28 char MSEnergizeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

### 9.18.2.29 char MSIRTrain (const byte *port*, 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 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 Lowspeed 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\\_MSIRTrain.nxc](#).

### 9.18.2.30 char MSIRTrainEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *func*) [inline]

MSIRTrainEx 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](#).
- addr* 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\\_MSIRTrainEx.nxc](#).

### 9.18.2.31 char MSPFComboDirect (const byte *port*, 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](#).
- 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](#).

**9.18.2.32 char MSPFComboDirectEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

MSPFComboDirectEx 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](#).
- addr* 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\\_MSPFComboDirectEx.nxc](#).

**9.18.2.33 char MSPFComboPWM (const byte *port*, 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](#).

*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](#).

#### 9.18.2.34 char MSPFComboPWMEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

MSPFComboPWMEx function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the mindsensors NLink 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](#).

*addr* 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\\_MSPFComboPWMEx.nxc](#).

#### 9.18.2.35 char MSPFRawOutput (const byte *port*, 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 NLink 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\\_MSPFRawOutput.nxc](#).

**9.18.2.36 char MSPFRawOutputEx (const byte *port*, const byte *addr*, const byte *nibble0*, const byte *nibble1*, const byte *nibble2*) [inline]**

MSPFRawOutputEx 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](#).

*addr* 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\\_MSPFRawOutputEx.nxc](#).

**9.18.2.37 char MSPFRepeat (const byte *port*, 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 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\\_MSPFRepeat.nxc](#).

### 9.18.2.38 `char MSPFRepeatEx (const byte port, const byte addr, const byte count, const unsigned int delay) [inline]`

MSPFRepeatEx function. Repeat sending the last Power Function command using the mindsensors NR-Link 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](#).
- addr* 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\\_MSPFRepeatEx.nxc](#).

### 9.18.2.39 `char MSPFSingleOutputCST (const byte port, 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](#).
- 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](#).

**9.18.2.40** `char MSPFSingleOutputCSTEx (const byte port, const byte addr, const byte channel, const byte out, const byte func) [inline]`

MSPFSingleOutputCSTEx 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](#).  
*addr* 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\\_MSPFSingleOutputCSTEx.nxc](#).

**9.18.2.41** `char MSPFSingleOutputPWM (const byte port, 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](#).  
*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](#).

#### 9.18.2.42 `char MSPFSingleOutputPWMEx (const byte port, const byte addr, const byte channel, const byte out, const byte func) [inline]`

MSPFSingleOutputPWMEx 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](#).
- addr* 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\\_MSPFSingleOutputPWMEx.nxc](#).

#### 9.18.2.43 `char MSPFSinglePin (const byte port, 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](#).
- 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](#).

**9.18.2.44** `char MSPFSinglePinEx (const byte port, const byte addr, const byte channel, const byte out, const byte pin, const byte func, bool cont) [inline]`

MSPFSinglePinEx 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](#).
- addr* 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\\_MSPFSinglePinEx.nxc](#).

**9.18.2.45** `char MSPFTrain (const byte port, 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](#).
- 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](#).

#### 9.18.2.46 `char MSPFTrainEx (const byte port, const byte addr, const byte channel, const byte func) [inline]`

MSPFTrainEx 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](#).
- addr* 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\\_MSPFTrainEx.nxc](#).

#### 9.18.2.47 `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](#).

#### 9.18.2.48 `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](#).

**9.18.2.49 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](#).

**9.18.2.50 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](#).

**9.18.2.51 void MSRCXBoot (void) [inline]**

MSRCXBoot function. Send the Boot command to an RCX.

**Examples:**

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

**9.18.2.52 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](#).

**9.18.2.53 void MSRCXClearAllEvents (void) [inline]**

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

**Examples:**

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

**9.18.2.54 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](#).

**9.18.2.55 void MSRCXClearMsg (void) [inline]**

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

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

**9.18.2.56 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](#).

**9.18.2.57 void MSRCXClearSound (void) [inline]**

MSRCXClearSound function. Send the ClearSound command to an RCX.

**Examples:**

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



**9.18.2.58 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](#).

**9.18.2.59 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](#).

**9.18.2.60 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](#).

**9.18.2.61 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](#).

**9.18.2.62 void MSRCXDeleteSubs (void) [inline]**

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

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

**9.18.2.63 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](#).

**9.18.2.64 void MSRCXDeleteTasks (void) [inline]**

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

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

**9.18.2.65 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](#).

**9.18.2.66 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](#).

**9.18.2.67 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](#).

**9.18.2.68 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](#).

**9.18.2.69 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](#).

**9.18.2.70 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](#).

**9.18.2.71 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](#).

**9.18.2.72 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](#).

**9.18.2.73 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](#).

**9.18.2.74 void MSRCXMuteSound (void) [inline]**

MSRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

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

**9.18.2.75 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](#).

**9.18.2.76 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](#).

**9.18.2.77 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](#).

**9.18.2.78 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](#).

**9.18.2.79 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](#).

**9.18.2.80 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](#).

**9.18.2.81 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](#).

**9.18.2.82 void MSRCXPBTurnOff (void) [inline]**

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

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

**9.18.2.83 void MSRCXPing (void) [inline]**

MSRCXPing function. Send the Ping command to an RCX.

**Examples:**

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

**9.18.2.84 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](#).

**9.18.2.85 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](#).

**9.18.2.86 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](#).

**9.18.2.87 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](#).

**9.18.2.88 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](#).

**9.18.2.89 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](#).

**9.18.2.90 void MSRCXReset (void) [inline]**

MSRCXReset function. Send the Reset command to an RCX.

**Examples:**

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



**9.18.2.91 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](#).

**9.18.2.92 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](#).

**9.18.2.93 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](#).

**9.18.2.94 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](#).

### 9.18.2.95 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](#).

### 9.18.2.96 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](#).

### 9.18.2.97 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](#).

### 9.18.2.98 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](#).

**9.18.2.99 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](#).

**9.18.2.100 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](#).

**9.18.2.101 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](#).

**9.18.2.102 void MSRCXSetNRLinkPort (const byte *port*) [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](#).

**Examples:**

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

**9.18.2.103 void MSRCXSetNRLinkPortEx (const byte *port*, const byte *addr*) [inline]**

MSRCXSetNRLinkPortEx 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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Examples:**

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

**9.18.2.104 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](#).

**9.18.2.105 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](#).

**9.18.2.106 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](#).

**9.18.2.107 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](#).

**9.18.2.108 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](#).

**9.18.2.109 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](#).

**9.18.2.110 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](#).

**9.18.2.111 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](#).

**9.18.2.112 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](#).

**9.18.2.113 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](#).

**9.18.2.114 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](#).

**9.18.2.115 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](#).

**9.18.2.116 void MSRCXStopAllTasks (void) [inline]**

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

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

**9.18.2.117 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](#).

**9.18.2.118 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](#).

**9.18.2.119 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](#).

**9.18.2.120 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](#).



**9.18.2.121 void MSRCXUnlock (void) [inline]**

MSRCXUnlock function. Send the Unlock command to an RCX.

**Examples:**

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

**9.18.2.122 void MSRCXUnmuteSound (void) [inline]**

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

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

**9.18.2.123 int MSReadValue (const byte *port*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one or two byte value from a mindsensors sensor. The value must be stored with the least significant byte (LSB) first. 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](#).

*reg* The device register to read.

*numbytes* The number of bytes to read. Only 1 or 2 byte values are supported.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.124 int MSReadValueEx (const byte *port*, const byte *addr*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one or two byte value from a mindsensors sensor. The value must be stored with the least significant byte (LSB) first. 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](#).

*addr* 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 or 2 byte values are supported.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.125 void MSScoutCalibrateSensor (void) [inline]**

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

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

**9.18.2.126 void MSScoutMuteSound (void) [inline]**

MSScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

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

**9.18.2.127 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](#).

**9.18.2.128 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](#).

**9.18.2.129 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](#).

**9.18.2.130 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](#).

**9.18.2.131 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](#).

**9.18.2.132 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](#).

### 9.18.2.133 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](#).

### 9.18.2.134 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](#).

### 9.18.2.135 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](#).

### 9.18.2.136 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](#).

**9.18.2.137 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](#).

**9.18.2.138 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](#).

**9.18.2.139 void MSScoutUnmuteSound (void) [inline]**

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

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

**9.18.2.140 char NRLink2400 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400.nxc](#).

**9.18.2.141 char NRLink2400Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400Ex.nxc](#).

**9.18.2.142 char NRLink4800 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800.nxc](#).

**9.18.2.143** `char NRLink4800Ex (const byte port, const byte addr) [inline]`

Configure NRLink in 4800 baud mode. Configure the mindsensors NRLink device in 4800 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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800Ex.nxc](#).

**9.18.2.144** `char NRLinkFlush (const byte port) [inline]`

Flush NRLink buffers. Flush the mindsensors NRLink device buffers. 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 function call result.

**Examples:**

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

**9.18.2.145** `char NRLinkFlushEx (const byte port, const byte addr) [inline]`

Flush NRLink buffers. Flush the mindsensors NRLink device buffers. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.146 char NRLinkIRLong (const byte *port*) [inline]**

Configure NRLink in IR long mode. Configure the mindsensors NRLink device in IR long mode. 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 function call result.

**Examples:**

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

**9.18.2.147 char NRLinkIRLongEx (const byte *port*, const byte *addr*) [inline]**

Configure NRLink in IR long mode. Configure the mindsensors NRLink device in IR long mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.148 char NRLinkIRShort (const byte *port*) [inline]**

Configure NRLink in IR short mode. Configure the mindsensors NRLink device in IR short mode. 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 function call result.

**Examples:**

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



**9.18.2.149 char NRLinkIRShortEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.150 char NRLinkSetPF (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.151 char NRLinkSetPFEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.152 char NRLinkSetRCX (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.153 char NRLinkSetRCXEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.154 char NRLinkSetTrain (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.155** `char NRLinkSetTrainEx (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.156** `byte NRLinkStatus (const byte port) [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](#).

**Returns:**

The mindsensors NRLink status.

**Examples:**

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

**9.18.2.157** `byte NRLinkStatusEx (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors NRLink status.

**Examples:**

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

**9.18.2.158 char NRLinkTxRaw (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.159 char NRLinkTxRawEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.160 char PSPNxAnalog (const byte & *port*) [inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. 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.

**Examples:**

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

**9.18.2.161 char PSPNxAnalogEx (const byte & *port*, const byte & *addr*) [inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.162 char PSPNxDigital (const byte & *port*) [inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. 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.

**Examples:**

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

**9.18.2.163 char PSPNxDigitalEx (const byte & *port*, const byte & *addr*) [inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.164** `bool ReadNRLinkBytes (const byte port, 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](#).  
*data* A byte array that will contain the data read from the device on output.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.165** `bool ReadNRLinkBytesEx (const byte port, const byte addr, 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](#).  
*addr* 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\\_ReadNRLinkBytesEx.nxc](#).

**9.18.2.166** `bool ReadSensorMSAccel (const byte port, 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](#).  
*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](#).

**9.18.2.167 bool ReadSensorMSAccelEx (const byte *port*, const byte *addr*, 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](#).  
*addr* 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\\_ReadSensorMSAccelEx.nxc](#).

**9.18.2.168 bool ReadSensorMSPlayStation (const byte *port*, 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 Lowspeed port before using this function.

**Parameters:**

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

### 9.18.2.169 **bool ReadSensorMSPlayStationEx (const byte *port*, const byte *addr*, 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 Lowspeed port before using this function.

#### Parameters:

*port* The sensor port. See [Input port constants](#).  
*addr* 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\\_ReadSensorMSPlayStationEx.nxc](#).

### 9.18.2.170 **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](#).



**9.18.2.171 bool ReadSensorMSTilt (const byte *port*, 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 Low-speed port before using this function.

**Parameters:**

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

*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](#).

**9.18.2.172 bool ReadSensorMSTiltEx (const byte *port*, const byte *addr*, 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 Low-speed port before using this function.

**Parameters:**

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

*addr* 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\\_ReadSensorMSTiltEx.nxc](#).

**9.18.2.173 char RunNRLinkMacro (const byte *port*, const byte *macro*) [inline]**

Run NRLink macro. Run the specified mindsensors NRLink device macro. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*macro* The address of the macro to execute.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.174 char RunNRLinkMacroEx (const byte *port*, const byte *addr*, 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](#).

*addr* 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\\_RunNRLinkMacroEx.nxc](#).

**9.18.2.175 int SensorMSCompass (const byte & *port*) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

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

**Returns:**

The mindsensors compass value

**Examples:**

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

**9.18.2.176 int SensorMSCompassEx (const byte & *port*, const byte *addr*) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors compass value

**Examples:**

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

**9.18.2.177 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](#).

**9.18.2.178 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](#).

**9.18.2.179 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](#).

**9.18.2.180 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](#).

**9.18.2.181 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](#).

**9.18.2.182 char WriteNRLinkBytes (const byte *port*, 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](#).

*data* A byte array containing the data to write.

**Returns:**

The function call result.

**Examples:**

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

**9.18.2.183 char WriteNRLinkBytesEx (const byte *port*, const byte *addr*, 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](#).

*addr* 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\\_WriteNRLinkBytesEx.nxc](#).

## 9.19 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.*

### 9.19.1 Detailed Description

Macro wrappers for use in defining RIC byte arrays.

### 9.19.2 Define Documentation

#### 9.19.2.1 `#define RICArg(_arg) ((_arg)|0x1000)`

Output an RIC parameterized argument.

#### Parameters:

**`_arg`** The argument that you want to parameterize.

**Examples:**

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

**9.19.2.2 #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](#).

**9.19.2.3 #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](#).

**9.19.2.4 #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.

**9.19.2.5 #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.



**9.19.2.6 #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](#).

**9.19.2.7 #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.

**9.19.2.8 #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](#).

**9.19.2.9 #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](#).

**9.19.2.10** `#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.

**9.19.2.11** `#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](#).

**9.19.2.12** `#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.

**9.19.2.13** `#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).

**9.19.2.14** `#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](#).

**9.19.2.15** `#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.

**9.19.2.16** `#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](#).

**9.19.2.17** `#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](#).

**9.19.2.18 #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](#).

**9.19.2.19 #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

**9.19.2.20 #define RICSpriteData(...) \_\_VA\_ARGS\_\_**

Output RIC sprite data.

**Examples:**

[ex\\_dispgaout.nxc](#), [ex\\_dispgaoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

## 9.20 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"`

### 9.20.1 Detailed Description

Constant string names for all the NXT firmware modules.

### 9.20.2 Define Documentation

#### 9.20.2.1 `#define ButtonModuleName "Button.mod"`

The button module name

#### 9.20.2.2 `#define CommandModuleName "Command.mod"`

The command module name

#### Examples:

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

#### 9.20.2.3 `#define CommModuleName "Comm.mod"`

The Comm module name

#### 9.20.2.4 `#define DisplayModuleName "Display.mod"`

The display module name

#### 9.20.2.5 `#define InputModuleName "Input.mod"`

The input module name.

**9.20.2.6 #define IOCtrlModuleName "IOCtrl.mod"**

The IOCtrl module name

**9.20.2.7 #define LoaderModuleName "Loader.mod"**

The Loader module name

**9.20.2.8 #define LowSpeedModuleName "Low Speed.mod"**

The low speed module name

**9.20.2.9 #define OutputModuleName "Output.mod"**

The output module name

**9.20.2.10 #define SoundModuleName "Sound.mod"**

The sound module name

**Examples:**

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

**9.20.2.11 #define UIModuleName "Ui.mod"**

The Ui module name

## 9.21 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`

### 9.21.1 Detailed Description

Constant numeric IDs for all the NXT firmware modules.

### 9.21.2 Define Documentation

#### 9.21.2.1 `#define ButtonModuleID 0x00040001`

The button module ID

#### 9.21.2.2 `#define CommandModuleID 0x00010001`

The command module ID

#### Examples:

`ex_reladdr.nxc`, and `ex_sysiomapreadbyid.nxc`.

#### 9.21.2.3 `#define CommModuleID 0x00050001`

The Comm module ID

#### 9.21.2.4 `#define DisplayModuleID 0x000A0001`

The display module ID

#### 9.21.2.5 `#define InputModuleID 0x00030001`

The input module ID

**9.21.2.6 #define IOCtrlModuleID 0x00060001**

The IOCtrl module ID

**9.21.2.7 #define LoaderModuleID 0x00090001**

The Loader module ID

**9.21.2.8 #define LowSpeedModuleID 0x000B0001**

The low speed module ID

**9.21.2.9 #define OutputModuleID 0x00020001**

The output module ID

**9.21.2.10 #define SoundModuleID 0x00080001**

The sound module ID

**Examples:**

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

**9.21.2.11 #define UIModuleID 0x000C0001**

The Ui module ID



## 9.22 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.*

- [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`

#### 9.22.1 Detailed Description

Miscellaneous constants for use in NBC and NXC.

#### 9.22.2 Define Documentation

##### 9.22.2.1 `#define DEGREES_PER_RADIAN 180/PI`

Used for converting from radians to degrees

##### 9.22.2.2 `#define FALSE 0`

A false value

##### 9.22.2.3 `#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](#).

**9.22.2.4 #define PI 3.141593**

A constant for PI

**Examples:**

[ex\\_dispfnout.nxc](#), and [ex\\_string.nxc](#).

**9.22.2.5 #define RADIANS\_PER\_DEGREE PI/180**

Used for converting from degrees to radians

**9.22.2.6 #define TRUE 1**

A true value

**Examples:**

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

## 9.23 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.*

### 9.23.1 Detailed Description

Documentation for NXT devices made by companies other than LEGO such as HiTechnic, mindsensors.com, and CodaTex.

## 9.24 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.*

### 9.24.1 Detailed Description

Documentation for various Standard-C library routines.

## 9.25 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.*

### 9.25.1 Detailed Description

Documentation for a simple 3D graphics library. The library code was written by Arno van der Vegt.

### 9.25.2 Function Documentation

#### 9.25.2.1 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](#).

**9.25.2.2 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](#).

**9.25.2.3 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.

**9.25.2.4 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.

**9.25.2.5 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](#).

#### 9.25.2.6 `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.

#### 9.25.2.7 `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](#).

#### 9.25.2.8 `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](#).

#### 9.25.2.9 `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](#).



**9.25.2.10 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.

**9.25.2.11 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](#).

**9.25.2.12 void glEnd () [inline]**

Finish a polygon for the current object. Stop defining a polygon surface for the current graphics object.

**9.25.2.13 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.

**9.25.2.14 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](#).

### 9.25.2.15 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](#).

### 9.25.2.16 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](#).

### 9.25.2.17 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).

### 9.25.2.18 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](#).

**9.25.2.19 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](#).

**9.25.2.20 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.

**9.25.2.21 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.

**9.25.2.22 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.

## 9.26 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

### 9.26.1 Detailed Description

Short type aliases indicating signed/unsigned and bit count for each type.

### 9.26.2 Define Documentation

#### 9.26.2.1 `#define s16` int

Signed 16 bit type

#### 9.26.2.2 `#define s32` long

Signed 32 bit type

#### 9.26.2.3 `#define s8` char

Signed 8 bit type

#### 9.26.2.4 `#define u16` unsigned int

Unsigned 16 bit type

#### 9.26.2.5 `#define u32` unsigned long

Unsigned 32 bit type

#### 9.26.2.6 `#define u8` unsigned char

Unsigned 8 bit type

## 9.27 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

### 9.27.1 Detailed Description

Input port constants are used when calling NXC sensor control API functions.

### 9.27.2 Define Documentation

#### 9.27.2.1 #define S1 0

Input port 1

#### Examples:

```
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_CustomSensorActiveStatus.nxc, ex_CustomSensorPercentFullScale.nxc,
ex_CustomSensorZeroOffset.nxc, ex_DISTNxDistance.nxc, ex_DISTNxDistanceEx.nxc,
ex_DISTNxGP2D12.nxc, ex_DISTNxGP2D120.nxc, ex_DISTNxGP2D120Ex.nxc, ex_
DISTNxGP2D12Ex.nxc, ex_DISTNxGP2YA02.nxc, ex_DISTNxGP2YA02Ex.nxc, ex_
DISTNxGP2YA21.nxc, ex_DISTNxGP2YA21Ex.nxc, ex_DISTNxMaxDistance.nxc, ex_
DISTNxMaxDistanceEx.nxc, ex_DISTNxMinDistance.nxc, ex_DISTNxMinDistanceEx.nxc,
ex_DISTNxModuleType.nxc, ex_DISTNxModuleTypeEx.nxc, ex_DISTNxNumPoints.nxc,
ex_DISTNxNumPointsEx.nxc, ex_DISTNxVoltage.nxc, ex_DISTNxVoltageEx.nxc, ex_
GetInput.nxc, ex_GetLSInputBuffer.nxc, ex_GetLSOutputBuffer.nxc, ex_HTIRTrain.nxc,
ex_HTTPFComboDirect.nxc, ex_HTTPFComboPWM.nxc, ex_HTTPFRawOutput.nxc, ex_
HTTPFRepeat.nxc, ex_HTTPFSingleOutputCST.nxc, ex_HTTPFSingleOutputPWM.nxc,
ex_HTTPFSinglePin.nxc, ex_HTTPFTrain.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_I2CDeviceIdEx.nxc, ex_I2CDeviceInfo.nxc, ex_I2CDeviceInfoEx.nxc, ex_I2CRead.nxc, ex_
I2CSendCommand.nxc, ex_I2CSendCommandEx.nxc, ex_I2CStatus.nxc, ex_I2CVendorId.nxc,
ex_I2CVendorIdEx.nxc, ex_I2CVersion.nxc, ex_I2CVersionEx.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_MSADPAOffEx.nxc, ex_MSADPAOn.nxc,
ex_MSADPAOnEx.nxc, ex_MSDeenergize.nxc, ex_MSDeenergizeEx.nxc, ex_MSEnergize.nxc,
ex_MSEnergizeEx.nxc, ex_MSIRTrain.nxc, ex_MSIRTrainEx.nxc, ex_MSPFComboDirect.nxc,
ex_MSPFComboDirectEx.nxc, ex_MSPFComboPWM.nxc, ex_MSPFComboPWMLEx.nxc,
```

[ex\\_MSPFRawOutput.nxc](#), [ex\\_MSPFRawOutputEx.nxc](#), [ex\\_MSPFRepeat.nxc](#), [ex\\_MSPFRepeatEx.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputCSTEx.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSingleOutputPWMEEx.nxc](#), [ex\\_MSPFSinglePin.nxc](#), [ex\\_MSPFSinglePinEx.nxc](#), [ex\\_MSPFTrain.nxc](#), [ex\\_MSPFTrainEx.nxc](#), [ex\\_MSRCXAddToDatalog.nxc](#), [ex\\_MSRCXClearSensor.nxc](#), [ex\\_MSRCXSetNRLinkPort.nxc](#), [ex\\_MSRCXSetNRLinkPortEx.nxc](#), [ex\\_MSRCXSetSensorMode.nxc](#), [ex\\_MSRCXSetSensorType.nxc](#), [ex\\_MSRCXSumVar.nxc](#), [ex\\_MSReadValue.nxc](#), [ex\\_MSReadValueEx.nxc](#), [ex\\_NRLink2400.nxc](#), [ex\\_NRLink2400Ex.nxc](#), [ex\\_NRLink4800.nxc](#), [ex\\_NRLink4800Ex.nxc](#), [ex\\_NRLinkFlush.nxc](#), [ex\\_NRLinkFlushEx.nxc](#), [ex\\_NRLinkIRLong.nxc](#), [ex\\_NRLinkIRLongEx.nxc](#), [ex\\_NRLinkIRShort.nxc](#), [ex\\_NRLinkIRShortEx.nxc](#), [ex\\_NRLinkSetPF.nxc](#), [ex\\_NRLinkSetPFEx.nxc](#), [ex\\_NRLinkSetRCX.nxc](#), [ex\\_NRLinkSetRCXEx.nxc](#), [ex\\_NRLinkSetTrain.nxc](#), [ex\\_NRLinkSetTrainEx.nxc](#), [ex\\_NRLinkStatus.nxc](#), [ex\\_NRLinkStatusEx.nxc](#), [ex\\_NRLinkTxRaw.nxc](#), [ex\\_NRLinkTxRawEx.nxc](#), [ex\\_PSPNxAnalog.nxc](#), [ex\\_PSPNxAnalogEx.nxc](#), [ex\\_PSPNxDigital.nxc](#), [ex\\_PSPNxDigitalEx.nxc](#), [ex\\_readi2cregister.nxc](#), [ex\\_ReadNRLinkBytes.nxc](#), [ex\\_ReadNRLinkBytesEx.nxc](#), [ex\\_ReadSensorColorEx.nxc](#), [ex\\_ReadSensorColorRaw.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\\_ReadSensorMSAccelEx.nxc](#), [ex\\_ReadSensorMSPlayStation.nxc](#), [ex\\_ReadSensorMSPlayStationEx.nxc](#), [ex\\_ReadSensorMSRTClock.nxc](#), [ex\\_ReadSensorMSTilt.nxc](#), [ex\\_ReadSensorMSTiltEx.nxc](#), [ex\\_readsensorex.nxc](#), [ex\\_RemoteResetScaledValue.nxc](#), [ex\\_RemoteSetInputMode.nxc](#), [ex\\_RunNRLinkMacro.nxc](#), [ex\\_RunNRLinkMacroEx.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\\_SensorInvalid.nxc](#), [ex\\_SensorMode.nxc](#), [ex\\_SensorMSCompass.nxc](#), [ex\\_SensorMSCompassEx.nxc](#), [ex\\_SensorMSDROD.nxc](#), [ex\\_SensorMSPressure.nxc](#), [ex\\_SensorMSPressureRaw.nxc](#), [ex\\_SensorNormalized.nxc](#), [ex\\_SensorRaw.nxc](#), [ex\\_SensorScaled.nxc](#), [ex\\_SensorType.nxc](#), [ex\\_SensorValue.nxc](#), [ex\\_SensorValueBool.nxc](#), [ex\\_SensorValueRaw.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\\_SetSensorHTEOPD.nxc](#), [ex\\_SetSensorHTGyro.nxc](#), [ex\\_SetSensorLight.nxc](#), [ex\\_SetSensorLowspeed.nxc](#), [ex\\_SetSensorMode.nxc](#), [ex\\_SetSensorMSDROD.nxc](#), [ex\\_SetSensorMSPressure.nxc](#), [ex\\_SetSensorSound.nxc](#), [ex\\_SetSensorTouch.nxc](#), [ex\\_SetSensorType.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_syscommlscheckstatus.nxc](#), [ex\\_syscommlsread.nxc](#), [ex\\_syscommlswrite.nxc](#), [ex\\_syscommlswriteex.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_writei2cregister.nxc](#), [ex\\_WriteNRLinkBytes.nxc](#), and [ex\\_WriteNRLinkBytesEx.nxc](#).

### 9.27.2.2 #define S2 1

Input port 2

**9.27.2.3 #define S3 2**

Input port 3

**9.27.2.4 #define S4 3**

Input port 4

**Examples:**

[ex\\_I2CBytes.nxc](#), and [ex\\_sensorus.nxc](#).

## 9.28 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`

### 9.28.1 Detailed Description

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

See also:

[SetSensorType\(\)](#)

### 9.28.2 Define Documentation

#### 9.28.2.1 `#define SENSOR\_TYPE\_COLORBLUE IN_TYPE_COLORBLUE`

NXT 2.0 color sensor with blue light

#### 9.28.2.2 `#define SENSOR\_TYPE\_COLORFULL IN_TYPE_COLORFULL`

NXT 2.0 color sensor in full color mode

#### 9.28.2.3 `#define SENSOR\_TYPE\_COLORGREEN IN_TYPE_COLORGREEN`

NXT 2.0 color sensor with green light



**9.28.2.4 #define SENSOR\_TYPE\_COLORNONE IN\_TYPE\_COLORNONE**

NXT 2.0 color sensor with no light

**9.28.2.5 #define SENSOR\_TYPE\_COLORRED IN\_TYPE\_COLORRED**

NXT 2.0 color sensor with red light

**9.28.2.6 #define SENSOR\_TYPE\_CUSTOM IN\_TYPE\_CUSTOM**

NXT custom sensor

**9.28.2.7 #define SENSOR\_TYPE\_HIGHSPEED IN\_TYPE\_HISPEED**

NXT Hi-speed port (only S4)

**9.28.2.8 #define SENSOR\_TYPE\_LIGHT IN\_TYPE\_REFLECTION**

RCX light sensor

**9.28.2.9 #define SENSOR\_TYPE\_LIGHT\_ACTIVE IN\_TYPE\_LIGHT\_ACTIVE**

NXT light sensor with light

**9.28.2.10 #define SENSOR\_TYPE\_LIGHT\_INACTIVE IN\_TYPE\_LIGHT\_INACTIVE**

NXT light sensor without light

**9.28.2.11 #define SENSOR\_TYPE\_LOWSPEED IN\_TYPE\_LOWSPEED**

NXT I2C digital sensor

**Examples:**

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

**9.28.2.12 #define SENSOR\_TYPE\_LOWSPEED\_9V IN\_TYPE\_LOWSPEED\_9V**

NXT I2C digital sensor with 9V power

**9.28.2.13 #define SENSOR\_TYPE\_NONE IN\_TYPE\_NO\_SENSOR**

No sensor configured

**9.28.2.14 #define SENSOR\_TYPE\_ROTATION IN\_TYPE\_ANGLE**

RCX rotation sensor

**9.28.2.15 #define SENSOR\_TYPE\_SOUND\_DB IN\_TYPE\_SOUND\_DB**

NXT sound sensor with dB scaling

**Examples:**

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

**9.28.2.16 #define SENSOR\_TYPE\_SOUND\_DBA IN\_TYPE\_SOUND\_DBA**

NXT sound sensor with dBA scaling

**9.28.2.17 #define SENSOR\_TYPE\_TEMPERATURE IN\_TYPE\_TEMPERATURE**

RCX temperature sensor

**9.28.2.18 #define SENSOR\_TYPE\_TOUCH IN\_TYPE\_SWITCH**

NXT or RCX touch sensor

**Examples:**

[ex\\_HTRCXSetSensorType.nxc](#), [ex\\_MSRCXSetSensorType.nxc](#), and [ex\\_SetSensorType.nxc](#).

## 9.29 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

### 9.29.1 Detailed Description

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

See also:

[SetSensorMode\(\)](#)

### 9.29.2 Define Documentation

#### 9.29.2.1 #define SENSOR\_MODE\_BOOL IN\_MODE\_BOOLEAN

Boolean value (0 or 1)

Examples:

[ex\\_HTRCXSetSensorMode.nxc](#), and [ex\\_MSRCXSetSensorMode.nxc](#).

#### 9.29.2.2 #define SENSOR\_MODE\_CELSIUS IN\_MODE\_CELSIUS

RCX temperature sensor value in degrees celcius

#### 9.29.2.3 #define SENSOR\_MODE\_EDGE IN\_MODE\_TRANSITIONCNT

Counts the number of boolean transitions

#### 9.29.2.4 #define SENSOR\_MODE\_FAHRENHEIT IN\_MODE\_FAHRENHEIT

RCX temperature sensor value in degrees fahrenheit

#### 9.29.2.5 #define SENSOR\_MODE\_PERCENT IN\_MODE\_PCTFULLSCALE

Scaled value from 0 to 100

**9.29.2.6 #define SENSOR\_MODE\_PULSE IN\_MODE\_PERIODCOUNTER**

Counts the number of boolean periods

**9.29.2.7 #define SENSOR\_MODE\_RAW IN\_MODE\_RAW**

Raw value from 0 to 1023

**Examples:**

[ex\\_RemoteSetInputMode.nxc](#), and [ex\\_SetSensorMode.nxc](#).

**9.29.2.8 #define SENSOR\_MODE\_ROTATION IN\_MODE\_ANGLESTEP**

RCX rotation sensor (16 ticks per revolution)

## 9.30 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)`

### 9.30.1 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\(\)](#)

### 9.30.2 Define Documentation

#### 9.30.2.1 **#define \_SENSOR\_CFG(\_type, \_mode) (((\_type)<<8)+(\_mode))**

Macro for defining [SetSensor](#) combined type and mode constants

#### 9.30.2.2 **#define SENSOR\_CELSIUS \_SENSOR\_CFG(SENSOR\_TYPE\_TEMPERATURE, SENSOR\_MODE\_CELSIUS)**

RCX temperature sensor in celcius mode

#### 9.30.2.3 **#define SENSOR\_COLORBLUE \_SENSOR\_CFG(SENSOR\_TYPE\_COLORBLUE, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (blue) in percent mode

#### 9.30.2.4 **#define SENSOR\_COLORFULL \_SENSOR\_CFG(SENSOR\_TYPE\_COLORFULL, SENSOR\_MODE\_RAW)**

NXT 2.0 color sensor (full) in raw mode

#### 9.30.2.5 **#define SENSOR\_COLORGREEN \_SENSOR\_CFG(SENSOR\_TYPE\_COLORGREEN, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

#### 9.30.2.6 **#define SENSOR\_COLORNONE \_SENSOR\_CFG(SENSOR\_TYPE\_COLORNONE, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

#### 9.30.2.7 **#define SENSOR\_COLORRED \_SENSOR\_CFG(SENSOR\_TYPE\_COLORRED, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (red) in percent mode

#### 9.30.2.8 **#define SENSOR\_EDGE \_SENSOR\_CFG(SENSOR\_TYPE\_TOUCH, SENSOR\_MODE\_EDGE)**

Touch sensor in edge mode

#### 9.30.2.9 **#define SENSOR\_FAHRENHEIT \_SENSOR\_CFG(SENSOR\_TYPE\_TEMPERATURE, SENSOR\_MODE\_FAHRENHEIT)**

RCX temperature sensor in fahrenheit mode

**9.30.2.10** `#define SENSOR_LIGHT _SENSOR_CFG(SENSOR_TYPE_LIGHT,  
SENSOR_MODE_PERCENT)`

Light sensor in percent mode

**9.30.2.11** `#define SENSOR_LOWSPEED _SENSOR_CFG(SENSOR_TYPE_LOWSPEED,  
SENSOR_MODE_RAW)`

NXT I2C sensor without 9V power in raw mode

**9.30.2.12** `#define SENSOR_LOWSPEED_9V _SENSOR_CFG(SENSOR_TYPE_LOWSPEED -  
9V, SENSOR_MODE_RAW)`

NXT I2C sensor with 9V power in raw mode

**9.30.2.13** `#define SENSOR_NXTLIGHT _SENSOR_CFG(SENSOR_TYPE_LIGHT_ACTIVE,  
SENSOR_MODE_PERCENT)`

NXT light sensor in active mode

**9.30.2.14** `#define SENSOR_PULSE _SENSOR_CFG(SENSOR_TYPE_TOUCH,  
SENSOR_MODE_PULSE)`

Touch sensor in pulse mode

**9.30.2.15** `#define SENSOR_ROTATION _SENSOR_CFG(SENSOR_TYPE_ROTATION,  
SENSOR_MODE_ROTATION)`

RCX rotation sensor in rotation mode

**9.30.2.16** `#define SENSOR_SOUND _SENSOR_CFG(SENSOR_TYPE_SOUND_DB,  
SENSOR_MODE_PERCENT)`

NXT sound sensor (dB) in percent mode

**9.30.2.17** `#define SENSOR_TOUCH _SENSOR_CFG(SENSOR_TYPE_TOUCH,  
SENSOR_MODE_BOOL)`

Touch sensor in boolean mode

**Examples:**

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

## 9.31 Input module types

Types used by various input module functions.

### Data Structures

- struct [ColorSensorReadType](#)  
*Parameters for the ColorSensorRead system call.*

### 9.31.1 Detailed Description

Types used by various input module functions.



## 9.32 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 [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](#) (const byte port)  
*Get the custom sensor active status.*
- byte [CustomSensorPercentFullScale](#) (const byte port)  
*Get the custom sensor percent full scale.*
- unsigned int [CustomSensorZeroOffset](#) (const byte port)  
*Get the custom sensor zero offset.*
- void [SetCustomSensorActiveStatus](#) (const byte port, const byte &activeStatus)  
*Set active status.*
- void [SetCustomSensorPercentFullScale](#) (const byte port, const byte &pctFullScale)  
*Set percent full scale.*
- void [SetCustomSensorZeroOffset](#) (const byte port, const int &zeroOffset)  
*Set percent zero offset.*
- void [SetSensorBoolean](#) (const byte port, const bool &value)  
*Set sensor boolean value.*
- void [SetSensorDigiPinsDirection](#) (const byte port, const byte &direction)  
*Set digital pins direction.*
- void [SetSensorDigiPinsOutputLevel](#) (const byte port, const byte &outputLevel)  
*Set digital pins output level.*
- void [SetSensorDigiPinsStatus](#) (const byte port, const 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](#) (const byte port, const byte color)  
*Read a LEGO color sensor AD raw value.*
- bool [ColorBoolean](#) (const byte port, const byte color)  
*Read a LEGO color sensor boolean value.*
- long [ColorCalibration](#) (const byte port, const byte point, const byte color)

*Read a LEGO color sensor calibration point value.*

- byte [ColorCalibrationState](#) (const byte port)  
*Read LEGO color sensor calibration state.*
- unsigned int [ColorCalLimits](#) (const byte port, const byte point)  
*Read a LEGO color sensor calibration limit value.*
- unsigned int [ColorSensorRaw](#) (const byte port, const byte color)  
*Read a LEGO color sensor raw value.*
- unsigned int [ColorSensorValue](#) (const byte port, const byte color)  
*Read a LEGO color sensor scaled value.*

### 9.32.1 Detailed Description

Functions for accessing and modifying input module features.

### 9.32.2 Function Documentation

#### 9.32.2.1 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](#).

#### 9.32.2.2 unsigned int ColorADRaw (const byte port, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

##### Returns:

The AD raw value.

##### Warning:

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**9.32.2.3 bool ColorBoolean (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The boolean value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**9.32.2.4 long ColorCalibration (const byte *port*, const byte *point*, const 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](#). Must be a constant.

*point* The calibration point. See [Color calibration constants](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The calibration point value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

### 9.32.2.5 byte ColorCalibrationState (const 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](#). Must be a constant.

#### Returns:

The calibration state.

#### Warning:

This function requires an NXT 2.0 compatible firmware.

#### Examples:

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

### 9.32.2.6 unsigned int ColorCalLimits (const byte *port*, const 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](#). Must be a constant.

*point* The calibration point. See [Color calibration constants](#). Must be a constant.

#### Returns:

The calibration limit value.

#### Warning:

This function requires an NXT 2.0 compatible firmware.

#### Examples:

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

### 9.32.2.7 unsigned int ColorSensorRaw (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**9.32.2.8 unsigned int ColorSensorValue (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The scaled value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**9.32.2.9 byte CustomSensorActiveStatus (const 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](#).

#### 9.32.2.10 `byte CustomSensorPercentFullScale (const 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](#).

#### 9.32.2.11 `unsigned int CustomSensorZeroOffset (const 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](#).

#### 9.32.2.12 `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](#).



**9.32.2.13 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](#).

**9.32.2.14 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](#).

### 9.32.2.15 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](#).

### 9.32.2.16 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](#).

### 9.32.2.17 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](#).

**9.32.2.18 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](#).

**9.32.2.19 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](#).

**9.32.2.20 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](#).

**9.32.2.21 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](#).

**9.32.2.22 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](#).

**9.32.2.23 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](#).

**9.32.2.24 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](#).

**9.32.2.25 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](#).

**9.32.2.26 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](#).

### 9.32.2.27 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](#).

### 9.32.2.28 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](#).

### 9.32.2.29 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](#).

**9.32.2.30 void SetCustomSensorActiveStatus (const byte *port*, const byte & *activeStatus*)  
[inline]**

Set active status. Sets the active status value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*activeStatus* The new active status value.

**Examples:**

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

**9.32.2.31 void SetCustomSensorPercentFullScale (const byte *port*, const 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](#). Must be a constant.

*pctFullScale* The new percent full scale value.

**Examples:**

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

**9.32.2.32 void SetCustomSensorZeroOffset (const byte *port*, const int & *zeroOffset*) [inline]**

Set percent zero offset. Sets the zero offset value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*zeroOffset* The new zero offset value.

**Examples:**

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

**9.32.2.33 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](#).

**9.32.2.34 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](#).

**9.32.2.35 void SetSensorBoolean (const byte *port*, const bool & *value*) [inline]**

Set sensor boolean value. Sets the boolean value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*value* The new boolean value.

**9.32.2.36 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](#).

**9.32.2.37 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](#).

**9.32.2.38 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](#).

**9.32.2.39 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](#).

**9.32.2.40 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](#).

**9.32.2.41 void SetSensorDigiPinsDirection (const byte *port*, const 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](#). Must be a constant.

*direction* The new digital pins direction value.

**Examples:**

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

**9.32.2.42 void SetSensorDigiPinsOutputLevel (const byte *port*, const 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](#). Must be a constant.

*outputLevel* The new digital pins output level value.

**Examples:**

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

**9.32.2.43 void SetSensorDigiPinsStatus (const byte *port*, const 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](#). Must be a constant.

*status* The new digital pins status value.

**Examples:**

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

**9.32.2.44 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](#).

**9.32.2.45 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\\_HTRCXSetIRLinkPort.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), and [ex\\_SetSensorLowspeed.nxc](#).

**9.32.2.46 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](#).

**9.32.2.47 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](#).

**9.32.2.48 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\\_SetSensorTouch.nxc](#).

**9.32.2.49 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](#).

**9.32.2.50 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](#).

## 9.33 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)

### 9.33.1 Detailed Description

Read analog sensor values using these names. Returns the current scaled value of the sensor on the specified port.

### 9.33.2 Define Documentation

#### 9.33.2.1 #define [SENSOR\\_1](#) Sensor(S1)

Read the value of the analog sensor on port S1

#### 9.33.2.2 #define [SENSOR\\_2](#) Sensor(S2)

Read the value of the analog sensor on port S2

#### 9.33.2.3 #define [SENSOR\\_3](#) Sensor(S3)

Read the value of the analog sensor on port S3

#### 9.33.2.4 #define [SENSOR\\_4](#) Sensor(S4)

Read the value of the analog sensor on port S4

## 9.34 Output module functions

Functions for accessing and modifying output module features.

### Functions

- void [SetMotorPwnFreq](#) (byte n)  
*Set motor PWM frequency.*
- 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 [MotorPwnFreq](#) ()

*Get motor PWN frequency.*

### 9.34.1 Detailed Description

Functions for accessing and modifying output module features.

### 9.34.2 Function Documentation

#### 9.34.2.1 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](#).

#### 9.34.2.2 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](#).

### 9.34.2.3 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](#).

### 9.34.2.4 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](#).

#### Examples:

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

### 9.34.2.5 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](#).

#### Examples:

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

### 9.34.2.6 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](#).

#### Examples:

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

#### 9.34.2.7 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](#).

**Examples:**

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

#### 9.34.2.8 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](#).

**Examples:**

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

#### 9.34.2.9 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](#).

**Examples:**

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

#### 9.34.2.10 byte MotorPwnFreq () [inline]

Get motor PWN frequency. Get the current motor pulse width modulation frequency.

**Examples:**

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

#### 9.34.2.11 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](#).

##### Examples:

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

#### 9.34.2.12 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](#).

##### Examples:

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

#### 9.34.2.13 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](#).

##### Examples:

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

#### 9.34.2.14 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](#).

##### Examples:

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

**9.34.2.15 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](#).

**Examples:**

[ex\\_motorrotationcount.nxc](#), and [util\\_rpm.nxc](#).

**9.34.2.16 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](#).

**Examples:**

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

**9.34.2.17 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](#).

**Examples:**

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

**9.34.2.18 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](#).

**Examples:**

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

#### 9.34.2.19 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](#).

##### Examples:

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

#### 9.34.2.20 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](#).

#### 9.34.2.21 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](#).

#### 9.34.2.22 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](#).

**9.34.2.23 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](#).

**9.34.2.24 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](#).

**9.34.2.25 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](#).

#### 9.34.2.26 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](#).

#### 9.34.2.27 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](#).



**9.34.2.28 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](#).

**9.34.2.29 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](#).

**9.34.2.30 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](#).

**9.34.2.31 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](#).

**9.34.2.32 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](#).

**9.34.2.33 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](#).

**9.34.2.34 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](#).

**9.34.2.35 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](#).

### 9.34.2.36 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](#).

### 9.34.2.37 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](#).

### 9.34.2.38 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](#).

**9.34.2.39 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](#).

**9.34.2.40 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](#).

#### 9.34.2.41 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](#).

#### 9.34.2.42 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](#).

#### 9.34.2.43 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](#).

**9.34.2.44 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](#).

**9.34.2.45 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](#).

**9.34.2.46 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](#).

**9.34.2.47 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](#).

**9.34.2.48 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](#).

**9.34.2.49 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](#).

**9.34.2.50 void SetMotorPwnFreq (byte *n*) [inline]**

Set motor PWM frequency. Set the motor pulse width modulation frequency.

**Examples:**

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

**9.34.2.51 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](#).

## 9.35 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.*

### 9.35.1 Detailed Description

Types used by various display module functions.

## 9.36 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 addr)  
*Set the display font memory address.*
- void [SetDisplayDisplay](#) (unsigned long addr)  
*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.*

### 9.36.1 Detailed Description

Functions for accessing and modifying display module features.

### 9.36.2 Function Documentation

#### 9.36.2.1 `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](#).

#### 9.36.2.2 `void ClearLine (byte line) [inline]`

Clear a line on the LCD screen. This function lets you clear a single line on the NXT LCD.

#### Examples:

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

#### 9.36.2.3 `void ClearScreen () [inline]`

Clear LCD screen. This function lets you clear the NXT LCD to a blank screen.

#### Examples:

[ex\\_ClearScreen.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_string.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

#### 9.36.2.4 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](#).

#### 9.36.2.5 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](#).

#### 9.36.2.6 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](#).

#### 9.36.2.7 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](#).



**9.36.2.8 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\\_addr.nxc](#), [ex\\_addrx.nxc](#), [ex\\_DisplayFont.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

**9.36.2.9 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](#).

**9.36.2.10 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](#).

**9.36.2.11 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](#).

#### 9.36.2.12 `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](#).

**9.36.2.13 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](#).

**9.36.2.14 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](#).

### 9.36.2.15 `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](#).

### 9.36.2.16 `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](#).

### 9.36.2.17 `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](#).

**9.36.2.18** `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](#).

**9.36.2.19** `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](#).

**9.36.2.20** `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](#).

**9.36.2.21** `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\\_dispgout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysDatalogGetTimes.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), and [ex\\_SysWriteSemData.nxc](#).

**9.36.2.22 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](#).

### 9.36.2.23 `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](#).

### 9.36.2.24 `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](#).



**9.36.2.25 void ResetScreen () [inline]**

Reset LCD screen. This function lets you restore the standard NXT running program screen.

**Examples:**

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

**9.36.2.26 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](#).

**9.36.2.27 void SetDisplayDisplay (unsigned long *addr*) [inline]**

Set the display memory address. This function lets you set the current display memory address.

**Parameters:**

*addr* The new display memory address.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayDisplay.nxc](#).

**9.36.2.28 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](#).

**9.36.2.29 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](#).

**9.36.2.30 void SetDisplayFont (unsigned long *addr*) [inline]**

Set the display font memory address. This function lets you set the current display font memory address.

**Parameters:**

*addr* The new display font memory address.

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrx.nxc](#), [ex\\_DisplayFont.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

**9.36.2.31 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](#).

**9.36.2.32 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](#).

**9.36.2.33 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](#).

**9.36.2.34 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](#).

**9.36.2.35 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](#).

**9.36.2.36 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](#).

**9.36.2.37 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](#).

**9.36.2.38 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\\_dispfout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**9.36.2.39 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](#).

**9.36.2.40 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](#).

**9.36.2.41 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](#).

**9.36.2.42 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](#).

**9.36.2.43 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](#).

**9.36.2.44 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](#).

**9.36.2.45 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](#).

**9.36.2.46 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](#).

**9.36.2.47 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\\_addr.nxc](#), [ex\\_addrex.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\\_ClearLine.nxc](#), [ex\\_Copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_midstr.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SetDisplayFont.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\\_SysCommBTOnOff.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\\_TextOut.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

## 9.37 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.*

### 9.37.1 Detailed Description

Types used by various sound module functions.



## 9.38 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.*

### 9.38.1 Detailed Description

Functions for accessing and modifying sound module features.

### 9.38.2 Function Documentation

#### 9.38.2.1 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](#).

**9.38.2.2 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](#).

**9.38.2.3 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](#).

**9.38.2.4 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\\_PlayTone.nxc](#), and [ex\\_yield.nxc](#).

### 9.38.2.5 **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](#).

### 9.38.2.6 **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](#).

### 9.38.2.7 **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](#).

**9.38.2.8 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](#).

**9.38.2.9 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](#).

**9.38.2.10 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](#).

**9.38.2.11 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](#).

#### 9.38.2.12 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](#).

#### 9.38.2.13 void SetSoundVolume (byte *volume*) [inline]

Set sound volume. Set the sound volume.

See also:

[SoundVolume\(\)](#)

**Parameters:**

*volume* The new volume

**Examples:**

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

#### 9.38.2.14 unsigned int SoundDuration () [inline]

Get sound duration. Return the current sound duration.

See also:

[SetSoundDuration\(\)](#)

**Returns:**

The current sound duration.

**Examples:**

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

**9.38.2.15** `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](#).

**9.38.2.16** `unsigned int SoundFrequency () [inline]`

Get sound frequency. Return the current sound frequency.

**See also:**

[SetSoundFrequency\(\)](#)

**Returns:**

The current sound frequency.

**Examples:**

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

**9.38.2.17** `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](#).

**9.38.2.18** `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](#).

#### 9.38.2.19 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](#).

#### 9.38.2.20 byte SoundVolume () [inline]

Get volume. Return the current sound volume.

See also:

[SetSoundVolume\(\)](#)

Returns:

The current sound volume.

Examples:

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

#### 9.38.2.21 byte StopSound () [inline]

Stop sound. Stop playing of the current tone or file.

Returns:

The result

**Todo**

?

Examples:

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



**9.38.2.22 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](#).

**9.38.2.23 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](#).

**9.38.2.24 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](#).

**9.38.2.25 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](#).

## 9.39 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.*

### 9.39.1 Detailed Description

Types used by various low speed module functions.

## 9.40 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 [ReadI2CRegister](#) (byte port, byte reg, byte &out)  
*Read I2C register.*
- char [WriteI2CRegister](#) (byte port, byte reg, byte val)  
*Write I2C register.*
- 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.*

- string [I2CDeviceInfo](#) (byte port, byte info)

*Read I2C device information.*

- string [I2CDeviceInfoEx](#) (byte port, byte addr, byte info)

*Read I2C device information extra.*

- string [I2CVersion](#) (byte port)

*Read I2C device version.*

- string [I2CVersionEx](#) (byte port, byte addr)

*Read I2C device version extra.*

- string [I2CVendorId](#) (byte port)

*Read I2C device vendor.*

- string [I2CVendorIdEx](#) (byte port, byte addr)

*Read I2C device vendor extra.*

- string [I2CDeviceId](#) (byte port)

*Read I2C device identifier.*

- string [I2CDeviceIdEx](#) (byte port, byte addr)

*Read I2C device identifier extra.*

- long [I2CSendCommand](#) (byte port, byte cmd)

*Send an I2C command.*

- long [I2CSendCommandEx](#) (byte port, byte addr, byte cmd)

*Send an I2C command extra.*

### 9.40.1 Detailed Description

Functions for accessing and modifying low speed module features.

## 9.40.2 Function Documentation

### 9.40.2.1 `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](#).

### 9.40.2.2 `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](#).

**9.40.2.3 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](#).

**9.40.2.4 string I2CDeviceId (byte *port*) [inline]**

Read I2C device identifier. Read standard I2C device identifier. The I2C device must use address 0x02.

**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 string containing the device identifier.

**See also:**

[I2CDeviceIdEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

#### 9.40.2.5 string I2CDeviceIdEx (byte *port*, byte *addr*) [inline]

Read I2C device identifier extra. 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.

*addr* The I2C device address.

##### Returns:

A string containing the device identifier.

##### See also:

[I2CDeviceId](#)

##### Examples:

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

#### 9.40.2.6 string I2CDeviceInfo (byte *port*, byte *info*) [inline]

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device must use address 0x02.

##### 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.

*info* A value indicating the type of device information you are requesting. See [Standard I2C constants](#).

##### Returns:

A string containing the requested device information.

##### See also:

[I2CDeviceInfoEx](#)

##### Examples:

[ex\\_I2CDeviceInfo.nxc](#).

#### 9.40.2.7 string I2CDeviceInfoEx (byte *port*, byte *addr*, byte *info*) [inline]

Read I2C device information extra. 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.

**addr** 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.

**See also:**

[I2CDeviceInfo](#)

**Examples:**

[ex\\_I2CDeviceInfoEx.nxc](#).

**9.40.2.8 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](#).

**9.40.2.9 long I2CSendCommand (byte port, 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 must use address 0x02.



**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.

*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](#).

**9.40.2.10 long I2CSendCommandEx (byte port, byte addr, byte cmd) [inline]**

Send an I2C command extra. 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.

*addr* 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\\_I2CSendCommandEx.nxc](#).

**9.40.2.11 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](#).

**9.40.2.12 string I2CVendorId (byte *port*) [inline]**

Read I2C device vendor. Read standard I2C device vendor. The I2C device must use address 0x02.

**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 string containing the device vendor.

**See also:**

[I2CVendorIdEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

**9.40.2.13 string I2CVendorIdEx (byte *port*, byte *addr*) [inline]**

Read I2C device vendor extra. 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.

*addr* The I2C device address.

**Returns:**

A string containing the device vendor.

**See also:**

[I2CVendorId](#)

**Examples:**

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

**9.40.2.14 string I2CVersion (byte *port*) [inline]**

Read I2C device version. Read standard I2C device version. The I2C device must use address 0x02.

**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 string containing the device version.

**See also:**

[I2CVersionEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

**9.40.2.15 string I2CVersionEx (byte *port*, byte *addr*) [inline]**

Read I2C device version extra. 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.

***addr*** The I2C device address.

**Returns:**

A string containing the device version.

**See also:**

[I2CVersion](#)

**Examples:**

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

**9.40.2.16 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](#).

#### 9.40.2.17 `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](#).

#### 9.40.2.18 `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](#).

**9.40.2.19 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](#).

**9.40.2.20 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](#).

**9.40.2.21 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](#).

**9.40.2.22 char ReadI2CRegister (byte port, 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.
- 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](#).

**9.40.2.23 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\\_readsensorex.nxc](#).

**9.40.2.24 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](#).

#### 9.40.2.25 char WriteI2CRegister (byte *port*, 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.

*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](#).



## 9.41 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 [LSState](#) ()   
*Get I2C state.*
- byte [LSSpeed](#) ()   
*Get I2C speed.*
- byte [LSNoRestartOnRead](#) ()   
*Get I2C no restart on read setting.*

### 9.41.1 Detailed Description

Low level functions for accessing low speed module features.

### 9.41.2 Function Documentation

#### 9.41.2.1 `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](#).

#### 9.41.2.2 `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](#).

#### 9.41.2.3 `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](#).

**9.41.2.4 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](#).

**9.41.2.5 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](#).

**9.41.2.6 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](#).

#### 9.41.2.7 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](#).

#### 9.41.2.8 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](#).

#### 9.41.2.9 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](#).

#### 9.41.2.10 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](#).

**9.41.2.11 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](#).

**9.41.2.12 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](#).

**9.41.2.13 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](#).

**9.41.2.14** `byte LSState () [inline]`

Get I2C state. This method returns the value of the I2C state.

**Returns:**

The I2C state. See [LSState constants](#).

**Examples:**

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

## 9.42 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).*

### 9.42.1 Detailed Description

System call functions for accessing low speed module features.

### 9.42.2 Function Documentation

#### 9.42.2.1 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](#).

#### 9.42.2.2 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](#).

**9.42.2.3 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](#).

**9.42.2.4 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](#).



## 9.43 IOCtrl module types

Types used by various IOCtrl module functions. Types used by various IOCtrl module functions.

## 9.44 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.*

### 9.44.1 Detailed Description

Functions for accessing and modifying IOCtrl module features.

### 9.44.2 Function Documentation

#### 9.44.2.1 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](#).

#### 9.44.2.2 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](#).

#### 9.44.2.3 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](#).

## 9.45 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.*

### 9.45.1 Detailed Description

Types used by various Command module functions.

## 9.46 Command module functions

Functions for accessing and modifying Command module features.

### Modules

- [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.*
- 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.*

### 9.46.1 Detailed Description

Functions for accessing and modifying Command module features.

## 9.46.2 Function Documentation

### 9.46.2.1 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](#).

### 9.46.2.2 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](#).

### 9.46.2.3 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](#).

### 9.46.2.4 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](#).

**9.46.2.5 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](#).

**9.46.2.6 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](#).

**9.46.2.7 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](#).

#### 9.46.2.8 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](#).

#### 9.46.2.9 void StartTask (task *t*) [inline]

Start a task. Start the specified task.

##### Parameters:

*t* The task to start.

##### Examples:

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

#### 9.46.2.10 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\\_stop.nxc](#).

#### 9.46.2.11 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](#).

#### 9.46.2.12 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](#).

**9.46.2.13 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](#).

**9.46.2.14 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](#).

#### 9.46.2.15 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](#).

#### 9.46.2.16 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](#).

#### 9.46.2.17 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](#).

**9.46.2.18 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](#).

**9.46.2.19 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\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**9.46.2.20 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](#).

**9.46.2.21 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\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**9.46.2.22 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](#).

**9.46.2.23 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](#).

**9.46.2.24 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](#).

**9.46.2.25 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](#).

**9.46.2.26 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\\_addr.nxc](#), [ex\\_addrex.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\\_dispfout.nxc](#), [ex\\_dispfunc.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetDisplayFont.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\\_wait.nxc](#), [ex\\_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**9.46.2.27 void Yield () [inline]**

Yield to another task. Make a task yield to another concurrently running task.

**Examples:**

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

## 9.47 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.*
- 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.*

### 9.47.1 Detailed Description

Functions for use with NXC array types.

## 9.47.2 Function Documentation

### 9.47.2.1 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.

#### Examples:

[ex\\_arraybuild.nxc](#), [ex\\_SysCommHSWrite.nxc](#), and [ex\\_SysDatalogWrite.nxc](#).

### 9.47.2.2 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.

#### Examples:

[ex\\_arrayinit.nxc](#), and [ex\\_sysdrawgraphic.nxc](#).

### 9.47.2.3 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.

#### Examples:

[ex\\_arraylen.nxc](#), and [ex\\_syslistfiles.nxc](#).

**9.47.2.4 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.

**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).

**Examples:**

[ex\\_ArrayMax.nxc](#), and [ex\\_ArraySort.nxc](#).

**9.47.2.5 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.

**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).

**Examples:**

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

**9.47.2.6 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.

**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).

**Examples:**

[ex\\_ArrayMin.nxc](#), and [ex\\_ArraySort.nxc](#).

#### 9.47.2.7 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.

**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).

**Examples:**

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

#### 9.47.2.8 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.

**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).

**Examples:**

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

#### 9.47.2.9 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.

**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*).

**Examples:**

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

**9.47.2.10 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.

**Examples:**

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

**9.47.2.11 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.

**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*).

**Examples:**

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

**9.47.2.12** `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.

**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).

**Examples:**

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

## 9.48 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 [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.*

### 9.48.1 Detailed Description

Types used by various Comm module functions.

## 9.49 Comm module functions

Functions for accessing and modifying Comm module features.

### Functions

- 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](#) (const byte conn)  
*Check bluetooth status.*
- char [BluetoothWrite](#) (const byte conn, byte buffer[ ])  
*Write to a bluetooth connection.*
- 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 [RS485Control](#) (byte cmd, byte baud, unsigned int mode)  
*Control the RS485 port.*
- bool [RS485DataAvailable](#) (void)  
*Check for RS485 available data.*
- char [RS485Exit](#) (void)  
*Exit RS485.*
- char [RS485Init](#) (void)  
*Initialize RS485.*
- char [RS485Read](#) (byte &buffer[ ])  
*Read RS485 data.*

- bool [RS485SendingData](#) (void)  
*Is RS485 sending data.*
- void [RS485Status](#) (bool &sendingData, bool &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.*

- int [HSMode](#) (void)  
*Get hi-speed port mode.*
- 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](#) (const byte hsFlags)   
*Set hi-speed port flags.*
- void [SetHSSpeed](#) (const byte hsSpeed)   
*Set hi-speed port speed.*
- void [SetHSState](#) (const byte hsState)   
*Set hi-speed port state.*
- void [SetHSMode](#) (const unsigned int hsMode)   
*Set hi-speed port 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.*

### 9.49.1 Detailed Description

Functions for accessing and modifying Comm module features.

### 9.49.2 Function Documentation

#### 9.49.2.1 `char BluetoothStatus (const 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).

##### Returns:

The bluetooth status for the specified connection.

##### Examples:

[ex\\_bluetoothstatus.nxc](#), and [ex\\_SysCommBTConnection.nxc](#).

#### 9.49.2.2 `char BluetoothWrite (const 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).

*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.](#)

**9.49.2.3 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.](#)

**9.49.2.4 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.](#)

**9.49.2.5 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.](#)

**9.49.2.6 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.](#)

#### 9.49.2.7 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](#).

#### 9.49.2.8 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](#).

#### 9.49.2.9 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](#).

#### 9.49.2.10 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](#).

**9.49.2.11 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](#).

**9.49.2.12 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](#).

**9.49.2.13 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](#).

**9.49.2.14 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](#).

**9.49.2.15 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](#).

**9.49.2.16 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](#).



**9.49.2.17 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](#).

**9.49.2.18 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](#).

**9.49.2.19 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](#).

**9.49.2.20 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](#).

**9.49.2.21 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](#).

**9.49.2.22 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](#).

**9.49.2.23 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](#).

**9.49.2.24 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](#).

**9.49.2.25 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](#).

**9.49.2.26 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](#).

**9.49.2.27 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](#).

**9.49.2.28 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](#).

**9.49.2.29 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](#).

**9.49.2.30 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](#).

**9.49.2.31 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](#).

**9.49.2.32 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](#).

**9.49.2.33 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](#).

**9.49.2.34 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](#).

**9.49.2.35 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](#).

**9.49.2.36 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](#).

**Examples:**

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

**9.49.2.37 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](#).

**9.49.2.38 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](#).

**9.49.2.39 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](#).

**9.49.2.40 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](#).

**9.49.2.41 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.

**9.49.2.42 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](#).

#### 9.49.2.43 **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](#).

#### 9.49.2.44 **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.

#### 9.49.2.45 **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](#).

**9.49.2.46 char RemoteKeepAlive (byte *conn*) [inline]**

Send a KeepAlive message. This method sends a KeepAlive direct command to the device on the specified connection. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**9.49.2.47 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*queue* The mailbox to read. See [Mailbox constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**9.49.2.48 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

**9.49.2.49 char RemotePlaySoundFile (byte *conn*, string *filename*, bool *bloop*) [inline]**

Send a PlaySoundFile message. Send the PlaySoundFile direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

**9.49.2.50 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

**9.49.2.51 char RemoteResetMotorPosition (byte *conn*, byte *port*, bool *brelative*) [inline]**

Send a ResetMotorPosition message. Send the ResetMotorPosition direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- 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](#).

**9.49.2.52 char RemoteResetScaledValue (byte *conn*, byte *port*) [inline]**

Send a ResetScaledValue message. Send the ResetScaledValue direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- port* The input port to reset.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**9.49.2.53 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- 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](#).

#### 9.49.2.54 **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 [BluetoothStatus](#) to determine when this write request is completed.

##### Parameters:

*conn* The connection slot (0..3).  
*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](#).

#### 9.49.2.55 **char RemoteStartProgram (byte *conn*, string *filename*) [inline]**

Send a StartProgram message. Send the StartProgram direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

##### Parameters:

*conn* The connection slot (0..3).  
*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](#).

#### 9.49.2.56 **char RemoteStopProgram (byte *conn*) [inline]**

Send a StopProgram message. Send the StopProgram direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

##### Parameters:

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**9.49.2.57 char RemoteStopSound (byte *conn*) [inline]**

Send a StopSound message. Send the StopSound direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**9.49.2.58 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.

**Examples:**

[ex\\_RS485Control.nxc](#).

**9.49.2.59 bool RS485DataAvailable (void) [inline]**

Check for RS485 available data. Check the RS485 hi-speed port for available data.

**Returns:**

A value indicating whether data is available or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485DataAvailable.nxc](#).

**9.49.2.60 char RS485Exit (void) [inline]**

Exit 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\\_RS485Exit.nxc](#).

**9.49.2.61 char RS485Init (void) [inline]**

Initialize RS485. Initialize 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\\_RS485Init.nxc](#).

**9.49.2.62 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\\_RS485Read.nxc](#).

**9.49.2.63 bool RS485SendingData (void) [inline]**

Is RS485 sending data. Check whether the RS485 is actively sending data.

**Returns:**

A value indicating whether data is being sent or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485SendingData.nxc](#).

**9.49.2.64 void RS485Status (bool & *sendingData*, bool & *dataAvail*) [inline]**

Check RS485 status. Check the status of the RS485 hi-speed port.

**Parameters:**

*sendingData* A boolean value set to true on output if data is being sent.

*dataAvail* A boolean value set to true on output if data is available to be read.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Status.nxc](#).

#### 9.49.2.65 `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\\_RS485Uart.nxc](#).

#### 9.49.2.66 `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\\_RS485Write.nxc](#).

#### 9.49.2.67 `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](#).

**9.49.2.68 char SendRemoteBool (byte *conn*, byte *queue*, bool *bval*) [inline]**

Send a boolean value to a remote mailbox. Send a boolean value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

**9.49.2.69 char SendRemoteNumber (byte *conn*, byte *queue*, long *val*) [inline]**

Send a numeric value to a remote mailbox. Send a numeric value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

**9.49.2.70 char SendRemoteString (byte *conn*, byte *queue*, string *str*) [inline]**

Send a string value to a remote mailbox. Send a string value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

**9.49.2.71 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](#).

**9.49.2.72 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](#).

**9.49.2.73 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](#).

**9.49.2.74 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.

**Examples:**

[ex\\_SendRS485Bool.nxc](#).

**9.49.2.75 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\\_SendRS485Number.nxc](#).

**9.49.2.76 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\\_SendRS485String.nxc](#).

**9.49.2.77 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](#).

**9.49.2.78 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](#).

**9.49.2.79 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](#).

**9.49.2.80 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](#).

**9.49.2.81 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](#).

**9.49.2.82 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](#).

**9.49.2.83 void SetHSFlags (const 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](#).

**9.49.2.84 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](#).

**9.49.2.85 void SetHSInputBufferInPtr (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\\_SetHSInputBufferInPtr.nxc](#).

**9.49.2.86 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](#).

**9.49.2.87 void SetHSMode (const 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](#).

**Examples:**

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

**9.49.2.88 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](#).

**9.49.2.89 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](#).

**9.49.2.90 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](#).

**9.49.2.91 void SetHSSpeed (const 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](#).

**9.49.2.92 void SetHSState (const 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](#).

**9.49.2.93 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](#).

**9.49.2.94 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](#).

**9.49.2.95 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](#).



**9.49.2.96 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.](#)

**9.49.2.97 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.](#)

**9.49.2.98 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.](#)

**9.49.2.99 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.](#)

**9.49.2.100 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](#).

**9.49.2.101 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](#).

**9.49.2.102 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](#).

**9.49.2.103 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](#).

**9.49.2.104 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](#).

**9.49.2.105 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](#).

**9.49.2.106 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](#).

**9.49.2.107 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](#).

**9.49.2.108 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](#).

**9.49.2.109 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](#).

**9.49.2.110 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](#).

**9.49.2.111 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](#).

**9.49.2.112 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](#).

**9.49.2.113 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](#).

**9.49.2.114   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.](#)

**9.49.2.115   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.](#)

**9.49.2.116   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.](#)

**9.49.2.117   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.](#)

**9.49.2.118 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](#).

**9.49.2.119 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](#).

**9.49.2.120 byte USBState (void) [inline]**

Get USB state. This method returns the value of the USB state.

**Returns:**

The USB state.

## 9.50 Button module types

Types used by various Button module functions.

### Data Structures

- struct [ReadButtonType](#)

*Parameters for the ReadButton system call.*

### 9.50.1 Detailed Description

Types used by various Button module functions.



## 9.51 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.*

### 9.51.1 Detailed Description

Functions for accessing and modifying Button module features.

### 9.51.2 Function Documentation

#### 9.51.2.1 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](#).

#### 9.51.2.2 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](#).

#### 9.51.2.3 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](#).

**9.51.2.4 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](#).

**9.51.2.5 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](#).

**9.51.2.6 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](#).

**9.51.2.7 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](#).

**9.51.2.8 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](#).

**9.51.2.9 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](#).

**9.51.2.10 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](#).

**9.51.2.11 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](#).

**9.51.2.12 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](#).

**9.51.2.13 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](#).

**9.51.2.14 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](#).

**9.51.2.15 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](#).

**9.51.2.16 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](#).

## 9.52 Ui module types

Types used by various Ui module functions.

### Data Structures

- struct [SetSleepTimeoutType](#)

*Parameters for the SetSleepTimeout system call.*

### 9.52.1 Detailed Description

Types used by various Ui module functions.

## 9.53 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.*

### 9.53.1 Detailed Description

Functions for accessing and modifying Ui module features.

### 9.53.2 Function Documentation

#### 9.53.2.1 `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](#).

#### 9.53.2.2 `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](#).

#### 9.53.2.3 `byte BatteryState (void) [inline]`

Get battery state. Return battery state information (0..4).

**Returns:**

The battery state (0..4)

**Examples:**

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

#### 9.53.2.4 `byte BluetoothState (void) [inline]`

Get bluetooth state. Return the bluetooth state.

**Returns:**

The bluetooth state. See [BluetoothState constants](#).

**Examples:**

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

**9.53.2.5 byte CommandFlags (void) [inline]**

Get command flags. Return the command flags.

**Returns:**

Command flags. See [CommandFlags constants](#)

**Examples:**

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

**9.53.2.6 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](#).

**9.53.2.7 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](#).

### 9.53.2.8 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](#).

### 9.53.2.9 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](#).

### 9.53.2.10 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](#).

### 9.53.2.11 void SetBatteryState (byte *state*) [inline]

Set battery state. Set battery state information.

#### Parameters:

*state* The desired battery state (0..4).

#### Examples:

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

**9.53.2.12 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](#).

**9.53.2.13 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](#).

**9.53.2.14 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](#), and [ex\\_SetLongAbort.nxc](#).

**9.53.2.15 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](#).

**9.53.2.16 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](#).

**9.53.2.17 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](#).

**9.53.2.18 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](#).

**9.53.2.19 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](#).

**9.53.2.20 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](#).

**9.53.2.21 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](#).

**Examples:**

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

**9.53.2.22 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](#).

**9.53.2.23 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](#).

#### 9.53.2.24 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](#).

#### 9.53.2.25 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](#).

#### 9.53.2.26 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](#).

#### 9.53.2.27 byte UIButton (void) [inline]

Read UI button. Return user interface button information.

**Returns:**

A UI button value. See [UIButton constants](#).

**Examples:**

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



**9.53.2.28 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](#).

**9.53.2.29 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](#).

**9.53.2.30 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](#).

**9.53.2.31 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](#).

## 9.54 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 [ListFilesType](#)  
*Parameters for the ListFiles system call.*

### 9.54.1 Detailed Description

Types used by various Loader module functions.

## 9.55 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 [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 `SysListFiles` (`ListFilesType` &args)  
*List files.*

### 9.55.1 Detailed Description

Functions for accessing and modifying Loader module features.

### 9.55.2 Function Documentation

#### 9.55.2.1 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](#).

**9.55.2.2 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](#).

**9.55.2.3 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](#).

**9.55.2.4 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](#).

**9.55.2.5 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\\_DeleteFile.nxc](#).

**9.55.2.6 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](#).

**9.55.2.7 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\\_FindNextFile.nxc](#).

**9.55.2.8 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](#).

**9.55.2.9 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\\_OpenFileAppend.nxc](#).

**9.55.2.10 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\\_OpenFileRead.nxc](#).

**9.55.2.11 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](#).

**9.55.2.12 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\\_Read.nxc](#).

**9.55.2.13 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](#).

**9.55.2.14 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](#).

**9.55.2.15 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](#).

**9.55.2.16 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.

**Examples:**

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

**9.55.2.17 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.

*newsize* The new size for the file.

**Returns:**

The function call result. See [Loader module error codes](#).

**Examples:**

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

**9.55.2.18 unsigned int ResolveHandle (string *filename*, byte & *handle*, bool & *writeable*) [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.

*writeable* 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](#).

**9.55.2.19 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](#).

**9.55.2.20 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](#).

**9.55.2.21 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](#).

**9.55.2.22 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](#).

**9.55.2.23 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](#).

**9.55.2.24 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](#).

**9.55.2.25 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](#).

**9.55.2.26 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](#).

**9.55.2.27 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](#).

**9.55.2.28 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](#).

**9.55.2.29 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](#).

**9.55.2.30 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](#).

**9.55.2.31 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](#).

**9.55.2.32 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](#).

**9.55.2.33 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](#).

**9.55.2.34 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](#).

**9.55.2.35 void SysListFiles (ListFileType & args) [inline]**

List files. This function lets you retrieve a list of files on the NXT using the values specified via the [ListFileType](#) structure.

**Parameters:**

*args* The [ListFileType](#) structure containing the needed parameters.

**Examples:**

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

**9.55.2.36 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](#).

**9.55.2.37 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\\_Write.nxc](#).

### 9.55.2.38 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](#).

### 9.55.2.39 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](#).

### 9.55.2.40 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](#).

**9.55.2.41 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](#).

**9.55.2.42 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](#).

## 9.56 cmath API

Standard C cmath API functions.

### 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.*

### 9.56.1 Detailed Description

Standard C cmath API functions.

### 9.56.2 Define Documentation

#### 9.56.2.1 #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.

#### 9.56.2.2 #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.

#### 9.56.2.3 #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.



**9.56.2.4 #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.

**9.56.2.5 #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.

**9.56.2.6 #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.

**9.56.2.7 #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  $\_Y/\_X$ .

**9.56.2.8 #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  $\_X$ .

**9.56.2.9 #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  $\_X$ .

**9.56.2.10 #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.

**9.56.2.11 #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.

**9.56.2.12 #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.

**9.56.2.13 #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.

**9.56.2.14 #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.

**9.56.2.15 #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.

**9.56.2.16 #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.

**9.56.2.17 #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.

**9.56.2.18 #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.

**9.56.2.19** `#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*.

**9.56.2.20** `#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*.

**9.56.2.21** `#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*.

**9.56.2.22 #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.

**9.56.2.23 #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.

**9.56.2.24 #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.

**9.56.2.25** `#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`.

**9.56.2.26** `#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`.

**9.56.2.27** `#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`.



**9.56.2.28 #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.

**9.56.2.29 #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.

**9.56.2.30 #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.

### 9.56.3 Function Documentation

#### 9.56.3.1 `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](#).

#### 9.56.3.2 `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](#).

#### 9.56.3.3 `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](#).

**9.56.3.4 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](#).

**9.56.3.5 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](#).

### 9.56.3.6 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](#).

### 9.56.3.7 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 [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2d.nxc](#).

### 9.56.3.8 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](#).

**9.56.3.9 byte bcd2dec (byte *bcd*) [inline]**

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

**Returns:**

The decimal equivalent of the binary coded decimal byte

**Examples:**

[ex\\_bcd2dec.nxc](#).

**9.56.3.10 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](#).

**9.56.3.11 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\\_cos.nxc](#).

**9.56.3.12 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\\_cosd.nxc](#).

**9.56.3.13 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](#).

**9.56.3.14 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.

**9.56.3.15 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](#).

**9.56.3.16 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](#).

### 9.56.3.17 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](#).

### 9.56.3.18 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](#).

### 9.56.3.19 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](#).

**9.56.3.20 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](#).

**9.56.3.21 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](#).

### 9.56.3.22 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](#).

### 9.56.3.23 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](#).

### 9.56.3.24 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.nxc](#).

**9.56.3.25 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.nxc](#).

**9.56.3.26 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](#).

**9.56.3.27 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.

### 9.56.3.28 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](#).

### 9.56.3.29 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](#).

### 9.56.3.30 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](#).

**9.56.3.31 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](#).

**9.56.3.32 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.

**9.56.3.33 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\\_trunc.nxc](#).

## 9.57 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.*
- byte [fopen](#) (string filename, const string mode)  
*Open file.*
- int [fflush](#) (byte handle)  
*Flush file.*
- 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.*

### 9.57.1 Detailed Description

Standard C cstdio API functions.

### 9.57.2 Define Documentation

#### 9.57.2.1 #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](#).

### 9.57.2.2 `#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](#).

## 9.57.3 Function Documentation

### 9.57.3.1 `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](#).

### 9.57.3.2 `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](#).



### 9.57.3.3 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](#).

### 9.57.3.4 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](#).

### 9.57.3.5 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](#).

### 9.57.3.6 `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](#).

### 9.57.3.7 `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](#).

### 9.57.3.8 `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](#).

**9.57.3.9 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](#).

**9.57.3.10 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](#).

### 9.57.3.11 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:

Currently always returns -1.

#### Examples:

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

### 9.57.3.12 int getchar () [inline]

Get character from stdin. Returns the next character from the standard input (stdin). It is equivalent togetc 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](#).

### 9.57.3.13 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](#).

**9.57.3.14 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.

**9.57.3.15 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](#).

**9.57.3.16 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](#).

**9.57.3.17 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](#).

## 9.58 fseek origin constants

Constants for use in calls to `fseek`.

### Defines

- `#define SEEK_SET 0`
- `#define SEEK_CUR 1`
- `#define SEEK_END 2`

### 9.58.1 Detailed Description

Constants for use in calls to `fseek`.

### 9.58.2 Define Documentation

#### 9.58.2.1 `#define SEEK_CUR 1`

Seek from the current file position

#### Examples:

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

#### 9.58.2.2 `#define SEEK_END 2`

Seek from the end of the file

#### 9.58.2.3 `#define SEEK_SET 0`

Seek from the beginning of the file

#### Examples:

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

## 9.59 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.*
- unsigned int [rand](#) ()  
*Generate random number.*
- int [Random](#) (unsigned int n=0)  
*Generate random number.*
- void [SysRandomNumber](#) ([RandomNumberType](#) &args)  
*Draw a random number.*
- 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.*

## 9.59.1 Detailed Description

Standard C cstdlib API functions and types.

## 9.59.2 Function Documentation

### 9.59.2.1 `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](#).

### 9.59.2.2 `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](#).

### 9.59.2.3 `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](#).

**9.59.2.4 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](#).

**9.59.2.5 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](#).

**9.59.2.6 `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](#).

**9.59.2.7 `long labs (long n) [inline]`**

Absolute value. Return the absolute value of parameter *n*.

**Parameters:**

*n* Integral value.

**Returns:**

The absolute value of *n*.

### 9.59.2.8 `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](#).

### 9.59.2.9 `unsigned int rand () [inline]`

Generate random number. Returns a pseudo-random integral number in the range 0 to `RAND_MAX`.

#### Returns:

An integer value between 0 and `RAND_MAX`.

#### 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](#).

### 9.59.2.10 `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\\_Random.nxc](#), [ex\\_string.nxc](#), [ex\\_sysdrawellipse.nxc](#), and [ex\\_wait.nxc](#).

**9.59.2.11 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](#).

**9.59.2.12 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](#).

**9.59.2.13 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](#).

**9.59.2.14 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](#).

## 9.60 cstdlib API types

Standard C cstdlib API types.

### Data Structures

- struct [RandomNumberType](#)  
*Parameters for the RandomNumber system call.*
- struct [div\\_t](#)  
*Output type of the div function.*
- struct [ldiv\\_t](#)  
*Output type of the ldiv function.*

### 9.60.1 Detailed Description

Standard C cstdlib API types.



## 9.61 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.*
- 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 [addr](#) (variant data)  
*Get the absolute address of a variable.*
- unsigned long [reladdr](#) (variant data)  
*Get the relative address of a variable.*
- unsigned long [addrex](#) (variant data, bool relative)  
*Get the absolute or relative address of a variable.*

### 9.61.1 Detailed Description

Standard C cstring API functions.

### 9.61.2 Function Documentation

#### 9.61.2.1 unsigned long addr (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\\_addr.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

#### 9.61.2.2 unsigned long address (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\\_address.nxc](#).

### 9.61.2.3 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](#).

### 9.61.2.4 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](#).

### 9.61.2.5 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](#).

#### 9.61.2.6 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](#).

#### 9.61.2.7 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](#), and [ex\\_string.nxc](#).

#### 9.61.2.8 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\\_addr.nxc](#), [ex\\_addrex.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_FormatNum.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

### 9.61.2.9 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](#).

### 9.61.2.10 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](#).

### 9.61.2.11 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](#).

**9.61.2.12 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](#).

**9.61.2.13 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](#).

**9.61.2.14 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](#), and [ex\\_string.nxc](#).

### 9.61.2.15 unsigned long reladdr (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\\_reladdr.nxc](#).

### 9.61.2.16 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](#).

### 9.61.2.17 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](#).

**9.61.2.18 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\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), and [util\\_battery\\_1.nxc](#).

**9.61.2.19 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](#).

**9.61.2.20 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](#).

**9.61.2.21 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](#).

**9.61.2.22 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](#).

**9.61.2.23 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](#).

**9.61.2.24 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](#).

**9.61.2.25 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](#).

**9.61.2.26 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](#).

**9.61.2.27 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](#).

**9.61.2.28 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](#).

**9.61.2.29 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](#).

**9.61.2.30 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](#).

**9.61.2.31 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\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

## 9.62 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.*

### 9.62.1 Detailed Description

Standard C ctype API functions.

## 9.62.2 Function Documentation

### 9.62.2.1 `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](#).

### 9.62.2.2 `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](#).

### 9.62.2.3 `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](#).



#### 9.62.2.4 `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](#).

#### 9.62.2.5 `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](#).

#### 9.62.2.6 `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](#).

#### 9.62.2.7 `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](#).

#### 9.62.2.8 `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](#).

#### 9.62.2.9 `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](#).

#### 9.62.2.10 `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](#).

**9.62.2.11 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](#).

**9.62.2.12 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](#).

**9.62.2.13 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](#).

## 9.63 Array operation constants

Constants for use with the [ArrayOp](#) function.

### 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`

### 9.63.1 Detailed Description

Constants for use with the [ArrayOp](#) function.

### 9.63.2 Define Documentation

#### 9.63.2.1 `#define OPARR_MAX 0x05`

Calculate the maximum value of the elements in the numeric input array

#### Examples:

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

#### 9.63.2.2 `#define OPARR_MEAN 0x01`

Calculate the mean value for the elements in the numeric input array

#### 9.63.2.3 `#define OPARR_MIN 0x04`

Calculate the minimum value of the elements in the numeric input array

#### 9.63.2.4 `#define OPARR_SORT 0x06`

Sort the elements in the numeric input array

#### 9.63.2.5 `#define OPARR_STD 0x03`

Calculate the standard deviation of the elements in the numeric input array

#### 9.63.2.6 `#define OPARR_SUM 0x00`

Calculate the sum of the elements in the numeric input array

**9.63.2.7 #define OPARR\_SUMSQR 0x02**

Calculate the sum of the squares of the elements in the numeric input array

## 9.64 System Call function constants

Constants for use in the [SysCall\(\)](#) function.

### 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` [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

### 9.64.1 Detailed Description

Constants for use in the [SysCall\(\)](#) function.

### 9.64.2 Define Documentation

#### 9.64.2.1 `#define` ColorSensorRead 34

Read data from the NXT 2.0 color sensor

#### 9.64.2.2 `#define` CommBTCheckStatus 28

Check the bluetooth status

#### 9.64.2.3 `#define` CommBTConnection 36

Connect or disconnect to a known bluetooth device

#### 9.64.2.4 `#define` CommBTOnOff 35

Turn the bluetooth radio on or off



**9.64.2.5 #define CommBTRead 30**

Read from a bluetooth connection

**9.64.2.6 #define CommBTWrite 29**

Write to a bluetooth connections

**9.64.2.7 #define CommExecuteFunction 81**

Execute one of the Comm module's internal functions

**9.64.2.8 #define CommHSCheckStatus 39**

Check the status of the hi-speed port

**9.64.2.9 #define CommHSControl 88**

Control the hi-speed port

**9.64.2.10 #define CommHSRead 38**

Read data from the hi-speed port

**9.64.2.11 #define CommHSWrite 37**

Write data to the hi-speed port

**9.64.2.12 #define CommLSCheckStatus 23**

Check the status of a lowspeed (aka I2C) device

**9.64.2.13 #define CommLSRead 22**

Read from a lowspeed (aka I2C) device

**9.64.2.14 #define CommLSWrite 21**

Write to a lowspeed (aka I2C) device

**9.64.2.15 #define CommLSWriteEx 89**

Write to a lowspeed (aka I2C) device with optional restart on read

**9.64.2.16 #define ComputeCalibValue 42**

Compute a calibration value

**9.64.2.17 #define DatalogGetTimes 45**

Get datalog timing information

**9.64.2.18 #define DatalogWrite 44**

Write to the datalog

**9.64.2.19 #define DisplayExecuteFunction 80**

Execute one of the Display module's internal functions

**9.64.2.20 #define DrawCircle 16**

Draw a circle on the LCD screen

**9.64.2.21 #define DrawEllipse 94**

Draw an ellipse on the LCD screen

**9.64.2.22 #define DrawFont 95**

Draw text using a custom RIC-based font to the LCD screen

**9.64.2.23 #define DrawGraphic 18**

Draw a graphic image on the LCD screen

**9.64.2.24 #define DrawGraphicArray 92**

Draw a graphic image from a byte array to the LCD screen

**Examples:**

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

**9.64.2.25 #define DrawLine 15**

Draw a line on the LCD screen

**9.64.2.26 #define DrawPoint 14**

Draw a single pixel on the LCD screen

**9.64.2.27 #define DrawPolygon 93**

Draw a polygon on the LCD screen

**9.64.2.28 #define DrawRect 17**

Draw a rectangle on the LCD screen

**9.64.2.29 #define DrawText 13**

Draw text to one of 8 LCD lines

**Examples:**

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

**9.64.2.30 #define FileClose 5**

Close the specified file

**9.64.2.31 #define FileDelete 8**

Delete a file

**9.64.2.32 #define FileFindFirst 83**

Start a search for a file using a filename pattern

**9.64.2.33 #define FileFindNext 84**

Continue searching for a file

**9.64.2.34 #define FileOpenAppend 2**

Open a file for appending to the end of the file

**9.64.2.35 #define FileOpenRead 0**

Open a file for reading

**9.64.2.36 #define FileOpenReadLinear 87**

Open a linear file for reading

**9.64.2.37 #define FileOpenWrite 1**

Open a file for writing (creates a new file)

**9.64.2.38 #define FileOpenWriteLinear 85**

Open a linear file for writing

**9.64.2.39 #define FileOpenWriteNonLinear 86**

Open a non-linear file for writing

**9.64.2.40 #define FileRead 3**

Read from the specified file

**9.64.2.41 #define FileRename 7**

Rename a file

**9.64.2.42 #define FileResize 91**

Resize a file (not yet implemented)

**9.64.2.43 #define FileResolveHandle 6**

Get a file handle for the specified filename if it is already open

**9.64.2.44 #define FileSeek 90**

Seek to a specific position in an open file

**9.64.2.45 #define FileWrite 4**

Write to the specified file

**9.64.2.46 #define GetStartTick 25**

Get the current system tick count

**9.64.2.47 #define IOMapRead 32**

Read data from one of the firmware module's IOMap structures using the module's name

**9.64.2.48 #define IOMapReadByID 78**

Read data from one of the firmware module's IOMap structures using the module's ID

**9.64.2.49 #define IOMapWrite 33**

Write data to one of the firmware module's IOMap structures using the module's name

**9.64.2.50 #define IOMapWriteByID 79**

Write data to one of the firmware module's IOMap structures using the module's ID

**9.64.2.51 #define KeepAlive 31**

Reset the NXT sleep timer

**9.64.2.52 #define ListFiles 47**

List files that match the specified filename pattern

**9.64.2.53 #define LoaderExecuteFunction 82**

Execute one of the Loader module's internal functions

**9.64.2.54 #define MessageRead 27**

Read a message from a mailbox

**9.64.2.55 #define MessageWrite 26**

Write a message to a mailbox

**9.64.2.56 #define RandomNumber 24**

Generate a random number

**9.64.2.57 #define ReadButton 20**

Read the current button state

**9.64.2.58 #define ReadSemData 40**

Read motor semaphore data

**9.64.2.59 #define SetScreenMode 19**

Set the screen mode

**9.64.2.60 #define SetSleepTimeoutVal 46**

Set the NXT sleep timeout value

**9.64.2.61 #define SoundGetState 11**

Get the current sound module state

**9.64.2.62 #define SoundPlayFile 9**

Play a sound or melody file

**9.64.2.63 #define SoundPlayTone 10**

Play a simple tone with the specified frequency and duration

**9.64.2.64 #define SoundSetState 12**

Set the sound module state

**9.64.2.65 #define UpdateCalibCacheInfo 43**

Update sensor calibration cache information

**9.64.2.66 #define WriteSemData 41**

Write motor semaphore data

## 9.65 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

### 9.65.1 Detailed Description

Line numbers for use with DrawText system function.

See also:

[SysDrawText\(\)](#), [TextOut\(\)](#), [NumOut\(\)](#)

### 9.65.2 Define Documentation

#### 9.65.2.1 #define LCD\_LINE1 56

The 1st line of the LCD screen

Examples:

[ex\\_addr.nxc](#), [ex\\_addrex.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\\_ClearLine.nxc](#), [ex\\_contrast.nxc](#), [ex\\_Copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_syscall.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysCommBTOnOff.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\\_sysmessageread.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), and [ex\\_SysWriteSemData.nxc](#).

#### 9.65.2.2 #define LCD\_LINE2 48

The 2nd line of the LCD screen

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrex.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\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_sysfileread.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**9.65.2.3 #define LCD\_LINE3 40**

The 3rd line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SysCommBTConnection.nxc](#), and [ex\\_TextOut.nxc](#).

**9.65.2.4 #define LCD\_LINE4 32**

The 4th line of the LCD screen

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrex.nxc](#), [ex\\_arraybuild.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), and [ex\\_SysDatalogGetTimes.nxc](#).

**9.65.2.5 #define LCD\_LINE5 24**

The 5th line of the LCD screen

**Examples:**

[ex\\_arraybuild.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), and [ex\\_SysDatalogGetTimes.nxc](#).

**9.65.2.6 #define LCD\_LINE6 16**

The 6th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), and [ex\\_syslistfiles.nxc](#).



**9.65.2.7 #define LCD\_LINE7 8**

The 7th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), and [ex\\_string.nxc](#).

**9.65.2.8 #define LCD\_LINE8 0**

The 8th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), and [ex\\_string.nxc](#).

## 9.66 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

### 9.66.1 Detailed Description

Constants for use with the [Wait\(\)](#) function.

See also:

[Wait\(\)](#)

### 9.66.2 Define Documentation

#### 9.66.2.1 `#define MIN_1 60000`

1 minute

**Examples:**

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

#### 9.66.2.2 `#define MS_1 1`

1 millisecond

#### 9.66.2.3 `#define MS_10 10`

10 milliseconds

#### 9.66.2.4 `#define MS_100 100`

100 milliseconds

**Examples:**

[ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

#### 9.66.2.5 `#define MS_150 150`

150 milliseconds

#### 9.66.2.6 `#define MS_2 2`

2 milliseconds

#### 9.66.2.7 `#define MS_20 20`

20 milliseconds

**Examples:**

[ex\\_dispgayout.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).

**9.66.2.8 #define MS\_200 200**

200 milliseconds

**Examples:**

[ex\\_dispgoutex.nxc](#), and [ex\\_playtones.nxc](#).

**9.66.2.9 #define MS\_250 250**

250 milliseconds

**9.66.2.10 #define MS\_3 3**

3 milliseconds

**9.66.2.11 #define MS\_30 30**

30 milliseconds

**9.66.2.12 #define MS\_300 300**

300 milliseconds

**9.66.2.13 #define MS\_350 350**

350 milliseconds

**9.66.2.14 #define MS\_4 4**

4 milliseconds

**9.66.2.15 #define MS\_40 40**

40 milliseconds

**9.66.2.16 #define MS\_400 400**

400 milliseconds

**9.66.2.17 #define MS\_450 450**

450 milliseconds

**9.66.2.18 #define MS\_5 5**

5 milliseconds

**Examples:**

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

**9.66.2.19 #define MS\_50 50**

50 milliseconds

**Examples:**

[ex\\_CircleOut.nxc](#), and [ex\\_playtones.nxc](#).

**9.66.2.20 #define MS\_500 500**

500 milliseconds

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_playsound.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

**9.66.2.21 #define MS\_6 6**

6 milliseconds

**9.66.2.22 #define MS\_60 60**

60 milliseconds

**9.66.2.23 #define MS\_600 600**

600 milliseconds

**9.66.2.24 #define MS\_7 7**

7 milliseconds

**9.66.2.25 #define MS\_70 70**

70 milliseconds

**9.66.2.26 #define MS\_700 700**

700 milliseconds

**9.66.2.27 #define MS\_8 8**

8 milliseconds

**9.66.2.28 #define MS\_80 80**

80 milliseconds

**9.66.2.29 #define MS\_800 800**

800 milliseconds

**9.66.2.30 #define MS\_9 9**

9 milliseconds

**9.66.2.31 #define MS\_90 90**

90 milliseconds

**9.66.2.32 #define MS\_900 900**

900 milliseconds

**9.66.2.33 #define SEC\_1 1000**

1 second

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_sysdrawpolygon.nxc](#), [ex\\_wait.nxc](#), and [ex\\_yield.nxc](#).

**9.66.2.34 #define SEC\_10 10000**

10 seconds

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrx.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_SysCommBTConnection.nxc](#), and [ex\\_SysCommHSControl.nxc](#).

**9.66.2.35 #define SEC\_15 15000**

15 seconds

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_memcmp.nxc](#).

**9.66.2.36 #define SEC\_2 2000**

2 seconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**9.66.2.37 #define SEC\_20 20000**

20 seconds

**9.66.2.38 #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](#).

**9.66.2.39 #define SEC\_30 30000**

30 seconds

**9.66.2.40 #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](#).

**9.66.2.41 #define SEC\_5 5000**

5 seconds

**Examples:**

[ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_ClearLine.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_isNAN.nxc](#), [ex\\_labs.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](#), and [ex\\_wait.nxc](#).

**9.66.2.42 #define SEC\_6 6000**

6 seconds

**Examples:**

[ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), and [ex\\_strtoul.nxc](#).

**9.66.2.43 #define SEC\_7 7000**

7 seconds

**9.66.2.44 #define SEC\_8 8000**

8 seconds

**9.66.2.45 #define SEC\_9 9000**

9 seconds

**Examples:**

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



## 9.67 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

### 9.67.1 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\(\)](#)

### 9.67.2 Define Documentation

#### 9.67.2.1 #define MAILBOX1 0

Mailbox number 1

#### Examples:

[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](#).

#### 9.67.2.2 #define MAILBOX10 9

Mailbox number 10

#### 9.67.2.3 #define MAILBOX2 1

Mailbox number 2

**9.67.2.4 #define MAILBOX3 2**

Mailbox number 3

**9.67.2.5 #define MAILBOX4 3**

Mailbox number 4

**9.67.2.6 #define MAILBOX5 4**

Mailbox number 5

**9.67.2.7 #define MAILBOX6 5**

Mailbox number 6

**9.67.2.8 #define MAILBOX7 6**

Mailbox number 7

**9.67.2.9 #define MAILBOX8 7**

Mailbox number 8

**9.67.2.10 #define MAILBOX9 8**

Mailbox number 9

## 9.68 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

### 9.68.1 Detailed Description

Constants defining possible VM states.

### 9.68.2 Define Documentation

#### 9.68.2.1 #define `BREAKOUT_REQ` 3

VM should break out of current thread

#### 9.68.2.2 #define `CLUMP_DONE` 1

VM has finished executing thread

#### 9.68.2.3 #define `CLUMP_SUSPEND` 2

VM should suspend thread

#### 9.68.2.4 #define `ROTATE_QUEUE` 5

VM should rotate queue

#### 9.68.2.5 #define `STOP_REQ` 4

VM should stop executing program

#### 9.68.2.6 #define `TIMES_UP` 6

VM time is up

## 9.69 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`

### 9.69.1 Detailed Description

Constants defining various fatal error conditions.

### 9.69.2 Define Documentation

#### 9.69.2.1 `#define ERR_ARG -1`

0xFF Bad arguments

#### 9.69.2.2 `#define ERR_BAD_POOL_SIZE -10`

0xF6 VarsCmd.PoolSize > POOL\_MAX\_SIZE

#### 9.69.2.3 `#define ERR_BAD_PTR -6`

0xFA Someone passed us a bad pointer!

#### 9.69.2.4 `#define ERR_CLUMP_COUNT -7`

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT\_A\_CLUMP)

**9.69.2.5 #define ERR\_DEFAULT\_OFFSETS -14**

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

**9.69.2.6 #define ERR\_FILE -3**

0xFD Malformed file contents

**9.69.2.7 #define ERR\_INSANE\_OFFSET -9**

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount \* 2)

**9.69.2.8 #define ERR\_INSTR -2**

0xFE Illegal bytecode instruction

**9.69.2.9 #define ERR\_LOADER\_ERR -11**

0xF5 LOADER\_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

**9.69.2.10 #define ERR\_MEM -5**

0xFB Insufficient memory available

**9.69.2.11 #define ERR\_MEMMGR\_FAIL -15**

0xF1 (UBYTE \*)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV\_ARRAY[0].Offset

**9.69.2.12 #define ERR\_NO\_ACTIVE\_CLUMP -13**

0xF3 VarsCmd.RunQ.Head == NOT\_A\_CLUMP

**9.69.2.13 #define ERR\_NO\_CODE -8**

0xF8 VarsCmd.CodespaceCount == 0

**9.69.2.14 #define ERR\_NON\_FATAL -16**

Fatal errors are greater than this value

**9.69.2.15 #define ERR\_SPOTCHECK\_FAIL -12**

0xF4 ((UBYTE\*)(VarsCmd.pCodespace) < pData) (c\_cmd.c 1893)

**9.69.2.16 #define ERR\_VER -4**

0xFC Version mismatch between firmware and compiler

## 9.70 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

### 9.70.1 Detailed Description

Constants defining general error conditions.

### 9.70.2 Define Documentation

#### 9.70.2.1 #define [ERR\\_INVALID\\_FIELD](#) -17

0xEF Attempted to access invalid field of a structure

#### 9.70.2.2 #define [ERR\\_INVALID\\_PORT](#) -16

0xF0 Bad input or output port specified

#### 9.70.2.3 #define [ERR\\_INVALID\\_QUEUE](#) -18

0xEE Illegal queue ID specified

#### 9.70.2.4 #define [ERR\\_INVALID\\_SIZE](#) -19

0xED Illegal size specified

#### 9.70.2.5 #define [ERR\\_NO\\_PROG](#) -20

0xEC No active program

## 9.71 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`

### 9.71.1 Detailed Description

Constants defining communication error conditions.

### 9.71.2 Define Documentation

#### 9.71.2.1 `#define ERR_COMM_BUFFER_FULL -34`

0xDE No room in comm buffer

#### 9.71.2.2 `#define ERR_COMM_BUS_ERR -35`

0xDD Something went wrong on the communications bus

#### 9.71.2.3 `#define ERR_COMM_CHAN_INVALID -33`

0xDF Specified channel/connection is not valid

#### 9.71.2.4 `#define ERR_COMM_CHAN_NOT_READY -32`

0xE0 Specified channel/connection not configured or busy



## 9.72 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`

### 9.72.1 Detailed Description

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

### 9.72.2 Define Documentation

#### 9.72.2.1 `#define ERR_RC_BAD_PACKET -65`

0xBF Clearly insane packet

#### 9.72.2.2 `#define ERR_RC_FAILED -67`

0xBD Request failed (i.e. specified file not found)

#### 9.72.2.3 `#define ERR_RC_ILLEGAL_VAL -64`

0xC0 Data contains out-of-range values

#### 9.72.2.4 `#define ERR_RC_UNKNOWN_CMD -66`

0xBE Unknown command opcode

## 9.73 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`

### 9.73.1 Detailed Description

Constants defining various states of the command module virtual machine.

### 9.73.2 Define Documentation

#### 9.73.2.1 `#define PROG_ABORT 4`

Program has been aborted

#### 9.73.2.2 `#define PROG_ERROR 3`

A program error has occurred

#### 9.73.2.3 `#define PROG_IDLE 0`

Program state is idle

#### 9.73.2.4 `#define PROG_OK 1`

Program state is okay

#### 9.73.2.5 `#define PROG_RESET 5`

Program has been reset

#### 9.73.2.6 `#define PROG_RUNNING 2`

Program is running

## 9.74 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

### 9.74.1 Detailed Description

Constant offsets into the Command module IOMAP structure.

### 9.74.2 Define Documentation

#### 9.74.2.1 #define [CommandOffsetActivateFlag](#) 30

Offset to the activate flag

#### 9.74.2.2 #define [CommandOffsetAwake](#) 29

Offset to the VM's awake state

#### 9.74.2.3 #define [CommandOffsetDeactivateFlag](#) 31

Offset to the deactivate flag

#### 9.74.2.4 #define [CommandOffsetFileName](#) 32

Offset to the running program's filename

#### 9.74.2.5 #define [CommandOffsetFormatString](#) 0

Offset to the format string

**9.74.2.6 #define CommandOffsetMemoryPool 52**

Offset to the VM's memory pool

**Examples:**

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

**9.74.2.7 #define CommandOffsetOffsetDS 24**

Offset to the running program's data space (DS)

**9.74.2.8 #define CommandOffsetOffsetDVA 26**

Offset to the running program's DOPE vector address (DVA)

**9.74.2.9 #define CommandOffsetPRCHandler 16**

Offset to the RC Handler function pointer

**9.74.2.10 #define CommandOffsetProgStatus 28**

Offset to the running program's status

**9.74.2.11 #define CommandOffsetSyncTick 32824**

Offset to the VM sync tick

**9.74.2.12 #define CommandOffsetSyncTime 32820**

Offset to the VM sync time

**9.74.2.13 #define CommandOffsetTick 20**

Offset to the VM's current tick

**Examples:**

[ex\\_sysiomapread.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

## 9.75 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.*

### 9.75.1 Detailed Description

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

## 9.76 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`

### 9.76.1 Detailed Description

Use these constants to power down the NXT or boot it into SAMBA (aka firmware download) mode.

### 9.76.2 Define Documentation

#### 9.76.2.1 `#define IOCTRL_BOOT 0xA55A`

Reboot the NXT into SAMBA mode

#### 9.76.2.2 `#define IOCTRL_POWERDOWN 0x5A00`

Power down the NXT

## 9.77 IOCtrl module IOMAP offsets

Constant offsets into the IOCtrl module IOMAP structure.

### Defines

- #define `IOCtrlOffsetPowerOn` 0

#### 9.77.1 Detailed Description

Constant offsets into the IOCtrl module IOMAP structure.

#### 9.77.2 Define Documentation

##### 9.77.2.1 #define `IOCtrlOffsetPowerOn` 0

Offset to power on field

## 9.78 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`

### 9.78.1 Detailed Description

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

### 9.78.2 Define Documentation

#### 9.78.2.1 `#define EOF -1`

A constant representing end of file

#### 9.78.2.2 `#define NULL 0`

A constant representing NULL



## 9.79 Loader module IOMAP offsets

Constant offsets into the Loader module IOMAP structure.

### Defines

- `#define LoaderOffsetPFunc 0`
- `#define LoaderOffsetFreeUserFlash 4`

### 9.79.1 Detailed Description

Constant offsets into the Loader module IOMAP structure.

### 9.79.2 Define Documentation

#### 9.79.2.1 `#define LoaderOffsetFreeUserFlash 4`

Offset to the amount of free user flash

#### 9.79.2.2 `#define LoaderOffsetPFunc 0`

Offset to the Loader module function pointer

## 9.80 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

### 9.80.1 Detailed Description

Error codes returned by functions in the Loader module (file access).

### 9.80.2 Define Documentation

#### 9.80.2.1 #define LDR\_APPENDNOTPOSSIBLE 0x8D00

Only datafiles can be appended to.

**9.80.2.2 #define LDR\_BTBUSY 0x9400**

The bluetooth system is busy.

**9.80.2.3 #define LDR\_BTCONNECTFAIL 0x9500**

Bluetooth connection attempt failed.

**9.80.2.4 #define LDR\_BTTIMEOUT 0x9600**

A timeout in the bluetooth system has occurred.

**9.80.2.5 #define LDR\_ENDOFFILE 0x8500**

The end of the file has been reached.

**9.80.2.6 #define LDR\_EOFEXPECTED 0x8400**

EOF expected.

**9.80.2.7 #define LDR\_FILEEXISTS 0x8F00**

A file with the same name already exists.

**9.80.2.8 #define LDR\_FILEISBUSY 0x8B00**

The file is already being used.

**9.80.2.9 #define LDR\_FILEISFULL 0x8E00**

The allocated file size has been filled.

**9.80.2.10 #define LDR\_FILENOTFOUND 0x8700**

No files matched the search criteria.

**9.80.2.11 #define LDR\_FILETX\_CLOSEERROR 0x9B00**

Error transmitting file: attempt to close file failed.

**9.80.2.12 #define LDR\_FILETX\_DSTEXISTS 0x9800**

Error transmitting file: destination file exists.

**9.80.2.13 #define LDR\_FILETX\_SRCMISSING 0x9900**

Error transmitting file: source file is missing.

**9.80.2.14 #define LDR\_FILETX\_STREAMERROR 0x9A00**

Error transmitting file: a stream error occurred.

**9.80.2.15 #define LDR\_FILETX\_TIMEOUT 0x9700**

Error transmitting file: a timeout occurred.

**9.80.2.16 #define LDR\_HANDLEALREADYCLOSED 0x8800**

The file handle has already been closed.

**9.80.2.17 #define LDR\_ILLEGALFILENAME 0x9200**

Filename length too long or attempted open a system file (\*.rxe, \*.rtm, or \*.sys) for writing as a datafile.

**9.80.2.18 #define LDR\_ILLEGALHANDLE 0x9300**

Invalid file handle.

**9.80.2.19 #define LDR\_INPROGRESS 0x0001**

The function is executing but has not yet completed.

**9.80.2.20 #define LDR\_INVALIDSEEK 0x9C00**

Invalid file seek operation.

**9.80.2.21 #define LDR\_MODULENOTFOUND 0x9000**

No modules matched the specified search criteria.

**9.80.2.22 #define LDR\_NOLINEARSPACE 0x8900**

Not enough linear flash memory is available.

**9.80.2.23 #define LDR\_NOMOREFILES 0x8300**

The maximum number of files has been reached.

**9.80.2.24 #define LDR\_NOMOREHANDLES 0x8100**

All available file handles are in use.

**9.80.2.25 #define LDR\_NOSPACE 0x8200**

Not enough free flash memory for the specified file size.

**9.80.2.26 #define LDR\_NOTLINEARFILE 0x8600**

The specified file is not linear.

**9.80.2.27 #define LDR\_NOWRITEBUFFERS 0x8C00**

No more write buffers are available.

**9.80.2.28 #define LDR\_OUTOFBOUNDARY 0x9100**

Specified IOMap offset is outside the bounds of the IOMap.

**9.80.2.29 #define LDR\_REQPIN 0x0002**

A PIN exchange request is in progress.

**9.80.2.30 #define LDR\_SUCCESS 0x0000**

The function completed successfully.

**Examples:**

[ex\\_syscommbtcheckstatus.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_sysfilerename.nxc](#), and [ex\\_sysfileresolvehandle.nxc](#).

**9.80.2.31 #define LDR\_UNDEFINEDERROR 0x8A00**

An undefined error has occurred.

## 9.81 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

### 9.81.1 Detailed Description

Constants defining the functions provided by the Loader module.

### 9.81.2 Define Documentation

#### 9.81.2.1 #define [LDR\\_CMD\\_BOOTCMD](#) 0x97

Reboot the NXT into SAMBA mode

**9.81.2.2 #define LDR\_CMD\_BTFACTORYRESET 0xA4**

Reset bluetooth configuration to factory defaults

**9.81.2.3 #define LDR\_CMD\_BTGETADR 0x9A**

Get the NXT's bluetooth brick address

**9.81.2.4 #define LDR\_CMD\_CLOSE 0x84**

Close a file handle

**9.81.2.5 #define LDR\_CMD\_CLOSEMODHANDLE 0x92**

Close a module handle

**9.81.2.6 #define LDR\_CMD\_CROPDATAFILE 0x8D**

Crop a data file to its used space

**9.81.2.7 #define LDR\_CMD\_DELETE 0x85**

Delete a file

**9.81.2.8 #define LDR\_CMD\_DELETEUSERFLASH 0xA0**

Delete all files from user flash memory

**9.81.2.9 #define LDR\_CMD\_DEVICEINFO 0x9B**

Read device information

**9.81.2.10 #define LDR\_CMD\_FINDFIRST 0x86**

Find the first file matching the specified pattern

**9.81.2.11 #define LDR\_CMD\_FINDFIRSTMODULE 0x90**

Find the first module matching the specified pattern

**9.81.2.12 #define LDR\_CMD\_FINDNEXT 0x87**

Find the next file matching the specified pattern

**9.81.2.13 #define LDR\_CMD\_FINDNEXTMODULE 0x91**

Find the next module matching the specified pattern

**9.81.2.14 #define LDR\_CMD\_IOMAPREAD 0x94**

Read data from a module IOMAP

**9.81.2.15 #define LDR\_CMD\_IOMAPWRITE 0x95**

Write data to a module IOMAP

**9.81.2.16 #define LDR\_CMD\_OPENAPPENDDATA 0x8C**

Open a data file for appending

**9.81.2.17 #define LDR\_CMD\_OPENREAD 0x80**

Open a file for reading

**9.81.2.18 #define LDR\_CMD\_OPENREADLINEAR 0x8A**

Open a linear file for reading

**9.81.2.19 #define LDR\_CMD\_OPENWRITE 0x81**

Open a file for writing

**9.81.2.20 #define LDR\_CMD\_OPENWRITEDATA 0x8B**

Open a data file for writing

**9.81.2.21 #define LDR\_CMD\_OPENWRITELINEAR 0x89**

Open a linear file for writing

**9.81.2.22 #define LDR\_CMD\_POLLCMD 0xA2**

Poll command

**9.81.2.23 #define LDR\_CMD\_POLLCMDLEN 0xA1**

Read poll command length



**9.81.2.24 #define LDR\_CMD\_READ 0x82**

Read from a file

**9.81.2.25 #define LDR\_CMD\_RENAMEFILE 0xA3**

Rename a file

**9.81.2.26 #define LDR\_CMD\_RESIZEDATAFILE 0xD0**

Resize a data file

**9.81.2.27 #define LDR\_CMD\_SEEKFROMCURRENT 0xD2**

Seek from the current position

**9.81.2.28 #define LDR\_CMD\_SEEKFROMEND 0xD3**

Seek from the end of the file

**9.81.2.29 #define LDR\_CMD\_SEEKFROMSTART 0xD1**

Seek from the start of the file

**9.81.2.30 #define LDR\_CMD\_SETBRICKNAME 0x98**

Set the NXT's brick name

**9.81.2.31 #define LDR\_CMD\_VERSIONS 0x88**

Read firmware version information

**9.81.2.32 #define LDR\_CMD\_WRITE 0x83**

Write to a file

## 9.82 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.*

### 9.82.1 Detailed Description

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

## 9.83 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

### 9.83.1 Detailed Description

Constants for use with the [SoundFlags\(\)](#) function.

See also:

[SoundFlags\(\)](#)

### 9.83.2 Define Documentation

#### 9.83.2.1 #define SOUND\_FLAGS\_IDLE 0x00

R - Sound is idle

#### 9.83.2.2 #define SOUND\_FLAGS\_RUNNING 0x02

R - Currently processing a tone or file

#### 9.83.2.3 #define SOUND\_FLAGS\_UPDATE 0x01

W - Make changes take effect

Examples:

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

## 9.84 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`

### 9.84.1 Detailed Description

Constants for use with the [SoundState\(\)](#) function.

See also:

[SoundState\(\)](#)

### 9.84.2 Define Documentation

#### 9.84.2.1 `#define SOUND_STATE_FILE 0x02`

R - Processing a file of sound/melody data

#### 9.84.2.2 `#define SOUND_STATE_IDLE 0x00`

R - Idle, ready for start sound (SOUND\_UPDATE)

Examples:

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

#### 9.84.2.3 `#define SOUND_STATE_STOP 0x04`

W - Stop sound immediately and close hardware

Examples:

[ex\\_SetSoundModuleState.nxc](#), and [ex\\_syssoundsetstate.nxc](#).

#### 9.84.2.4 `#define SOUND_STATE_TONE 0x03`

R - Processing a play tone request

## 9.85 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

### 9.85.1 Detailed Description

Constants for use with the [SoundMode\(\)](#) function.

See also:

[SoundMode\(\)](#)

### 9.85.2 Define Documentation

#### 9.85.2.1 #define SOUND\_MODE\_LOOP 0x01

W - Play file until writing SOUND\_STATE\_STOP into SoundState

#### 9.85.2.2 #define SOUND\_MODE\_ONCE 0x00

W - Only play file once

Examples:

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

#### 9.85.2.3 #define SOUND\_MODE\_TONE 0x02

W - Play tone specified in Frequency for Duration ms

## 9.86 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

### 9.86.1 Detailed Description

Constant offsets into the Sound module IOMAP structure.

### 9.86.2 Define Documentation

#### 9.86.2.1 `#define SoundOffsetDuration` 2

RW - [Tone](#) duration [mS] (2 bytes)

#### 9.86.2.2 `#define SoundOffsetFlags` 26

RW - Play flag - described above (1 byte) [SoundFlags constants](#)

#### 9.86.2.3 `#define SoundOffsetFreq` 0

RW - [Tone](#) frequency [Hz] (2 bytes)

#### 9.86.2.4 `#define SoundOffsetMode` 28

RW - Play mode - described above (1 byte) [SoundMode constants](#)

#### 9.86.2.5 `#define SoundOffsetSampleRate` 4

RW - Sound file sample rate [2000..16000] (2 bytes)

### Examples:

[ex\\_sysiomapwrite.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**9.86.2.6 #define SoundOffsetSoundFilename 6**

RW - Sound/melody filename (20 bytes)

**9.86.2.7 #define SoundOffsetState 27**

RW - Play state - described above (1 byte) [SoundState constants](#)

**9.86.2.8 #define SoundOffsetVolume 29**

RW - Sound/melody volume [0..4] 0 = off (1 byte)

## 9.87 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`

### 9.87.1 Detailed Description

Constants defining miscellaneous sound module aspects.

### 9.87.2 Define Documentation

#### 9.87.2.1 `#define FREQUENCY_MAX 14080`

Maximum frequency [Hz]

#### 9.87.2.2 `#define FREQUENCY_MIN 220`

Minimum frequency [Hz]

#### 9.87.2.3 `#define SAMPLERATE_DEFAULT 8000`

Default sample rate [sps]

#### 9.87.2.4 `#define SAMPLERATE_MAX 16000`

Max sample rate [sps]

#### 9.87.2.5 `#define SAMPLERATE_MIN 2000`

Min sample rate [sps]



## 9.88 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

### 9.88.1 Detailed Description

Constants for use in the `SoundPlayTone()` API function.

See also:

[SoundPlayTone\(\)](#)

### 9.88.2 Define Documentation

#### 9.88.2.1 `#define TONE_A3` 220

Third octave A

#### 9.88.2.2 `#define TONE_A4` 440

Fourth octave A

Examples:

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

#### 9.88.2.3 `#define TONE_A5` 880

Fifth octave A

#### 9.88.2.4 `#define TONE_A6` 1760

Sixth octave A

#### 9.88.2.5 `#define TONE_A7` 3520

Seventh octave A

#### 9.88.2.6 `#define TONE_AS3` 233

Third octave A sharp

**9.88.2.7 #define TONE\_AS4 466**

Fourth octave A sharp

**9.88.2.8 #define TONE\_AS5 932**

Fifth octave A sharp

**9.88.2.9 #define TONE\_AS6 1865**

Sixth octave A sharp

**9.88.2.10 #define TONE\_AS7 3729**

Seventh octave A sharp

**9.88.2.11 #define TONE\_B3 247**

Third octave B

**9.88.2.12 #define TONE\_B4 494**

Fourth octave B

**9.88.2.13 #define TONE\_B5 988**

Fifth octave B

**9.88.2.14 #define TONE\_B6 1976**

Sixth octave B

**9.88.2.15 #define TONE\_B7 3951**

Seventh octave B

**9.88.2.16 #define TONE\_C4 262**

Fourth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**9.88.2.17 #define TONE\_C5 523**

Fifth octave C

**Examples:**

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

**9.88.2.18 #define TONE\_C6 1047**

Sixth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**9.88.2.19 #define TONE\_C7 2093**

Seventh octave C

**9.88.2.20 #define TONE\_CS4 277**

Fourth octave C sharp

**9.88.2.21 #define TONE\_CS5 554**

Fifth octave C sharp

**9.88.2.22 #define TONE\_CS6 1109**

Sixth octave C sharp

**9.88.2.23 #define TONE\_CS7 2217**

Seventh octave C sharp

**9.88.2.24 #define TONE\_D4 294**

Fourth octave D

**9.88.2.25 #define TONE\_D5 587**

Fifth octave D

**9.88.2.26 #define TONE\_D6 1175**

Sixth octave D

**9.88.2.27 #define TONE\_D7 2349**

Seventh octave D

**9.88.2.28 #define TONE\_DS4 311**

Fourth octave D sharp

**9.88.2.29 #define TONE\_DS5 622**

Fifth octave D sharp

**9.88.2.30 #define TONE\_DS6 1245**

Sixth octave D sharp

**9.88.2.31 #define TONE\_DS7 2489**

Seventh octave D sharp

**9.88.2.32 #define TONE\_E4 330**

Fourth octave E

**Examples:**

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

**9.88.2.33 #define TONE\_E5 659**

Fifth octave E

**Examples:**

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

**9.88.2.34 #define TONE\_E6 1319**

Sixth octave E

**9.88.2.35 #define TONE\_E7 2637**

Seventh octave E

**9.88.2.36 #define TONE\_F4 349**

Fourth octave F

**9.88.2.37 #define TONE\_F5 698**

Fifth octave F

**9.88.2.38 #define TONE\_F6 1397**

Sixth octave F

**9.88.2.39 #define TONE\_F7 2794**

Seventh octave F

**9.88.2.40 #define TONE\_FS4 370**

Fourth octave F sharp

**9.88.2.41 #define TONE\_FS5 740**

Fifth octave F sharp

**9.88.2.42 #define TONE\_FS6 1480**

Sixth octave F sharp

**9.88.2.43 #define TONE\_FS7 2960**

Seventh octave F sharp

**9.88.2.44 #define TONE\_G4 392**

Fourth octave G

**Examples:**

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

**9.88.2.45 #define TONE\_G5 784**

Fifth octave G

**Examples:**

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

**9.88.2.46 #define TONE\_G6 1568**

Sixth octave G

**9.88.2.47 #define TONE\_G7 3136**

Seventh octave G

**9.88.2.48 #define TONE\_GS4 415**

Fourth octave G sharp

**9.88.2.49 #define TONE\_GS5 831**

Fifth octave G sharp

**9.88.2.50 #define TONE\_GS6 1661**

Sixth octave G sharp

**9.88.2.51 #define TONE\_GS7 3322**

Seventh octave G sharp

## 9.89 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.*

### 9.89.1 Detailed Description

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



## 9.90 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`

### 9.90.1 Detailed Description

Constants to specify which button to use with button module functions.

See also:

[ButtonPressed\(\)](#), [ButtonState\(\)](#), [ButtonCount\(\)](#), [ReadButtonEx\(\)](#), [SysReadButton\(\)](#), [ReadButtonType](#)

### 9.90.2 Define Documentation

#### 9.90.2.1 `#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](#).

#### 9.90.2.2 `#define BTN2 1`

The right button.

#### 9.90.2.3 `#define BTN3 2`

The left button.

#### 9.90.2.4 `#define BTN4 3`

The enter button.

**9.90.2.5 #define BTNCENTER BTN4**

The enter button.

**Examples:**

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

**9.90.2.6 #define BTNEXIT BTN1**

The exit button.

**Examples:**

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

**9.90.2.7 #define BTNLEFT BTN3**

The left button.

**Examples:**

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

**9.90.2.8 #define BTNRIGHT BTN2**

The right button.

**Examples:**

[ex\\_ButtonPressed.nxc](#), and [ex\\_sysreadbutton.nxc](#).

**9.90.2.9 #define NO\_OF\_BTNS 4**

The number of NXT buttons.

## 9.91 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`

### 9.91.1 Detailed Description

Constants for use with the [ButtonState\(\)](#) function. The `_EV` values can be combined together using a bitwise OR operation.

See also:

[ButtonState\(\)](#)

### 9.91.2 Define Documentation

#### 9.91.2.1 `#define BTNSTATE_LONG_PRESSED_EV 0x04`

Button is in the long pressed state.

Examples:

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

#### 9.91.2.2 `#define BTNSTATE_LONG_RELEASED_EV 0x08`

Button is in the long released state.

#### 9.91.2.3 `#define BTNSTATE_NONE 0x10`

The default button state.

#### 9.91.2.4 `#define BTNSTATE_PRESSED_EV 0x01`

Button is in the pressed state.

Examples:

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

**9.91.2.5 #define BTNSTATE\_PRESSED\_STATE 0x80**

A bitmask for the button pressed state

**9.91.2.6 #define BTNSTATE\_SHORT\_RELEASED\_EV 0x02**

Button is in the short released state.

## 9.92 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)`

### 9.92.1 Detailed Description

Constant offsets into the Button module IOMAP structure.

#### 9.92.2 Define Documentation

##### 9.92.2.1 `#define ButtonOffsetLongPressCnt(b) (((b)*8)+1)`

Offset to the LongPressCnt field. This field stores the long press count.

##### 9.92.2.2 `#define ButtonOffsetLongRelCnt(b) (((b)*8)+3)`

Offset to the LongRelCnt field. This field stores the long release count.

##### 9.92.2.3 `#define ButtonOffsetPressedCnt(b) (((b)*8)+0)`

Offset to the PressedCnt field. This field stores the press count.

##### 9.92.2.4 `#define ButtonOffsetRelCnt(b) (((b)*8)+4)`

Offset to the RelCnt field. This field stores the release count.

##### 9.92.2.5 `#define ButtonOffsetShortRelCnt(b) (((b)*8)+2)`

Offset to the ShortRelCnt field. This field stores the short release count.

##### 9.92.2.6 `#define ButtonOffsetState(b) ((b)+32)`

Offset to the State field. This field stores the current button state.

## 9.93 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.*

### 9.93.1 Detailed Description

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

## 9.94 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`

### 9.94.1 Detailed Description

Constants for use with the [CommandFlags\(\)](#) function.

See also:

[CommandFlags\(\)](#)

### 9.94.2 Define Documentation

#### 9.94.2.1 `#define UI_FLAGS_BUSY 0x40`

R - UI busy running or datalogging (popup disabled)

#### 9.94.2.2 `#define UI_FLAGS_DISABLE_EXIT 0x04`

RW - Disable exit button

#### 9.94.2.3 `#define UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER 0x02`

RW - Disable left, right and enter button

#### 9.94.2.4 `#define UI_FLAGS_ENABLE_STATUS_UPDATE 0x80`

W - Enable status line to be updated

#### 9.94.2.5 `#define UI_FLAGS_EXECUTE_LMS_FILE 0x20`

W - Execute LMS file in "LMSfilename" (Try It)

**9.94.2.6 #define UI\_FLAGS\_REDRAW\_STATUS 0x08**

W - Redraw entire status line

**Examples:**

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

**9.94.2.7 #define UI\_FLAGS\_RESET\_SLEEP\_TIMER 0x10**

W - Reset sleep timeout timer

**9.94.2.8 #define UI\_FLAGS\_UPDATE 0x01**

W - Make changes take effect



## 9.95 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`

### 9.95.1 Detailed Description

Constants for use with the [UIState\(\)](#) function.

See also:

[UIState\(\)](#)

### 9.95.2 Define Documentation

#### 9.95.2.1 `#define UI_STATE_BT_ERROR 16`

R - BT error

#### 9.95.2.2 `#define UI_STATE_CONNECT_REQUEST 12`

RW - Request for connection accept

#### 9.95.2.3 `#define UI_STATE_DRAW_MENU 6`

RW - Execute function and draw menu icons

#### 9.95.2.4 `#define UI_STATE_ENTER_PRESSED 10`

RW - Load selected function and next menu id

**9.95.2.5 #define UI\_STATE\_EXECUTE\_FILE 13**

RW - Execute file in "LMSfilename"

**9.95.2.6 #define UI\_STATE\_EXECUTING\_FILE 14**

R - Executing file in "LMSfilename"

**9.95.2.7 #define UI\_STATE\_EXIT\_PRESSED 11**

RW - Load selected function and next menu id

**9.95.2.8 #define UI\_STATE\_INIT\_DISPLAY 0**

RW - Init display and load font, menu etc.

**9.95.2.9 #define UI\_STATE\_INIT\_INTRO 2**

R - Display intro

**9.95.2.10 #define UI\_STATE\_INIT\_LOW\_BATTERY 1**

R - Low battery voltage at power on

**9.95.2.11 #define UI\_STATE\_INIT\_MENU 4**

RW - Init menu system

**9.95.2.12 #define UI\_STATE\_INIT\_WAIT 3**

RW - Wait for initialization end

**9.95.2.13 #define UI\_STATE\_LEFT\_PRESSED 8**

RW - Load selected function and next menu id

**9.95.2.14 #define UI\_STATE\_LOW\_BATTERY 15**

R - Low battery at runtime

**Examples:**

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

**9.95.2.15 #define UI\_STATE\_NEXT\_MENU 5**

RW - Next menu icons ready for drawing

**9.95.2.16 #define UI\_STATE\_RIGHT\_PRESSED 9**

RW - Load selected function and next menu id

**9.95.2.17 #define UI\_STATE\_TEST\_BUTTONS 7**

RW - Wait for buttons to be pressed

## 9.96 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`

### 9.96.1 Detailed Description

Constants for use with the [UIButton\(\)](#) function.

See also:

[UIButton\(\)](#)

### 9.96.2 Define Documentation

#### 9.96.2.1 `#define UI_BUTTON_ENTER 2`

W - Insert enter button

Examples:

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

#### 9.96.2.2 `#define UI_BUTTON_EXIT 4`

W - Insert exit button

#### 9.96.2.3 `#define UI_BUTTON_LEFT 1`

W - Insert left arrow button

#### 9.96.2.4 `#define UI_BUTTON_NONE 0`

R - Button inserted are executed

#### 9.96.2.5 `#define UI_BUTTON_RIGHT 3`

W - Insert right arrow button

## 9.97 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`

### 9.97.1 Detailed Description

Constants for use with the [BluetoothState\(\)](#) function.

See also:

[BluetoothState\(\)](#)

### 9.97.2 Define Documentation

#### 9.97.2.1 `#define UI_BT_CONNECT_REQUEST 0x40`

RW - BT get connect accept in progress

#### 9.97.2.2 `#define UI_BT_ERROR_ATTENTION 0x08`

W - BT error attention

#### 9.97.2.3 `#define UI_BT_PIN_REQUEST 0x80`

RW - BT get pin code

#### 9.97.2.4 `#define UI_BT_STATE_CONNECTED 0x02`

RW - BT connected to something

#### 9.97.2.5 `#define UI_BT_STATE_OFF 0x04`

RW - BT power off

Examples:

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

**9.97.2.6 #define UI\_BT\_STATE\_VISIBLE 0x01**

RW - BT visible

## 9.98 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`

### 9.98.1 Detailed Description

Constants for use with the [VMRunState\(\)](#) function.

See also:

[VMRunState\(\)](#)

### 9.98.2 Define Documentation

#### 9.98.2.1 `#define UI_VM_IDLE 0`

VM\_IDLE: Just sitting around. Request to run program will lead to ONE of the VM\_RUN\* states.

#### 9.98.2.2 `#define UI_VM_RESET1 4`

VM\_RESET2: Final clean up and return to IDLE

#### 9.98.2.3 `#define UI_VM_RESET2 5`

VM\_RESET1: Initialize state variables and some I/O devices -- executed when programs end

#### 9.98.2.4 `#define UI_VM_RUN_FREE 1`

VM\_RUN\_FREE: Attempt to run as many instructions as possible within our timeslice

#### 9.98.2.5 `#define UI_VM_RUN_PAUSE 3`

VM\_RUN\_PAUSE: Program still "active", but someone has asked us to pause

#### 9.98.2.6 `#define UI_VM_RUN_SINGLE 2`

VM\_RUN\_SINGLE: Run exactly one instruction per timeslice

## 9.99 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`

### 9.99.1 Detailed Description

Constant offsets into the Ui module IOMAP structure.

### 9.99.2 Define Documentation

#### 9.99.2.1 `#define UIOffsetAbortFlag 40`

RW - Long Abort (true == use long press to abort) (1 byte)

#### 9.99.2.2 `#define UIOffsetBatteryState 30`

W - Battery state (0..4 capacity) (1 byte)

#### 9.99.2.3 `#define UIOffsetBatteryVoltage 4`

R - Battery voltage in millivolts (2 bytes)

#### 9.99.2.4 `#define UIOffsetBluetoothState 31`

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)



**9.99.2.5 #define UIOffsetButton 28**

RW - Insert button (buttons enumerated above) (1 byte)

**9.99.2.6 #define UIOffsetError 37**

W - Error code (1 byte)

**9.99.2.7 #define UIOffsetFlags 26**

RW - Update command flags (flags enumerated above) (1 byte)

**9.99.2.8 #define UIOffsetForceOff 39**

W - Force off (> 0 = off) (1 byte)

**9.99.2.9 #define UIOffsetLMSfilename 6**

W - LMS filename to execute (Try It) (20 bytes)

**9.99.2.10 #define UIOffsetOBPPointer 38**

W - Actual OBP step (0 - 4) (1 byte)

**9.99.2.11 #define UIOffsetPMenu 0**

W - Pointer to menu file (4 bytes)

**9.99.2.12 #define UIOffsetRechargeable 35**

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

**9.99.2.13 #define UIOffsetRunState 29**

W - VM Run state (0 = stopped, 1 = running) (1 byte)

**9.99.2.14 #define UIOffsetSleepTimeout 33**

RW - Sleep timeout time (min) (1 byte)

**9.99.2.15 #define UIOffsetSleepTimer 34**

RW - Sleep timer (min) (1 byte)

**9.99.2.16 #define UIOffsetState 27**

RW - UI state (states enumerated above) (1 byte)

**9.99.2.17 #define UIOffsetUsbState 32**

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

**9.99.2.18 #define UIOffsetVolume 36**

RW - Volume used in UI (0 - 4) (1 byte)

## 9.100 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`

### 9.100.1 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](#)

### 9.100.2 Define Documentation

#### 9.100.2.1 `#define IN_1 0x00`

Input port 1

#### 9.100.2.2 `#define IN_2 0x01`

Input port 2

#### 9.100.2.3 `#define IN_3 0x02`

Input port 3

#### 9.100.2.4 `#define IN_4 0x03`

Input port 4

## 9.101 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

### 9.101.1 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\(\)](#)

### 9.101.2 Define Documentation

#### 9.101.2.1 #define IN\_TYPE\_ANGLE 0x04

RCX rotation sensor

#### 9.101.2.2 #define IN\_TYPE\_COLORBLUE 0x10

NXT 2.0 color sensor with blue light

#### 9.101.2.3 #define IN\_TYPE\_COLOREXIT 0x12

NXT 2.0 color sensor internal state

**9.101.2.4 #define IN\_TYPE\_COLORFULL 0x0D**

NXT 2.0 color sensor in full color mode

**9.101.2.5 #define IN\_TYPE\_COLORGREEN 0x0F**

NXT 2.0 color sensor with green light

**9.101.2.6 #define IN\_TYPE\_COLORNONE 0x11**

NXT 2.0 color sensor with no light

**9.101.2.7 #define IN\_TYPE\_COLORRED 0x0E**

NXT 2.0 color sensor with red light

**9.101.2.8 #define IN\_TYPE\_CUSTOM 0x09**

NXT custom sensor

**9.101.2.9 #define IN\_TYPE\_HISPEED 0x0C**

NXT Hi-speed port (only S4)

**9.101.2.10 #define IN\_TYPE\_LIGHT\_ACTIVE 0x05**

NXT light sensor with light

**9.101.2.11 #define IN\_TYPE\_LIGHT\_INACTIVE 0x06**

NXT light sensor without light

**9.101.2.12 #define IN\_TYPE\_LOWSPEED 0x0A**

NXT I2C digital sensor

**9.101.2.13 #define IN\_TYPE\_LOWSPEED\_9V 0x0B**

NXT I2C digital sensor with 9V power

**9.101.2.14 #define IN\_TYPE\_NO\_SENSOR 0x00**

No sensor configured

**9.101.2.15 #define IN\_TYPE\_REFLECTION 0x03**

RCX light sensor

**9.101.2.16 #define IN\_TYPE\_SOUND\_DB 0x07**

NXT sound sensor with dB scaling

**9.101.2.17 #define IN\_TYPE\_SOUND\_DBA 0x08**

NXT sound sensor with dBA scaling

**9.101.2.18 #define IN\_TYPE\_SWITCH 0x01**

NXT or RCX touch sensor

**9.101.2.19 #define IN\_TYPE\_TEMPERATURE 0x02**

RCX temperature sensor

## 9.102 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`

### 9.102.1 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\(\)](#)

### 9.102.2 Define Documentation

#### 9.102.2.1 `#define IN_MODE_ANGLESTEP 0xE0`

RCX rotation sensor (16 ticks per revolution)

#### 9.102.2.2 `#define IN_MODE_BOOLEAN 0x20`

Boolean value (0 or 1)

#### 9.102.2.3 `#define IN_MODE_CELSIUS 0xA0`

RCX temperature sensor value in degrees celcius

#### 9.102.2.4 `#define IN_MODE_FAHRENHEIT 0xC0`

RCX temperature sensor value in degrees fahrenheit

#### 9.102.2.5 `#define IN_MODE_MODEMASK 0xE0`

Mask for the mode without any slope value

**9.102.2.6 #define IN\_MODE\_PCTFULLSCALE 0x80**

Scaled value from 0 to 100

**9.102.2.7 #define IN\_MODE\_PERIODCOUNTER 0x60**

Counts the number of boolean periods

**9.102.2.8 #define IN\_MODE\_RAW 0x00**

Raw value from 0 to 1023

**9.102.2.9 #define IN\_MODE\_SLOPEMASK 0x1F**

Mask for slope parameter added to mode

**9.102.2.10 #define IN\_MODE\_TRANSITIONCNT 0x40**

Counts the number of boolean transitions



## 9.103 Input field constants

Constants for use with [SetInput\(\)](#) and [GetInput\(\)](#).

### Defines

- `#define Type 0`
- `#define InputMode 1`
- `#define RawValue 2`
- `#define NormalizedValue 3`
- `#define ScaledValue 4`
- `#define InvalidData 5`

### 9.103.1 Detailed Description

Constants for use with [SetInput\(\)](#) and [GetInput\(\)](#). Each sensor has six fields that are used to define its state. The field constants are described in the table above.

### 9.103.2 Define Documentation

#### 9.103.2.1 `#define InputMode 1`

Input mode field. Contains one of the sensor mode constants. Read/write.

#### 9.103.2.2 `#define InvalidData 5`

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

#### 9.103.2.3 `#define NormalizedValue 3`

Normalized value field. Contains the current normalized analog sensor value. Read only.

#### 9.103.2.4 `#define RawValue 2`

Raw value field. Contains the current raw analog sensor value. Read only.

#### 9.103.2.5 `#define ScaledValue 4`

Scaled value field. Contains the current scaled analog sensor value. Read/write.

#### 9.103.2.6 `#define Type 0`

Type field. Contains one of the sensor type constants. Read/write.

### Examples:

[ex\\_GetInput.nxc](#), and [ex\\_SetInput.nxc](#).

## 9.104 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`

### 9.104.1 Detailed Description

Constants for use with color sensor value arrays to index RGB and blank return values.

See also:

[ReadSensorColorEx\(\)](#), [ReadSensorColorRaw\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

### 9.104.2 Define Documentation

#### 9.104.2.1 `#define INPUT_BLANK 3`

Access the blank value from color sensor value arrays

#### 9.104.2.2 `#define INPUT_BLUE 2`

Access the blue value from color sensor value arrays

#### 9.104.2.3 `#define INPUT_GREEN 1`

Access the green value from color sensor value arrays

#### 9.104.2.4 `#define INPUT_NO_OF_COLORS 4`

The number of entries in the color sensor value arrays

#### 9.104.2.5 `#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](#).

## 9.105 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

### 9.105.1 Detailed Description

Constants for use with the ColorValue returned by the color sensor in full color mode.

See also:

[SensorValue\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

### 9.105.2 Define Documentation

#### 9.105.2.1 #define INPUT\_BLACKCOLOR 1

The color value is black

#### 9.105.2.2 #define INPUT\_BLUECOLOR 2

The color value is blue

#### 9.105.2.3 #define INPUT\_GREENCOLOR 3

The color value is green

#### 9.105.2.4 #define INPUT\_REDCOLOR 5

The color value is red

#### 9.105.2.5 #define INPUT\_WHITECOLOR 6

The color value is white

#### 9.105.2.6 #define INPUT\_YELLOWCOLOR 4

The color value is yellow

## 9.106 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

### 9.106.1 Detailed Description

Constants for use with the color calibration state function.

See also:

[ColorCalibrationState\(\)](#)

### 9.106.2 Define Documentation

#### 9.106.2.1 #define INPUT\_RESETCAL 0x80

Unused calibration state constant

#### 9.106.2.2 #define INPUT\_RUNNINGCAL 0x20

Unused calibration state constant

#### 9.106.2.3 #define INPUT\_SENSORCAL 0x01

The state returned while the color sensor is calibrating

#### 9.106.2.4 #define INPUT\_SENSOROFF 0x02

The state returned once calibration has completed

#### 9.106.2.5 #define INPUT\_STARTCAL 0x40

Unused calibration state constant

## 9.107 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`

### 9.107.1 Detailed Description

Constants for use with the color calibration functions.

See also:

[ColorCalibration\(\)](#), [ColorCalLimits\(\)](#)

### 9.107.2 Define Documentation

#### 9.107.2.1 `#define INPUT_CAL_POINT_0 0`

Calibration point 0

Examples:

[ex\\_ColorCalibration.nxc](#), and [ex\\_ColorCalLimits.nxc](#).

#### 9.107.2.2 `#define INPUT_CAL_POINT_1 1`

Calibration point 1

#### 9.107.2.3 `#define INPUT_CAL_POINT_2 2`

Calibration point 2

#### 9.107.2.4 `#define INPUT_NO_OF_POINTS 3`

The number of calibration points

## 9.108 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 InputOffsetColorSensorBoolean(p, nc) (80+((p)*84)+76+((nc)*2))`
- `#define InputOffsetColorCalibrationState(p) (80+((p)*84)+80)`

### 9.108.1 Detailed Description

Constant offsets into the Input module IOMAP structure.

### 9.108.2 Define Documentation

#### 9.108.2.1 `#define InputOffsetADRaw(p) (((p)*20)+2)`

Read the AD raw sensor value (2 bytes) uword

#### 9.108.2.2 `#define InputOffsetColorADRaw(p, nc) (80+((p)*84)+52+((nc)*2))`

Read AD raw color sensor values

#### 9.108.2.3 `#define InputOffsetColorCalibration(p, np, nc) (80+((p)*84)+0+((np)*16)+((nc)*4))`

Read/write color calibration point values

#### 9.108.2.4 `#define InputOffsetColorCalibrationState(p) (80+((p)*84)+80)`

Read color sensor calibration state

**9.108.2.5 #define InputOffsetColorCalLimits(p, np) (80+((p)\*84)+48+((np)\*2))**

Read/write color calibration limits

**9.108.2.6 #define InputOffsetColorSensorBoolean(p, nc) (80+((p)\*84)+76+((nc)\*2))**

Read color sensor boolean values

**9.108.2.7 #define InputOffsetColorSensorRaw(p, nc) (80+((p)\*84)+60+((nc)\*2))**

Read raw color sensor values

**9.108.2.8 #define InputOffsetColorSensorValue(p, nc) (80+((p)\*84)+68+((nc)\*2))**

Read scaled color sensor values

**9.108.2.9 #define InputOffsetCustomActiveStatus(p) (((p)\*20)+15)**

Read/write the active or inactive state of the custom sensor

**9.108.2.10 #define InputOffsetCustomPctFullScale(p) (((p)\*20)+14)**

Read/write the Pct full scale of the custom sensor

**9.108.2.11 #define InputOffsetCustomZeroOffset(p) (((p)\*20)+0)**

Read/write the zero offset of a custom sensor (2 bytes) uword

**9.108.2.12 #define InputOffsetDigiPinsDir(p) (((p)\*20)+11)**

Read/write the direction of the Digital pins (1 is output, 0 is input)

**9.108.2.13 #define InputOffsetDigiPinsIn(p) (((p)\*20)+12)**

Read/write the status of the digital pins

**9.108.2.14 #define InputOffsetDigiPinsOut(p) (((p)\*20)+13)**

Read/write the output level of the digital pins

**9.108.2.15 #define InputOffsetInvalidData(p) (((p)\*20)+16)**

Indicates whether data is invalid (1) or valid (0)

**9.108.2.16 #define InputOffsetSensorBoolean(p) (((p)\*20)+10)**

Read the sensor boolean value

**9.108.2.17 #define InputOffsetSensorMode(p) (((p)\*20)+9)**

Read/write the sensor mode

**9.108.2.18 #define InputOffsetSensorRaw(p) (((p)\*20)+4)**

Read the raw sensor value (2 bytes) uword

**9.108.2.19 #define InputOffsetSensorType(p) (((p)\*20)+8)**

Read/write the sensor type

**9.108.2.20 #define InputOffsetSensorValue(p) (((p)\*20)+6)**

Read/write the scaled sensor value (2 bytes) sword



## 9.109 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`

### 9.109.1 Detailed Description

Output port constants are used when calling motor control API functions.

### 9.109.2 Define Documentation

#### 9.109.2.1 `#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_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_RemoteResetMotorPosition.nxc`, `ex_RemoteSetOutputState.nxc`, `ex_rotatemotor.nxc`, `ex_rotatemotorpid.nxc`, and `ex_yield.nxc`.

#### 9.109.2.2 `#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_onrevsyncexpid.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`.

#### 9.109.2.3 `#define OUT_ABC 0x06`

Output ports A, B, and C

**9.109.2.4 #define OUT\_AC 0x04**

Output ports A and C

**9.109.2.5 #define OUT\_B 0x01**

Output port B

**9.109.2.6 #define OUT\_BC 0x05**

Output ports B and C

**9.109.2.7 #define OUT\_C 0x02**

Output port C

## 9.110 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`

### 9.110.1 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\(\)](#)

### 9.110.2 Define Documentation

#### 9.110.2.1 `#define PID_0 0`

PID zero

#### 9.110.2.2 `#define PID_1 32`

PID one

#### 9.110.2.3 `#define PID_2 64`

PID two

#### 9.110.2.4 `#define PID_3 96`

PID three

#### 9.110.2.5 `#define PID_4 128`

PID four

**9.110.2.6 #define PID\_5 160**

PID five

**9.110.2.7 #define PID\_6 192**

PID six

**9.110.2.8 #define PID\_7 224**

PID seven

## 9.111 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`

### 9.111.1 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\(\)](#)

### 9.111.2 Define Documentation

#### 9.111.2.1 `#define UF_PENDING_UPDATES 0x80`

Are there any pending motor updates?

#### 9.111.2.2 `#define UF_UPDATE_MODE 0x01`

Commits changes to the OutputMode output property

#### 9.111.2.3 `#define UF_UPDATE_PID_VALUES 0x10`

Commits changes to the PID motor regulation properties

#### 9.111.2.4 `#define UF_UPDATE_RESET_BLOCK_COUNT 0x20`

Resets the NXT-G block-relative rotation counter

#### 9.111.2.5 `#define UF_UPDATE_RESET_COUNT 0x08`

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

**9.111.2.6 #define UF\_UPDATE\_RESET\_ROTATION\_COUNT 0x40**

Resets the program-relative (user) rotation counter

**9.111.2.7 #define UF\_UPDATE\_SPEED 0x02**

Commits changes to the Power output property

**9.111.2.8 #define UF\_UPDATE\_TACHO\_LIMIT 0x04**

Commits changes to the TachoLimit output property

## 9.112 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`

### 9.112.1 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...

### 9.112.2 Define Documentation

#### 9.112.2.1 `#define RESET_ALL 0x68`

Reset all three tachometer counters

#### 9.112.2.2 `#define RESET_BLOCK_COUNT 0x20`

Reset the NXT-G block tachometer counter

#### 9.112.2.3 `#define RESET_BLOCKANDTACHO 0x28`

Reset both the internal counter and the NXT-G block counter

#### 9.112.2.4 `#define RESET_COUNT 0x08`

Reset the internal tachometer counter

#### 9.112.2.5 `#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\\_onfwdsyncex.nxc](#), [ex\\_onfwdsyncexpid.nxc](#), [ex\\_onrevex.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), [ex\\_onrevsyncex.nxc](#), and [ex\\_onrevsyncexpid.nxc](#).

#### 9.112.2.6 `#define RESET_ROTATION_COUNT 0x40`

Reset the rotation counter



## 9.113 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`

### 9.113.1 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\(\)](#)

### 9.113.2 Define Documentation

#### 9.113.2.1 `#define OUT_MODE_BRAKE 0x02`

Uses electronic braking to outputs

#### 9.113.2.2 `#define OUT_MODE_COAST 0x00`

No power and no braking so motors rotate freely.

#### 9.113.2.3 `#define OUT_MODE_MOTORON 0x01`

Enables PWM power to the outputs given the Power setting

Examples:

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

#### 9.113.2.4 `#define OUT_MODE_REGMETHOD 0xF0`

Mask for unimplemented regulation mode

#### 9.113.2.5 `#define OUT_MODE_REGULATED 0x04`

Enables active power regulation using the RegMode value

## 9.114 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_RAMPDOWNTOLIMIT 0x20`

### 9.114.1 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\(\)](#)

### 9.114.2 Define Documentation

#### 9.114.2.1 `#define OUT_OPTION_HOLDATLIMIT 0x10`

Option to have the firmware hold the motor when it reaches the tachometer limit

#### 9.114.2.2 `#define OUT_OPTION_RAMPDOWNTOLIMIT 0x20`

Option to have the firmware rampdown the motor power as it approaches the tachometer limit (not implemented yet)

## 9.115 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`

### 9.115.1 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\(\)](#)

### 9.115.2 Define Documentation

#### 9.115.2.1 `#define OUT_RUNSTATE_HOLD 0x60`

Set motor run state to hold at the current position.

#### 9.115.2.2 `#define OUT_RUNSTATE_IDLE 0x00`

Disable all power to motors.

#### 9.115.2.3 `#define OUT_RUNSTATE_RAMPDOWN 0x40`

Enable ramping down from a current Power to a new (lower) Power over a specified TachoLimit goal.

#### 9.115.2.4 `#define OUT_RUNSTATE_RAMPUP 0x10`

Enable ramping up from a current Power to a new (higher) Power over a specified TachoLimit goal.

#### 9.115.2.5 `#define OUT_RUNSTATE_RUNNING 0x20`

Enable power to motors at the specified Power level.

Examples:

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

## 9.116 Output port regulation mode constants

Use these constants to configure the desired regulation mode for the specified motor(s): none, speed regulation, or multi-motor synchronization.

### Defines

- `#define OUT_REGMODE_IDLE 0`
- `#define OUT_REGMODE_SPEED 1`
- `#define OUT_REGMODE_SYNC 2`

### 9.116.1 Detailed Description

Use these constants to configure the desired regulation mode for the specified motor(s): none, speed regulation, or multi-motor synchronization.

See also:

[SetOutput\(\)](#)

### 9.116.2 Define Documentation

#### 9.116.2.1 `#define OUT_REGMODE_IDLE 0`

No motor regulation.

Examples:

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

#### 9.116.2.2 `#define OUT_REGMODE_SPEED 1`

Regulate a motor's speed (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](#).

#### 9.116.2.3 `#define OUT_REGMODE_SYNC 2`

Synchronize the rotation of two motors.

## 9.117 Output field constants

Constants for use with [SetOutput\(\)](#) and [GetOutput\(\)](#).

### Defines

- `#define UpdateFlags 0`  
*Update flags field.*
- `#define OutputMode 1`  
*Mode field.*
- `#define Power 2`  
*Power field.*
- `#define ActualSpeed 3`  
*Actual speed field.*
- `#define TachoCount 4`  
*Internal tachometer count field.*
- `#define TachoLimit 5`  
*Tachometer limit field.*
- `#define RunState 6`  
*Run state field.*
- `#define TurnRatio 7`  
*Turn ratio field.*
- `#define RegMode 8`  
*Regulation mode field.*
- `#define Overload 9`  
*Overload field.*
- `#define RegPValue 10`  
*Proportional field.*
- `#define RegIValue 11`  
*Integral field.*
- `#define RegDValue 12`  
*Derivative field.*
- `#define BlockTachoCount 13`  
*NXT-G block tachometer count field.*
- `#define RotationCount 14`

*Rotation counter field.*

- `#define OutputOptions 15`

*Options field.*

### 9.117.1 Detailed Description

Constants for use with [SetOutput\(\)](#) and [GetOutput\(\)](#).

See also:

[SetOutput\(\)](#), [GetOutput\(\)](#)

### 9.117.2 Define Documentation

#### 9.117.2.1 `#define ActualSpeed 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 Power value when auto-regulation code in the firmware responds to a load on the output.

#### 9.117.2.2 `#define BlockTachoCount 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 [UpdateFlags](#) description for information about how to use block-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) flag in [UpdateFlags](#) to request that the firmware reset the [BlockTachoCount](#). The sign of [BlockTachoCount](#) 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.

#### 9.117.2.3 `#define OutputMode 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 [OutputMode](#) value. Use [UF\\_UPDATE\\_MODE](#) with [UpdateFlags](#) to commit changes to this field.

#### 9.117.2.4 `#define OutputOptions 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 (not yet implemented).

#### 9.117.2.5 #define Overload 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 [RunState](#), an [OutputMode](#) which includes [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), and its [RegMode](#) must be set to [OUT\\_REGMODE\\_SPEED](#).

#### 9.117.2.6 #define Power 2

Power field. Contains the desired power level (-100 to 100). Read/write. Specify the power level of the output. The absolute value of Power is a percentage of the full power of the motor. The sign of Power 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 [UpdateFlags](#) to commit changes to this field.

#### 9.117.2.7 #define RegDValue 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.

#### 9.117.2.8 #define RegIValue 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.

#### 9.117.2.9 #define RegMode 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 [OutputMode](#) field. Unlike the [OutputMode](#) field, [RegMode](#) is not a bitfield. Only one [RegMode](#) value can be set at a time. Speed regulation means that the firmware tries to maintain a certain speed based on the [Power](#) 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 [ActualSpeed](#) property. When using speed regulation, do not set [Power](#) 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 synch regardless of physical loads. Use this mode to maintain a straight path for a mobile robot automatically. Also use this mode with the [TurnRatio](#) 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.

#### 9.117.2.10 #define RegPValue 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 al-

gorithm. Set [UF\\_UPDATE\\_PID\\_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

#### 9.117.2.11 #define RotationCount 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 [UpdateFlags](#) description for information about how to use program-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) flag in [UpdateFlags](#) to request that the firmware reset the RotationCount. The sign of RotationCount 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.

#### 9.117.2.12 #define RunState 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 RunState to [OUT\\_RUNSTATE\\_RUNNING](#) to enable power to any output. Use [OUT\\_RUNSTATE\\_RAMPUP](#) to enable automatic ramping to a new [Power](#) level greater than the current [Power](#) level. Use [OUT\\_RUNSTATE\\_RAMPDOWN](#) to enable automatic ramping to a new [Power](#) level less than the current [Power](#) level. Both the rampup and rampdown bits must be used in conjunction with appropriate [TachoLimit](#) and [Power](#) values. In this case the firmware smoothly increases or decreases the actual power to the new [Power](#) level over the total number of degrees of rotation specified in [TachoLimit](#).

#### 9.117.2.13 #define TachoCount 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 [TachoLimit](#) and the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag. Set the [UF\\_UPDATE\\_RESET\\_COUNT](#) flag in [UpdateFlags](#) to reset TachoCount and cancel any [TachoLimit](#). The sign of TachoCount indicates the motor rotation direction.

#### 9.117.2.14 #define TachoLimit 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 [UpdateFlags](#) field to commit changes to the TachoLimit. 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.

#### Examples:

[ex\\_getoutput.nxc](#), and [ex\\_setoutput.nxc](#).

#### 9.117.2.15 #define TurnRatio 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: [OutputMode](#) must include [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), [RegMode](#) must be set to [OUT\\_REGMODE\\_SYNC](#), [RunState](#) must not be [OUT\\_RUNSTATE\\_IDLE](#), and [Power](#) must be non-zero. There are only three valid combinations of left and right motors for use with TurnRatio: [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 TurnRatio 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.

#### 9.117.2.16 #define UpdateFlags 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.

## 9.118 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 OutputOffsetPwnFreq 96`

### 9.118.1 Detailed Description

Constant offsets into the Output module IOMAP structure.

### 9.118.2 Define Documentation

#### 9.118.2.1 `#define OutputOffsetActualSpeed(p) (((p)*32)+21)`

R - Holds the current motor speed (1 byte) sbyte

#### 9.118.2.2 `#define OutputOffsetBlockTachoCount(p) (((p)*32)+4)`

R - Holds current number of counts for the current output block (4 bytes) slong

#### 9.118.2.3 `#define OutputOffsetFlags(p) (((p)*32)+18)`

RW - Holds flags for which data should be updated (1 byte) ubyte

#### 9.118.2.4 `#define OutputOffsetMode(p) (((p)*32)+19)`

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

**9.118.2.5 #define OutputOffsetMotorRPM(p) (((p)\*32)+16)**

Not updated, will be removed later !! (2 bytes) sword

**9.118.2.6 #define OutputOffsetOptions(p) (((p)\*32)+29)**

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

**9.118.2.7 #define OutputOffsetOverloaded(p) (((p)\*32)+27)**

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

**9.118.2.8 #define OutputOffsetPwnFreq 96**

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

**9.118.2.9 #define OutputOffsetRegDParameter(p) (((p)\*32)+24)**

RW - Holds the D-constant used in the regulation (1 byte) ubyte

**9.118.2.10 #define OutputOffsetRegIParameter(p) (((p)\*32)+23)**

RW - Holds the I-constant used in the regulation (1 byte) ubyte

**9.118.2.11 #define OutputOffsetRegMode(p) (((p)\*32)+26)**

RW - Tells which regulation mode should be used (1 byte) ubyte

**9.118.2.12 #define OutputOffsetRegPParameter(p) (((p)\*32)+22)**

RW - Holds the P-constant used in the regulation (1 byte) ubyte

**9.118.2.13 #define OutputOffsetRotationCount(p) (((p)\*32)+8)**

R - Holds current number of counts for the rotation counter to the output (4 bytes) slong

**9.118.2.14 #define OutputOffsetRunState(p) (((p)\*32)+25)**

RW - Holds the current RunState in the output module (1 byte) ubyte

**9.118.2.15 #define OutputOffsetSpeed(p) (((p)\*32)+20)**

RW - Holds the wanted speed (1 byte) sbyte

**9.118.2.16 #define OutputOffsetSyncTurnParameter(p) (((p)\*32)+28)**

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

**9.118.2.17 #define OutputOffsetTachoCount(p) (((p)\*32)+0)**

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes) slong

**9.118.2.18 #define OutputOffsetTachoLimit(p) (((p)\*32)+12)**

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

## 9.119 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.*

- [Ultrasonic sensor constants](#)

*Constants for use with the ultrasonic sensor.*

- [Temperature sensor constants](#)

*Constants for use with the temperature sensor.*

### 9.119.1 Detailed Description

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

## 9.120 LSSState constants

Constants for the low speed module LSSState 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`

### 9.120.1 Detailed Description

Constants for the low speed module LSSState function. These values are combined together using a bitwise OR operation.

See also:

[LSSState\(\)](#)

### 9.120.2 Define Documentation

#### 9.120.2.1 `#define COM_CHANNEL_FOUR_ACTIVE 0x08`

Low speed channel 4 is active

#### 9.120.2.2 `#define COM_CHANNEL_NONE_ACTIVE 0x00`

None of the low speed channels are active

#### 9.120.2.3 `#define COM_CHANNEL_ONE_ACTIVE 0x01`

Low speed channel 1 is active

#### 9.120.2.4 `#define COM_CHANNEL_THREE_ACTIVE 0x04`

Low speed channel 3 is active

#### 9.120.2.5 `#define COM_CHANNEL_TWO_ACTIVE 0x02`

Low speed channel 2 is active

## 9.121 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`

### 9.121.1 Detailed Description

Constants for the low speed module LSChannelState function.

See also:

[LSChannelState\(\)](#)

### 9.121.2 Define Documentation

#### 9.121.2.1 `#define LOWSPEED_COMMUNICATING 3`

Channel is actively communicating

#### 9.121.2.2 `#define LOWSPEED_DONE 5`

Channel is done communicating

#### 9.121.2.3 `#define LOWSPEED_ERROR 4`

Channel is in an error state

#### 9.121.2.4 `#define LOWSPEED_IDLE 0`

Channel is idle

Examples:

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

#### 9.121.2.5 `#define LOWSPEED_INIT 1`

Channel is being initialized

**9.121.2.6 #define LOWSPEED\_LOAD\_BUFFER 2**

Channel buffer is loading



## 9.122 LSMoDe constants

Constants for the low speed module LSMoDe function.

### Defines

- #define [LOWSPEED\\_TRANSMITTING](#) 1
- #define [LOWSPEED\\_RECEIVING](#) 2
- #define [LOWSPEED\\_DATA\\_RECEIVED](#) 3

### 9.122.1 Detailed Description

Constants for the low speed module LSMoDe function.

See also:

[LSMoDe\(\)](#)

### 9.122.2 Define Documentation

#### 9.122.2.1 #define LOWSPEED\_DATA\_RECEIVED 3

Lowspeed port is in data received mode

#### 9.122.2.2 #define LOWSPEED\_RECEIVING 2

Lowspeed port is in receiving mode

#### 9.122.2.3 #define LOWSPEED\_TRANSMITTING 1

Lowspeed port is in transmitting mode

## 9.123 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

### 9.123.1 Detailed Description

Constants for the low speed module LSErrorType function.

See also:

[LSErrorType\(\)](#)

### 9.123.2 Define Documentation

#### 9.123.2.1 #define LOWSPEED\_CH\_NOT\_READY 1

Lowspeed port is not ready

#### 9.123.2.2 #define LOWSPEED\_NO\_ERROR 0

Lowspeed port has no error

#### 9.123.2.3 #define LOWSPEED\_RX\_ERROR 3

Lowspeed port encountered an error while receiving data

#### 9.123.2.4 #define LOWSPEED\_TX\_ERROR 2

Lowspeed port encountered an error while transmitting data

## 9.124 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

### 9.124.1 Detailed Description

Constant offsets into the low speed module IOMAP structure.

### 9.124.2 Define Documentation

#### 9.124.2.1 #define `LowSpeedOffsetChannelState(p)` ((p)+156)

R - Lowspeed channgel state (1 byte)

#### 9.124.2.2 #define `LowSpeedOffsetErrorType(p)` ((p)+160)

R - Lowspeed port error type (1 byte)

#### 9.124.2.3 #define `LowSpeedOffsetInBufBuf(p)` (((p)\*19)+0)

RW - Input buffer data buffer field offset (16 bytes)

#### 9.124.2.4 #define `LowSpeedOffsetInBufBytesToRx(p)` (((p)\*19)+18)

RW - Input buffer bytes to receive field offset (1 byte)

#### 9.124.2.5 #define `LowSpeedOffsetInBufInPtr(p)` (((p)\*19)+16)

RW - Input buffer in pointer field offset (1 byte)

**9.124.2.6 #define LowSpeedOffsetInBufOutPtr(p) (((p)\*19)+17)**

RW - Input buffer out pointer field offset (1 byte)

**9.124.2.7 #define LowSpeedOffsetMode(p) ((p)+152)**

R - Lowspeed port mode (1 byte)

**9.124.2.8 #define LowSpeedOffsetNoRestartOnRead 166**

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

**9.124.2.9 #define LowSpeedOffsetOutBufBuf(p) (((p)\*19)+76)**

RW - Output buffer data buffer field offset (16 bytes)

**9.124.2.10 #define LowSpeedOffsetOutBufBytesToRx(p) (((p)\*19)+94)**

RW - Output buffer bytes to receive field offset (1 byte)

**9.124.2.11 #define LowSpeedOffsetOutBufInPtr(p) (((p)\*19)+92)**

RW - Output buffer in pointer field offset (1 byte)

**9.124.2.12 #define LowSpeedOffsetOutBufOutPtr(p) (((p)\*19)+93)**

RW - Output buffer out pointer field offset (1 byte)

**9.124.2.13 #define LowSpeedOffsetSpeed 165**

R - Lowspeed speed (unused)

**9.124.2.14 #define LowSpeedOffsetState 164**

R - Lowspeed state (all channels)

## 9.125 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`

### 9.125.1 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\(\)](#)

### 9.125.2 Define Documentation

#### 9.125.2.1 `#define LSREAD_NO_RESTART_1 0x01`

No restart on read for channel 1

#### 9.125.2.2 `#define LSREAD_NO_RESTART_2 0x02`

No restart on read for channel 2

#### 9.125.2.3 `#define LSREAD_NO_RESTART_3 0x04`

No restart on read for channel 3

#### 9.125.2.4 `#define LSREAD_NO_RESTART_4 0x08`

No restart on read for channel 4

#### 9.125.2.5 `#define LSREAD_NO_RESTART_MASK 0x10`

No restart mask

#### 9.125.2.6 `#define LSREAD_RESTART_ALL 0x00`

Restart on read for all channels (default)

**9.125.2.7 #define LSREAD\_RESTART\_NONE 0x0F**

No restart on read for all channels

## 9.126 Standard I2C constants

Constants for use with standard I2C devices.

### Defines

- `#define I2C_REG_VERSION 0x00`
- `#define I2C_REG_VENDOR_ID 0x08`
- `#define I2C_REG_DEVICE_ID 0x10`
- `#define I2C_REG_CMD 0x41`

### 9.126.1 Detailed Description

Constants for use with standard I2C devices.

### 9.126.2 Define Documentation

#### 9.126.2.1 `#define I2C_REG_CMD 0x41`

Standard NXT I2C device command register

#### Examples:

[ex\\_MSReadValue.nxc](#), [ex\\_MSReadValueEx.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

#### 9.126.2.2 `#define I2C_REG_DEVICE_ID 0x10`

Standard NXT I2C device ID register

#### Examples:

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

#### 9.126.2.3 `#define I2C_REG_VENDOR_ID 0x08`

Standard NXT I2C vendor ID register

#### Examples:

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

#### 9.126.2.4 `#define I2C_REG_VERSION 0x00`

Standard NXT I2C version register

#### Examples:

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

## 9.127 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`

### 9.127.1 Detailed Description

Constants for use with the ultrasonic sensor.

### 9.127.2 Define Documentation

#### 9.127.2.1 `#define US_CMD_CONTINUOUS 0x02`

Command to put the ultrasonic sensor into continuous polling mode (default)

#### 9.127.2.2 `#define US_CMD_EVENTCAPTURE 0x03`

Command to put the ultrasonic sensor into event capture mode

#### 9.127.2.3 `#define US_CMD_OFF 0x00`

Command to turn off the ultrasonic sensor

#### Examples:

[ex\\_writei2cregister.nxc](#).

#### 9.127.2.4 `#define US_CMD_SINGLESHOT 0x01`

Command to put the ultrasonic sensor into single shot mode



**9.127.2.5 #define US\_CMD\_WARMRESET 0x04**

Command to warm reset the ultrasonic sensor

**9.127.2.6 #define US\_REG\_ACTUAL\_ZERO 0x50**

The register address used to store the actual zero value

**9.127.2.7 #define US\_REG\_CM\_INTERVAL 0x40**

The register address used to store the CM interval

**9.127.2.8 #define US\_REG\_FACTORY\_ACTUAL\_ZERO 0x11**

The register address containing the factory setting for the actual zero value

**9.127.2.9 #define US\_REG\_FACTORY\_SCALE\_DIVISOR 0x13**

The register address containing the factory setting for the scale divisor value

**9.127.2.10 #define US\_REG\_FACTORY\_SCALE\_FACTOR 0x12**

The register address containing the factory setting for the scale factor value

**9.127.2.11 #define US\_REG\_MEASUREMENT\_UNITS 0x14**

The register address containing the measurement units (degrees C or F)

**9.127.2.12 #define US\_REG\_SCALE\_DIVISOR 0x52**

The register address used to store the scale divisor value

**9.127.2.13 #define US\_REG\_SCALE\_FACTOR 0x51**

The register address used to store the scale factor value

## 9.128 Temperature sensor constants

Constants for use with the temperature sensor.

### Defines

- `#define TEMP_RES_12BIT 0x60`
- `#define TEMP_RES_11BIT 0x40`
- `#define TEMP_RES_10BIT 0x20`
- `#define TEMP_RES_9BIT 0x00`
- `#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_I2C_ADDRESS 0x98`
- `#define TEMP_REG_TEMP 0x00`
- `#define TEMP_REG_CONFIG 0x01`
- `#define TEMP_REG_TLOW 0x02`
- `#define TEMP_REG_THIGH 0x03`

### 9.128.1 Detailed Description

Constants for use with the temperature sensor.

### 9.128.2 Define Documentation

#### 9.128.2.1 `#define TEMP_FQ_1 0x00`

Set fault queue 1

#### 9.128.2.2 `#define TEMP_FQ_2 0x08`

Set fault queue 2

#### 9.128.2.3 `#define TEMP_FQ_4 0x10`

Set fault queue 4

#### 9.128.2.4 `#define TEMP_FQ_6 0x18`

Set fault queue 6

**9.128.2.5 #define TEMP\_I2C\_ADDRESS 0x98**

The temperature sensor's I2C address

**9.128.2.6 #define TEMP\_OS\_ONESHOT 0x80**

Set the sensor into oneshot mode

**9.128.2.7 #define TEMP\_POL\_HIGH 0x04**

Set polarity to high

**9.128.2.8 #define TEMP\_POL\_LOW 0x00**

Set polarity to low

**9.128.2.9 #define TEMP\_REG\_CONFIG 0x01**

The register for reading/writing sensor configuration values

**9.128.2.10 #define TEMP\_REG\_TEMP 0x00**

The register where temperature values can be read

**9.128.2.11 #define TEMP\_REG\_THIGH 0x03**

The register where temperature high values can be read

**9.128.2.12 #define TEMP\_REG\_TLOW 0x02**

The register where temperature low values can be read

**9.128.2.13 #define TEMP\_RES\_10BIT 0x20**

Set the resolution to 10 bit

**9.128.2.14 #define TEMP\_RES\_11BIT 0x40**

Set the resolution to 11 bit

**9.128.2.15 #define TEMP\_RES\_12BIT 0x60**

Set the resolution to 12 bit

**9.128.2.16 #define TEMP\_RES\_9BIT 0x00**

Set the resolution to 9 bit

**9.128.2.17 #define TEMP\_SD\_CONTINUOUS 0x00**

Set the mode to continuous

**9.128.2.18 #define TEMP\_SD\_SHUTDOWN 0x01**

Set the mode to shutdown

**9.128.2.19 #define TEMP\_TM\_COMPARATOR 0x00**

Set the thermostat mode to comparator

**9.128.2.20 #define TEMP\_TM\_INTERRUPT 0x02**

Set the thermostat mode to interrupt

## 9.129 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

### 9.129.1 Detailed Description

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

### 9.129.2 Define Documentation

#### 9.129.2.1 #define BITMAP\_1 0

Bitmap 1

#### 9.129.2.2 #define BITMAP\_2 1

Bitmap 2

#### 9.129.2.3 #define BITMAP\_3 2

Bitmap 3

#### 9.129.2.4 #define BITMAP\_4 3

Bitmap 4

#### 9.129.2.5 #define BITMAPS 4

The number of bitmap bits

**9.129.2.6 #define DISPLAY\_HEIGHT 64**

The height of the LCD screen in pixels

**Examples:**

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

**9.129.2.7 #define DISPLAY\_MENUICONS\_X\_DIFF 31****9.129.2.8 #define DISPLAY\_MENUICONS\_X\_OFFS 7****9.129.2.9 #define DISPLAY\_MENUICONS\_Y 40****9.129.2.10 #define DISPLAY\_WIDTH 100**

The width of the LCD screen in pixels

**Examples:**

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

**9.129.2.11 #define FRAME\_SELECT 0**

Center icon select frame

**9.129.2.12 #define MENUICON\_CENTER 1**

Center icon

**9.129.2.13 #define MENUICON\_LEFT 0**

Left icon

**9.129.2.14 #define MENUICON\_RIGHT 2**

Right icon

**9.129.2.15 #define MENUICONS 3**

The number of menu icons

**9.129.2.16 #define MENUTEXT 2**

Center icon text

**9.129.2.17 #define SCREEN\_BACKGROUND 0**

Entire screen

**9.129.2.18 #define SCREEN\_LARGE 1**

Entire screen except status line

**9.129.2.19 #define SCREEN\_MODE\_CLEAR 0x01**

Clear the screen

See also:

[SetScreenMode\(\)](#)

**9.129.2.20 #define SCREEN\_MODE\_RESTORE 0x00**

Restore the screen

See also:

[SetScreenMode\(\)](#)

**9.129.2.21 #define SCREEN\_SMALL 2**

Screen between menu icons and status line

**9.129.2.22 #define SCREENS 3**

The number of screen bits

**9.129.2.23 #define SPECIALS 5**

The number of special bit values

**9.129.2.24 #define STATUSICON\_BATTERY 3**

Battery status icon collection

**9.129.2.25 #define STATUSICON\_BLUETOOTH 0**

BlueTooth status icon collection

**9.129.2.26 #define STATUSICON\_USB 1**

USB status icon collection



**9.129.2.27 #define STATUSICON\_VM 2**

VM status icon collection

**9.129.2.28 #define STATUSICONS 4**

The number of status icons

**9.129.2.29 #define STATUSTEXT 1**

Status text (BT name)

**9.129.2.30 #define STEPICON\_1 0**

Left most step icon

**9.129.2.31 #define STEPICON\_2 1****9.129.2.32 #define STEPICON\_3 2****9.129.2.33 #define STEPICON\_4 3****9.129.2.34 #define STEPICON\_5 4**

Right most step icon

**9.129.2.35 #define STEPICONS 5****9.129.2.36 #define STEPLINE 3**

Step collection lines

**9.129.2.37 #define TOPLINE 4**

Top status underline

## 9.130 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

### 9.130.1 Detailed Description

Constants that are for use with the DisplayExecuteFunction system call.

### 9.130.2 Define Documentation

#### 9.130.2.1 #define `DISPLAY_CHAR` 0x04

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

#### 9.130.2.2 #define `DISPLAY_ERASE_ALL` 0x00

W - erase entire screen (CMD,x,x,x,x,x)

#### Examples:

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

#### 9.130.2.3 #define `DISPLAY_ERASE_LINE` 0x05

W - erase a single line (CMD,x,LINE,x,x,x)

#### 9.130.2.4 #define `DISPLAY_FILL_REGION` 0x06

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

#### 9.130.2.5 #define `DISPLAY_FRAME` 0x07

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**9.130.2.6 #define DISPLAY\_HORIZONTAL\_LINE 0x02**

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

**Examples:**

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

**9.130.2.7 #define DISPLAY\_PIXEL 0x01**

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

**9.130.2.8 #define DISPLAY\_VERTICAL\_LINE 0x03**

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

## 9.131 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)`

#### 9.131.1 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\(\)](#), [Font-TextOut\(\)](#), [FontNumOut\(\)](#), [GraphicOut\(\)](#), [GraphicArrayOut\(\)](#)

#### 9.131.2 Define Documentation

##### 9.131.2.1 `#define DRAW_OPT_CLEAR (0x0004)`

Clear pixels while drawing (aka draw in white)

**9.131.2.2 #define DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_SCREEN (0x0002)**

Clear the screen except for the status line before drawing

**9.131.2.3 #define DRAW\_OPT\_CLEAR\_PIXELS (0x0004)**

Clear pixels while drawing (aka draw in white)

**9.131.2.4 #define DRAW\_OPT\_CLEAR\_SCREEN\_MODES (0x0003)**

Bit mask for the clear screen modes

**9.131.2.5 #define DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN (0x0001)**

Clear the entire screen before drawing

**Examples:**

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

**9.131.2.6 #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](#).

**9.131.2.7 #define DRAW\_OPT\_INVERT (0x0004)**

Invert text or graphics

**Examples:**

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

**9.131.2.8 #define DRAW\_OPT\_LOGICAL\_AND (0x0008)**

Draw pixels using a logical AND operation

**Examples:**

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

**9.131.2.9 #define DRAW\_OPT\_LOGICAL\_COPY (0x0000)**

Draw pixels using a logical copy operation

**9.131.2.10 #define DRAW\_OPT\_LOGICAL\_OPERATIONS (0x0018)**

Bit mask for the logical drawing operations

**9.131.2.11 #define DRAW\_OPT\_LOGICAL\_OR (0x0010)**

Draw pixels using a logical OR operation

**Examples:**

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

**9.131.2.12 #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](#).

**9.131.2.13 #define DRAW\_OPT\_NORMAL (0x0000)**

Normal drawing

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispfunc.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**9.131.2.14 #define DRAW\_OPT\_POLYGON\_POLYLINE (0x0400)**

When drawing polygons, do not close (i.e., draw a polyline instead)

## 9.132 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)`

### 9.132.1 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\(\)](#)

### 9.132.2 Define Documentation

#### 9.132.2.1 `#define DRAW_OPT_FONT_DIR_B2TL (0x0100)`

Font bottom to top left align

#### 9.132.2.2 `#define DRAW_OPT_FONT_DIR_B2TR (0x0140)`

Font bottom to top right align

#### 9.132.2.3 `#define DRAW_OPT_FONT_DIR_L2RB (0x0000)`

Font left to right bottom align

Examples:

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

**9.132.2.4 #define DRAW\_OPT\_FONT\_DIR\_L2RT (0x0040)**

Font left to right top align

**Examples:**

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**9.132.2.5 #define DRAW\_OPT\_FONT\_DIR\_R2LB (0x0080)**

Font right to left bottom align

**9.132.2.6 #define DRAW\_OPT\_FONT\_DIR\_R2LT (0x00C0)**

Font right to left top align

**9.132.2.7 #define DRAW\_OPT\_FONT\_DIR\_T2BL (0x0180)**

Font top to bottom left align

**Examples:**

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

**9.132.2.8 #define DRAW\_OPT\_FONT\_DIR\_T2BR (0x01C0)**

Font top to bottom right align

**9.132.2.9 #define DRAW\_OPT\_FONT DIRECTIONS (0x01C0)**

Bit mask for the font direction bits

**9.132.2.10 #define DRAW\_OPT\_FONT\_WRAP (0x0200)**

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

**Examples:**

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



## 9.133 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`

### 9.133.1 Detailed Description

Constants that are for use with the display flags functions.

See also:

[SetDisplayFlags\(\)](#), [DisplayFlags\(\)](#)

### 9.133.2 Define Documentation

#### 9.133.2.1 `#define DISPLAY_BUSY 0x80`

R - Refresh in progress

#### 9.133.2.2 `#define DISPLAY_ON 0x01`

W - Display on

#### 9.133.2.3 `#define DISPLAY_POPUP 0x08`

W - Use popup display memory

Examples:

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

#### 9.133.2.4 `#define DISPLAY_REFRESH 0x02`

W - Enable refresh

#### 9.133.2.5 `#define DISPLAY_REFRESH_DISABLED 0x40`

R - Refresh disabled

## 9.134 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`

### 9.134.1 Detailed Description

Constants that are for use with the display contrast API functions.

See also:

[SetDisplayContrast\(\)](#), [DisplayContrast\(\)](#)

### 9.134.2 Define Documentation

#### 9.134.2.1 `#define DISPLAY_CONTRAST_DEFAULT 0x5A`

Default display contrast value

Examples:

[ex\\_contrast.nxc](#), and [ex\\_SetDisplayContrast.nxc](#).

#### 9.134.2.2 `#define DISPLAY_CONTRAST_MAX 0x7F`

Maximum display contrast value

Examples:

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

## 9.135 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`

### 9.135.1 Detailed Description

Constants that are for use with getting/setting display data.

See also:

[SetDisplayNormal\(\)](#), [GetDisplayNormal\(\)](#), [SetDisplayPopup\(\)](#), [GetDisplayPopup\(\)](#)

### 9.135.2 Define Documentation

#### 9.135.2.1 `#define TEXTLINE_1 0`

Text line 1

Examples:

[ex\\_GetDisplayNormal.nxc](#), [ex\\_GetDisplayPopup.nxc](#), [ex\\_SetDisplayNormal.nxc](#), and [ex\\_SetDisplayPopup.nxc](#).

#### 9.135.2.2 `#define TEXTLINE_2 1`

Text line 2

#### 9.135.2.3 `#define TEXTLINE_3 2`

Text line 3

#### 9.135.2.4 `#define TEXTLINE_4 3`

Text line 4

**9.135.2.5 #define TEXTLINE\_5 4**

Text line 5

**9.135.2.6 #define TEXTLINE\_6 5**

Text line 6

**9.135.2.7 #define TEXTLINE\_7 6**

Text line 7

**9.135.2.8 #define TEXTLINE\_8 7**

Text line 8

**9.135.2.9 #define TEXTLINES 8**

The number of text lines on the LCD

## 9.136 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`

### 9.136.1 Detailed Description

Constant offsets into the display module IOMAP structure.

### 9.136.2 Define Documentation

#### 9.136.2.1 `#define DisplayOffsetContrast 1719`

Adjust the display contrast with this field

#### 9.136.2.2 `#define DisplayOffsetDisplay 104`

Display content copied to physical display every 17 mS

#### 9.136.2.3 `#define DisplayOffsetEraseMask 4`

Section erase mask (executed first)

#### 9.136.2.4 `#define DisplayOffsetFlags 117`

Update flags enumerated above

**9.136.2.5 #define DisplayOffsetNormal(l, w) (((l)\*100)+(w)+119)**

Raw display memory for normal screen

**9.136.2.6 #define DisplayOffsetPBitmaps(p) (((p)\*4)+68)**

Pointer to free bitmap files

**9.136.2.7 #define DisplayOffsetPFont 12**

Pointer to font file

**9.136.2.8 #define DisplayOffsetPFunc 0**

Simple draw entry

**9.136.2.9 #define DisplayOffsetPMenuIcons(p) (((p)\*4)+88)**

Pointer to menu icon images (NULL == none)

**9.136.2.10 #define DisplayOffsetPMenuText 84**

Pointer to menu icon text (NULL == none)

**9.136.2.11 #define DisplayOffsetPopup(l, w) (((l)\*100)+(w)+919)**

Raw display memory for popup screen

**9.136.2.12 #define DisplayOffsetPScreens(p) (((p)\*4)+56)**

Pointer to screen bitmap file

**9.136.2.13 #define DisplayOffsetPStatusIcons 52**

Pointer to status icon collection file

**9.136.2.14 #define DisplayOffsetPStatusText 48**

Pointer to status text string

**9.136.2.15 #define DisplayOffsetPStepIcons 100**

Pointer to step icon collection file

**9.136.2.16 #define DisplayOffsetPTextLines(p) (((p)\*4)+16)**

Pointer to text strings

**9.136.2.17 #define DisplayOffsetStatusIcons(p) ((p)+108)**

Index in status icon collection file (index = 0 -> none)

**9.136.2.18 #define DisplayOffsetStepIcons(p) ((p)+112)**

Index in step icon collection file (index = 0 -> none)

**9.136.2.19 #define DisplayOffsetTextLinesCenterFlags 118**

Mask to center TextLines

**9.136.2.20 #define DisplayOffsetUpdateMask 8**

Section update mask (executed next)

## 9.137 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.*

- [Bluetooth state status constants](#)

*Constants related to the bluetooth state status.*

- [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.*

### 9.137.1 Detailed Description

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



## 9.138 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\\_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

### 9.138.1 Detailed Description

Miscellaneous constants related to the Comm module.

### 9.138.2 Define Documentation

#### 9.138.2.1 #define BT\_CMD\_BYTE 1

Size of Bluetooth command

#### 9.138.2.2 #define BT\_DEFAULT\_INQUIRY\_MAX 0

Bluetooth default inquiry Max (0 == unlimited)

#### 9.138.2.3 #define BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO 15

Bluetooth inquiry timeout (15\*1.28 sec = 19.2 sec)

#### 9.138.2.4 #define MAX\_BT\_MSG\_SIZE 60000

Max Bluetooth Message Size

#### 9.138.2.5 #define SIZE\_OF\_BDADDR 7

Size of Bluetooth Address

**9.138.2.6 #define SIZE\_OF\_BRICK\_NAME 8**

Size of NXT Brick name

**9.138.2.7 #define SIZE\_OF\_BT\_CONNECT\_TABLE 4**

Size of Bluetooth connection table -- Index 0 is always incoming connection

**9.138.2.8 #define SIZE\_OF\_BT\_DEVICE\_TABLE 30**

Size of Bluetooth device table

**9.138.2.9 #define SIZE\_OF\_BT\_NAME 16**

Size of Bluetooth name

**9.138.2.10 #define SIZE\_OF\_BT\_PINCODE 16**

Size of Bluetooth PIN

**9.138.2.11 #define SIZE\_OF\_BTBUF 128**

Size of Bluetooth buffer

**9.138.2.12 #define SIZE\_OF\_CLASS\_OF\_DEVICE 4**

Size of class of device

**9.138.2.13 #define SIZE\_OF\_HSBUF 128**

Size of High Speed Port 4 buffer

**9.138.2.14 #define SIZE\_OF\_USBBUF 64**

Size of USB Buffer in bytes

**9.138.2.15 #define SIZE\_OF\_USBDATA 62**

Size of USB Buffer available for data

**9.138.2.16 #define USB\_PROTOCOL\_OVERHEAD 2**

Size of USB Overhead in bytes -- Command type byte + Command

## 9.139 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`

### 9.139.1 Detailed Description

Constants related to the bluetooth state.

### 9.139.2 Define Documentation

#### 9.139.2.1 `#define BT_ARM_CMD_MODE 1`

BtState constant bluetooth command mode

#### 9.139.2.2 `#define BT_ARM_DATA_MODE 2`

BtState constant bluetooth data mode

#### 9.139.2.3 `#define BT_ARM_OFF 0`

BtState constant bluetooth off

## 9.140 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`

### 9.140.1 Detailed Description

Constants related to the bluetooth state status.

### 9.140.2 Define Documentation

#### 9.140.2.1 `#define BT_BRICK_PORT_OPEN 0x02`

BtStateStatus port open bit

#### 9.140.2.2 `#define BT_BRICK_VISIBILITY 0x01`

BtStateStatus brick visibility bit

#### 9.140.2.3 `#define BT_CONNECTION_0_ENABLE 0x10`

BtStateStatus connection 0 enable/disable bit

#### 9.140.2.4 `#define BT_CONNECTION_1_ENABLE 0x20`

BtStateStatus connection 1 enable/disable bit

#### 9.140.2.5 `#define BT_CONNECTION_2_ENABLE 0x40`

BtStateStatus connection 2 enable/disable bit

#### 9.140.2.6 `#define BT_CONNECTION_3_ENABLE 0x80`

BtStateStatus connection 3 enable/disable bit

## 9.141 Bluetooth hardware status constants

Constants related to the bluetooth hardware status.

### Defines

- #define [BT\\_ENABLE](#) 0x00
- #define [BT\\_DISABLE](#) 0x01

### 9.141.1 Detailed Description

Constants related to the bluetooth hardware status.

### 9.141.2 Define Documentation

#### 9.141.2.1 #define BT\_DISABLE 0x01

BtHwStatus bluetooth disable

#### 9.141.2.2 #define BT\_ENABLE 0x00

BtHwStatus bluetooth enable

## 9.142 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.*
- [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.*

### 9.142.1 Detailed Description

Constants related to the hi-speed port.

## 9.143 Hi-speed port flags constants

Constants related to the hi-speed port flags.

### Defines

- #define [HS\\_UPDATE](#) 1

#### 9.143.1 Detailed Description

Constants related to the hi-speed port flags.

#### 9.143.2 Define Documentation

##### 9.143.2.1 #define HS\_UPDATE 1

HsFlags high speed update required

## 9.144 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

### 9.144.1 Detailed Description

Constants related to the hi-speed port state.

### 9.144.2 Define Documentation

#### 9.144.2.1 #define HS\_DISABLE 4

HsState disable

#### 9.144.2.2 #define HS\_ENABLE 5

HsState enable

#### 9.144.2.3 #define HS\_INIT\_RECEIVER 2

HsState initialize receiver

#### 9.144.2.4 #define HS\_INITIALISE 1

HsState initialize

#### 9.144.2.5 #define HS\_SEND\_DATA 3

HsState send data



## 9.145 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

### 9.145.1 Detailed Description

Constants for use with the SysCommHSControl API function.

See also:

[SysCommHSControl\(\)](#)

### 9.145.2 Define Documentation

#### 9.145.2.1 #define HS\_CTRL\_EXIT 2

Disable the high speed port

#### 9.145.2.2 #define HS\_CTRL\_INIT 0

Enable the high speed port

Examples:

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

#### 9.145.2.3 #define HS\_CTRL\_UART 1

Setup the high speed port UART configuration

Examples:

[ex\\_RS485Control.nxc](#).

## 9.146 Hi-speed port baud rate constants

Constants for configuring the hi-speed port baud rate.

### 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`

### 9.146.1 Detailed Description

Constants for configuring the hi-speed port baud rate.

### 9.146.2 Define Documentation

#### 9.146.2.1 `#define HS_BAUD_115200 12`

HsMode 115200 Baud

#### Examples:

[ex\\_RS485Uart.nxc](#).

#### 9.146.2.2 `#define HS_BAUD_1200 0`

HsMode 1200 Baud

#### 9.146.2.3 `#define HS_BAUD_14400 6`

HsMode 14400 Baud

#### 9.146.2.4 `#define HS_BAUD_19200 7`

HsMode 19200 Baud

**9.146.2.5 #define HS\_BAUD\_230400 13**

HsMode 230400 Baud

**9.146.2.6 #define HS\_BAUD\_2400 1**

HsMode 2400 Baud

**9.146.2.7 #define HS\_BAUD\_28800 8**

HsMode 28800 Baud

**9.146.2.8 #define HS\_BAUD\_3600 2**

HsMode 3600 Baud

**9.146.2.9 #define HS\_BAUD\_38400 9**

HsMode 38400 Baud

**9.146.2.10 #define HS\_BAUD\_460800 14**

HsMode 460800 Baud

**9.146.2.11 #define HS\_BAUD\_4800 3**

HsMode 4800 Baud

**9.146.2.12 #define HS\_BAUD\_57600 10**

HsMode 57600 Baud

**9.146.2.13 #define HS\_BAUD\_7200 4**

HsMode 7200 Baud

**9.146.2.14 #define HS\_BAUD\_76800 11**

HsMode 76800 Baud

**9.146.2.15 #define HS\_BAUD\_921600 15**

HsMode 921600 Baud

**9.146.2.16 #define HS\_BAUD\_9600 5**

HsMode 9600 Baud

**Examples:**

[ex\\_RS485Control.nxc](#).

## 9.147 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

### 9.147.1 Detailed Description

Constants referring to HsMode (number of data bits).

### 9.147.2 Define Documentation

#### 9.147.2.1 #define HS\_MODE\_5\_DATA 0x0000

HsMode 5 data bits

#### 9.147.2.2 #define HS\_MODE\_6\_DATA 0x0040

HsMode 6 data bits

#### 9.147.2.3 #define HS\_MODE\_7\_DATA 0x0080

HsMode 7 data bits

#### 9.147.2.4 #define HS\_MODE\_8\_DATA 0x00C0

HsMode 8 data bits

## 9.148 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

### 9.148.1 Detailed Description

Constants referring to HsMode (number of stop bits).

### 9.148.2 Define Documentation

#### 9.148.2.1 #define HS\_MODE\_10\_STOP 0x0000

HsMode 1 stop bit

#### 9.148.2.2 #define HS\_MODE\_15\_STOP 0x1000

HsMode 1.5 stop bits

#### 9.148.2.3 #define HS\_MODE\_20\_STOP 0x2000

HsMode 2 stop bits

## 9.149 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

### 9.149.1 Detailed Description

Constants referring to HsMode (parity).

### 9.149.2 Define Documentation

#### 9.149.2.1 #define HS\_MODE\_E\_PARITY 0x0000

HsMode Even parity

#### 9.149.2.2 #define HS\_MODE\_M\_PARITY 0x0600

HsMode Mark parity

#### 9.149.2.3 #define HS\_MODE\_N\_PARITY 0x0800

HsMode No parity

#### 9.149.2.4 #define HS\_MODE\_O\_PARITY 0x0200

HsMode Odd parity

#### 9.149.2.5 #define HS\_MODE\_S\_PARITY 0x0400

HsMode Space parity

## 9.150 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)`

### 9.150.1 Detailed Description

Constants that combine data bits, parity, and stop bits into a single value.

### 9.150.2 Define Documentation

#### 9.150.2.1 `#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

#### 9.150.2.2 `#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\\_RS485Control.nxc](#), [ex\\_RS485Uart.nxc](#), and [ex\\_SetHSMODE.nxc](#).



## 9.151 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`

### 9.151.1 Detailed Description

Constants referring to DeviceStatus within DeviceTable.

### 9.151.2 Define Documentation

#### 9.151.2.1 `#define BT_DEVICE_AWAY 0x80`

Bluetooth device away

#### 9.151.2.2 `#define BT_DEVICE_EMPTY 0x00`

Bluetooth device table empty

#### 9.151.2.3 `#define BT_DEVICE_KNOWN 0x02`

Bluetooth device known

#### 9.151.2.4 `#define BT_DEVICE_NAME 0x40`

Bluetooth device name

#### 9.151.2.5 `#define BT_DEVICE_UNKNOWN 0x01`

Bluetooth device unknown

## 9.152 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`

### 9.152.1 Detailed Description

Constants for all the Comm module interface functions executable via SysCommExecuteFunction.

See also:

[SysCommExecuteFunction\(\)](#)

### 9.152.2 Define Documentation

#### 9.152.2.1 `#define INTF_BTOff 13`

Turn off the bluetooth radio

Examples:

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

#### 9.152.2.2 `#define INTF_BTOn 12`

Turn on the bluetooth radio

**9.152.2.3 #define INTF\_CONNECT 3**

Connect to one of the known devices

**9.152.2.4 #define INTF\_CONNECTBYNAME 18**

Connect to a bluetooth device by name

**9.152.2.5 #define INTF\_CONNECTREQ 17**

Connection request from another device

**9.152.2.6 #define INTF\_DISCONNECT 4**

Disconnect from one of the connected devices

**9.152.2.7 #define INTF\_DISCONNECTALL 5**

Disconnect all devices

**9.152.2.8 #define INTF\_EXTREAD 15**

External read request

**9.152.2.9 #define INTF\_FACTORYRESET 11**

Reset bluetooth settings to factory values

**9.152.2.10 #define INTF\_OPENSTREAM 9**

Open a bluetooth stream

**9.152.2.11 #define INTF\_PINREQ 16**

Bluetooth PIN request

**9.152.2.12 #define INTF\_REMOVEDEVICE 6**

Remove a device from the known devices table

**9.152.2.13 #define INTF\_SEARCH 1**

Search for bluetooth devices

**9.152.2.14 #define INTF\_SENDDATA 10**

Send data over a bluetooth connection

**9.152.2.15 #define INTF\_SENDFILE 0**

Send a file via bluetooth to another device

**9.152.2.16 #define INTF\_SETBTNAME 14**

Set the bluetooth name

**9.152.2.17 #define INTF\_SETCMDMODE 8**

Set bluetooth into command mode

**9.152.2.18 #define INTF\_STOPSEARCH 2**

Stop searching for bluetooth devices

**9.152.2.19 #define INTF\_VISIBILITY 7**

Set the bluetooth visibility on or off

## 9.153 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`

### 9.153.1 Detailed Description

Constants for Comm module status codes.

### 9.153.2 Define Documentation

#### 9.153.2.1 `#define BT_CMD_READY 0x02`

A constant representing bluetooth direct command

#### 9.153.2.2 `#define HS_CMD_READY 0x04`

A constant representing high speed direct command

#### 9.153.2.3 `#define LR_COULD_NOT_SAVE 0x51`

Bluetooth list result could not save

#### 9.153.2.4 `#define LR_ENTRY_REMOVED 0x53`

Bluetooth list result entry removed

#### 9.153.2.5 `#define LR_STORE_IS_FULL 0x52`

Bluetooth list result store is full

#### 9.153.2.6 `#define LR_SUCCESS 0x50`

Bluetooth list result success

**9.153.2.7 #define LR\_UNKNOWN\_ADDR 0x54**

Bluetooth list result unknown address

**9.153.2.8 #define USB\_CMD\_READY 0x01**

A constant representing usb direct command

## 9.154 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 [CommOffsetHsMode](#) 1895

### 9.154.1 Detailed Description

Constant offsets into the Comm module IOMAP structure.

### 9.154.2 Define Documentation

#### 9.154.2.1 **#define CommOffsetBrickDataBdAddr 1144**

Offset to Bluetooth address (7 bytes)

#### 9.154.2.2 **#define CommOffsetBrickDataBluecoreVersion 1142**

Offset to Bluecore version (2 bytes)

#### 9.154.2.3 **#define CommOffsetBrickDataBtHwStatus 1152**

Offset to BtHwStatus (1 byte)

#### 9.154.2.4 **#define CommOffsetBrickDataBtStateStatus 1151**

Offset to BtStateStatus (1 byte)

#### 9.154.2.5 **#define CommOffsetBrickDataName 1126**

Offset to brick name (16 bytes)

#### 9.154.2.6 **#define CommOffsetBrickDataTimeOutValue 1153**

Offset to data timeout value (1 byte)

#### 9.154.2.7 **#define CommOffsetBtConnectTableBdAddr(p) (((p)\*47)+974)**

Offset to Bluetooth connect table address (7 bytes)

#### 9.154.2.8 **#define CommOffsetBtConnectTableClassOfDevice(p) (((p)\*47)+954)**

Offset to Bluetooth connect table device class (4 bytes)

#### 9.154.2.9 **#define CommOffsetBtConnectTableHandleNr(p) (((p)\*47)+981)**

Offset to Bluetooth connect table handle (1 byte)

#### 9.154.2.10 **#define CommOffsetBtConnectTableLinkQuality(p) (((p)\*47)+983)**

Offset to Bluetooth connect table link quality (1 byte)



**9.154.2.11 #define CommOffsetBtConnectTableName(p) (((p)\*47)+938)**

Offset to Bluetooth connect table name (16 bytes)

**9.154.2.12 #define CommOffsetBtConnectTablePinCode(p) (((p)\*47)+958)**

Offset to Bluetooth connect table pin code (16 bytes)

**9.154.2.13 #define CommOffsetBtConnectTableStreamStatus(p) (((p)\*47)+982)**

Offset to Bluetooth connect table stream status (1 byte)

**9.154.2.14 #define CommOffsetBtDeviceCnt 1889**

Offset to Bluetooth device count (1 byte)

**9.154.2.15 #define CommOffsetBtDeviceNameCnt 1890**

Offset to Bluetooth device name count (1 byte)

**9.154.2.16 #define CommOffsetBtDeviceTableBdAddr(p) (((p)\*31)+28)**

Offset to Bluetooth device table address (7 bytes)

**9.154.2.17 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)\*31)+24)**

Offset to Bluetooth device table device class (4 bytes)

**9.154.2.18 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)\*31)+35)**

Offset to Bluetooth device table status (1 byte)

**9.154.2.19 #define CommOffsetBtDeviceTableName(p) (((p)\*31)+8)**

Offset to BT device table name (16 bytes)

**9.154.2.20 #define CommOffsetBtInBufBuf 1157**

Offset to Bluetooth input buffer data (128 bytes)

**9.154.2.21 #define CommOffsetBtInBufInPtr 1285**

Offset to Bluetooth input buffer front pointer (1 byte)

**9.154.2.22 #define CommOffsetBtInBufOutPtr 1286**

Offset to Bluetooth output buffer back pointer (1 byte)

**9.154.2.23 #define CommOffsetBtOutBufBuf 1289**

Offset to Bluetooth output buffer offset data (128 bytes)

**9.154.2.24 #define CommOffsetBtOutBufInPtr 1417**

Offset to Bluetooth output buffer front pointer (1 byte)

**9.154.2.25 #define CommOffsetBtOutBufOutPtr 1418**

Offset to Bluetooth output buffer back pointer (1 byte)

**9.154.2.26 #define CommOffsetHsFlags 1891**

Offset to High Speed flags (1 byte)

**9.154.2.27 #define CommOffsetHsInBufBuf 1421**

Offset to High Speed input buffer data (128 bytes)

**9.154.2.28 #define CommOffsetHsInBufInPtr 1549**

Offset to High Speed input buffer front pointer (1 byte)

**9.154.2.29 #define CommOffsetHsInBufOutPtr 1550**

Offset to High Speed input buffer back pointer (1 byte)

**9.154.2.30 #define CommOffsetHsMode 1895**

Offset to High Speed mode (2 bytes)

**9.154.2.31 #define CommOffsetHsOutBufBuf 1553**

Offset to High Speed output buffer data (128 bytes)

**9.154.2.32 #define CommOffsetHsOutBufInPtr 1681**

Offset to High Speed output buffer front pointer (1 byte)

**9.154.2.33 #define CommOffsetHsOutBufOutPtr 1682**

Offset to High Speed output buffer back pointer (1 byte)

**9.154.2.34 #define CommOffsetHsSpeed 1892**

Offset to High Speed speed (1 byte)

**9.154.2.35 #define CommOffsetHsState 1893**

Offset to High Speed state (1 byte)

**9.154.2.36 #define CommOffsetPFunc 0**

Offset to the Comm module first function pointer (4 bytes)

**9.154.2.37 #define CommOffsetPFuncTwo 4**

Offset to the Comm module second function pointer (4 bytes)

**9.154.2.38 #define CommOffsetUsbInBufBuf 1685**

Offset to Usb input buffer data (64 bytes)

**9.154.2.39 #define CommOffsetUsbInBufInPtr 1749**

Offset to Usb input buffer front pointer (1 byte)

**9.154.2.40 #define CommOffsetUsbInBufOutPtr 1750**

Offset to Usb input buffer back pointer (1 byte)

**9.154.2.41 #define CommOffsetUsbOutBufBuf 1753**

Offset to Usb output buffer data (64 bytes)

**9.154.2.42 #define CommOffsetUsbOutBufInPtr 1817**

Offset to Usb output buffer front pointer (1 byte)

**9.154.2.43 #define CommOffsetUsbOutBufOutPtr 1818**

Offset to Usb output buffer back pointer (1 byte)

**9.154.2.44 #define CommOffsetUsbPollBufBuf 1821**

Offset to Usb Poll buffer data (64 bytes)

**9.154.2.45 #define CommOffsetUsbPollBufInPtr 1885**

Offset to Usb Poll buffer front pointer (1 byte)

**9.154.2.46 #define CommOffsetUsbPollBufOutPtr 1886**

Offset to Usb Poll buffer back pointer (1 byte)

**9.154.2.47 #define CommOffsetUsbState 1894**

Offset to Usb State (1 byte)

## 9.155 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.*

### 9.155.1 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.

## 9.156 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`

### 9.156.1 Detailed Description

Constants for use when choosing RCX outputs.

### 9.156.2 Define Documentation

#### 9.156.2.1 `#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`.

#### 9.156.2.2 `#define RCX_OUT_AB 0x03`

RCX Outputs A and B

#### 9.156.2.3 `#define RCX_OUT_ABC 0x07`

RCX Outputs A, B, and C

**9.156.2.4 #define RCX\_OUT\_AC 0x05**

RCX Outputs A and C

**9.156.2.5 #define RCX\_OUT\_B 0x02**

RCX Output B

**9.156.2.6 #define RCX\_OUT\_BC 0x06**

RCX Outputs B and C

**9.156.2.7 #define RCX\_OUT\_C 0x04**

RCX Output C

## 9.157 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`

### 9.157.1 Detailed Description

Constants for use when configuring RCX output mode.

### 9.157.2 Define Documentation

#### 9.157.2.1 `#define RCX_OUT_FLOAT 0`

Set RCX output to float

#### 9.157.2.2 `#define RCX_OUT_OFF 0x40`

Set RCX output to off

#### 9.157.2.3 `#define RCX_OUT_ON 0x80`

Set RCX output to on

### Examples:

[ex\\_HTRCXSetGlobalOutput.nxc](#), [ex\\_HTRCXSetOutput.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), and [ex\\_MSRCXSetOutput.nxc](#).



## 9.158 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`

### 9.158.1 Detailed Description

Constants for use when configuring RCX output direction.

### 9.158.2 Define Documentation

#### 9.158.2.1 `#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`.

#### 9.158.2.2 `#define RCX_OUT_REV 0`

Set RCX output direction to reverse

#### 9.158.2.3 `#define RCX_OUT_TOGGLE 0x40`

Set RCX output direction to toggle

## 9.159 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`

### 9.159.1 Detailed Description

Constants for use when configuring RCX output power.

### 9.159.2 Define Documentation

#### 9.159.2.1 `#define RCX_OUT_FULL 7`

Set RCX output power level to full

#### Examples:

`ex_HTRCXSetPower.nxc`, and `ex_MSRCXSetPower.nxc`.

#### 9.159.2.2 `#define RCX_OUT_HALF 3`

Set RCX output power level to half

#### 9.159.2.3 `#define RCX_OUT_LOW 0`

Set RCX output power level to low

## 9.160 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

### 9.160.1 Detailed Description

Constants for use when simulating RCX IR remote messages.

### 9.160.2 Define Documentation

#### 9.160.2.1 #define [RCX\\_RemoteKeysReleased](#) 0x0000

All remote keys have been released

#### 9.160.2.2 #define [RCX\\_RemoteOutABackward](#) 0x4000

Set output A backward

#### 9.160.2.3 #define [RCX\\_RemoteOutAForward](#) 0x0800

Set output A forward

#### 9.160.2.4 #define [RCX\\_RemoteOutBBackward](#) 0x8000

Set output B backward

**9.160.2.5 #define RCX\_RemoteOutBForward 0x1000**

Set output B forward

**9.160.2.6 #define RCX\_RemoteOutCBackward 0x0001**

Set output C backward

**9.160.2.7 #define RCX\_RemoteOutCForward 0x2000**

Set output C forward

**9.160.2.8 #define RCX\_RemotePBMessage1 0x0100**

Send PB message 1

**9.160.2.9 #define RCX\_RemotePBMessage2 0x0200**

Send PB message 2

**9.160.2.10 #define RCX\_RemotePBMessage3 0x0400**

Send PB message 3

**9.160.2.11 #define RCX\_RemotePlayASound 0x0080**

Play a sound

**Examples:**

[ex\\_HTRCXRemote.nxc](#), and [ex\\_MSRCXRemote.nxc](#).

**9.160.2.12 #define RCX\_RemoteSelProgram1 0x0002**

Select program 1

**9.160.2.13 #define RCX\_RemoteSelProgram2 0x0004**

Select program 2

**9.160.2.14 #define RCX\_RemoteSelProgram3 0x0008**

Select program 3

**9.160.2.15 #define RCX\_RemoteSelProgram4 0x0010**

Select program 4

**9.160.2.16 #define RCX\_RemoteSelProgram5 0x0020**

Select program 5

**9.160.2.17 #define RCX\_RemoteStopOutOff 0x0040**

Stop and turn off outputs

## 9.161 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

### 9.161.1 Detailed Description

Constants for use when playing standard RCX and Scout sounds.

### 9.161.2 Define Documentation

#### 9.161.2.1 #define SOUND\_CLICK 0

Play the standard key click sound

#### 9.161.2.2 #define SOUND\_DOUBLE\_BEEP 1

Play the standard double beep sound

#### 9.161.2.3 #define SOUND\_DOWN 2

Play the standard sweep down sound

#### Examples:

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

#### 9.161.2.4 #define SOUND\_FAST\_UP 5

Play the standard fast up sound

#### Examples:

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

**9.161.2.5 #define SOUND\_LOW\_BEEP 4**

Play the standard low beep sound

**Examples:**

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

**9.161.2.6 #define SOUND\_UP 3**

Play the standard sweep up sound

**Examples:**

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

## 9.162 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.*

### 9.162.1 Detailed Description

Constants for use when controlling the Scout brick.



## 9.163 Scout light constants

Constants for use when controlling the Scout light settings.

### Defines

- `#define SCOUT_LIGHT_ON 0x80`
- `#define SCOUT_LIGHT_OFF 0`

### 9.163.1 Detailed Description

Constants for use when controlling the Scout light settings.

### 9.163.2 Define Documentation

#### 9.163.2.1 `#define SCOUT_LIGHT_OFF 0`

Turn off the scout light

#### 9.163.2.2 `#define SCOUT_LIGHT_ON 0x80`

Turn on the scout light

### Examples:

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

## 9.164 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

### 9.164.1 Detailed Description

Constants for use when playing standard Scout sounds.

### 9.164.2 Define Documentation

#### 9.164.2.1 #define SCOUT\_SOUND\_1\_BLINK 17

Play the Scout 1 blink sound

#### 9.164.2.2 #define SCOUT\_SOUND\_2\_BLINK 18

Play the Scout 2 blink sound

#### 9.164.2.3 #define SCOUT\_SOUND\_COUNTER1 19

Play the Scout counter 1 sound

**9.164.2.4 #define SCOUT\_SOUND\_COUNTER2 20**

Play the Scout counter 2 sound

**9.164.2.5 #define SCOUT\_SOUND\_ENTER\_BRIGHT 14**

Play the Scout enter bright sound

**9.164.2.6 #define SCOUT\_SOUND\_ENTER\_DARK 16**

Play the Scout enter dark sound

**9.164.2.7 #define SCOUT\_SOUND\_ENTER\_NORMAL 15**

Play the Scout enter normal sound

**9.164.2.8 #define SCOUT\_SOUND\_ENTERSA 7**

Play the Scout enter standalone sound

**9.164.2.9 #define SCOUT\_SOUND\_KEYERROR 8**

Play the Scout key error sound

**9.164.2.10 #define SCOUT\_SOUND\_MAIL\_RECEIVED 24**

Play the Scout mail received sound

**9.164.2.11 #define SCOUT\_SOUND\_NONE 9**

Play the Scout none sound

**9.164.2.12 #define SCOUT\_SOUND\_REMOTE 6**

Play the Scout remote sound

**9.164.2.13 #define SCOUT\_SOUND\_SPECIAL1 25**

Play the Scout special 1 sound

**9.164.2.14 #define SCOUT\_SOUND\_SPECIAL2 26**

Play the Scout special 2 sound

**9.164.2.15 #define SCOUT\_SOUND\_SPECIAL3 27**

Play the Scout special 3 sound

**9.164.2.16 #define SCOUT\_SOUND\_TIMER1 21**

Play the Scout timer 1 sound

**9.164.2.17 #define SCOUT\_SOUND\_TIMER2 22**

Play the Scout timer 2 sound

**9.164.2.18 #define SCOUT\_SOUND\_TIMER3 23**

Play the Scout timer 3 sound

**9.164.2.19 #define SCOUT\_SOUND\_TOUCH1\_PRES 10**

Play the Scout touch 1 pressed sound

**9.164.2.20 #define SCOUT\_SOUND\_TOUCH1\_REL 11**

Play the Scout touch 1 released sound

**9.164.2.21 #define SCOUT\_SOUND\_TOUCH2\_PRES 12**

Play the Scout touch 2 pressed sound

**9.164.2.22 #define SCOUT\_SOUND\_TOUCH2\_REL 13**

Play the Scout touch 2 released sound

## 9.165 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

### 9.165.1 Detailed Description

Constants for use when choosing standard Scout sound sets.

### 9.165.2 Define Documentation

#### 9.165.2.1 #define SCOUT\_SNDSET\_ALARM 3

Set sound set to alarm

#### 9.165.2.2 #define SCOUT\_SNDSET\_BASIC 1

Set sound set to basic

#### 9.165.2.3 #define SCOUT\_SNDSET\_BUG 2

Set sound set to bug

#### 9.165.2.4 #define SCOUT\_SNDSET\_NONE 0

Set sound set to none

#### 9.165.2.5 #define SCOUT\_SNDSET\_RANDOM 4

Set sound set to random

#### 9.165.2.6 #define SCOUT\_SNDSET\_SCIENCE 5

Set sound set to science

## 9.166 Scout mode constants

Constants for use when setting the scout mode.

### Defines

- `#define SCOUT_MODE_STANDALONE 0`
- `#define SCOUT_MODE_POWER 1`

### 9.166.1 Detailed Description

Constants for use when setting the scout mode.

### 9.166.2 Define Documentation

#### 9.166.2.1 `#define SCOUT_MODE_POWER 1`

Enter power mode

#### Examples:

[ex\\_HTScoutSetScoutMode.nxc](#), and [ex\\_MSScoutSetScoutMode.nxc](#).

#### 9.166.2.2 `#define SCOUT_MODE_STANDALONE 0`

Enter stand alone mode

## 9.167 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

### 9.167.1 Detailed Description

Constants for use when setting the scout motion rule.

### 9.167.2 Define Documentation

#### 9.167.2.1 #define SCOUT\_MR\_CIRCLE\_LEFT 4

Motion rule circle left

#### 9.167.2.2 #define SCOUT\_MR\_CIRCLE\_RIGHT 3

Motion rule circle right

#### 9.167.2.3 #define SCOUT\_MR\_FORWARD 1

Motion rule forward

#### Examples:

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

#### 9.167.2.4 #define SCOUT\_MR\_LOOP\_A 5

Motion rule loop A

#### 9.167.2.5 #define SCOUT\_MR\_LOOP\_AB 7

Motion rule loop A then B

#### 9.167.2.6 #define SCOUT\_MR\_LOOP\_B 6

Motion rule loop B

**9.167.2.7 #define SCOUT\_MR\_NO\_MOTION 0**

Motion rule none

**9.167.2.8 #define SCOUT\_MR\_ZIGZAG 2**

Motion rule zigzag



## 9.168 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`

### 9.168.1 Detailed Description

Constants for use when setting the scout touch rule.

### 9.168.2 Define Documentation

#### 9.168.2.1 `#define SCOUT_TR_AVOID 2`

Touch rule avoid

#### 9.168.2.2 `#define SCOUT_TR_IGNORE 0`

Touch rule ignore

#### 9.168.2.3 `#define SCOUT_TR_OFF_WHEN 4`

Touch rule off when

#### 9.168.2.4 `#define SCOUT_TR_REVERSE 1`

Touch rule reverse

### Examples:

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

#### 9.168.2.5 `#define SCOUT_TR_WAIT_FOR 3`

Touch rule wait for

## 9.169 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`

### 9.169.1 Detailed Description

Constants for use when setting the scout light rule.

### 9.169.2 Define Documentation

#### 9.169.2.1 `#define SCOUT_LR_AVOID 3`

Light rule avoid

#### 9.169.2.2 `#define SCOUT_LR_IGNORE 0`

Light rule ignore

#### Examples:

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

#### 9.169.2.3 `#define SCOUT_LR_OFF_WHEN 5`

Light rule off when

#### 9.169.2.4 `#define SCOUT_LR_SEEK_DARK 2`

Light rule seek dark

#### 9.169.2.5 `#define SCOUT_LR_SEEK_LIGHT 1`

Light rule seek light

#### 9.169.2.6 `#define SCOUT_LR_WAIT_FOR 4`

Light rule wait for

## 9.170 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`

### 9.170.1 Detailed Description

Constants for use when setting the scout transmit rule.

### 9.170.2 Define Documentation

#### 9.170.2.1 `#define SCOUT_TGS_LONG 2`

Transmit level long

#### 9.170.2.2 `#define SCOUT_TGS_MEDIUM 1`

Transmit level medium

#### 9.170.2.3 `#define SCOUT_TGS_SHORT 0`

Transmit level short

### Examples:

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

## 9.171 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`

### 9.171.1 Detailed Description

Constants for use when setting the scout special effect.

### 9.171.2 Define Documentation

#### 9.171.2.1 `#define SCOUT_FXR_ALARM 2`

Alarm special effects

#### 9.171.2.2 `#define SCOUT_FXR_BUG 1`

Bug special effects

#### Examples:

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

#### 9.171.2.3 `#define SCOUT_FXR_NONE 0`

No special effects

#### 9.171.2.4 `#define SCOUT_FXR_RANDOM 3`

Random special effects

#### 9.171.2.5 `#define SCOUT_FXR_SCIENCE 4`

Science special effects

## 9.172 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`

### 9.172.1 Detailed Description

Constants for use when specifying RCX and Scout sources.

## 9.172.2 Define Documentation

### 9.172.2.1 `#define RCX_BatteryLevelSrc 34`

The RCX battery level source

### 9.172.2.2 `#define RCX_ClickCounterSrc 27`

The RCX event click counter source

### 9.172.2.3 `#define RCX_ConstantSrc 2`

The RCX constant value source

#### Examples:

```
ex_HTRCXEvent.nxc,      ex_HTRCXSetEvent.nxc,      ex_HTRCXSetMaxPower.nxc,      ex_
HTRCXSetPower.nxc,      ex_HTScoutSendVLL.nxc,      ex_HTScoutSetEventFeedback.nxc,
ex_HTScoutSetSensorClickTime.nxc,      ex_HTScoutSetSensorHysteresis.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.
```

### 9.172.2.4 `#define RCX_CounterSrc 21`

The RCX counter source

### 9.172.2.5 `#define RCX_DatalogRawDirectSrc 42`

The RCX direct datalog raw source

### 9.172.2.6 `#define RCX_DatalogRawIndirectSrc 41`

The RCX indirect datalog raw source

### 9.172.2.7 `#define RCX_DatalogSrcDirectSrc 38`

The RCX direct datalog source source

### 9.172.2.8 `#define RCX_DatalogSrcIndirectSrc 37`

The RCX indirect datalog source source

### 9.172.2.9 `#define RCX_DatalogValueDirectSrc 40`

The RCX direct datalog value source

**9.172.2.10 #define RCX\_DatalogValueIndirectSrc 39**

The RCX indirect datalog value source

**9.172.2.11 #define RCX\_DurationSrc 31**

The RCX event duration source

**9.172.2.12 #define RCX\_EventStateSrc 25**

The RCX event static source

**9.172.2.13 #define RCX\_FirmwareVersionSrc 35**

The RCX firmware version source

**9.172.2.14 #define RCX\_GlobalMotorStatusSrc 17**

The RCX global motor status source

**9.172.2.15 #define RCX\_HysteresisSrc 30**

The RCX event hysteresis source

**9.172.2.16 #define RCX\_IndirectVarSrc 36**

The RCX indirect variable source

**9.172.2.17 #define RCX\_InputBooleanSrc 13**

The RCX input boolean source

**9.172.2.18 #define RCX\_InputModeSrc 11**

The RCX input mode source

**9.172.2.19 #define RCX\_InputRawSrc 12**

The RCX input raw source

**9.172.2.20 #define RCX\_InputTypeSrc 10**

The RCX input type source

**9.172.2.21 #define RCX\_InputValueSrc 9**

The RCX input value source

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#), [ex\\_MSRCXAddToDatalog.nxc](#), and [ex\\_MSRCXSumVar.nxc](#).

**9.172.2.22 #define RCX\_LowerThresholdSrc 29**

The RCX event lower threshold source

**9.172.2.23 #define RCX\_MessageSrc 15**

The RCX message source

**9.172.2.24 #define RCX\_OutputStatusSrc 3**

The RCX output status source

**9.172.2.25 #define RCX\_ProgramSlotSrc 8**

The RCX program slot source

**9.172.2.26 #define RCX\_RandomSrc 4**

The RCX random number source

**Examples:**

[ex\\_MSRCXSet.nxc](#), and [ex\\_MSRCXSubVar.nxc](#).

**9.172.2.27 #define RCX\_ScoutCounterLimitSrc 22**

The Scout counter limit source

**9.172.2.28 #define RCX\_ScoutEventFBSrc 24**

The Scout event feedback source

**9.172.2.29 #define RCX\_ScoutLightParamsSrc 19**

The Scout light parameters source

**9.172.2.30 #define RCX\_ScoutRulesSrc 18**

The Scout rules source



**9.172.2.31 #define RCX\_ScoutTimerLimitSrc 20**

The Scout timer limit source

**9.172.2.32 #define RCX\_TaskEventsSrc 23**

The RCX task events source

**9.172.2.33 #define RCX\_TenMSTimerSrc 26**

The RCX 10ms timer source

**9.172.2.34 #define RCX\_TimerSrc 1**

The RCX timer source

**9.172.2.35 #define RCX\_UARTSetupSrc 33**

The RCX UART setup source

**9.172.2.36 #define RCX\_UpperThresholdSrc 28**

The RCX event upper threshold source

**9.172.2.37 #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](#), [ex\\_MSRCXSgnVar.nxc](#), [ex\\_-](#)  
[MSScoutSetSensorLowerLimit.nxc](#), and [ex\\_MSScoutSetSensorUpperLimit.nxc](#).

**9.172.2.38 #define RCX\_WatchSrc 14**

The RCX watch source

## 9.173 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

### 9.173.1 Detailed Description

Constants for use when specifying RCX and Scout opcodes.

### 9.173.2 Define Documentation

#### 9.173.2.1 #define [RCX\\_AbsVarOp](#) 0x74

Absolute value function

#### 9.173.2.2 #define [RCX\\_AndVarOp](#) 0x84

AND function

**9.173.2.3 #define RCX\_AutoOffOp 0xb1**

Set auto off timer

**9.173.2.4 #define RCX\_BatteryLevelOp 0x30**

Read the battery level

**9.173.2.5 #define RCX\_BootModeOp 0x65**

Set into book mode

**9.173.2.6 #define RCX\_CalibrateEventOp 0x04**

Calibrate event

**9.173.2.7 #define RCX\_ClearAllEventsOp 0x06**

Clear all events

**9.173.2.8 #define RCX\_ClearCounterOp 0xb7**

Clear a counter

**9.173.2.9 #define RCX\_ClearMsgOp 0x90**

Clear message

**9.173.2.10 #define RCX\_ClearSensorOp 0xd1**

Clear a sensor

**9.173.2.11 #define RCX\_ClearSoundOp 0x80**

Clear sound

**9.173.2.12 #define RCX\_ClearTimerOp 0xa1**

Clear a timer

**9.173.2.13 #define RCX\_DatalogOp 0x62**

Datalog the specified source/value

**9.173.2.14 #define RCX\_DecCounterOp 0xa7**

Decrement a counter

**9.173.2.15 #define RCX\_DeleteSubOp 0xc1**

Delete a subroutine

**9.173.2.16 #define RCX\_DeleteSubsOp 0x70**

Delete subroutines

**9.173.2.17 #define RCX\_DeleteTaskOp 0x61**

Delete a task

**9.173.2.18 #define RCX\_DeleteTasksOp 0x40**

Delete tasks

**9.173.2.19 #define RCX\_DirectEventOp 0x03**

Fire an event

**9.173.2.20 #define RCX\_DisplayOp 0x33**

Set LCD display value

**9.173.2.21 #define RCX\_DivVarOp 0x44**

Divide function

**9.173.2.22 #define RCX\_GOutputDirOp 0x77**

Set global motor direction

**9.173.2.23 #define RCX\_GOutputModeOp 0x67**

Set global motor mode

**9.173.2.24 #define RCX\_GOutputPowerOp 0xa3**

Set global motor power levels

**9.173.2.25 #define RCX\_IncCounterOp 0x97**

Increment a counter

**9.173.2.26 #define RCX\_InputModeOp 0x42**

Set the input mode

**9.173.2.27 #define RCX\_InputTypeOp 0x32**

Set the input type

**9.173.2.28 #define RCX\_IRModeOp 0x31**

Set the IR transmit mode

**9.173.2.29 #define RCX\_LightOp 0x87**

Light opcode

**9.173.2.30 #define RCX\_LSblinkTimeOp 0xe3**

Set the light sensor blink time

**9.173.2.31 #define RCX\_LSCalibrateOp 0xc0**

Calibrate the light sensor

**9.173.2.32 #define RCX\_LSHysteresisOp 0xd3**

Set the light sensor hysteresis

**9.173.2.33 #define RCX\_LSLowerThreshOp 0xc3**

Set the light sensor lower threshold

**9.173.2.34 #define RCX\_LSupperThreshOp 0xb3**

Set the light sensor upper threshold

**9.173.2.35 #define RCX\_MessageOp 0xf7**

Set message

**9.173.2.36 #define RCX\_MulVarOp 0x54**

Multiply function

**9.173.2.37 #define RCX\_MuteSoundOp 0xd0**

Mute sound

**9.173.2.38 #define RCX\_OnOffFloatOp 0x21**

Control motor state - on, off, float

**9.173.2.39 #define RCX\_OrVarOp 0x94**

OR function

**9.173.2.40 #define RCX\_OutputDirOp 0xe1**

Set the motor direction

**9.173.2.41 #define RCX\_OutputPowerOp 0x13**

Set the motor power level

**9.173.2.42 #define RCX\_PBTurnOffOp 0x60**

Turn off the brick

**9.173.2.43 #define RCX\_PingOp 0x10**

Ping the brick

**9.173.2.44 #define RCX\_PlaySoundOp 0x51**

Play a sound

**9.173.2.45 #define RCX\_PlayToneOp 0x23**

Play a tone

**9.173.2.46 #define RCX\_PlayToneVarOp 0x02**

Play a tone using a variable

**9.173.2.47 #define RCX\_PollMemoryOp 0x63**

Poll a memory location

**9.173.2.48 #define RCX\_PollOp 0x12**

Poll a source/value combination

**9.173.2.49 #define RCX\_RemoteOp 0xd2**

Execute simulated remote control buttons

**9.173.2.50 #define RCX\_ScoutOp 0x47**

Scout opcode

**9.173.2.51 #define RCX\_ScoutRulesOp 0xd5**

Set Scout rules

**9.173.2.52 #define RCX\_SelectProgramOp 0x91**

Select a program slot

**9.173.2.53 #define RCX\_SendUARTDataOp 0xc2**

Send data via IR using UART settings

**9.173.2.54 #define RCX\_SetCounterOp 0xd4**

Set counter value

**9.173.2.55 #define RCX\_SetDatalogOp 0x52**

Set the datalog size

**9.173.2.56 #define RCX\_SetEventOp 0x93**

Set an event

**9.173.2.57 #define RCX\_SetFeedbackOp 0x83**

Set Scout feedback



**9.173.2.58 #define RCX\_SetPriorityOp 0xd7**

Set task priority

**9.173.2.59 #define RCX\_SetSourceValueOp 0x05**

Set a source/value

**9.173.2.60 #define RCX\_SetTimerLimitOp 0xc4**

Set timer limit

**9.173.2.61 #define RCX\_SetVarOp 0x14**

Set function

**9.173.2.62 #define RCX\_SetWatchOp 0x22**

Set the watch source/value

**9.173.2.63 #define RCX\_SgnVarOp 0x64**

Sign function

**9.173.2.64 #define RCX\_SoundOp 0x57**

Sound opcode

**9.173.2.65 #define RCX\_StartTaskOp 0x71**

Start a task

**9.173.2.66 #define RCX\_StopAllTasksOp 0x50**

Stop all tasks

**9.173.2.67 #define RCX\_StopTaskOp 0x81**

Stop a task

**9.173.2.68 #define RCX\_SubVarOp 0x34**

Subtract function

**9.173.2.69 #define RCX\_SumVarOp 0x24**

Sum function

**9.173.2.70 #define RCX\_UnlockFirmOp 0xa5**

Unlock the firmware

**9.173.2.71 #define RCX\_UnlockOp 0x15**

Unlock the brick

**9.173.2.72 #define RCX\_UnmuteSoundOp 0xe0**

Unmute sound

**9.173.2.73 #define RCX\_UploadDatalogOp 0xa4**

Upload datalog contents

**9.173.2.74 #define RCX\_ViewSourceValOp 0xe5**

View a source/value

**9.173.2.75 #define RCX\_VLLOp 0xe2**

Send visual light link (VLL) data

## 9.174 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.*

### 9.174.1 Detailed Description

Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.

## 9.175 Power Function command constants

Constants that are for sending Power Function commands.

### Defines

- `#define PF_CMD_STOP 0`
- `#define PF_CMD_FWD 1`
- `#define PF_CMD_REV 2`
- `#define PF_CMD_BRAKE 3`

### 9.175.1 Detailed Description

Constants that are for sending Power Function commands.

### 9.175.2 Define Documentation

#### 9.175.2.1 `#define PF_CMD_BRAKE 3`

Power function command brake

#### 9.175.2.2 `#define PF_CMD_FWD 1`

Power function command forward

#### Examples:

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_MSPFComboDirectEx.nxc](#).

#### 9.175.2.3 `#define PF_CMD_REV 2`

Power function command reverse

#### 9.175.2.4 `#define PF_CMD_STOP 0`

Power function command stop

#### Examples:

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_MSPFComboDirectEx.nxc](#).

## 9.176 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`

### 9.176.1 Detailed Description

Constants that are for specifying Power Function channels.

### 9.176.2 Define Documentation

#### 9.176.2.1 `#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_MSPFComboDirectEx.nxc`, `ex_MSPFComboPWM.nxc`, `ex_-`  
`MSPFComboPWMEEx.nxc`, `ex_MSPFSingleOutputCST.nxc`, `ex_MSPFSingleOutputCSTEx.nxc`,  
`ex_MSPFSingleOutputPWM.nxc`, `ex_MSPFSingleOutputPWMEEx.nxc`, `ex_MSPFSinglePin.nxc`,  
`ex_MSPFSinglePinEx.nxc`, `ex_MSPFTrain.nxc`, and `ex_MSPFTrainEx.nxc`.

#### 9.176.2.2 `#define PF_CHANNEL_2 1`

Power function channel 2

#### 9.176.2.3 `#define PF_CHANNEL_3 2`

Power function channel 3

#### 9.176.2.4 `#define PF_CHANNEL_4 3`

Power function channel 4

## 9.177 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`

### 9.177.1 Detailed Description

Constants that are for choosing Power Function modes.

### 9.177.2 Define Documentation

#### 9.177.2.1 `#define PF_MODE_COMBO_DIRECT 1`

Power function mode combo direct

#### 9.177.2.2 `#define PF_MODE_COMBO_PWM 4`

Power function mode combo pulse width modulation (PWM)

#### 9.177.2.3 `#define PF_MODE_SINGLE_OUTPUT_CST 6`

Power function mode single output clear, set, toggle (CST)

#### 9.177.2.4 `#define PF_MODE_SINGLE_OUTPUT_PWM 4`

Power function mode single output pulse width modulation (PWM)

#### 9.177.2.5 `#define PF_MODE_SINGLE_PIN_CONT 2`

Power function mode single pin continuous

#### 9.177.2.6 `#define PF_MODE_SINGLE_PIN_TIME 3`

Power function mode single pin timed

#### 9.177.2.7 `#define PF_MODE_TRAIN 0`

Power function mode IR Train

## 9.178 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`

### 9.178.1 Detailed Description

Constants that are for sending PF/IR Train functions.

### 9.178.2 Define Documentation

#### 9.178.2.1 `#define TRAIN_FUNC_DECR_SPEED 2`

PF/IR Train function decrement speed

#### 9.178.2.2 `#define TRAIN_FUNC_INCR_SPEED 1`

PF/IR Train function increment speed

#### Examples:

`ex_HTIRTrain.nxc`, `ex_HTPFTrain.nxc`, `ex_MSIRTrain.nxc`, `ex_MSIRTrainEx.nxc`, `ex_MSPFTrain.nxc`, and `ex_MSPFTrainEx.nxc`.

#### 9.178.2.3 `#define TRAIN_FUNC_STOP 0`

PF/IR Train function stop

#### 9.178.2.4 `#define TRAIN_FUNC_TOGGLE_LIGHT 4`

PF/IR Train function toggle light

## 9.179 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`

### 9.179.1 Detailed Description

Constants that are for specifying IR Train channels.

### 9.179.2 Define Documentation

#### 9.179.2.1 `#define TRAIN_CHANNEL_1 0`

IR Train channel 1

#### Examples:

[ex\\_HTIRTrain.nxc](#), [ex\\_MSIRTrain.nxc](#), and [ex\\_MSIRTrainEx.nxc](#).

#### 9.179.2.2 `#define TRAIN_CHANNEL_2 1`

IR Train channel 2

#### 9.179.2.3 `#define TRAIN_CHANNEL_3 2`

IR Train channel 3

#### 9.179.2.4 `#define TRAIN_CHANNEL_ALL 3`

IR Train channel all



## 9.180 Power Function output constants

Constants that are for choosing a Power Function output.

### Defines

- `#define PF_OUT_A 0`
- `#define PF_OUT_B 1`

### 9.180.1 Detailed Description

Constants that are for choosing a Power Function output.

### 9.180.2 Define Documentation

#### 9.180.2.1 `#define PF_OUT_A 0`

Power function output A

#### Examples:

`ex_HTPFSingleOutputCST.nxc`, `ex_HTPFSingleOutputPWM.nxc`, `ex_HTPFSinglePin.nxc`, `ex_MSPFSingleOutputCST.nxc`, `ex_MSPFSingleOutputCSTEx.nxc`, `ex_MSPFSingleOutputPWM.nxc`, `ex_MSPFSingleOutputPWMEEx.nxc`, `ex_MSPFSinglePin.nxc`, and `ex_MSPFSinglePinEx.nxc`.

#### 9.180.2.2 `#define PF_OUT_B 1`

Power function output B

## 9.181 Power Function pin constants

Constants that are for choosing a Power Function pin.

### Defines

- `#define PF_PIN_C1 0`
- `#define PF_PIN_C2 1`

### 9.181.1 Detailed Description

Constants that are for choosing a Power Function pin.

### 9.181.2 Define Documentation

#### 9.181.2.1 `#define PF_PIN_C1 0`

Power function pin C1

#### Examples:

[ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFSinglePinEx.nxc](#).

#### 9.181.2.2 `#define PF_PIN_C2 1`

Power function pin C2

## 9.182 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`

### 9.182.1 Detailed Description

Constants that are for sending Power Function single pin functions.

### 9.182.2 Define Documentation

#### 9.182.2.1 `#define PF_FUNC_CLEAR 1`

Power function single pin - clear

#### 9.182.2.2 `#define PF_FUNC_NOCHANGE 0`

Power function single pin - no change

#### 9.182.2.3 `#define PF_FUNC_SET 2`

Power function single pin - set

### Examples:

[ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFSinglePinEx.nxc](#).

#### 9.182.2.4 `#define PF_FUNC_TOGGLE 3`

Power function single pin - toggle

## 9.183 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

### 9.183.1 Detailed Description

Constants that are for specifying Power Function CST options.

### 9.183.2 Define Documentation

#### 9.183.2.1 #define [PF\\_CST\\_CLEAR1\\_CLEAR2](#) 0

Power function CST clear 1 and clear 2

#### 9.183.2.2 #define [PF\\_CST\\_CLEAR1\\_SET2](#) 2

Power function CST clear 1 and set 2

#### 9.183.2.3 #define [PF\\_CST\\_DECREMENT\\_PWM](#) 5

Power function CST decrement PWM

#### 9.183.2.4 #define [PF\\_CST\\_FULL\\_FWD](#) 6

Power function CST full forward

#### 9.183.2.5 #define [PF\\_CST\\_FULL\\_REV](#) 7

Power function CST full reverse

#### 9.183.2.6 #define [PF\\_CST\\_INCREMENT\\_PWM](#) 4

Power function CST increment PWM

**9.183.2.7 #define PF\_CST\_SET1\_CLEAR2 1**

Power function CST set 1 and clear 2

**9.183.2.8 #define PF\_CST\_SET1\_SET2 3**

Power function CST set 1 and set 2

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), and [ex\\_MSPFSingleOutputCSTEx.nxc](#).

**9.183.2.9 #define PF\_CST\_TOGGLE\_DIR 8**

Power function CST toggle direction

## 9.184 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`

### 9.184.1 Detailed Description

Constants that are for specifying Power Function PWM options.

### 9.184.2 Define Documentation

#### 9.184.2.1 `#define PF_PWM_BRAKE 8`

Power function PWM brake

#### 9.184.2.2 `#define PF_PWM_FLOAT 0`

Power function PWM float

#### 9.184.2.3 `#define PF_PWM_FWD1 1`

Power function PWM foward level 1

#### 9.184.2.4 `#define PF_PWM_FWD2 2`

Power function PWM foward level 2

#### 9.184.2.5 `#define PF_PWM_FWD3 3`

Power function PWM foward level 3

**9.184.2.6 #define PF\_PWM\_FWD4 4**

Power function PWM forward level 4

**9.184.2.7 #define PF\_PWM\_FWD5 5**

Power function PWM forward level 5

**Examples:**

[ex\\_HTTPFComboPWM.nxc](#), [ex\\_HTTPFSingleOutputPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#),  
[ex\\_MSPFComboPWMEEx.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), and [ex\\_-  
MSPFSingleOutputPWMEEx.nxc](#).

**9.184.2.8 #define PF\_PWM\_FWD6 6**

Power function PWM forward level 6

**9.184.2.9 #define PF\_PWM\_FWD7 7**

Power function PWM forward level 7

**9.184.2.10 #define PF\_PWM\_REV1 15**

Power function PWM reverse level 1

**9.184.2.11 #define PF\_PWM\_REV2 14**

Power function PWM reverse level 2

**9.184.2.12 #define PF\_PWM\_REV3 13**

Power function PWM reverse level 3

**9.184.2.13 #define PF\_PWM\_REV4 12**

Power function PWM reverse level 4

**Examples:**

[ex\\_HTTPFComboPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#), and [ex\\_MSPFComboPWMEEx.nxc](#).

**9.184.2.14 #define PF\_PWM\_REV5 11**

Power function PWM reverse level 5

**9.184.2.15 #define PF\_PWM\_REV6 10**

Power function PWM reverse level 6

**9.184.2.16 #define PF\_PWM\_REV7 9**

Power function PWM reverse level 7



## 9.185 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.*

### 9.185.1 Detailed Description

Constants that are for use with HiTechnic devices.

## 9.186 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`

### 9.186.1 Detailed Description

Constants that are for use with the HiTechnic IRSeeker2 device.

### 9.186.2 Define Documentation

#### 9.186.2.1 `#define HTIR2_MODE_1200 0`

Set IRSeeker2 to 1200 mode

#### Examples:

[ex\\_SetHTIRSeeker2Mode.nxc](#), and [ex\\_SetSensorBoolean.nxc](#).

#### 9.186.2.2 `#define HTIR2_MODE_600 1`

Set IRSeeker2 to 600 mode

#### 9.186.2.3 `#define HTIR2_REG_AC01 0x4A`

IRSeeker 2 AC 01 register

#### 9.186.2.4 `#define HTIR2_REG_AC02 0x4B`

IRSeeker 2 AC 02 register

**9.186.2.5 #define HTIR2\_REG\_AC03 0x4C**

IRSeeker 2 AC 03 register

**9.186.2.6 #define HTIR2\_REG\_AC04 0x4D**

IRSeeker 2 AC 04 register

**9.186.2.7 #define HTIR2\_REG\_AC05 0x4E**

IRSeeker 2 AC 05 register

**9.186.2.8 #define HTIR2\_REG\_ACDIR 0x49**

IRSeeker 2 AC direction register

**9.186.2.9 #define HTIR2\_REG\_DC01 0x43**

IRSeeker 2 DC 01 register

**9.186.2.10 #define HTIR2\_REG\_DC02 0x44**

IRSeeker 2 DC 02 register

**9.186.2.11 #define HTIR2\_REG\_DC03 0x45**

IRSeeker 2 DC 03 register

**9.186.2.12 #define HTIR2\_REG\_DC04 0x46**

IRSeeker 2 DC 04 register

**9.186.2.13 #define HTIR2\_REG\_DC05 0x47**

IRSeeker 2 DC 05 register

**9.186.2.14 #define HTIR2\_REG\_DCAVG 0x48**

IRSeeker 2 DC average register

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**9.186.2.15 #define HTIR2\_REG\_DCDIR 0x42**

IRSeeker 2 DC direction register

**9.186.2.16 #define HTIR2\_REG\_MODE 0x41**

IRSeeker 2 mode register

## 9.187 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`

### 9.187.1 Detailed Description

Constants that are for use with the HiTechnic IRReceiver device.

### 9.187.2 Define Documentation

#### 9.187.2.1 `#define HT_CH1_A 0`

Use IRReceiver channel 1 output A

#### Examples:

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

#### 9.187.2.2 `#define HT_CH1_B 1`

Use IRReceiver channel 1 output B

#### 9.187.2.3 `#define HT_CH2_A 2`

Use IRReceiver channel 2 output A

#### 9.187.2.4 `#define HT_CH2_B 3`

Use IRReceiver channel 2 output B

#### 9.187.2.5 `#define HT_CH3_A 4`

Use IRReceiver channel 3 output A

**9.187.2.6 #define HT\_CH3\_B 5**

Use IRReceiver channel 3 output B

**9.187.2.7 #define HT\_CH4\_A 6**

Use IRReceiver channel 4 output A

**9.187.2.8 #define HT\_CH4\_B 7**

Use IRReceiver channel 4 output B

## 9.188 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`

### 9.188.1 Detailed Description

Constants that are for use with the HiTechnic Color2 device.

### 9.188.2 Define Documentation

#### 9.188.2.1 `#define HT_CMD_COLOR2_50HZ 0x35`

Set the Color2 sensor to 50Hz mode

#### 9.188.2.2 `#define HT_CMD_COLOR2_60HZ 0x36`

Set the Color2 sensor to 60Hz mode

#### 9.188.2.3 `#define HT_CMD_COLOR2_ACTIVE 0x00`

Set the Color2 sensor to active mode

#### Examples:

[ex\\_I2CSendCommand.nxc](#), [ex\\_I2CSendCommandEx.nxc](#), and [ex\\_SetHTColor2Mode.nxc](#).

#### 9.188.2.4 `#define HT_CMD_COLOR2_BLCAL 0x42`

Set the Color2 sensor to black level calibration mode

#### 9.188.2.5 `#define HT_CMD_COLOR2_FAR 0x46`

Set the Color2 sensor to far mode

**9.188.2.6 #define HT\_CMD\_COLOR2\_LED\_HI 0x48**

Set the Color2 sensor to LED high mode

**9.188.2.7 #define HT\_CMD\_COLOR2\_LED\_LOW 0x4C**

Set the Color2 sensor to LED low mode

**9.188.2.8 #define HT\_CMD\_COLOR2\_NEAR 0x4E**

Set the Color2 sensor to near mode

**9.188.2.9 #define HT\_CMD\_COLOR2\_PASSIVE 0x01**

Set the Color2 sensor to passive mode

**9.188.2.10 #define HT\_CMD\_COLOR2\_RAW 0x03**

Set the Color2 sensor to raw mode

**9.188.2.11 #define HT\_CMD\_COLOR2\_WBCAL 0x43**

Set the Color2 sensor to white level calibration mode



## 9.189 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.*

### Defines

- #define [MS\\_CMD\\_ENERGIZED](#) 0x45
- #define [MS\\_CMD\\_DEENERGIZED](#) 0x44
- #define [MS\\_CMD\\_ADPA\\_ON](#) 0x4E
- #define [MS\\_CMD\\_ADPA\\_OFF](#) 0x4F

### 9.189.1 Detailed Description

Constants that are for use with MindSensors devices.

### 9.189.2 Define Documentation

#### 9.189.2.1 #define MS\_CMD\_ADPA\_OFF 0x4F

Turn MindSensors ADPA mode off

#### 9.189.2.2 #define MS\_CMD\_ADPA\_ON 0x4E

Turn MindSensors ADPA mode on

#### 9.189.2.3 #define MS\_CMD\_DEENERGIZED 0x44

De-energize the MindSensors device

#### 9.189.2.4 #define MS\_CMD\_ENERGIZED 0x45

Energize the MindSensors device

## 9.190 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

### 9.190.1 Detailed Description

Constants that are for use with the MindSensors DIST-Nx device.

### 9.190.2 Define Documentation

#### 9.190.2.1 #define `DIST_CMD_CUSTOM` 0x35

Set the DIST-Nx to a custom mode

#### 9.190.2.2 #define `DIST_CMD_GP2D12` 0x31

Set the DIST-Nx to GP2D12 mode

#### 9.190.2.3 #define `DIST_CMD_GP2D120` 0x32

Set the DIST-Nx to GP2D120 mode

#### 9.190.2.4 #define `DIST_CMD_GP2YA02` 0x34

Set the DIST-Nx to GP2YA02 mode

#### 9.190.2.5 #define `DIST_CMD_GP2YA21` 0x33

Set the DIST-Nx to GP2YA21 mode

**9.190.2.6 #define DIST\_REG\_DIST 0x42**

The DIST-Nx distance register

**9.190.2.7 #define DIST\_REG\_DIST1 0x58**

The DIST-Nx distance 1 register

**9.190.2.8 #define DIST\_REG\_DIST\_MAX 0x54**

The DIST-Nx maximum distance register

**9.190.2.9 #define DIST\_REG\_DIST\_MIN 0x52**

The DIST-Nx minimum distance register

**9.190.2.10 #define DIST\_REG\_MODULE\_TYPE 0x50**

The DIST-Nx module type register

**9.190.2.11 #define DIST\_REG\_NUM\_POINTS 0x51**

The DIST-Nx number of points register

**9.190.2.12 #define DIST\_REG\_VOLT 0x44**

The DIST-Nx voltage register

**9.190.2.13 #define DIST\_REG\_VOLT1 0x56**

The DIST-Nx voltage 1 register

## 9.191 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`

#### 9.191.1 Detailed Description

Constants that are for use with the MindSensors PSP-Nx device.

#### 9.191.2 Define Documentation

##### 9.191.2.1 `#define PSP_CMD_ANALOG 0x73`

Set the PSP-Nx to analog mode

##### 9.191.2.2 `#define PSP_CMD_DIGITAL 0x41`

Set the PSP-Nx to digital mode

##### 9.191.2.3 `#define PSP_REG_BTNSET1 0x42`

The PSP-Nx button set 1 register

##### 9.191.2.4 `#define PSP_REG_BTNSET2 0x43`

The PSP-Nx button set 2 register

**9.191.2.5 #define PSP\_REG\_XLEFT 0x44**

The PSP-Nx X left register

**9.191.2.6 #define PSP\_REG\_XRIGHT 0x46**

The PSP-Nx X right register

**9.191.2.7 #define PSP\_REG\_YLEFT 0x45**

The PSP-Nx Y left register

**9.191.2.8 #define PSP\_REG\_YRIGHT 0x47**

The PSP-Nx Y right register

## 9.192 MindSensors PSP-Nx button set 1 constants

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

### Defines

- #define `PSP_BTNSET1_LEFT` 0x01
- #define `PSP_BTNSET1_DOWN` 0x02
- #define `PSP_BTNSET1_RIGHT` 0x04
- #define `PSP_BTNSET1_UP` 0x08
- #define `PSP_BTNSET1_R3` 0x20
- #define `PSP_BTNSET1_L3` 0x40

### 9.192.1 Detailed Description

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

### 9.192.2 Define Documentation

#### 9.192.2.1 #define `PSP_BTNSET1_DOWN` 0x02

The PSP-Nx button set 1 down arrow

#### 9.192.2.2 #define `PSP_BTNSET1_L3` 0x40

The PSP-Nx button set 1 L3

#### 9.192.2.3 #define `PSP_BTNSET1_LEFT` 0x01

The PSP-Nx button set 1 left arrow

#### 9.192.2.4 #define `PSP_BTNSET1_R3` 0x20

The PSP-Nx button set 1 R3

#### 9.192.2.5 #define `PSP_BTNSET1_RIGHT` 0x04

The PSP-Nx button set 1 right arrow

#### 9.192.2.6 #define `PSP_BTNSET1_UP` 0x08

The PSP-Nx button set 1 up arrow

## 9.193 MindSensors PSP-Nx button set 2 constants

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

### Defines

- #define `PSP_BTNSET2_SQUARE` 0x01
- #define `PSP_BTNSET2_CROSS` 0x02
- #define `PSP_BTNSET2_CIRCLE` 0x04
- #define `PSP_BTNSET2_TRIANGLE` 0x08
- #define `PSP_BTNSET2_R1` 0x10
- #define `PSP_BTNSET2_L1` 0x20
- #define `PSP_BTNSET2_R2` 0x40
- #define `PSP_BTNSET2_L2` 0x80

### 9.193.1 Detailed Description

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

### 9.193.2 Define Documentation

#### 9.193.2.1 #define `PSP_BTNSET2_CIRCLE` 0x04

The PSP-Nx button set 2 circle

#### 9.193.2.2 #define `PSP_BTNSET2_CROSS` 0x02

The PSP-Nx button set 2 cross

#### 9.193.2.3 #define `PSP_BTNSET2_L1` 0x20

The PSP-Nx button set 2 L1

#### 9.193.2.4 #define `PSP_BTNSET2_L2` 0x80

The PSP-Nx button set 2 L2

#### 9.193.2.5 #define `PSP_BTNSET2_R1` 0x10

The PSP-Nx button set 2 R1

#### 9.193.2.6 #define `PSP_BTNSET2_R2` 0x40

The PSP-Nx button set 2 R2

**9.193.2.7 #define PSP\_BTNSET2\_SQUARE 0x01**

The PSP-Nx button set 2 square

**9.193.2.8 #define PSP\_BTNSET2\_TRIANGLE 0x08**

The PSP-Nx button set 2 triangle



## 9.194 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`

### 9.194.1 Detailed Description

Constants that are for use with the MindSensors nRLink device.

### 9.194.2 Define Documentation

#### 9.194.2.1 `#define NRLINK_CMD_2400 0x44`

Set NLink to 2400 baud

#### 9.194.2.2 `#define NRLINK_CMD_4800 0x48`

Set NLink to 4800 baud

#### 9.194.2.3 `#define NRLINK_CMD_FLUSH 0x46`

Flush the NLink

#### 9.194.2.4 `#define NRLINK_CMD_IR_LONG 0x4C`

Set the NLink to long range IR

#### 9.194.2.5 `#define NRLINK_CMD_IR_SHORT 0x53`

Set the NLink to short range IR

**9.194.2.6 #define NRLINK\_CMD\_RUN\_MACRO 0x52**

Run an NRLink macro

**9.194.2.7 #define NRLINK\_CMD\_SET\_PF 0x50**

Set the NRLink to Power Function mode

**9.194.2.8 #define NRLINK\_CMD\_SET\_RCX 0x58**

Set the NRLink to RCX mode

**9.194.2.9 #define NRLINK\_CMD\_SET\_TRAIN 0x54**

Set the NRLink to IR Train mode

**9.194.2.10 #define NRLINK\_CMD\_TX\_RAW 0x55**

Set the NRLink to transmit raw bytes

**9.194.2.11 #define NRLINK\_REG\_BYTES 0x40**

The NRLink bytes register

**9.194.2.12 #define NRLINK\_REG\_DATA 0x42**

The NRLink data register

**9.194.2.13 #define NRLINK\_REG\_EEPROM 0x50**

The NRLink eeprom register

## 9.195 Data type limits

Constants that define various data type limits.

### Defines

- #define CHAR\_BIT 8
- #define SCHAR\_MIN -127
- #define SCHAR\_MAX 127
- #define UCHAR\_MAX 255
- #define CHAR\_MIN -127
- #define CHAR\_MAX 127
- #define SHRT\_MIN -32767
- #define SHRT\_MAX 32767
- #define USHRT\_MAX 65535
- #define INT\_MIN -32767
- #define INT\_MAX 32767
- #define UINT\_MAX 65535
- #define LONG\_MIN -2147483647
- #define LONG\_MAX 2147483647
- #define ULONG\_MAX 4294967295
- #define RAND\_MAX 32768

### 9.195.1 Detailed Description

Constants that define various data type limits.

### 9.195.2 Define Documentation

#### 9.195.2.1 #define CHAR\_BIT 8

The number of bits in the char type

#### 9.195.2.2 #define CHAR\_MAX 127

The maximum value of the char type

#### 9.195.2.3 #define CHAR\_MIN -127

The minimum value of the char type

#### 9.195.2.4 #define INT\_MAX 32767

The maximum value of the int type

#### 9.195.2.5 #define INT\_MIN -32767

The minimum value of the int type

**9.195.2.6 #define LONG\_MAX 2147483647**

The maximum value of the long type

**9.195.2.7 #define LONG\_MIN -2147483647**

The minimum value of the long type

**9.195.2.8 #define RAND\_MAX 32768**

The maximum unsigned int random number returned by [rand](#)

**9.195.2.9 #define SCHAR\_MAX 127**

The maximum value of the signed char type

**9.195.2.10 #define SCHAR\_MIN -127**

The minimum value of the signed char type

**9.195.2.11 #define SHRT\_MAX 32767**

The maximum value of the short type

**9.195.2.12 #define SHRT\_MIN -32767**

The minimum value of the short type

**9.195.2.13 #define UCHAR\_MAX 255**

The maximum value of the unsigned char type

**9.195.2.14 #define UINT\_MAX 65535**

The maximum value of the unsigned int type

**9.195.2.15 #define ULONG\_MAX 4294967295**

The maximum value of the unsigned long type

**9.195.2.16 #define USHRT\_MAX 65535**

The maximum value of the unsigned short type

## 9.196 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

### 9.196.1 Detailed Description

Constants that are used to specify the polygon surface begin mode.

### 9.196.2 Define Documentation

#### 9.196.2.1 #define GL\_CIRCLE 4

Use circle mode.

#### Examples:

[glCircleDemo.nxc](#).

#### 9.196.2.2 #define GL\_LINE 2

Use line mode.

#### 9.196.2.3 #define GL\_POINT 3

Use point mode.

#### 9.196.2.4 #define GL\_POLYGON 1

Use polygon mode.

#### Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

## 9.197 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`

### 9.197.1 Detailed Description

Constants that are used to specify a graphics library action.

### 9.197.2 Define Documentation

#### 9.197.2.1 `#define GL_ROTATE_X 4`

Rotate around the X axis.

##### Examples:

[glRotateDemo.nxc](#).

#### 9.197.2.2 `#define GL_ROTATE_Y 5`

Rotate around the Y axis.

##### Examples:

[glRotateDemo.nxc](#).

#### 9.197.2.3 `#define GL_ROTATE_Z 6`

Rotate around the Z axis.

#### 9.197.2.4 `#define GL_SCALE_X 7`

Scale along the X axis.

##### Examples:

[glScaleDemo.nxc](#).

**9.197.2.5 #define GL\_SCALE\_Y 8**

Scale along the Y axis.

**9.197.2.6 #define GL\_SCALE\_Z 9**

Scale along the Z axis.

**9.197.2.7 #define GL\_TRANSLATE\_X 1**

Translate along the X axis.

**Examples:**

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

**9.197.2.8 #define GL\_TRANSLATE\_Y 2**

Translate along the Y axis.

**Examples:**

[glTranslateDemo.nxc](#).

**9.197.2.9 #define GL\_TRANSLATE\_Z 3**

Translate along the Z axis.

**Examples:**

[glTranslateDemo.nxc](#).

## 9.198 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`

### 9.198.1 Detailed Description

Constants that are used to configure the graphics library settings.

### 9.198.2 Define Documentation

#### 9.198.2.1 `#define GL_CAMERA_DEPTH 3`

Set the camera depth.

#### 9.198.2.2 `#define GL_CIRCLE_SIZE 1`

Set the circle size.

#### 9.198.2.3 `#define GL_CULL_MODE 2`

Set the cull mode.

#### Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

#### 9.198.2.4 `#define GL_ZOOM_FACTOR 4`

Set the zoom factor.



## 9.199 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

### 9.199.1 Detailed Description

Constants to use when setting the graphics library cull mode.

### 9.199.2 Define Documentation

#### 9.199.2.1 #define GL\_CULL\_BACK 2

Cull lines in back.

#### 9.199.2.2 #define GL\_CULL\_FRONT 3

Cull lines in front.

#### 9.199.2.3 #define GL\_CULL\_NONE 4

Do not cull any lines.

### Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).



# Chapter 10

## Data Structure Documentation

### 10.1 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](#)

#### 10.1.1 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](#).

#### 10.1.2 Field Documentation

##### 10.1.2.1 int ColorSensorReadType::ColorValue

The color value returned by the sensor. See the [Color values](#) group.

**Examples:**

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

**10.1.2.2 bool ColorSensorReadType::Invalid**

Are the sensor values valid?

**10.1.2.3 unsigned int ColorSensorReadType::NormalizedArray[]**

Normalized color values returned by the sensor. See the [Color sensor array indices](#) group.

**10.1.2.4 byte ColorSensorReadType::Port**

The sensor port. See the constants in the [Input port constants](#) group.

**Examples:**

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

**10.1.2.5 unsigned int ColorSensorReadType::RawArray[]**

Raw color values returned by the sensor. See the [Color sensor array indices](#) group.

**10.1.2.6 char ColorSensorReadType::Result**

The function call result. [NO\\_ERR](#) means it succeeded.

**Examples:**

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

**10.1.2.7 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](#)

## 10.2 CommBTCheckStatusType Struct Reference

Parameters for the CommBTCheckStatus system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Connection](#)

### 10.2.1 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.](#)

### 10.2.2 Field Documentation

#### 10.2.2.1 byte CommBTCheckStatusType::Connection

The connection to check.

**Examples:**

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

#### 10.2.2.2 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](#)

## 10.3 CommBTConnectionType Struct Reference

Parameters for the CommBTConnection system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [Action](#)
- string [Name](#)
- byte [ConnectionSlot](#)

### 10.3.1 Detailed Description

Parameters for the CommBTConnection system call. This structure is used when calling the [SysCommBT-Connection](#) system call function.

See also:

[SysCommBTConnection\(\)](#)

Examples:

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

### 10.3.2 Field Documentation

#### 10.3.2.1 byte CommBTConnectionType::Action

The connection action (connect or disconnect).

Examples:

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

#### 10.3.2.2 byte CommBTConnectionType::ConnectionSlot

The connection slot to connect or disconnect.

Examples:

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

#### 10.3.2.3 string CommBTConnectionType::Name

The name of the device to connect or disconnect.

Examples:

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

#### 10.3.2.4 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](#)

## 10.4 CommBTONOffType Struct Reference

Parameters for the CommBTONOff system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- bool [PowerState](#)

### 10.4.1 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.](#)

### 10.4.2 Field Documentation

#### 10.4.2.1 bool CommBTONOffType::PowerState

If true then turn on bluetooth, otherwise, turn it off.

**Examples:**

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

#### 10.4.2.2 unsigned int CommBTONOffType::Result

The function call result.

**Examples:**

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)



## 10.5 CommBTWriteType Struct Reference

Parameters for the CommBTWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Connection](#)
- byte [Buffer](#) []

### 10.5.1 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](#).

### 10.5.2 Field Documentation

#### 10.5.2.1 byte CommBTWriteType::Buffer[]

The data to write to the connection.

**Examples:**

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

#### 10.5.2.2 byte CommBTWriteType::Connection

The connection to use.

**Examples:**

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

#### 10.5.2.3 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](#)

## 10.6 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](#)

### 10.6.1 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](#).

**10.6.2 Field Documentation****10.6.2.1 byte CommExecuteFunctionType::Cmd**

The command to execute.

**Examples:**

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

**10.6.2.2 string CommExecuteFunctionType::Name**

The name parameter, see table.

**10.6.2.3 byte CommExecuteFunctionType::Param1**

The first parameter, see table.

**10.6.2.4 byte CommExecuteFunctionType::Param2**

The second parameter, see table.

**10.6.2.5 byte CommExecuteFunctionType::Param3**

The third parameter, see table.

**10.6.2.6 unsigned int CommExecuteFunctionType::Result**

The function call result. Possible values include [Loader module error codes](#).

**10.6.2.7 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](#)

## 10.7 CommHSCheckStatusType Struct Reference

Parameters for the CommHSCheckStatus system call.

```
#include <NXCDefs.h>
```

### Data Fields

- bool [SendingData](#)
- bool [DataAvailable](#)

### 10.7.1 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.](#)

### 10.7.2 Field Documentation

#### 10.7.2.1 bool CommHSCheckStatusType::DataAvailable

Is data available for reading?

**Examples:**

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

#### 10.7.2.2 bool CommHSCheckStatusType::SendingData

Is data currently being sent?

**Examples:**

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.8 CommHSControlType Struct Reference

Parameters for the CommHSControl system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Command](#)
- byte [BaudRate](#)
- unsigned int [Mode](#)

### 10.8.1 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](#).

### 10.8.2 Field Documentation

#### 10.8.2.1 byte CommHSControlType::BaudRate

The hi-speed port baud rate. See [Hi-speed port baud rate constants](#).

#### 10.8.2.2 byte CommHSControlType::Command

The hi-speed port configuration command. See [Hi-speed port SysCommHSControl constants](#).

**Examples:**

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

#### 10.8.2.3 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](#).

#### 10.8.2.4 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](#)

## 10.9 CommHSReadWriteType Struct Reference

Parameters for the CommHSReadWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Status](#)
- byte [Buffer](#) []

### 10.9.1 Detailed Description

Parameters for the CommHSReadWrite system call. This structure is used when calling the [SysCommH-  
SRead](#) and [SysCommHSWrite](#) system call functions.

**See also:**

[SysCommHSRead\(\)](#), [SysCommHSWrite\(\)](#)

**Examples:**

[ex\\_SysCommHSRead.nxc](#), and [ex\\_SysCommHSWrite.nxc](#).

### 10.9.2 Field Documentation

#### 10.9.2.1 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](#).

#### 10.9.2.2 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](#)

## 10.10 CommLSCheckStatusType Struct Reference

Parameters for the CommLSCheckStatus system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [BytesReady](#)

### 10.10.1 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](#).

### 10.10.2 Field Documentation

#### 10.10.2.1 byte CommLSCheckStatusType::BytesReady

The number of bytes ready to read from the specified port.

#### 10.10.2.2 byte CommLSCheckStatusType::Port

The port to which the I2C device is connected.

**Examples:**

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

#### 10.10.2.3 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](#)



## 10.11 CommLSReadType Struct Reference

Parameters for the CommLSRead system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [BufferLen](#)

### 10.11.1 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](#).

### 10.11.2 Field Documentation

#### 10.11.2.1 byte CommLSReadType::Buffer[]

The buffer used to store the bytes read from the I2C device.

**Examples:**

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

#### 10.11.2.2 byte CommLSReadType::BufferLen

The size of the output buffer on input. This field is not updated during the function call.

**Examples:**

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

#### 10.11.2.3 byte CommLSReadType::Port

The port to which the I2C device is connected.

**Examples:**

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

#### 10.11.2.4 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](#)

## 10.12 CommLSWriteExType Struct Reference

Parameters for the CommLSWriteEx system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)
- bool [NoRestartOnRead](#)

### 10.12.1 Detailed Description

Parameters for the CommLSWriteEx system call. This structure is used when calling the [SysCommLSWriteEx](#) system call function.

**See also:**

[SysCommLSWriteEx\(\)](#)

**Examples:**

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

### 10.12.2 Field Documentation

#### 10.12.2.1 byte CommLSWriteExType::Buffer[]

The buffer written to the I2C device.

**Examples:**

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

#### 10.12.2.2 bool CommLSWriteExType::NoRestartOnRead

Should a restart occur before reading from the device?

**Examples:**

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

#### 10.12.2.3 byte CommLSWriteExType::Port

The port to which the I2C device is connected.

**Examples:**

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

#### 10.12.2.4 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](#).

#### 10.12.2.5 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](#)

## 10.13 CommLSWriteType Struct Reference

Parameters for the CommLSWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)

### 10.13.1 Detailed Description

Parameters for the CommLSWrite system call. This structure is used when calling the [SysCommLSWrite](#) system call function.

**See also:**

[SysCommLSWrite\(\)](#)

**Examples:**

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

### 10.13.2 Field Documentation

#### 10.13.2.1 byte CommLSWriteType::Buffer[]

The buffer containing data to be written to the I2C device.

**Examples:**

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

#### 10.13.2.2 byte CommLSWriteType::Port

The port to which the I2C device is connected.

**Examples:**

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

#### 10.13.2.3 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](#).

#### 10.13.2.4 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\\_syscommlswrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.14 ComputeCalibValueType Struct Reference

Parameters for the ComputeCalibValue system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- string [Name](#)
- unsigned int [RawVal](#)

### 10.14.1 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](#).

### 10.14.2 Field Documentation

#### 10.14.2.1 string ComputeCalibValueType::Name

The name of the sensor calibration cache.

**Todo**

?,

**Examples:**

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

#### 10.14.2.2 unsigned int ComputeCalibValueType::RawVal

The raw value.

**Todo**

?,

**Examples:**

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

### 10.14.2.3 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](#)



## 10.15 DatalogGetTimesType Struct Reference

Parameters for the DatalogGetTimes system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned long [SyncTime](#)
- unsigned long [SyncTick](#)

### 10.15.1 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.](#)

### 10.15.2 Field Documentation

#### 10.15.2.1 unsigned long DatalogGetTimesType::SyncTick

The datalog synchronized tick.

**Examples:**

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

#### 10.15.2.2 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](#)

## 10.16 DatalogWriteType Struct Reference

Parameters for the DatalogWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Message](#) []

### 10.16.1 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.](#)

### 10.16.2 Field Documentation

#### 10.16.2.1 byte DatalogWriteType::Message[ ]

A buffer containing data to write to the datalog.

**Examples:**

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

#### 10.16.2.2 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](#)

## 10.17 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](#)

### 10.17.1 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_HORIZONTAL_LINE	draw horizontal line	(true/false,X1,Y1,X2,x)
DISPLAY_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_LINE	erase a single line	(x,LINE,x,x,x)
DISPLAY_FILL_REGION	fill screen region	(true/false,X1,Y1,X2,Y2)
DISPLAY_FILLED_FRAME	draw a frame (on / off)	(true/false,X1,Y1,X2,Y2)

See also:

[SysDisplayExecuteFunction\(\)](#)

Examples:

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

### 10.17.2 Field Documentation

#### 10.17.2.1 byte DisplayExecuteFunctionType::Cmd

The command to execute.

Examples:

[ex\\_dispfunc.nxc](#), and [ex\\_sysdisplayexecutefunction.nxc](#).

**10.17.2.2 bool DisplayExecuteFunctionType::On**

The On parameter, see table.

**Examples:**

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

**10.17.2.3 byte DisplayExecuteFunctionType::Status**

The function call result, always [NO\\_ERR](#).

**10.17.2.4 byte DisplayExecuteFunctionType::X1**

The X1 parameter, see table.

**Examples:**

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

**10.17.2.5 byte DisplayExecuteFunctionType::X2**

The X2 parameter, see table.

**Examples:**

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

**10.17.2.6 byte DisplayExecuteFunctionType::Y1**

The Y1 parameter, see table.

**Examples:**

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

**10.17.2.7 byte DisplayExecuteFunctionType::Y2**

The Y2 parameter, see table.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.18 div\_t Struct Reference

Output type of the div function.

```
#include <NXCDefs.h>
```

### Data Fields

- [int quot](#)
- [int rem](#)

### 10.18.1 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.](#)

### 10.18.2 Field Documentation

#### 10.18.2.1 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.](#)

#### 10.18.2.2 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](#)

## 10.19 DrawCircleType Struct Reference

Parameters for the DrawCircle system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Center
- byte [Size](#)
- unsigned long [Options](#)

### 10.19.1 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](#).

### 10.19.2 Field Documentation

#### 10.19.2.1 LocationType DrawCircleType::Center

The location of the circle center.

Examples:

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

#### 10.19.2.2 unsigned long DrawCircleType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.19.2.3 char DrawCircleType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 10.19.2.4 byte DrawCircleType::Size

The circle radius.

**Examples:**

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.20 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](#)

### 10.20.1 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.](#)

### 10.20.2 Field Documentation

#### 10.20.2.1 LocationType DrawEllipseType::Center

The location of the ellipse center.

Examples:

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

#### 10.20.2.2 unsigned long DrawEllipseType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.20.2.3 char DrawEllipseType::Result

The function call result. [NO\\_ERR](#) means it succeeded.



#### 10.20.2.4 byte DrawEllipseType::SizeX

The horizontal ellipse radius.

**Examples:**

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

#### 10.20.2.5 byte DrawEllipseType::SizeY

The vertical ellipse radius.

**Examples:**

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.21 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](#)

### 10.21.1 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](#).

### 10.21.2 Field Documentation

#### 10.21.2.1 string DrawFontType::Filename

The filename of the RIC-based font file.

Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

#### 10.21.2.2 LocationType DrawFontType::Location

The location in X, LCD line number coordinates.

Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

#### 10.21.2.3 unsigned long DrawFontType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex\\_dispftout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

#### 10.21.2.4 char DrawFontType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 10.21.2.5 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](#)

## 10.22 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](#)

### 10.22.1 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](#).

### 10.22.2 Field Documentation

#### 10.22.2.1 byte DrawGraphicArrayType::Data[ ]

A byte array containing the RIC opcodes. [RIC Macro Wrappers](#)

Examples:

[ex\\_dispgout.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

#### 10.22.2.2 LocationType DrawGraphicArrayType::Location

The location on screen.

Examples:

[ex\\_dispgout.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

### 10.22.2.3 unsigned long DrawGraphicArrayType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

#### Examples:

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

### 10.22.2.4 char DrawGraphicArrayType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

### 10.22.2.5 long DrawGraphicArrayType::Variables[ ]

The variables passed as RIC arguments.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.23 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](#)

### 10.23.1 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.](#)

### 10.23.2 Field Documentation

#### 10.23.2.1 string DrawGraphicType::Filename

The RIC file name.

Examples:

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

#### 10.23.2.2 LocationType DrawGraphicType::Location

The location on screen.

Examples:

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

### 10.23.2.3 unsigned long DrawGraphicType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

**Examples:**

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

### 10.23.2.4 char DrawGraphicType::Result

The function call result. Possible values include [Loader module error codes](#), [ERR\\_FILE](#), and [NO\\_ERR](#).

### 10.23.2.5 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](#)

## 10.24 DrawLineType Struct Reference

Parameters for the DrawLine system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) StartLoc
- [LocationType](#) EndLoc
- unsigned long [Options](#)

### 10.24.1 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.](#)

### 10.24.2 Field Documentation

#### 10.24.2.1 LocationType DrawLineType::EndLoc

The location of the ending point.

Examples:

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

#### 10.24.2.2 unsigned long DrawLineType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.24.2.3 char DrawLineType::Result

The function call result. [NO\\_ERR](#) means it succeeded.



#### 10.24.2.4 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](#)

## 10.25 DrawPointType Struct Reference

Parameters for the DrawPoint system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Location
- unsigned long [Options](#)

### 10.25.1 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.](#)

### 10.25.2 Field Documentation

#### 10.25.2.1 LocationType DrawPointType::Location

The point location on screen.

Examples:

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

#### 10.25.2.2 unsigned long DrawPointType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.25.2.3 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](#)

## 10.26 DrawPolygonType Struct Reference

Parameters for the DrawPolygon system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Points []
- unsigned long [Options](#)

### 10.26.1 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](#).

### 10.26.2 Field Documentation

#### 10.26.2.1 unsigned long DrawPolygonType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.26.2.2 LocationType DrawPolygonType::Points[ ]

An array of [LocationType](#) structures which define the polygon's shape.

Examples:

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

#### 10.26.2.3 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](#)

## 10.27 DrawRectType Struct Reference

Parameters for the DrawRect system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Location
- [SizeType](#) Size
- unsigned long [Options](#)

### 10.27.1 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](#).

### 10.27.2 Field Documentation

#### 10.27.2.1 LocationType DrawRectType::Location

The top left corner location.

Examples:

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

#### 10.27.2.2 unsigned long DrawRectType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.27.2.3 char DrawRectType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 10.27.2.4 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](#)

## 10.28 DrawTextType Struct Reference

Parameters for the DrawText system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- [LocationType](#) Location
- string [Text](#)
- unsigned long [Options](#)

### 10.28.1 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](#).

### 10.28.2 Field Documentation

#### 10.28.2.1 LocationType DrawTextType::Location

The location in X, LCD line number coordinates.

Examples:

[ex\\_syscall.nxc](#), and [ex\\_sysdrawtext.nxc](#).

#### 10.28.2.2 unsigned long DrawTextType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

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

#### 10.28.2.3 char DrawTextType::Result

The function call result. [NO\\_ERR](#) means it succeeded.

#### 10.28.2.4 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](#)

## 10.29 FileCloseType Struct Reference

Parameters for the FileClose system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)

### 10.29.1 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](#).

### 10.29.2 Field Documentation

#### 10.29.2.1 byte FileCloseType::FileHandle

The file handle to close.

**Examples:**

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

#### 10.29.2.2 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](#)



## 10.30 FileDeleteType Struct Reference

Parameters for the FileDelete system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- string [Filename](#)

### 10.30.1 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.](#)

### 10.30.2 Field Documentation

#### 10.30.2.1 string FileDeleteType::Filename

The name of the file to delete.

**Examples:**

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

#### 10.30.2.2 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](#)

## 10.31 FileFindType Struct Reference

Parameters for the FileFind system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Filename](#)
- unsigned long [Length](#)

### 10.31.1 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](#).

### 10.31.2 Field Documentation

#### 10.31.2.1 byte FileFindType::FileHandle

The returned file handle to be used to continue iterations. Close it after usage.

**Examples:**

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

#### 10.31.2.2 string FileFindType::Filename

The pattern to match file name, then the returned found file name.

**Examples:**

[ex\\_sysfilefindfirst.nxc](#), and [ex\\_sysfilefindnext.nxc](#).

#### 10.31.2.3 unsigned long FileFindType::Length

The found file length.

#### 10.31.2.4 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](#)

## 10.32 FileOpenType Struct Reference

Parameters for the FileOpen system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Filename](#)
- unsigned long [Length](#)

### 10.32.1 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](#).

### 10.32.2 Field Documentation

#### 10.32.2.1 byte FileOpenType::FileHandle

The returned file handle to use for subsequent file operations.

#### 10.32.2.2 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](#).

#### 10.32.2.3 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](#).

**10.32.2.4 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](#)

## 10.33 FileReadWriteType Struct Reference

Parameters for the FileReadWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Buffer](#)
- unsigned long [Length](#)

### 10.33.1 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](#).

### 10.33.2 Field Documentation

#### 10.33.2.1 string FileReadWriteType::Buffer

The buffer to store read bytes or containing bytes to write.

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

#### 10.33.2.2 byte FileReadWriteType::FileHandle

The file handle to access.

**Examples:**

[ex\\_sysfileread.nxc](#), and [ex\\_sysfilewrite.nxc](#).

#### 10.33.2.3 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](#).

#### 10.33.2.4 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](#)

## 10.34 FileRenameType Struct Reference

Parameters for the FileRename system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- string [OldFilename](#)
- string [NewFilename](#)

### 10.34.1 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.](#)

### 10.34.2 Field Documentation

#### 10.34.2.1 string FileRenameType::NewFilename

The new name to give to the file.

**Examples:**

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

#### 10.34.2.2 string FileRenameType::OldFilename

The name of the file to be renamed.

**Examples:**

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

#### 10.34.2.3 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](#)

## 10.35 FileResizeType Struct Reference

Parameters for the FileResize system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- unsigned int [NewSize](#)

### 10.35.1 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](#).

### 10.35.2 Field Documentation

#### 10.35.2.1 byte FileResizeType::FileHandle

The handle of the file to resize.

**Examples:**

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

#### 10.35.2.2 unsigned int FileResizeType::NewSize

The new file size.

**Examples:**

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

#### 10.35.2.3 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](#)

## 10.36 FileResolveHandleType Struct Reference

Parameters for the FileResolveHandle system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- bool [WriteHandle](#)
- string [Filename](#)

### 10.36.1 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](#).

### 10.36.2 Field Documentation

#### 10.36.2.1 byte FileResolveHandleType::FileHandle

The returned resolved file handle.

#### 10.36.2.2 string FileResolveHandleType::Filename

The name of the file for which to resolve a handle.

Examples:

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

#### 10.36.2.3 unsigned int FileResolveHandleType::Result

The function call result. Possible values include [LDR\\_HANDLEALREADYCLOSED](#) and [LDR\\_SUCCESS](#).

Examples:

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

#### 10.36.2.4 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](#)

## 10.37 FileSeekType Struct Reference

Parameters for the FileSeek system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- byte [Origin](#)
- long [Length](#)

### 10.37.1 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](#).

### 10.37.2 Field Documentation

#### 10.37.2.1 byte FileSeekType::FileHandle

The handle of the file to seek in.

**Examples:**

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

#### 10.37.2.2 long FileSeekType::Length

The offset from the origin to seek to.

**Examples:**

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

#### 10.37.2.3 byte FileSeekType::Origin

The origin of the file seek operation. See [fseek origin constants](#).

**Examples:**

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

#### 10.37.2.4 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](#)

## 10.38 GetStartTickType Struct Reference

Parameters for the GetStartTick system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned long [Result](#)

### 10.38.1 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](#).

### 10.38.2 Field Documentation

#### 10.38.2.1 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](#)

## 10.39 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](#) []

### 10.39.1 Detailed Description

Parameters for the IOMapReadByID system call. This structure is used when calling the [SysIOMapReadByID](#) system call function.

See also:

[SysIOMapReadByID\(\)](#)

Examples:

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

### 10.39.2 Field Documentation

#### 10.39.2.1 byte IOMapReadByIDType::Buffer[]

The buffer used to store read bytes.

Examples:

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

#### 10.39.2.2 unsigned int IOMapReadByIDType::Count

The number of bytes to read.

Examples:

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

#### 10.39.2.3 unsigned long IOMapReadByIDType::ModuleID

The identifier of the module to read from. See the [NXT firmware module IDs](#) group.

Examples:

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).



#### 10.39.2.4 unsigned int IOMapReadByIDType::Offset

The offset in the module IOMap where to start reading.

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

#### 10.39.2.5 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](#)

## 10.40 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](#) [ ]

### 10.40.1 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](#).

### 10.40.2 Field Documentation

#### 10.40.2.1 byte IOMapReadType::Buffer[ ]

The buffer used to store read bytes.

#### 10.40.2.2 unsigned int IOMapReadType::Count

The number of bytes to read.

Examples:

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

#### 10.40.2.3 string IOMapReadType::ModuleName

The name of the module to read from. See the [NXT firmware module names](#) group.

Examples:

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

#### 10.40.2.4 unsigned int IOMapReadType::Offset

The offset in the module IOMap where to start reading.

**Examples:**

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

#### 10.40.2.5 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](#)

## 10.41 IOMapWriteByIDType Struct Reference

Parameters for the IOMapWriteByID system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [ModuleID](#)
- unsigned int [Offset](#)
- byte [Buffer](#) []

### 10.41.1 Detailed Description

Parameters for the IOMapWriteByID system call. This structure is used when calling the [SysIOMapWriteByID](#) system call function.

**See also:**

[SysIOMapWriteByID\(\)](#)

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

### 10.41.2 Field Documentation

#### 10.41.2.1 byte IOMapWriteByIDType::Buffer[]

The buffer containing bytes to write.

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

#### 10.41.2.2 unsigned long IOMapWriteByIDType::ModuleID

The identifier of the module to write to. See the [NXT firmware module IDs](#) group.

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

#### 10.41.2.3 unsigned int IOMapWriteByIDType::Offset

The offset in the module IOMap where to start writing.

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

#### 10.41.2.4 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](#)

## 10.42 IOMapWriteType Struct Reference

Parameters for the IOMapWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- string [ModuleName](#)
- unsigned int [Offset](#)
- byte [Buffer](#) []

### 10.42.1 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](#).

### 10.42.2 Field Documentation

#### 10.42.2.1 byte IOMapWriteType::Buffer[]

The buffer containing bytes to write.

**Examples:**

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

#### 10.42.2.2 string IOMapWriteType::ModuleName

The name of the module to write to. See the [NXT firmware module names](#) group.

**Examples:**

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

#### 10.42.2.3 unsigned int IOMapWriteType::Offset

The offset in the module IOMap where to start writing.

**Examples:**

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

#### 10.42.2.4 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](#)

## 10.43 KeepAliveType Struct Reference

Parameters for the KeepAlive system call.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned long [Result](#)

### 10.43.1 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](#).

### 10.43.2 Field Documentation

#### 10.43.2.1 unsigned long KeepAliveType::Result

The current sleep timeout in milliseconds.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)



## 10.44 ldiv\_t Struct Reference

Output type of the ldiv function.

```
#include <NXCDefs.h>
```

### Data Fields

- long [quot](#)
- long [rem](#)

#### 10.44.1 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.](#)

#### 10.44.2 Field Documentation

##### 10.44.2.1 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.](#)

##### 10.44.2.2 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](#)

## 10.45 ListFilesType Struct Reference

Parameters for the ListFiles system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- string [Pattern](#)
- string [FileList](#) [ ]

### 10.45.1 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.](#)

### 10.45.2 Field Documentation

#### 10.45.2.1 string ListFilesType::FileList[ ]

An array of strings containing the list of filenames that matched the file search pattern.

**Examples:**

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

#### 10.45.2.2 string ListFilesType::Pattern

The file search pattern.

**Examples:**

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

#### 10.45.2.3 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](#)

## 10.46 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](#)

### 10.46.1 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](#).

## 10.46.2 Field Documentation

### 10.46.2.1 byte LoaderExecuteFunctionType::Buffer[ ]

The Buffer parameter, see table.

### 10.46.2.2 byte LoaderExecuteFunctionType::Cmd

The command to execute.

#### Examples:

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

### 10.46.2.3 string LoaderExecuteFunctionType::Filename

The Filename parameter, see table.

### 10.46.2.4 unsigned long LoaderExecuteFunctionType::Length

The Length parameter, see table.

### 10.46.2.5 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](#)

## 10.47 LocationType Struct Reference

A point on the NXT LCD screen.

```
#include <NXCDefs.h>
```

### Data Fields

- [int X](#)
- [int Y](#)

### 10.47.1 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](#).

### 10.47.2 Field Documentation

#### 10.47.2.1 int LocationType::X

The X coordinate. Valid range is from 0 to 99 inclusive.

**Examples:**

[ex\\_dispgtout.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](#).

#### 10.47.2.2 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\\_dispgtout.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](#)

## 10.48 MessageReadType Struct Reference

Parameters for the MessageRead system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [QueueID](#)
- bool [Remove](#)
- string [Message](#)

### 10.48.1 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.](#)

### 10.48.2 Field Documentation

#### 10.48.2.1 string MessageReadType::Message

The contents of the mailbox/queue.

**Examples:**

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

#### 10.48.2.2 byte MessageReadType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

**Examples:**

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

#### 10.48.2.3 bool MessageReadType::Remove

If true, remove the read message from the queue.

**Examples:**

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

#### 10.48.2.4 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](#)



## 10.49 MessageWriteType Struct Reference

Parameters for the MessageWrite system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [QueueID](#)
- string [Message](#)

### 10.49.1 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](#).

### 10.49.2 Field Documentation

#### 10.49.2.1 string MessageWriteType::Message

The message to write.

**Examples:**

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

#### 10.49.2.2 byte MessageWriteType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

**Examples:**

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

#### 10.49.2.3 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](#)

## 10.50 RandomNumberType Struct Reference

Parameters for the RandomNumber system call.

```
#include <NXCDefs.h>
```

### Data Fields

- int [Result](#)

### 10.50.1 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](#).

### 10.50.2 Field Documentation

#### 10.50.2.1 int RandomNumberType::Result

The random number.

Examples:

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

## 10.51 ReadButtonType Struct Reference

Parameters for the ReadButton system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- byte [Index](#)
- bool [Pressed](#)
- byte [Count](#)
- bool [Reset](#)

### 10.51.1 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](#).

### 10.51.2 Field Documentation

#### 10.51.2.1 byte ReadButtonType::Count

The returned button pressed count.

#### 10.51.2.2 byte ReadButtonType::Index

The requested button index. See the [Button name constants](#) group.

Examples:

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

#### 10.51.2.3 bool ReadButtonType::Pressed

The returned button state.

Examples:

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

#### 10.51.2.4 `bool ReadButtonType::Reset`

If true, the count is reset after reading.

#### 10.51.2.5 `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](#)

## 10.52 ReadSemDataType Struct Reference

Parameters for the ReadSemData system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [SemData](#)
- bool [Request](#)

### 10.52.1 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.](#)

### 10.52.2 Field Documentation

#### 10.52.2.1 bool ReadSemDataType::Request

Which semaphore am I reading from, usage or request?

**Examples:**

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

#### 10.52.2.2 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](#)

## 10.53 SetScreenModeType Struct Reference

Parameters for the SetScreenMode system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [ScreenMode](#)

### 10.53.1 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](#).

### 10.53.2 Field Documentation

#### 10.53.2.1 char SetScreenModeType::Result

The function call result, always [NO\\_ERR](#).

#### 10.53.2.2 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](#)

## 10.54 SetSleepTimeoutType Struct Reference

Parameters for the SetSleepTimeout system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- unsigned long [TheSleepTimeoutMS](#)

### 10.54.1 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.](#)

### 10.54.2 Field Documentation

#### 10.54.2.1 char SetSleepTimeoutType::Result

The result of the system call function.

#### 10.54.2.2 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](#)

## 10.55 SizeType Struct Reference

Width and height dimensions for the DrawRect system call.

```
#include <NXCDefs.h>
```

### Data Fields

- int [Width](#)
- int [Height](#)

### 10.55.1 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](#)

### 10.55.2 Field Documentation

#### 10.55.2.1 int SizeType::Height

The rectangle height.

**Examples:**

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

#### 10.55.2.2 int SizeType::Width

The rectangle width.

**Examples:**

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

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)



## 10.56 SoundGetStateType Struct Reference

Parameters for the SoundGetState system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [State](#)
- byte [Flags](#)

### 10.56.1 Detailed Description

Parameters for the SoundGetState system call. This structure is used when calling the [SysSoundGetState](#) system call function.

**See also:**

[SysSoundGetState\(\)](#)

**Examples:**

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

### 10.56.2 Field Documentation

#### 10.56.2.1 byte SoundGetStateType::Flags

The returned sound flags. See the [SoundFlags constants](#) group.

#### 10.56.2.2 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](#)

## 10.57 SoundPlayFileType Struct Reference

Parameters for the SoundPlayFile system call.

```
#include <NXCDefs.h>
```

### Data Fields

- char [Result](#)
- string [Filename](#)
- bool [Loop](#)
- byte [SoundLevel](#)

### 10.57.1 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](#).

### 10.57.2 Field Documentation

#### 10.57.2.1 string SoundPlayFileType::Filename

The name of the file to play.

**Examples:**

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

#### 10.57.2.2 bool SoundPlayFileType::Loop

If true, loops at end of file.

**Examples:**

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

#### 10.57.2.3 char SoundPlayFileType::Result

The function call result, always [NO\\_ERR](#).

#### 10.57.2.4 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](#)

## 10.58 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](#)

### 10.58.1 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.](#)

### 10.58.2 Field Documentation

#### 10.58.2.1 unsigned int SoundPlayToneType::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

**Examples:**

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

#### 10.58.2.2 unsigned int SoundPlayToneType::Frequency

The tone frequency. See the [Tone constants](#) group.

**Examples:**

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

#### 10.58.2.3 bool SoundPlayToneType::Loop

If true, loops forever.

**Examples:**

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

**10.58.2.4 char SoundPlayToneType::Result**

The function call result, always [NO\\_ERR](#).

**10.58.2.5 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](#)

## 10.59 SoundSetStateType Struct Reference

Parameters for the SoundSetState system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- byte [State](#)
- byte [Flags](#)

### 10.59.1 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](#).

### 10.59.2 Field Documentation

#### 10.59.2.1 byte SoundSetStateType::Flags

The new sound flags. See the [SoundFlags constants](#) group.

#### 10.59.2.2 byte SoundSetStateType::Result

The function call result, same as State.

#### 10.59.2.3 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](#)

## 10.60 Tone Struct Reference

Type used with the PlayTones API function.

```
#include <NXCDefs.h>
```

### Data Fields

- unsigned int [Frequency](#)
- unsigned int [Duration](#)

### 10.60.1 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](#).

### 10.60.2 Field Documentation

#### 10.60.2.1 unsigned int Tone::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

#### 10.60.2.2 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](#)

## 10.61 UpdateCalibCacheInfoType Struct Reference

Parameters for the UpdateCalibCacheInfo system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [Result](#)
- string [Name](#)
- unsigned int [MinVal](#)
- unsigned int [MaxVal](#)

### 10.61.1 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.](#)

### 10.61.2 Field Documentation

#### 10.61.2.1 unsigned int UpdateCalibCacheInfoType::MaxVal

The maximum calibrated value.

**Examples:**

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

#### 10.61.2.2 unsigned int UpdateCalibCacheInfoType::MinVal

The minimum calibrated value.

**Examples:**

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

#### 10.61.2.3 string UpdateCalibCacheInfoType::Name

The name of the sensor calibration cache.

**Todo**

?,



**Examples:**

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

**10.61.2.4 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](#)

## 10.62 WriteSemDataType Struct Reference

Parameters for the WriteSemData system call.

```
#include <NXCDefs.h>
```

### Data Fields

- byte [SemData](#)
- bool [Request](#)
- byte [NewVal](#)
- bool [ClearBits](#)

### 10.62.1 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](#).

### 10.62.2 Field Documentation

#### 10.62.2.1 bool WriteSemDataType::ClearBits

Should I clear existing bits?

**Examples:**

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

#### 10.62.2.2 byte WriteSemDataType::NewVal

The new semaphore data.

**Examples:**

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

#### 10.62.2.3 bool WriteSemDataType::Request

Which semaphore am I writing to, usage or request?

**Examples:**

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

#### 10.62.2.4 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](#)



# Chapter 11

## File Documentation

### 11.1 NBCCCommon.h File Reference

Constants and macros common to both NBC and NXC.

#### Defines

- `#define TRUE 1`
- `#define FALSE 0`
- `#define NA 0xFFFF`
- `#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 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 [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 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 [Type](#) 0
- #define [InputMode](#) 1
- #define [RawValue](#) 2
- #define [NormalizedValue](#) 3
- #define [ScaledValue](#) 4
- #define [InvalidData](#) 5
- #define [INPUT\\_DIGI0](#) 1
- #define [INPUT\\_DIGI1](#) 2
- #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 [InputOffsetColorSensorBoolean](#)(p, nc) (80+((p)\*84)+76+((nc)\*2))
- #define [InputOffsetColorCalibrationState](#)(p) (80+((p)\*84)+80)
- #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\\_RUNSTATE\\_IDLE](#) 0x00
- #define [OUT\\_RUNSTATE\\_RAMPU](#) 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 [UpdateFlags](#) 0

*Update flags field.*



- #define [OutputMode](#) 1  
*Mode field.*
- #define [Power](#) 2  
*Power field.*
- #define [ActualSpeed](#) 3  
*Actual speed field.*
- #define [TachoCount](#) 4  
*Internal tachometer count field.*
- #define [TachoLimit](#) 5  
*Tachometer limit field.*
- #define [RunState](#) 6  
*Run state field.*
- #define [TurnRatio](#) 7  
*Turn ratio field.*
- #define [RegMode](#) 8  
*Regulation mode field.*
- #define [Overload](#) 9  
*Overload field.*
- #define [RegPValue](#) 10  
*Proportional field.*
- #define [RegIValue](#) 11  
*Integral field.*
- #define [RegDValue](#) 12  
*Derivative field.*
- #define [BlockTachoCount](#) 13  
*NXT-G block tachometer count field.*
- #define [RotationCount](#) 14  
*Rotation counter field.*
- #define [OutputOptions](#) 15  
*Options 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 [OutputOffsetPwnFreq](#) 96
- #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\\_REG\\_VERSION](#) 0x00
- #define [I2C\\_REG\\_VENDOR\\_ID](#) 0x08
- #define [I2C\\_REG\\_DEVICE\\_ID](#) 0x10
- #define [I2C\\_REG\\_CMD](#) 0x41
- #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\\_12BIT](#) 0x60
- #define [TEMP\\_RES\\_11BIT](#) 0x40
- #define [TEMP\\_RES\\_10BIT](#) 0x20
- #define [TEMP\\_RES\\_9BIT](#) 0x00
- #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\\_I2C\\_ADDRESS](#) 0x98
- #define [TEMP\\_REG\\_TEMP](#) 0x00
- #define [TEMP\\_REG\\_CONFIG](#) 0x01
- #define [TEMP\\_REG\\_TLOW](#) 0x02
- #define [TEMP\\_REG\\_THIGH](#) 0x03
- #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\\_OFFS](#) 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 [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 [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\\_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\\_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 [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\\_REMOVEDDEVICE](#) 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 [CommOffsetHsMode](#) 1895
- #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\_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 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 MS\_CMD\_ENERGIZED 0x45
- #define MS\_CMD\_DEENERGIZED 0x44
- #define MS\_CMD\_ADPA\_ON 0x4E
- #define MS\_CMD\_ADPA\_OFF 0x4F
- #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](#) 0x01
- #define [PSP\\_BTNSET1\\_DOWN](#) 0x02
- #define [PSP\\_BTNSET1\\_RIGHT](#) 0x04
- #define [PSP\\_BTNSET1\\_UP](#) 0x08
- #define [PSP\\_BTNSET1\\_R3](#) 0x20
- #define [PSP\\_BTNSET1\\_L3](#) 0x40
- #define [PSP\\_BTNSET2\\_SQUARE](#) 0x01
- #define [PSP\\_BTNSET2\\_CROSS](#) 0x02
- #define [PSP\\_BTNSET2\\_CIRCLE](#) 0x04
- #define [PSP\\_BTNSET2\\_TRIANGLE](#) 0x08
- #define [PSP\\_BTNSET2\\_R1](#) 0x10
- #define [PSP\\_BTNSET2\\_L1](#) 0x20
- #define [PSP\\_BTNSET2\\_R2](#) 0x40
- #define [PSP\\_BTNSET2\\_L2](#) 0x80
- #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 [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](#) -127
- #define [SCHAR\\_MAX](#) 127
- #define [UCHAR\\_MAX](#) 255
- #define [CHAR\\_MIN](#) -127
- #define [CHAR\\_MAX](#) 127
- #define [SHRT\\_MIN](#) -32767
- #define [SHRT\\_MAX](#) 32767
- #define [USHRT\\_MAX](#) 65535
- #define [INT\\_MIN](#) -32767
- #define [INT\\_MAX](#) 32767
- #define [UINT\\_MAX](#) 65535
- #define [LONG\\_MIN](#) -2147483647
- #define [LONG\\_MAX](#) 2147483647
- #define [ULONG\\_MAX](#) 4294967295
- #define [RAND\\_MAX](#) 32768
- #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

### 11.1.1 Detailed Description

Constants and macros common to both NBC and NXC. [NBCCommon.h](#) contains declarations for the NBC and NXC NXT 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) 2009-2010 John Hansen. All Rights Reserved.

-----

**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2010-04-30

**Version:**

52

### 11.1.2 Define Documentation

#### 11.1.2.1 #define ActualSpeed 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 Power value when auto-regulation code in the firmware responds to a load on the output.

#### 11.1.2.2 #define BITMAP\_1 0

Bitmap 1

#### 11.1.2.3 #define BITMAP\_2 1

Bitmap 2

**11.1.2.4 #define BITMAP\_3 2**

Bitmap 3

**11.1.2.5 #define BITMAP\_4 3**

Bitmap 4

**11.1.2.6 #define BITMAPS 4**

The number of bitmap bits

**11.1.2.7 #define BlockTachoCount 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 [UpdateFlags](#) description for information about how to use block-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_BLOCK\\_COUNT](#) flag in UpdateFlags to request that the firmware reset the BlockTachoCount. The sign of BlockTachoCount 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.

**11.1.2.8 #define BREAKOUT\_REQ 3**

VM should break out of current thread

**11.1.2.9 #define BT\_ARM\_CMD\_MODE 1**

BtState constant bluetooth command mode

**11.1.2.10 #define BT\_ARM\_DATA\_MODE 2**

BtState constant bluetooth data mode

**11.1.2.11 #define BT\_ARM\_OFF 0**

BtState constant bluetooth off

**11.1.2.12 #define BT\_BRICK\_PORT\_OPEN 0x02**

BtStateStatus port open bit

**11.1.2.13 #define BT\_BRICK\_VISIBILITY 0x01**

BtStateStatus brick visibility bit

**11.1.2.14 #define BT\_CMD\_BYTE 1**

Size of Bluetooth command

**11.1.2.15 #define BT\_CMD\_READY 0x02**

A constant representing bluetooth direct command

**11.1.2.16 #define BT\_CONNECTION\_0\_ENABLE 0x10**

BtStateStatus connection 0 enable/disable bit

**11.1.2.17 #define BT\_CONNECTION\_1\_ENABLE 0x20**

BtStateStatus connection 1 enable/disable bit

**11.1.2.18 #define BT\_CONNECTION\_2\_ENABLE 0x40**

BtStateStatus connection 2 enable/disable bit

**11.1.2.19 #define BT\_CONNECTION\_3\_ENABLE 0x80**

BtStateStatus connection 3 enable/disable bit

**11.1.2.20 #define BT\_DEFAULT\_INQUIRY\_MAX 0**

Bluetooth default inquiry Max (0 == unlimited)

**11.1.2.21 #define BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO 15**

Bluetooth inquiry timeout (15\*1.28 sec = 19.2 sec)

**11.1.2.22 #define BT\_DEVICE\_AWAY 0x80**

Bluetooth device away

**11.1.2.23 #define BT\_DEVICE\_EMPTY 0x00**

Bluetooth device table empty

**11.1.2.24 #define BT\_DEVICE\_KNOWN 0x02**

Bluetooth device known

**11.1.2.25 #define BT\_DEVICE\_NAME 0x40**

Bluetooth device name

**11.1.2.26 #define BT\_DEVICE\_UNKNOWN 0x01**

Bluetooth device unknown

**11.1.2.27 #define BT\_DISABLE 0x01**

BtHwStatus bluetooth disable

**11.1.2.28 #define BT\_ENABLE 0x00**

BtHwStatus bluetooth enable

**11.1.2.29 #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](#).

**11.1.2.30 #define BTN2 1**

The right button.

**11.1.2.31 #define BTN3 2**

The left button.

**11.1.2.32 #define BTN4 3**

The enter button.

**11.1.2.33 #define BTNCENTER BTN4**

The enter button.

**Examples:**

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

**11.1.2.34 #define BTNEXIT BTN1**

The exit button.

**Examples:**

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

**11.1.2.35 #define BTNLEFT BTN3**

The left button.

**Examples:**

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

**11.1.2.36 #define BTNRIGHT BTN2**

The right button.

**Examples:**

[ex\\_ButtonPressed.nxc](#), and [ex\\_sysreadbutton.nxc](#).

**11.1.2.37 #define BTNSTATE\_LONG\_PRESSED\_EV 0x04**

Button is in the long pressed state.

**Examples:**

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

**11.1.2.38 #define BTNSTATE\_LONG\_RELEASED\_EV 0x08**

Button is in the long released state.

**11.1.2.39 #define BTNSTATE\_NONE 0x10**

The default button state.

**11.1.2.40 #define BTNSTATE\_PRESSED\_EV 0x01**

Button is in the pressed state.

**Examples:**

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

**11.1.2.41 #define BTNSTATE\_PRESSED\_STATE 0x80**

A bitmask for the button pressed state

**11.1.2.42 #define BTNSTATE\_SHORT\_RELEASED\_EV 0x02**

Button is in the short released state.

**11.1.2.43 #define ButtonModuleID 0x00040001**

The button module ID

**11.1.2.44 #define ButtonModuleName "Button.mod"**

The button module name

**11.1.2.45 #define ButtonOffsetLongPressCnt(b) (((b)\*8)+1)**

Offset to the LongPressCnt field. This field stores the long press count.

**11.1.2.46 #define ButtonOffsetLongRelCnt(b) (((b)\*8)+3)**

Offset to the LongRelCnt field. This field stores the long release count.

**11.1.2.47 #define ButtonOffsetPressedCnt(b) (((b)\*8)+0)**

Offset to the PressedCnt field. This field stores the press count.

**11.1.2.48 #define ButtonOffsetRelCnt(b) (((b)\*8)+4)**

Offset to the RelCnt field. This field stores the release count.

**11.1.2.49 #define ButtonOffsetShortRelCnt(b) (((b)\*8)+2)**

Offset to the ShortRelCnt field. This field stores the short release count.

**11.1.2.50 #define ButtonOffsetState(b) ((b)+32)**

Offset to the State field. This field stores the current button state.

**11.1.2.51 #define CHAR\_BIT 8**

The number of bits in the char type

**11.1.2.52 #define CHAR\_MAX 127**

The maximum value of the char type

**11.1.2.53 #define CHAR\_MIN -127**

The minimum value of the char type

**11.1.2.54 #define CLUMP\_DONE 1**

VM has finished executing thread

**11.1.2.55 #define CLUMP\_SUSPEND 2**

VM should suspend thread

**11.1.2.56 #define ColorSensorRead 34**

Read data from the NXT 2.0 color sensor

**11.1.2.57 #define COM\_CHANNEL\_FOUR\_ACTIVE 0x08**

Low speed channel 4 is active

**11.1.2.58 #define COM\_CHANNEL\_NONE\_ACTIVE 0x00**

None of the low speed channels are active

**11.1.2.59 #define COM\_CHANNEL\_ONE\_ACTIVE 0x01**

Low speed channel 1 is active

**11.1.2.60 #define COM\_CHANNEL\_THREE\_ACTIVE 0x04**

Low speed channel 3 is active

**11.1.2.61 #define COM\_CHANNEL\_TWO\_ACTIVE 0x02**

Low speed channel 2 is active

**11.1.2.62 #define CommandModuleID 0x00010001**

The command module ID

**Examples:**

[ex\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).



**11.1.2.63 #define CommandModuleName "Command.mod"**

The command module name

**Examples:**

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

**11.1.2.64 #define CommandOffsetActivateFlag 30**

Offset to the activate flag

**11.1.2.65 #define CommandOffsetAwake 29**

Offset to the VM's awake state

**11.1.2.66 #define CommandOffsetDeactivateFlag 31**

Offset to the deactivate flag

**11.1.2.67 #define CommandOffsetFileName 32**

Offset to the running program's filename

**11.1.2.68 #define CommandOffsetFormatString 0**

Offset to the format string

**11.1.2.69 #define CommandOffsetMemoryPool 52**

Offset to the VM's memory pool

**Examples:**

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

**11.1.2.70 #define CommandOffsetOffsetDS 24**

Offset to the running program's data space (DS)

**11.1.2.71 #define CommandOffsetOffsetDVA 26**

Offset to the running program's DOPE vector address (DVA)

**11.1.2.72 #define CommandOffsetPRCHandler 16**

Offset to the RC Handler function pointer

**11.1.2.73 #define CommandOffsetProgStatus 28**

Offset to the running program's status

**11.1.2.74 #define CommandOffsetSyncTick 32824**

Offset to the VM sync tick

**11.1.2.75 #define CommandOffsetSyncTime 32820**

Offset to the VM sync time

**11.1.2.76 #define CommandOffsetTick 20**

Offset to the VM's current tick

**Examples:**

[ex\\_sysiomapread.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**11.1.2.77 #define CommBTCheckStatus 28**

Check the bluetooth status

**11.1.2.78 #define CommBTConnection 36**

Connect or disconnect to a known bluetooth device

**11.1.2.79 #define CommBTOnOff 35**

Turn the bluetooth radio on or off

**11.1.2.80 #define CommBTRead 30**

Read from a bluetooth connection

**11.1.2.81 #define CommBTWrite 29**

Write to a bluetooth connections

**11.1.2.82 #define CommExecuteFunction 81**

Execute one of the Comm module's internal functions

**11.1.2.83 #define CommHSCheckStatus 39**

Check the status of the hi-speed port

**11.1.2.84 #define CommHSControl 88**

Control the hi-speed port

**11.1.2.85 #define CommHSRead 38**

Read data from the hi-speed port

**11.1.2.86 #define CommHSWrite 37**

Write data to the hi-speed port

**11.1.2.87 #define CommLSCheckStatus 23**

Check the status of a lowspeed (aka I2C) device

**11.1.2.88 #define CommLSRead 22**

Read from a lowspeed (aka I2C) device

**11.1.2.89 #define CommLSWrite 21**

Write to a lowspeed (aka I2C) device

**11.1.2.90 #define CommLSWriteEx 89**

Write to a lowspeed (aka I2C) device with optional restart on read

**11.1.2.91 #define CommModuleID 0x00050001**

The Comm module ID

**11.1.2.92 #define CommModuleName "Comm.mod"**

The Comm module name

**11.1.2.93 #define CommOffsetBrickDataBdAddr 1144**

Offset to Bluetooth address (7 bytes)

**11.1.2.94 #define CommOffsetBrickDataBluecoreVersion 1142**

Offset to Bluecore version (2 bytes)

**11.1.2.95 #define CommOffsetBrickDataBtHwStatus 1152**

Offset to BtHwStatus (1 byte)

**11.1.2.96 #define CommOffsetBrickDataBtStateStatus 1151**

Offset to BtStateStatus (1 byte)

**11.1.2.97 #define CommOffsetBrickDataName 1126**

Offset to brick name (16 bytes)

**11.1.2.98 #define CommOffsetBrickDataTimeOutValue 1153**

Offset to data timeout value (1 byte)

**11.1.2.99 #define CommOffsetBtConnectTableBdAddr(p) (((p)\*47)+974)**

Offset to Bluetooth connect table address (7 bytes)

**11.1.2.100 #define CommOffsetBtConnectTableClassOfDevice(p) (((p)\*47)+954)**

Offset to Bluetooth connect table device class (4 bytes)

**11.1.2.101 #define CommOffsetBtConnectTableHandleNr(p) (((p)\*47)+981)**

Offset to Bluetooth connect table handle (1 byte)

**11.1.2.102 #define CommOffsetBtConnectTableLinkQuality(p) (((p)\*47)+983)**

Offset to Bluetooth connect table link quality (1 byte)

**11.1.2.103 #define CommOffsetBtConnectTableName(p) (((p)\*47)+938)**

Offset to Bluetooth connect table name (16 bytes)

**11.1.2.104 #define CommOffsetBtConnectTablePinCode(p) (((p)\*47)+958)**

Offset to Bluetooth connect table pin code (16 bytes)

**11.1.2.105 #define CommOffsetBtConnectTableStreamStatus(p) (((p)\*47)+982)**

Offset to Bluetooth connect table stream status (1 byte)

**11.1.2.106 #define CommOffsetBtDeviceCnt 1889**

Offset to Bluetooth device count (1 byte)

**11.1.2.107 #define CommOffsetBtDeviceNameCnt 1890**

Offset to Bluetooth device name count (1 byte)

**11.1.2.108 #define CommOffsetBtDeviceTableBdAddr(p) (((p)\*31)+28)**

Offset to Bluetooth device table address (7 bytes)

**11.1.2.109 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)\*31)+24)**

Offset to Bluetooth device table device class (4 bytes)

**11.1.2.110 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)\*31)+35)**

Offset to Bluetooth device table status (1 byte)

**11.1.2.111 #define CommOffsetBtDeviceTableName(p) (((p)\*31)+8)**

Offset to BT device table name (16 bytes)

**11.1.2.112 #define CommOffsetBtInBufBuf 1157**

Offset to Bluetooth input buffer data (128 bytes)

**11.1.2.113 #define CommOffsetBtInBufInPtr 1285**

Offset to Bluetooth input buffer front pointer (1 byte)

**11.1.2.114 #define CommOffsetBtInBufOutPtr 1286**

Offset to Bluetooth output buffer back pointer (1 byte)

**11.1.2.115 #define CommOffsetBtOutBufBuf 1289**

Offset to Bluetooth output buffer offset data (128 bytes)

**11.1.2.116 #define CommOffsetBtOutBufInPtr 1417**

Offset to Bluetooth output buffer front pointer (1 byte)

**11.1.2.117 #define CommOffsetBtOutBufOutPtr 1418**

Offset to Bluetooth output buffer back pointer (1 byte)

**11.1.2.118 #define CommOffsetHsFlags 1891**

Offset to High Speed flags (1 byte)

**11.1.2.119 #define CommOffsetHsInBufBuf 1421**

Offset to High Speed input buffer data (128 bytes)

**11.1.2.120 #define CommOffsetHsInBufInPtr 1549**

Offset to High Speed input buffer front pointer (1 byte)

**11.1.2.121 #define CommOffsetHsInBufOutPtr 1550**

Offset to High Speed input buffer back pointer (1 byte)

**11.1.2.122 #define CommOffsetHsMode 1895**

Offset to High Speed mode (2 bytes)

**11.1.2.123 #define CommOffsetHsOutBufBuf 1553**

Offset to High Speed output buffer data (128 bytes)

**11.1.2.124 #define CommOffsetHsOutBufInPtr 1681**

Offset to High Speed output buffer front pointer (1 byte)

**11.1.2.125 #define CommOffsetHsOutBufOutPtr 1682**

Offset to High Speed output buffer back pointer (1 byte)

**11.1.2.126 #define CommOffsetHsSpeed 1892**

Offset to High Speed speed (1 byte)

**11.1.2.127 #define CommOffsetHsState 1893**

Offset to High Speed state (1 byte)

**11.1.2.128 #define CommOffsetPFunc 0**

Offset to the Comm module first function pointer (4 bytes)

**11.1.2.129 #define CommOffsetPFuncTwo 4**

Offset to the Comm module second function pointer (4 bytes)

**11.1.2.130 #define CommOffsetUsbInBufBuf 1685**

Offset to Usb input buffer data (64 bytes)

**11.1.2.131 #define CommOffsetUsbInBufInPtr 1749**

Offset to Usb input buffer front pointer (1 byte)

**11.1.2.132 #define CommOffsetUsbInBufOutPtr 1750**

Offset to Usb input buffer back pointer (1 byte)

**11.1.2.133 #define CommOffsetUsbOutBufBuf 1753**

Offset to Usb output buffer data (64 bytes)

**11.1.2.134 #define CommOffsetUsbOutBufInPtr 1817**

Offset to Usb output buffer front pointer (1 byte)

**11.1.2.135 #define CommOffsetUsbOutBufOutPtr 1818**

Offset to Usb output buffer back pointer (1 byte)

**11.1.2.136 #define CommOffsetUsbPollBufBuf 1821**

Offset to Usb Poll buffer data (64 bytes)

**11.1.2.137 #define CommOffsetUsbPollBufInPtr 1885**

Offset to Usb Poll buffer front pointer (1 byte)

**11.1.2.138 #define CommOffsetUsbPollBufOutPtr 1886**

Offset to Usb Poll buffer back pointer (1 byte)

**11.1.2.139 #define CommOffsetUsbState 1894**

Offset to Usb State (1 byte)

**11.1.2.140 #define ComputeCalibValue 42**

Compute a calibration value

**11.1.2.141 #define DatalogGetTimes 45**

Get datalog timing information

**11.1.2.142 #define DatalogWrite 44**

Write to the datalog

**11.1.2.143 #define DEGREES\_PER\_RADIAN 180/PI**

Used for converting from radians to degrees

**11.1.2.144 #define DISPLAY\_BUSY 0x80**

R - Refresh in progress

**11.1.2.145 #define DISPLAY\_CHAR 0x04**

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

**11.1.2.146 #define DISPLAY\_CONTRAST\_DEFAULT 0x5A**

Default display contrast value

**Examples:**

[ex\\_contrast.nxc](#), and [ex\\_SetDisplayContrast.nxc](#).

**11.1.2.147 #define DISPLAY\_CONTRAST\_MAX 0x7F**

Maximum display contrast value

**Examples:**

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



**11.1.2.148 #define DISPLAY\_ERASE\_ALL 0x00**

W - erase entire screen (CMD,x,x,x,x,x)

**Examples:**

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

**11.1.2.149 #define DISPLAY\_ERASE\_LINE 0x05**

W - erase a single line (CMD,x,LINE,x,x,x)

**11.1.2.150 #define DISPLAY\_FILL\_REGION 0x06**

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**11.1.2.151 #define DISPLAY\_FRAME 0x07**

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

**11.1.2.152 #define DISPLAY\_HEIGHT 64**

The height of the LCD screen in pixels

**Examples:**

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

**11.1.2.153 #define DISPLAY\_HORIZONTAL\_LINE 0x02**

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

**Examples:**

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

**11.1.2.154 #define DISPLAY\_MENUICONS\_X\_DIFF 31****11.1.2.155 #define DISPLAY\_MENUICONS\_X\_OFFS 7****11.1.2.156 #define DISPLAY\_MENUICONS\_Y 40****11.1.2.157 #define DISPLAY\_ON 0x01**

W - Display on

**11.1.2.158 #define DISPLAY\_PIXEL 0x01**

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

**11.1.2.159 #define DISPLAY\_POPUP 0x08**

W - Use popup display memory

**Examples:**

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

**11.1.2.160 #define DISPLAY\_REFRESH 0x02**

W - Enable refresh

**11.1.2.161 #define DISPLAY\_REFRESH\_DISABLED 0x40**

R - Refresh disabled

**11.1.2.162 #define DISPLAY\_VERTICAL\_LINE 0x03**

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

**11.1.2.163 #define DISPLAY\_WIDTH 100**

The width of the LCD screen in pixels

**Examples:**

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

**11.1.2.164 #define DisplayExecuteFunction 80**

Execute one of the Display module's internal functions

**11.1.2.165 #define DisplayModuleID 0x000A0001**

The display module ID

**11.1.2.166 #define DisplayModuleName "Display.mod"**

The display module name

**11.1.2.167 #define DisplayOffsetContrast 1719**

Adjust the display contrast with this field

**11.1.2.168 #define DisplayOffsetDisplay 104**

Display content copied to physical display every 17 mS

**11.1.2.169 #define DisplayOffsetEraseMask 4**

Section erase mask (executed first)

**11.1.2.170 #define DisplayOffsetFlags 117**

Update flags enumerated above

**11.1.2.171 #define DisplayOffsetNormal(l, w) (((l)\*100)+(w)+119)**

Raw display memory for normal screen

**11.1.2.172 #define DisplayOffsetPBitmaps(p) (((p)\*4)+68)**

Pointer to free bitmap files

**11.1.2.173 #define DisplayOffsetPFont 12**

Pointer to font file

**11.1.2.174 #define DisplayOffsetPFunc 0**

Simple draw entry

**11.1.2.175 #define DisplayOffsetPMenuIcons(p) (((p)\*4)+88)**

Pointer to menu icon images (NULL == none)

**11.1.2.176 #define DisplayOffsetPMenuText 84**

Pointer to menu icon text (NULL == none)

**11.1.2.177 #define DisplayOffsetPopup(l, w) (((l)\*100)+(w)+919)**

Raw display memory for popup screen

**11.1.2.178 #define DisplayOffsetPScreens(p) (((p)\*4)+56)**

Pointer to screen bitmap file

**11.1.2.179 #define DisplayOffsetPStatusIcons 52**

Pointer to status icon collection file

**11.1.2.180 #define DisplayOffsetPStatusText 48**

Pointer to status text string

**11.1.2.181 #define DisplayOffsetPStepIcons 100**

Pointer to step icon collection file

**11.1.2.182 #define DisplayOffsetPTextLines(p) (((p)\*4)+16)**

Pointer to text strings

**11.1.2.183 #define DisplayOffsetStatusIcons(p) ((p)+108)**

Index in status icon collection file (index = 0 -> none)

**11.1.2.184 #define DisplayOffsetStepIcons(p) ((p)+112)**

Index in step icon collection file (index = 0 -> none)

**11.1.2.185 #define DisplayOffsetTextLinesCenterFlags 118**

Mask to center TextLines

**11.1.2.186 #define DisplayOffsetUpdateMask 8**

Section update mask (executed next)

**11.1.2.187 #define DIST\_CMD\_CUSTOM 0x35**

Set the DIST-Nx to a custom mode

**11.1.2.188 #define DIST\_CMD\_GP2D12 0x31**

Set the DIST-Nx to GP2D12 mode

**11.1.2.189 #define DIST\_CMD\_GP2D120 0x32**

Set the DIST-Nx to GP2D120 mode

**11.1.2.190 #define DIST\_CMD\_GP2YA02 0x34**

Set the DIST-Nx to GP2YA02 mode

**11.1.2.191 #define DIST\_CMD\_GP2YA21 0x33**

Set the DIST-Nx to GP2YA21 mode

**11.1.2.192 #define DIST\_REG\_DIST 0x42**

The DIST-Nx distance register

**11.1.2.193 #define DIST\_REG\_DIST1 0x58**

The DIST-Nx distance 1 register

**11.1.2.194 #define DIST\_REG\_DIST\_MAX 0x54**

The DIST-Nx maximum distance register

**11.1.2.195 #define DIST\_REG\_DIST\_MIN 0x52**

The DIST-Nx minimum distance register

**11.1.2.196 #define DIST\_REG\_MODULE\_TYPE 0x50**

The DIST-Nx module type register

**11.1.2.197 #define DIST\_REG\_NUM\_POINTS 0x51**

The DIST-Nx number of points register

**11.1.2.198 #define DIST\_REG\_VOLT 0x44**

The DIST-Nx voltage register

**11.1.2.199 #define DIST\_REG\_VOLT1 0x56**

The DIST-Nx voltage 1 register

**11.1.2.200 #define DRAW\_OPT\_CLEAR (0x0004)**

Clear pixels while drawing (aka draw in white)

**11.1.2.201 #define DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_SCREEN (0x0002)**

Clear the screen except for the status line before drawing

**11.1.2.202 #define DRAW\_OPT\_CLEAR\_PIXELS (0x0004)**

Clear pixels while drawing (aka draw in white)

**11.1.2.203 #define DRAW\_OPT\_CLEAR\_SCREEN\_MODES (0x0003)**

Bit mask for the clear screen modes

**11.1.2.204 #define DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN (0x0001)**

Clear the entire screen before drawing

**Examples:**

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

**11.1.2.205 #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](#).

**11.1.2.206 #define DRAW\_OPT\_FONT\_DIR\_B2TL (0x0100)**

Font bottom to top left align

**11.1.2.207 #define DRAW\_OPT\_FONT\_DIR\_B2TR (0x0140)**

Font bottom to top right align

**11.1.2.208 #define DRAW\_OPT\_FONT\_DIR\_L2RB (0x0000)**

Font left to right bottom align

**Examples:**

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

**11.1.2.209 #define DRAW\_OPT\_FONT\_DIR\_L2RT (0x0040)**

Font left to right top align

**Examples:**

[ex\\_dispfout.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**11.1.2.210 #define DRAW\_OPT\_FONT\_DIR\_R2LB (0x0080)**

Font right to left bottom align

**11.1.2.211 #define DRAW\_OPT\_FONT\_DIR\_R2LT (0x00C0)**

Font right to left top align

**11.1.2.212 #define DRAW\_OPT\_FONT\_DIR\_T2BL (0x0180)**

Font top to bottom left align

**Examples:**

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

**11.1.2.213 #define DRAW\_OPT\_FONT\_DIR\_T2BR (0x01C0)**

Font top to bottom right align

**11.1.2.214 #define DRAW\_OPT\_FONT\_DIRECTIONS (0x01C0)**

Bit mask for the font direction bits

**11.1.2.215 #define DRAW\_OPT\_FONT\_WRAP (0x0200)**

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

**Examples:**

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

**11.1.2.216 #define DRAW\_OPT\_INVERT (0x0004)**

Invert text or graphics

**Examples:**

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

**11.1.2.217 #define DRAW\_OPT\_LOGICAL\_AND (0x0008)**

Draw pixels using a logical AND operation

**Examples:**

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

**11.1.2.218 #define DRAW\_OPT\_LOGICAL\_COPY (0x0000)**

Draw pixels using a logical copy operation

**11.1.2.219 #define DRAW\_OPT\_LOGICAL\_OPERATIONS (0x0018)**

Bit mask for the logical drawing operations

**11.1.2.220 #define DRAW\_OPT\_LOGICAL\_OR (0x0010)**

Draw pixels using a logical OR operation

**Examples:**

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

**11.1.2.221 #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](#).

**11.1.2.222 #define DRAW\_OPT\_NORMAL (0x0000)**

Normal drawing

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispfunc.nxc](#), and [ex\\_sysdrawfont.nxc](#).

**11.1.2.223 #define DRAW\_OPT\_POLYGON\_POLYLINE (0x0400)**

When drawing polygons, do not close (i.e., draw a polyline instead)

**11.1.2.224 #define DrawCircle 16**

Draw a circle on the LCD screen

**11.1.2.225 #define DrawEllipse 94**

Draw an ellipse on the LCD screen

**11.1.2.226 #define DrawFont 95**

Draw text using a custom RIC-based font to the LCD screen



**11.1.2.227 #define DrawGraphic 18**

Draw a graphic image on the LCD screen

**11.1.2.228 #define DrawGraphicArray 92**

Draw a graphic image from a byte array to the LCD screen

**Examples:**

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

**11.1.2.229 #define DrawLine 15**

Draw a line on the LCD screen

**11.1.2.230 #define DrawPoint 14**

Draw a single pixel on the LCD screen

**11.1.2.231 #define DrawPolygon 93**

Draw a polygon on the LCD screen

**11.1.2.232 #define DrawRect 17**

Draw a rectangle on the LCD screen

**11.1.2.233 #define DrawText 13**

Draw text to one of 8 LCD lines

**Examples:**

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

**11.1.2.234 #define EOF -1**

A constant representing end of file

**11.1.2.235 #define ERR\_ARG -1**

0xFF Bad arguments

**11.1.2.236 #define ERR\_BAD\_POOL\_SIZE -10**

0xF6 VarsCmd.PoolSize > POOL\_MAX\_SIZE

**11.1.2.237 #define ERR\_BAD\_PTR -6**

0xFA Someone passed us a bad pointer!

**11.1.2.238 #define ERR\_CLUMP\_COUNT -7**

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT\_A\_CLUMP)

**11.1.2.239 #define ERR\_COMM\_BUFFER\_FULL -34**

0xDE No room in comm buffer

**11.1.2.240 #define ERR\_COMM\_BUS\_ERR -35**

0xDD Something went wrong on the communications bus

**11.1.2.241 #define ERR\_COMM\_CHAN\_INVALID -33**

0xDF Specified channel/connection is not valid

**11.1.2.242 #define ERR\_COMM\_CHAN\_NOT\_READY -32**

0xE0 Specified channel/connection not configured or busy

**11.1.2.243 #define ERR\_DEFAULT\_OFFSETS -14**

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

**11.1.2.244 #define ERR\_FILE -3**

0xFD Malformed file contents

**11.1.2.245 #define ERR\_INSANE\_OFFSET -9**

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount \* 2)

**11.1.2.246 #define ERR\_INSTR -2**

0xFE Illegal bytecode instruction

**11.1.2.247 #define ERR\_INVALID\_FIELD -17**

0xEF Attempted to access invalid field of a structure

**11.1.2.248 #define ERR\_INVALID\_PORT -16**

0xF0 Bad input or output port specified

**11.1.2.249 #define ERR\_INVALID\_QUEUE -18**

0xEE Illegal queue ID specified

**11.1.2.250 #define ERR\_INVALID\_SIZE -19**

0xED Illegal size specified

**11.1.2.251 #define ERR\_LOADER\_ERR -11**

0xF5 LOADER\_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

**11.1.2.252 #define ERR\_MEM -5**

0xFB Insufficient memory available

**11.1.2.253 #define ERR\_MEMMGR\_FAIL -15**

0xF1 (UBYTE \*)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV\_ARRAY[0].Offset

**11.1.2.254 #define ERR\_NO\_ACTIVE\_CLUMP -13**

0xF3 VarsCmd.RunQ.Head == NOT\_A\_CLUMP

**11.1.2.255 #define ERR\_NO\_CODE -8**

0xF8 VarsCmd.CodespaceCount == 0

**11.1.2.256 #define ERR\_NO\_PROG -20**

0xEC No active program

**11.1.2.257 #define ERR\_NON\_FATAL -16**

Fatal errors are greater than this value

**11.1.2.258 #define ERR\_RC\_BAD\_PACKET -65**

0xBF Clearly insane packet

**11.1.2.259 #define ERR\_RC\_FAILED -67**

0xBD Request failed (i.e. specified file not found)

**11.1.2.260 #define ERR\_RC\_ILLEGAL\_VAL -64**

0xC0 Data contains out-of-range values

**11.1.2.261 #define ERR\_RC\_UNKNOWN\_CMD -66**

0xBE Unknown command opcode

**11.1.2.262 #define ERR\_SPOTCHECK\_FAIL -12**

0xF4 ((UBYTE\*)(VarsCmd.pCodespace) < pData) (c\_cmd.c 1893)

**11.1.2.263 #define ERR\_VER -4**

0xFC Version mismatch between firmware and compiler

**11.1.2.264 #define FALSE 0**

A false value

**11.1.2.265 #define FileClose 5**

Close the specified file

**11.1.2.266 #define FileDelete 8**

Delete a file

**11.1.2.267 #define FileFindFirst 83**

Start a search for a file using a filename pattern

**11.1.2.268 #define FileFindNext 84**

Continue searching for a file

**11.1.2.269 #define FileOpenAppend 2**

Open a file for appending to the end of the file

**11.1.2.270 #define FileOpenRead 0**

Open a file for reading

**11.1.2.271 #define FileOpenReadLinear 87**

Open a linear file for reading

**11.1.2.272 #define FileOpenWrite 1**

Open a file for writing (creates a new file)

**11.1.2.273 #define FileOpenWriteLinear 85**

Open a linear file for writing

**11.1.2.274 #define FileOpenWriteNonLinear 86**

Open a non-linear file for writing

**11.1.2.275 #define FileRead 3**

Read from the specified file

**11.1.2.276 #define FileRename 7**

Rename a file

**11.1.2.277 #define FileResize 91**

Resize a file (not yet implemented)

**11.1.2.278 #define FileResolveHandle 6**

Get a file handle for the specified filename if it is already open

**11.1.2.279 #define FileSeek 90**

Seek to a specific position in an open file

**11.1.2.280 #define FileWrite 4**

Write to the specified file

**11.1.2.281 #define FRAME\_SELECT 0**

Center icon select frame

**11.1.2.282 #define FREQUENCY\_MAX 14080**

Maximum frequency [Hz]

**11.1.2.283 #define FREQUENCY\_MIN 220**

Minimum frequency [Hz]

**11.1.2.284 #define GetStartTick 25**

Get the current system tick count

**11.1.2.285 #define GL\_CAMERA\_DEPTH 3**

Set the camera depth.

**11.1.2.286 #define GL\_CIRCLE 4**

Use circle mode.

**Examples:**

[glCircleDemo.nxc](#).

**11.1.2.287 #define GL\_CIRCLE\_SIZE 1**

Set the circle size.

**11.1.2.288 #define GL\_CULL\_BACK 2**

Cull lines in back.

**11.1.2.289 #define GL\_CULL\_FRONT 3**

Cull lines in front.

**11.1.2.290 #define GL\_CULL\_MODE 2**

Set the cull mode.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**11.1.2.291 #define GL\_CULL\_NONE 4**

Do not cull any lines.

**Examples:**

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**11.1.2.292 #define GL\_LINE 2**

Use line mode.

**11.1.2.293 #define GL\_POINT 3**

Use point mode.

**11.1.2.294 #define GL\_POLYGON 1**

Use polygon mode.

**Examples:**

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

**11.1.2.295 #define GL\_ROTATE\_X 4**

Rotate around the X axis.

**Examples:**

[glRotateDemo.nxc](#).

**11.1.2.296 #define GL\_ROTATE\_Y 5**

Rotate around the Y axis.

**Examples:**

[glRotateDemo.nxc](#).

**11.1.2.297 #define GL\_ROTATE\_Z 6**

Rotate around the Z axis.

**11.1.2.298 #define GL\_SCALE\_X 7**

Scale along the X axis.

**Examples:**

[glScaleDemo.nxc](#).

**11.1.2.299 #define GL\_SCALE\_Y 8**

Scale along the Y axis.

**11.1.2.300 #define GL\_SCALE\_Z 9**

Scale along the Z axis.

**11.1.2.301 #define GL\_TRANSLATE\_X 1**

Translate along the X axis.

**Examples:**

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

**11.1.2.302 #define GL\_TRANSLATE\_Y 2**

Translate along the Y axis.

**Examples:**

[glTranslateDemo.nxc](#).

**11.1.2.303 #define GL\_TRANSLATE\_Z 3**

Translate along the Z axis.

**Examples:**

[glTranslateDemo.nxc](#).

**11.1.2.304 #define GL\_ZOOM\_FACTOR 4**

Set the zoom factor.

**11.1.2.305 #define HS\_BAUD\_115200 12**

HsMode 115200 Baud

**Examples:**

[ex\\_RS485Uart.nxc](#).



**11.1.2.306 #define HS\_BAUD\_1200 0**

HsMode 1200 Baud

**11.1.2.307 #define HS\_BAUD\_14400 6**

HsMode 14400 Baud

**11.1.2.308 #define HS\_BAUD\_19200 7**

HsMode 19200 Baud

**11.1.2.309 #define HS\_BAUD\_230400 13**

HsMode 230400 Baud

**11.1.2.310 #define HS\_BAUD\_2400 1**

HsMode 2400 Baud

**11.1.2.311 #define HS\_BAUD\_28800 8**

HsMode 28800 Baud

**11.1.2.312 #define HS\_BAUD\_3600 2**

HsMode 3600 Baud

**11.1.2.313 #define HS\_BAUD\_38400 9**

HsMode 38400 Baud

**11.1.2.314 #define HS\_BAUD\_460800 14**

HsMode 460800 Baud

**11.1.2.315 #define HS\_BAUD\_4800 3**

HsMode 4800 Baud

**11.1.2.316 #define HS\_BAUD\_57600 10**

HsMode 57600 Baud

**11.1.2.317 #define HS\_BAUD\_7200 4**

HsMode 7200 Baud

**11.1.2.318 #define HS\_BAUD\_76800 11**

HsMode 76800 Baud

**11.1.2.319 #define HS\_BAUD\_921600 15**

HsMode 921600 Baud

**11.1.2.320 #define HS\_BAUD\_9600 5**

HsMode 9600 Baud

**Examples:**

[ex\\_RS485Control.nxc.](#)

**11.1.2.321 #define HS\_CMD\_READY 0x04**

A constant representing high speed direct command

**11.1.2.322 #define HS\_CTRL\_EXIT 2**

Ddisable the high speed port

**11.1.2.323 #define HS\_CTRL\_INIT 0**

Enable the high speed port

**Examples:**

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

**11.1.2.324 #define HS\_CTRL\_UART 1**

Setup the high speed port UART configuration

**Examples:**

[ex\\_RS485Control.nxc.](#)

**11.1.2.325 #define HS\_DISABLE 4**

HsState disable

**11.1.2.326 #define HS\_ENABLE 5**

HsState enable

**11.1.2.327 #define HS\_INIT\_RECEIVER 2**

HsState initialize receiver

**11.1.2.328 #define HS\_INITIALISE 1**

HsState initialize

**11.1.2.329 #define HS\_MODE\_10\_STOP 0x0000**

HsMode 1 stop bit

**11.1.2.330 #define HS\_MODE\_15\_STOP 0x1000**

HsMode 1.5 stop bits

**11.1.2.331 #define HS\_MODE\_20\_STOP 0x2000**

HsMode 2 stop bits

**11.1.2.332 #define HS\_MODE\_5\_DATA 0x0000**

HsMode 5 data bits

**11.1.2.333 #define HS\_MODE\_6\_DATA 0x0040**

HsMode 6 data bits

**11.1.2.334 #define HS\_MODE\_7\_DATA 0x0080**

HsMode 7 data bits

**11.1.2.335 #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

**11.1.2.336 #define HS\_MODE\_8\_DATA 0x00C0**

HsMode 8 data bits

**11.1.2.337 #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\\_RS485Control.nxc](#), [ex\\_RS485Uart.nxc](#), and [ex\\_SetHSMODE.nxc](#).

**11.1.2.338 #define HS\_MODE\_E\_PARITY 0x0000**

HsMode Even parity

**11.1.2.339 #define HS\_MODE\_M\_PARITY 0x0600**

HsMode Mark parity

**11.1.2.340 #define HS\_MODE\_N\_PARITY 0x0800**

HsMode No parity

**11.1.2.341 #define HS\_MODE\_O\_PARITY 0x0200**

HsMode Odd parity

**11.1.2.342 #define HS\_MODE\_S\_PARITY 0x0400**

HsMode Space parity

**11.1.2.343 #define HS\_SEND\_DATA 3**

HsState send data

**11.1.2.344 #define HS\_UPDATE 1**

HsFlags high speed update required

**11.1.2.345 #define HT\_CH1\_A 0**

Use IRReceiver channel 1 output A

**Examples:**

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

**11.1.2.346 #define HT\_CH1\_B 1**

Use IRReceiver channel 1 output B

**11.1.2.347 #define HT\_CH2\_A 2**

Use IRReceiver channel 2 output A

**11.1.2.348 #define HT\_CH2\_B 3**

Use IRReceiver channel 2 output B

**11.1.2.349 #define HT\_CH3\_A 4**

Use IRReceiver channel 3 output A

**11.1.2.350 #define HT\_CH3\_B 5**

Use IRReceiver channel 3 output B

**11.1.2.351 #define HT\_CH4\_A 6**

Use IRReceiver channel 4 output A

**11.1.2.352 #define HT\_CH4\_B 7**

Use IRReceiver channel 4 output B

**11.1.2.353 #define HT\_CMD\_COLOR2\_50HZ 0x35**

Set the Color2 sensor to 50Hz mode

**11.1.2.354 #define HT\_CMD\_COLOR2\_60HZ 0x36**

Set the Color2 sensor to 60Hz mode

**11.1.2.355 #define HT\_CMD\_COLOR2\_ACTIVE 0x00**

Set the Color2 sensor to active mode

**Examples:**

[ex\\_I2CSendCommand.nxc](#), [ex\\_I2CSendCommandEx.nxc](#), and [ex\\_SetHTColor2Mode.nxc](#).

**11.1.2.356 #define HT\_CMD\_COLOR2\_BLCAL 0x42**

Set the Color2 sensor to black level calibration mode

**11.1.2.357 #define HT\_CMD\_COLOR2\_FAR 0x46**

Set the Color2 sensor to far mode

**11.1.2.358 #define HT\_CMD\_COLOR2\_LED\_HI 0x48**

Set the Color2 sensor to LED high mode

**11.1.2.359 #define HT\_CMD\_COLOR2\_LED\_LOW 0x4C**

Set the Color2 sensor to LED low mode

**11.1.2.360 #define HT\_CMD\_COLOR2\_NEAR 0x4E**

Set the Color2 sensor to near mode

**11.1.2.361 #define HT\_CMD\_COLOR2\_PASSIVE 0x01**

Set the Color2 sensor to passive mode

**11.1.2.362 #define HT\_CMD\_COLOR2\_RAW 0x03**

Set the Color2 sensor to raw mode

**11.1.2.363 #define HT\_CMD\_COLOR2\_WBCAL 0x43**

Set the Color2 sensor to white level calibration mode

**11.1.2.364 #define HTIR2\_MODE\_1200 0**

Set IRSeeker2 to 1200 mode

**Examples:**

[ex\\_SetHTIRSeeker2Mode.nxc](#), and [ex\\_SetSensorBoolean.nxc](#).

**11.1.2.365 #define HTIR2\_MODE\_600 1**

Set IRSeeker2 to 600 mode

**11.1.2.366 #define HTIR2\_REG\_AC01 0x4A**

IRSeeker 2 AC 01 register

**11.1.2.367 #define HTIR2\_REG\_AC02 0x4B**

IRSeeker 2 AC 02 register

**11.1.2.368 #define HTIR2\_REG\_AC03 0x4C**

IRSeeker 2 AC 03 register

**11.1.2.369 #define HTIR2\_REG\_AC04 0x4D**

IRSeeker 2 AC 04 register

**11.1.2.370 #define HTIR2\_REG\_AC05 0x4E**

IRSeeker 2 AC 05 register

**11.1.2.371 #define HTIR2\_REG\_ACDIR 0x49**

IRSeeker 2 AC direction register

**11.1.2.372 #define HTIR2\_REG\_DC01 0x43**

IRSeeker 2 DC 01 register

**11.1.2.373 #define HTIR2\_REG\_DC02 0x44**

IRSeeker 2 DC 02 register

**11.1.2.374 #define HTIR2\_REG\_DC03 0x45**

IRSeeker 2 DC 03 register

**11.1.2.375 #define HTIR2\_REG\_DC04 0x46**

IRSeeker 2 DC 04 register

**11.1.2.376 #define HTIR2\_REG\_DC05 0x47**

IRSeeker 2 DC 05 register

**11.1.2.377 #define HTIR2\_REG\_DCAVG 0x48**

IRSeeker 2 DC average register

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**11.1.2.378 #define HTIR2\_REG\_DCDIR 0x42**

IRSeeker 2 DC direction register

**11.1.2.379 #define HTIR2\_REG\_MODE 0x41**

IRSeeker 2 mode register

**11.1.2.380 #define I2C\_REG\_CMD 0x41**

Standard NXT I2C device command register

**Examples:**

[ex\\_MSReadValue.nxc](#), [ex\\_MSReadValueEx.nxc](#), [ex\\_readi2cregister.nxc](#), and [ex\\_writei2cregister.nxc](#).

**11.1.2.381 #define I2C\_REG\_DEVICE\_ID 0x10**

Standard NXT I2C device ID register

**Examples:**

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

**11.1.2.382 #define I2C\_REG\_VENDOR\_ID 0x08**

Standard NXT I2C vendor ID register

**Examples:**

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

**11.1.2.383 #define I2C\_REG\_VERSION 0x00**

Standard NXT I2C version register

**Examples:**

[ex\\_I2CDeviceInfo.nxc](#), and [ex\\_I2CDeviceInfoEx.nxc](#).

**11.1.2.384 #define IN\_1 0x00**

Input port 1

**11.1.2.385 #define IN\_2 0x01**

Input port 2

**11.1.2.386 #define IN\_3 0x02**

Input port 3

**11.1.2.387 #define IN\_4 0x03**

Input port 4



**11.1.2.388 #define IN\_MODE\_ANGLESTEP 0xE0**

RCX rotation sensor (16 ticks per revolution)

**11.1.2.389 #define IN\_MODE\_BOOLEAN 0x20**

Boolean value (0 or 1)

**11.1.2.390 #define IN\_MODE\_CELSIUS 0xA0**

RCX temperature sensor value in degrees celcius

**11.1.2.391 #define IN\_MODE\_FAHRENHEIT 0xC0**

RCX temperature sensor value in degrees fahrenheit

**11.1.2.392 #define IN\_MODE\_MODEMASK 0xE0**

Mask for the mode without any slope value

**11.1.2.393 #define IN\_MODE\_PCTFULLSCALE 0x80**

Scaled value from 0 to 100

**11.1.2.394 #define IN\_MODE\_PERIODCOUNTER 0x60**

Counts the number of boolean periods

**11.1.2.395 #define IN\_MODE\_RAW 0x00**

Raw value from 0 to 1023

**11.1.2.396 #define IN\_MODE\_SLOPEMASK 0x1F**

Mask for slope parameter added to mode

**11.1.2.397 #define IN\_MODE\_TRANSITIONCNT 0x40**

Counts the number of boolean transitions

**11.1.2.398 #define IN\_TYPE\_ANGLE 0x04**

RCX rotation sensor

**11.1.2.399 #define IN\_TYPE\_COLORBLUE 0x10**

NXT 2.0 color sensor with blue light

**11.1.2.400 #define IN\_TYPE\_COLOREXIT 0x12**

NXT 2.0 color sensor internal state

**11.1.2.401 #define IN\_TYPE\_COLORFULL 0x0D**

NXT 2.0 color sensor in full color mode

**11.1.2.402 #define IN\_TYPE\_COLORGREEN 0x0F**

NXT 2.0 color sensor with green light

**11.1.2.403 #define IN\_TYPE\_COLORNONE 0x11**

NXT 2.0 color sensor with no light

**11.1.2.404 #define IN\_TYPE\_COLORRED 0x0E**

NXT 2.0 color sensor with red light

**11.1.2.405 #define IN\_TYPE\_CUSTOM 0x09**

NXT custom sensor

**11.1.2.406 #define IN\_TYPE\_HISPEED 0x0C**

NXT Hi-speed port (only S4)

**11.1.2.407 #define IN\_TYPE\_LIGHT\_ACTIVE 0x05**

NXT light sensor with light

**11.1.2.408 #define IN\_TYPE\_LIGHT\_INACTIVE 0x06**

NXT light sensor without light

**11.1.2.409 #define IN\_TYPE\_LOWSPEED 0x0A**

NXT I2C digital sensor

**11.1.2.410 #define IN\_TYPE\_LOWSPEED\_9V 0x0B**

NXT I2C digital sensor with 9V power

**11.1.2.411 #define IN\_TYPE\_NO\_SENSOR 0x00**

No sensor configured

**11.1.2.412 #define IN\_TYPE\_REFLECTION 0x03**

RCX light sensor

**11.1.2.413 #define IN\_TYPE\_SOUND\_DB 0x07**

NXT sound sensor with dB scaling

**11.1.2.414 #define IN\_TYPE\_SOUND\_DBA 0x08**

NXT sound sensor with dBA scaling

**11.1.2.415 #define IN\_TYPE\_SWITCH 0x01**

NXT or RCX touch sensor

**11.1.2.416 #define IN\_TYPE\_TEMPERATURE 0x02**

RCX temperature sensor

**11.1.2.417 #define INPUT\_BLACKCOLOR 1**

The color value is black

**11.1.2.418 #define INPUT\_BLANK 3**

Access the blank value from color sensor value arrays

**11.1.2.419 #define INPUT\_BLUE 2**

Access the blue value from color sensor value arrays

**11.1.2.420 #define INPUT\_BLUECOLOR 2**

The color value is blue

**11.1.2.421 #define INPUT\_CAL\_POINT\_0 0**

Calibration point 0

**Examples:**

[ex\\_ColorCalibration.nxc](#), and [ex\\_ColorCalLimits.nxc](#).

**11.1.2.422 #define INPUT\_CAL\_POINT\_1 1**

Calibration point 1

**11.1.2.423 #define INPUT\_CAL\_POINT\_2 2**

Calibration point 2

**11.1.2.424 #define INPUT\_CUSTOM9V 0x01**

Custom sensor 9V

**11.1.2.425 #define INPUT\_CUSTOMACTIVE 0x02**

Custom sensor active

**11.1.2.426 #define INPUT\_CUSTOMINACTIVE 0x00**

Custom sensor inactive

**11.1.2.427 #define INPUT\_DIGI0 1**

Digital pin 0

**11.1.2.428 #define INPUT\_DIGI1 2**

Digital pin 1

**11.1.2.429 #define INPUT\_GREEN 1**

Access the green value from color sensor value arrays

**11.1.2.430 #define INPUT\_GREENCOLOR 3**

The color value is green

**11.1.2.431 #define INPUT\_INVALID\_DATA 0x01**

Invalid data flag

**11.1.2.432 #define INPUT\_NO\_OF\_COLORS 4**

The number of entries in the color sensor value arrays

**11.1.2.433 #define INPUT\_NO\_OF\_POINTS 3**

The number of calibration points

**11.1.2.434 #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](#).

**11.1.2.435 #define INPUT\_REDCOLOR 5**

The color value is red

**11.1.2.436 #define INPUT\_RESETCAL 0x80**

Unused calibration state constant

**11.1.2.437 #define INPUT\_RUNNINGCAL 0x20**

Unused calibration state constant

**11.1.2.438 #define INPUT\_SENSORCAL 0x01**

The state returned while the color sensor is calibrating

**11.1.2.439 #define INPUT\_SENSOROFF 0x02**

The state returned once calibration has completed

**11.1.2.440 #define INPUT\_STARTCAL 0x40**

Unused calibration state constant

**11.1.2.441 #define INPUT\_WHITECOLOR 6**

The color value is white

**11.1.2.442 #define INPUT\_YELLOWCOLOR 4**

The color value is yellow

**11.1.2.443 #define InputMode 1**

Input mode field. Contains one of the sensor mode constants. Read/write.

**11.1.2.444 #define InputModuleID 0x00030001**

The input module ID

**11.1.2.445 #define InputModuleName "Input.mod"**

The input module name.

**11.1.2.446 #define InputOffsetADRaw(p) (((p)\*20)+2)**

Read the AD raw sensor value (2 bytes) uword

**11.1.2.447 #define InputOffsetColorADRaw(p, nc) (80+((p)\*84)+52+((nc)\*2))**

Read AD raw color sensor values

**11.1.2.448 #define InputOffsetColorCalibration(p, np, nc) (80+((p)\*84)+0+((np)\*16)+((nc)\*4))**

Read/write color calibration point values

**11.1.2.449 #define InputOffsetColorCalibrationState(p) (80+((p)\*84)+80)**

Read color sensor calibration state

**11.1.2.450 #define InputOffsetColorCalLimits(p, np) (80+((p)\*84)+48+((np)\*2))**

Read/write color calibration limits

**11.1.2.451 #define InputOffsetColorSensorBoolean(p, nc) (80+((p)\*84)+76+((nc)\*2))**

Read color sensor boolean values

**11.1.2.452 #define InputOffsetColorSensorRaw(p, nc) (80+((p)\*84)+60+((nc)\*2))**

Read raw color sensor values

**11.1.2.453 #define InputOffsetColorSensorValue(p, nc) (80+((p)\*84)+68+((nc)\*2))**

Read scaled color sensor values

**11.1.2.454 #define InputOffsetCustomActiveStatus(p) (((p)\*20)+15)**

Read/write the active or inactive state of the custom sensor

**11.1.2.455 #define InputOffsetCustomPctFullScale(p) (((p)\*20)+14)**

Read/write the Pct full scale of the custom sensor

**11.1.2.456 #define InputOffsetCustomZeroOffset(p) (((p)\*20)+0)**

Read/write the zero offset of a custom sensor (2 bytes) uword

**11.1.2.457 #define InputOffsetDigiPinsDir(p) (((p)\*20)+11)**

Read/write the direction of the Digital pins (1 is output, 0 is input)

**11.1.2.458 #define InputOffsetDigiPinsIn(p) (((p)\*20)+12)**

Read/write the status of the digital pins

**11.1.2.459 #define InputOffsetDigiPinsOut(p) (((p)\*20)+13)**

Read/write the output level of the digital pins

**11.1.2.460 #define InputOffsetInvalidData(p) (((p)\*20)+16)**

Indicates whether data is invalid (1) or valid (0)

**11.1.2.461 #define InputOffsetSensorBoolean(p) (((p)\*20)+10)**

Read the sensor boolean value

**11.1.2.462 #define InputOffsetSensorMode(p) (((p)\*20)+9)**

Read/write the sensor mode

**11.1.2.463 #define InputOffsetSensorRaw(p) (((p)\*20)+4)**

Read the raw sensor value (2 bytes) uword

**11.1.2.464 #define InputOffsetSensorType(p) (((p)\*20)+8)**

Read/write the sensor type

**11.1.2.465 #define InputOffsetSensorValue(p) (((p)\*20)+6)**

Read/write the scaled sensor value (2 bytes) sword

**11.1.2.466 #define INT\_MAX 32767**

The maximum value of the int type

**11.1.2.467 #define INT\_MIN -32767**

The minimum value of the int type

**11.1.2.468 #define INTF\_BTOff 13**

Turn off the bluetooth radio

**Examples:**

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

**11.1.2.469 #define INTF\_BTOn 12**

Turn on the bluetooth radio

**11.1.2.470 #define INTF\_CONNECT 3**

Connect to one of the known devices

**11.1.2.471 #define INTF\_CONNECTBYNAME 18**

Connect to a bluetooth device by name

**11.1.2.472 #define INTF\_CONNECTREQ 17**

Connection request from another device

**11.1.2.473 #define INTF\_DISCONNECT 4**

Disconnect from one of the connected devices

**11.1.2.474 #define INTF\_DISCONNECTALL 5**

Disconnect all devices



**11.1.2.475 #define INTF\_EXTREAD 15**

External read request

**11.1.2.476 #define INTF\_FACTORYRESET 11**

Reset bluetooth settings to factory values

**11.1.2.477 #define INTF\_OPENSTREAM 9**

Open a bluetooth stream

**11.1.2.478 #define INTF\_PINREQ 16**

Bluetooth PIN request

**11.1.2.479 #define INTF\_REMOVEDEVICE 6**

Remove a device from the known devices table

**11.1.2.480 #define INTF\_SEARCH 1**

Search for bluetooth devices

**11.1.2.481 #define INTF\_SENDDATA 10**

Send data over a bluetooth connection

**11.1.2.482 #define INTF\_SENDFILE 0**

Send a file via bluetooth to another device

**11.1.2.483 #define INTF\_SETBTNAME 14**

Set the bluetooth name

**11.1.2.484 #define INTF\_SETCMDMODE 8**

Set bluetooth into command mode

**11.1.2.485 #define INTF\_STOPSEARCH 2**

Stop searching for bluetooth devices

**11.1.2.486 #define INTF\_VISIBILITY 7**

Set the bluetooth visibility on or off

**11.1.2.487 #define InvalidData 5**

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

**11.1.2.488 #define IOCTRL\_BOOT 0xA55A**

Reboot the NXT into SAMBA mode

**11.1.2.489 #define IOCTRL\_POWERDOWN 0x5A00**

Power down the NXT

**11.1.2.490 #define IOCtrlModuleID 0x00060001**

The IOCtrl module ID

**11.1.2.491 #define IOCtrlModuleName "IOCtrl.mod"**

The IOCtrl module name

**11.1.2.492 #define IOCtrlOffsetPowerOn 0**

Offset to power on field

**11.1.2.493 #define IOMapRead 32**

Read data from one of the firmware module's IOMap structures using the module's name

**11.1.2.494 #define IOMapReadByID 78**

Read data from one of the firmware module's IOMap structures using the module's ID

**11.1.2.495 #define IOMapWrite 33**

Write data to one of the firmware module's IOMap structures using the module's name

**11.1.2.496 #define IOMapWriteByID 79**

Write data to one of the firmware module's IOMap structures using the module's ID

**11.1.2.497 #define KeepAlive 31**

Reset the NXT sleep timer

**11.1.2.498 #define LCD\_LINE1 56**

The 1st line of the LCD screen

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrex.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\\_ClearLine.nxc](#), [ex\\_contrast.nxc](#), [ex\\_Copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_syscall.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysCommBTOnOff.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\\_sysmessageread.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), and [ex\\_SysWriteSemData.nxc](#).

**11.1.2.499 #define LCD\_LINE2 48**

The 2nd line of the LCD screen

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrex.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\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SubStr.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_sysfileread.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**11.1.2.500 #define LCD\_LINE3 40**

The 3rd line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SysCommBTConnection.nxc](#), and [ex\\_TextOut.nxc](#).

**11.1.2.501 #define LCD\_LINE4 32**

The 4th line of the LCD screen

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrex.nxc](#), [ex\\_arraybuild.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ButtonPressed.nxc](#), [ex\\_ctype.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), and [ex\\_SysDatalogGetTimes.nxc](#).

**11.1.2.502 #define LCD\_LINE5 24**

The 5th line of the LCD screen

**Examples:**

[ex\\_arraybuild.nxc](#), [ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), and [ex\\_SysDatalogGetTimes.nxc](#).

**11.1.2.503 #define LCD\_LINE6 16**

The 6th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), and [ex\\_syslistfiles.nxc](#).

**11.1.2.504 #define LCD\_LINE7 8**

The 7th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), and [ex\\_string.nxc](#).

**11.1.2.505 #define LCD\_LINE8 0**

The 8th line of the LCD screen

**Examples:**

[ex\\_ArraySort.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), and [ex\\_string.nxc](#).

**11.1.2.506 #define LDR\_APPENDNOTPOSSIBLE 0x8D00**

Only datafiles can be appended to.

**11.1.2.507 #define LDR\_BTBUSY 0x9400**

The bluetooth system is busy.

**11.1.2.508 #define LDR\_BTCONNECTFAIL 0x9500**

Bluetooth connection attempt failed.

**11.1.2.509 #define LDR\_BTTIMEOUT 0x9600**

A timeout in the bluetooth system has occurred.

**11.1.2.510 #define LDR\_CMD\_BOOTCMD 0x97**

Reboot the NXT into SAMBA mode

**11.1.2.511 #define LDR\_CMD\_BTFACTORYRESET 0xA4**

Reset bluetooth configuration to factory defaults

**11.1.2.512 #define LDR\_CMD\_BTGETADR 0x9A**

Get the NXT's bluetooth brick address

**11.1.2.513 #define LDR\_CMD\_CLOSE 0x84**

Close a file handle

**11.1.2.514 #define LDR\_CMD\_CLOSEMODHANDLE 0x92**

Close a module handle

**11.1.2.515 #define LDR\_CMD\_CROPDATAFILE 0x8D**

Crop a data file to its used space

**11.1.2.516 #define LDR\_CMD\_DELETE 0x85**

Delete a file

**11.1.2.517 #define LDR\_CMD\_DELETEUSERFLASH 0xA0**

Delete all files from user flash memory

**11.1.2.518 #define LDR\_CMD\_DEVICEINFO 0x9B**

Read device information

**11.1.2.519 #define LDR\_CMD\_FINDFIRST 0x86**

Find the first file matching the specified pattern

**11.1.2.520 #define LDR\_CMD\_FINDFIRSTMODULE 0x90**

Find the first module matching the specified pattern

**11.1.2.521 #define LDR\_CMD\_FINDNEXT 0x87**

Find the next file matching the specified pattern

**11.1.2.522 #define LDR\_CMD\_FINDNEXTMODULE 0x91**

Find the next module matching the specified pattern

**11.1.2.523 #define LDR\_CMD\_IOMAPREAD 0x94**

Read data from a module IOMAP

**11.1.2.524 #define LDR\_CMD\_IOMAPWRITE 0x95**

Write data to a module IOMAP

**11.1.2.525 #define LDR\_CMD\_OPENAPPENDDATA 0x8C**

Open a data file for appending

**11.1.2.526 #define LDR\_CMD\_OPENREAD 0x80**

Open a file for reading

**11.1.2.527 #define LDR\_CMD\_OPENREADLINEAR 0x8A**

Open a linear file for reading

**11.1.2.528 #define LDR\_CMD\_OPENWRITE 0x81**

Open a file for writing

**11.1.2.529 #define LDR\_CMD\_OPENWRITEDATA 0x8B**

Open a data file for writing

**11.1.2.530 #define LDR\_CMD\_OPENWRITELINEAR 0x89**

Open a linear file for writing

**11.1.2.531 #define LDR\_CMD\_POLLCMD 0xA2**

Poll command

**11.1.2.532 #define LDR\_CMD\_POLLCMDLEN 0xA1**

Read poll command length

**11.1.2.533 #define LDR\_CMD\_READ 0x82**

Read from a file

**11.1.2.534 #define LDR\_CMD\_RENAMEFILE 0xA3**

Rename a file

**11.1.2.535 #define LDR\_CMD\_RESIZEDATAFILE 0xD0**

Resize a data file

**11.1.2.536 #define LDR\_CMD\_SEEKFROMCURRENT 0xD2**

Seek from the current position

**11.1.2.537 #define LDR\_CMD\_SEEKFROMEND 0xD3**

Seek from the end of the file

**11.1.2.538 #define LDR\_CMD\_SEEKFROMSTART 0xD1**

Seek from the start of the file

**11.1.2.539 #define LDR\_CMD\_SETBRICKNAME 0x98**

Set the NXT's brick name

**11.1.2.540 #define LDR\_CMD\_VERSIONS 0x88**

Read firmware version information

**11.1.2.541 #define LDR\_CMD\_WRITE 0x83**

Write to a file

**11.1.2.542 #define LDR\_ENDOFFILE 0x8500**

The end of the file has been reached.

**11.1.2.543 #define LDR\_EOFEXPECTED 0x8400**

EOF expected.

**11.1.2.544 #define LDR\_FILEEXISTS 0x8F00**

A file with the same name already exists.

**11.1.2.545 #define LDR\_FILEISBUSY 0x8B00**

The file is already being used.

**11.1.2.546 #define LDR\_FILEISFULL 0x8E00**

The allocated file size has been filled.

**11.1.2.547 #define LDR\_FILENOTFOUND 0x8700**

No files matched the search criteria.

**11.1.2.548 #define LDR\_FILETX\_CLOSEERROR 0x9B00**

Error transmitting file: attempt to close file failed.

**11.1.2.549 #define LDR\_FILETX\_DSTEXISTS 0x9800**

Error transmitting file: destination file exists.

**11.1.2.550 #define LDR\_FILETX\_SRCMISSING 0x9900**

Error transmitting file: source file is missing.

**11.1.2.551 #define LDR\_FILETX\_STREAMERROR 0x9A00**

Error transmitting file: a stream error occurred.



**11.1.2.552 #define LDR\_FILETX\_TIMEOUT 0x9700**

Error transmitting file: a timeout occurred.

**11.1.2.553 #define LDR\_HANDLEALREADYCLOSED 0x8800**

The file handle has already been closed.

**11.1.2.554 #define LDR\_ILLEGALFILENAME 0x9200**

Filename length too long or attempted open a system file (\*.rx, \*.rtm, or \*.sys) for writing as a datafile.

**11.1.2.555 #define LDR\_ILLEGALHANDLE 0x9300**

Invalid file handle.

**11.1.2.556 #define LDR\_INPROGRESS 0x0001**

The function is executing but has not yet completed.

**11.1.2.557 #define LDR\_INVALIDSEEK 0x9C00**

Invalid file seek operation.

**11.1.2.558 #define LDR\_MODULENOTFOUND 0x9000**

No modules matched the specified search criteria.

**11.1.2.559 #define LDR\_NOLINEARSPACE 0x8900**

Not enough linear flash memory is available.

**11.1.2.560 #define LDR\_NOMOREFILES 0x8300**

The maximum number of files has been reached.

**11.1.2.561 #define LDR\_NOMOREHANDLES 0x8100**

All available file handles are in use.

**11.1.2.562 #define LDR\_NOSPACE 0x8200**

Not enough free flash memory for the specified file size.

**11.1.2.563 #define LDR\_NOTLINEARFILE 0x8600**

The specified file is not linear.

**11.1.2.564 #define LDR\_NOWRITEBUFFERS 0x8C00**

No more write buffers are available.

**11.1.2.565 #define LDR\_OUTOFBOUNDARY 0x9100**

Specified IOMap offset is outside the bounds of the IOMap.

**11.1.2.566 #define LDR\_REQPIN 0x0002**

A PIN exchange request is in progress.

**11.1.2.567 #define LDR\_SUCCESS 0x0000**

The function completed successfully.

**Examples:**

[ex\\_syscommbtcheckstatus.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_sysfilerename.nxc](#), and [ex\\_sysfileresolvehandle.nxc](#).

**11.1.2.568 #define LDR\_UNDEFINEDERROR 0x8A00**

An undefined error has occurred.

**11.1.2.569 #define ListFiles 47**

List files that match the specified filename pattern

**11.1.2.570 #define LoaderExecuteFunction 82**

Execute one of the Loader module's internal functions

**11.1.2.571 #define LoaderModuleID 0x00090001**

The Loader module ID

**11.1.2.572 #define LoaderModuleName "Loader.mod"**

The Loader module name

**11.1.2.573 #define LoaderOffsetFreeUserFlash 4**

Offset to the amount of free user flash

**11.1.2.574 #define LoaderOffsetPFunc 0**

Offset to the Loader module function pointer

**11.1.2.575 #define LONG\_MAX 2147483647**

The maximum value of the long type

**11.1.2.576 #define LONG\_MIN -2147483647**

The minimum value of the long type

**11.1.2.577 #define LOWSPEED\_CH\_NOT\_READY 1**

Lowspeed port is not ready

**11.1.2.578 #define LOWSPEED\_COMMUNICATING 3**

Channel is actively communicating

**11.1.2.579 #define LOWSPEED\_DATA\_RECEIVED 3**

Lowspeed port is in data received mode

**11.1.2.580 #define LOWSPEED\_DONE 5**

Channel is done communicating

**11.1.2.581 #define LOWSPEED\_ERROR 4**

Channel is in an error state

**11.1.2.582 #define LOWSPEED\_IDLE 0**

Channel is idle

**Examples:**

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

**11.1.2.583 #define LOWSPEED\_INIT 1**

Channel is being initialized

**11.1.2.584 #define LOWSPEED\_LOAD\_BUFFER 2**

Channel buffer is loading

**11.1.2.585 #define LOWSPEED\_NO\_ERROR 0**

Lowspeed port has no error

**11.1.2.586 #define LOWSPEED\_RECEIVING 2**

Lowspeed port is in receiving mode

**11.1.2.587 #define LOWSPEED\_RX\_ERROR 3**

Lowspeed port encountered an error while receiving data

**11.1.2.588 #define LOWSPEED\_TRANSMITTING 1**

Lowspeed port is in transmitting mode

**11.1.2.589 #define LOWSPEED\_TX\_ERROR 2**

Lowspeed port encountered an error while transmitting data

**11.1.2.590 #define LowSpeedModuleID 0x000B0001**

The low speed module ID

**11.1.2.591 #define LowSpeedModuleName "Low Speed.mod"**

The low speed module name

**11.1.2.592 #define LowSpeedOffsetChannelState(p) ((p)+156)**

R - Lowspeed channel state (1 byte)

**11.1.2.593 #define LowSpeedOffsetErrorType(p) ((p)+160)**

R - Lowspeed port error type (1 byte)

**11.1.2.594 #define LowSpeedOffsetInBufBuf(p) (((p)\*19)+0)**

RW - Input buffer data buffer field offset (16 bytes)

**11.1.2.595** `#define LowSpeedOffsetInBufBytesToRx(p) (((p)*19)+18)`

RW - Input buffer bytes to receive field offset (1 byte)

**11.1.2.596** `#define LowSpeedOffsetInBufInPtr(p) (((p)*19)+16)`

RW - Input buffer in pointer field offset (1 byte)

**11.1.2.597** `#define LowSpeedOffsetInBufOutPtr(p) (((p)*19)+17)`

RW - Input buffer out pointer field offset (1 byte)

**11.1.2.598** `#define LowSpeedOffsetMode(p) ((p)+152)`

R - Lowspeed port mode (1 byte)

**11.1.2.599** `#define LowSpeedOffsetNoRestartOnRead 166`

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

**11.1.2.600** `#define LowSpeedOffsetOutBufBuf(p) (((p)*19)+76)`

RW - Output buffer data buffer field offset (16 bytes)

**11.1.2.601** `#define LowSpeedOffsetOutBufBytesToRx(p) (((p)*19)+94)`

RW - Output buffer bytes to receive field offset (1 byte)

**11.1.2.602** `#define LowSpeedOffsetOutBufInPtr(p) (((p)*19)+92)`

RW - Output buffer in pointer field offset (1 byte)

**11.1.2.603** `#define LowSpeedOffsetOutBufOutPtr(p) (((p)*19)+93)`

RW - Output buffer out pointer field offset (1 byte)

**11.1.2.604** `#define LowSpeedOffsetSpeed 165`

R - Lowspeed speed (unused)

**11.1.2.605** `#define LowSpeedOffsetState 164`

R - Lowspeed state (all channels)

**11.1.2.606 #define LR\_COULD\_NOT\_SAVE 0x51**

Bluetooth list result could not save

**11.1.2.607 #define LR\_ENTRY\_REMOVED 0x53**

Bluetooth list result entry removed

**11.1.2.608 #define LR\_STORE\_IS\_FULL 0x52**

Bluetooth list result store is full

**11.1.2.609 #define LR\_SUCCESS 0x50**

Bluetooth list result success

**11.1.2.610 #define LR\_UNKNOWN\_ADDR 0x54**

Bluetooth list result unknown address

**11.1.2.611 #define LSREAD\_NO\_RESTART\_1 0x01**

No restart on read for channel 1

**11.1.2.612 #define LSREAD\_NO\_RESTART\_2 0x02**

No restart on read for channel 2

**11.1.2.613 #define LSREAD\_NO\_RESTART\_3 0x04**

No restart on read for channel 3

**11.1.2.614 #define LSREAD\_NO\_RESTART\_4 0x08**

No restart on read for channel 4

**11.1.2.615 #define LSREAD\_NO\_RESTART\_MASK 0x10**

No restart mask

**11.1.2.616 #define LSREAD\_RESTART\_ALL 0x00**

Restart on read for all channels (default)

**11.1.2.617 #define LSREAD\_RESTART\_NONE 0x0F**

No restart on read for all channels

**11.1.2.618 #define MAILBOX1 0**

Mailbox number 1

**Examples:**

```
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.
```

**11.1.2.619 #define MAILBOX10 9**

Mailbox number 10

**11.1.2.620 #define MAILBOX2 1**

Mailbox number 2

**11.1.2.621 #define MAILBOX3 2**

Mailbox number 3

**11.1.2.622 #define MAILBOX4 3**

Mailbox number 4

**11.1.2.623 #define MAILBOX5 4**

Mailbox number 5

**11.1.2.624 #define MAILBOX6 5**

Mailbox number 6

**11.1.2.625 #define MAILBOX7 6**

Mailbox number 7

**11.1.2.626 #define MAILBOX8 7**

Mailbox number 8

**11.1.2.627 #define MAILBOX9 8**

Mailbox number 9

**11.1.2.628 #define MAX\_BT\_MSG\_SIZE 60000**

Max Bluetooth Message Size

**11.1.2.629 #define MENUICON\_CENTER 1**

Center icon

**11.1.2.630 #define MENUICON\_LEFT 0**

Left icon

**11.1.2.631 #define MENUICON\_RIGHT 2**

Right icon

**11.1.2.632 #define MENUICONS 3**

The number of menu icons

**11.1.2.633 #define MENUTEXT 2**

Center icon text

**11.1.2.634 #define MessageRead 27**

Read a message from a mailbox

**11.1.2.635 #define MessageWrite 26**

Write a message to a mailbox

**11.1.2.636 #define MIN\_1 60000**

1 minute

**Examples:**

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

**11.1.2.637 #define MS\_1 1**

1 millisecond



**11.1.2.638 #define MS\_10 10**

10 milliseconds

**11.1.2.639 #define MS\_100 100**

100 milliseconds

**Examples:**

[ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**11.1.2.640 #define MS\_150 150**

150 milliseconds

**11.1.2.641 #define MS\_2 2**

2 milliseconds

**11.1.2.642 #define MS\_20 20**

20 milliseconds

**Examples:**

[ex\\_dispgayout.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).

**11.1.2.643 #define MS\_200 200**

200 milliseconds

**Examples:**

[ex\\_dispgoutex.nxc](#), and [ex\\_playtones.nxc](#).

**11.1.2.644 #define MS\_250 250**

250 milliseconds

**11.1.2.645 #define MS\_3 3**

3 milliseconds

**11.1.2.646 #define MS\_30 30**

30 milliseconds

**11.1.2.647 #define MS\_300 300**

300 milliseconds

**11.1.2.648 #define MS\_350 350**

350 milliseconds

**11.1.2.649 #define MS\_4 4**

4 milliseconds

**11.1.2.650 #define MS\_40 40**

40 milliseconds

**11.1.2.651 #define MS\_400 400**

400 milliseconds

**11.1.2.652 #define MS\_450 450**

450 milliseconds

**11.1.2.653 #define MS\_5 5**

5 milliseconds

**Examples:**

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

**11.1.2.654 #define MS\_50 50**

50 milliseconds

**Examples:**

[ex\\_CircleOut.nxc](#), and [ex\\_playtones.nxc](#).

**11.1.2.655 #define MS\_500 500**

500 milliseconds

**Examples:**

[alternating\\_tasks.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_playsound.nxc](#), [ex\\_yield.nxc](#), and [util\\_rpm.nxc](#).

**11.1.2.656 #define MS\_6 6**

6 milliseconds

**11.1.2.657 #define MS\_60 60**

60 milliseconds

**11.1.2.658 #define MS\_600 600**

600 milliseconds

**11.1.2.659 #define MS\_7 7**

7 milliseconds

**11.1.2.660 #define MS\_70 70**

70 milliseconds

**11.1.2.661 #define MS\_700 700**

700 milliseconds

**11.1.2.662 #define MS\_8 8**

8 milliseconds

**11.1.2.663 #define MS\_80 80**

80 milliseconds

**11.1.2.664 #define MS\_800 800**

800 milliseconds

**11.1.2.665 #define MS\_9 9**

9 milliseconds

**11.1.2.666 #define MS\_90 90**

90 milliseconds

**11.1.2.667 #define MS\_900 900**

900 milliseconds

**11.1.2.668 #define MS\_CMD\_ADPA\_OFF 0x4F**

Turn MindSensors ADPA mode off

**11.1.2.669 #define MS\_CMD\_ADPA\_ON 0x4E**

Turn MindSensors ADPA mode on

**11.1.2.670 #define MS\_CMD\_DEENERGIZED 0x44**

De-energize the MindSensors device

**11.1.2.671 #define MS\_CMD\_ENERGIZED 0x45**

Energize the MindSensors device

**11.1.2.672 #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](#).

**11.1.2.673 #define NO\_ERR 0**

Successful execution of the specified command

**Examples:**

[ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysCommBTOnOff.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](#), and [ex\\_sysmessageread.nxc](#).

**11.1.2.674 #define NO\_OF\_BTNS 4**

The number of NXT buttons.

**11.1.2.675 #define NormalizedValue 3**

Normalized value field. Contains the current normalized analog sensor value. Read only.

**11.1.2.676 #define NRLINK\_CMD\_2400 0x44**

Set NRLink to 2400 baud

**11.1.2.677 #define NRLINK\_CMD\_4800 0x48**

Set NRLink to 4800 baud

**11.1.2.678 #define NRLINK\_CMD\_FLUSH 0x46**

Flush the NRLink

**11.1.2.679 #define NRLINK\_CMD\_IR\_LONG 0x4C**

Set the NRLink to long range IR

**11.1.2.680 #define NRLINK\_CMD\_IR\_SHORT 0x53**

Set the NRLink to short range IR

**11.1.2.681 #define NRLINK\_CMD\_RUN\_MACRO 0x52**

Run an NRLink macro

**11.1.2.682 #define NRLINK\_CMD\_SET\_PF 0x50**

Set the NRLink to Power Function mode

**11.1.2.683 #define NRLINK\_CMD\_SET\_RCX 0x58**

Set the NRLink to RCX mode

**11.1.2.684 #define NRLINK\_CMD\_SET\_TRAIN 0x54**

Set the NRLink to IR Train mode

**11.1.2.685 #define NRLINK\_CMD\_TX\_RAW 0x55**

Set the NRLink to transmit raw bytes

**11.1.2.686 #define NRLINK\_REG\_BYTES 0x40**

The NRLink bytes register

**11.1.2.687 #define NRLINK\_REG\_DATA 0x42**

The NRLink data register

**11.1.2.688 #define NRLINK\_REG\_EEPROM 0x50**

The NRLink eeprom register

**11.1.2.689 #define NULL 0**

A constant representing NULL

**11.1.2.690 #define OPARR\_MAX 0x05**

Calculate the maximum value of the elements in the numeric input array

**Examples:**

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

**11.1.2.691 #define OPARR\_MEAN 0x01**

Calculate the mean value for the elements in the numeric input array

**11.1.2.692 #define OPARR\_MIN 0x04**

Calculate the minimum value of the elements in the numeric input array

**11.1.2.693 #define OPARR\_SORT 0x06**

Sort the elements in the numeric input array

**11.1.2.694 #define OPARR\_STD 0x03**

Calculate the standard deviation of the elements in the numeric input array

**11.1.2.695 #define OPARR\_SUM 0x00**

Calculate the sum of the elements in the numeric input array

**11.1.2.696 #define OPARR\_SUMSQR 0x02**

Calculate the sum of the squares of the elements in the numeric input array

**11.1.2.697 #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\\_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\\_RemoteResetMotorPosition.nxc](#), [ex\\_RemoteSetOutputState.nxc](#), [ex\\_rotatemotor.nxc](#), [ex\\_rotatemotorpid.nxc](#), and [ex\\_yield.nxc](#).

**11.1.2.698 #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\\_rotatemotorexpid.nxc](#), and [ex\\_setoutput.nxc](#).

**11.1.2.699 #define OUT\_ABC 0x06**

Output ports A, B, and C

**11.1.2.700 #define OUT\_AC 0x04**

Output ports A and C

**11.1.2.701 #define OUT\_B 0x01**

Output port B

**11.1.2.702 #define OUT\_BC 0x05**

Output ports B and C

**11.1.2.703 #define OUT\_C 0x02**

Output port C

**11.1.2.704 #define OUT\_MODE\_BRAKE 0x02**

Uses electronic braking to outputs

**11.1.2.705 #define OUT\_MODE\_COAST 0x00**

No power and no braking so motors rotate freely.

**11.1.2.706 #define OUT\_MODE\_MOTORON 0x01**

Enables PWM power to the outputs given the Power setting

**Examples:**

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

**11.1.2.707 #define OUT\_MODE\_REGMETHOD 0xF0**

Mask for unimplemented regulation mode

**11.1.2.708 #define OUT\_MODE\_REGULATED 0x04**

Enables active power regulation using the RegMode value

**11.1.2.709 #define OUT\_OPTION\_HOLDATLIMIT 0x10**

Option to have the firmware hold the motor when it reaches the tachometer limit

**11.1.2.710 #define OUT\_OPTION\_RAMPDOWNTOLIMIT 0x20**

Option to have the firmware rampdown the motor power as it approaches the tachometer limit (not implemented yet)

**11.1.2.711 #define OUT\_REGMODE\_IDLE 0**

No motor regulation.

**Examples:**

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

**11.1.2.712 #define OUT\_REGMODE\_SPEED 1**

Regulate a motor's speed (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](#).



**11.1.2.713 #define OUT\_REGMODE\_SYNC 2**

Synchronize the rotation of two motors.

**11.1.2.714 #define OUT\_RUNSTATE\_HOLD 0x60**

Set motor run state to hold at the current position.

**11.1.2.715 #define OUT\_RUNSTATE\_IDLE 0x00**

Disable all power to motors.

**11.1.2.716 #define OUT\_RUNSTATE\_RAMPDOWN 0x40**

Enable ramping down from a current Power to a new (lower) Power over a specified TachoLimit goal.

**11.1.2.717 #define OUT\_RUNSTATE\_RAMPUP 0x10**

Enable ramping up from a current Power to a new (higher) Power over a specified TachoLimit goal.

**11.1.2.718 #define OUT\_RUNSTATE\_RUNNING 0x20**

Enable power to motors at the specified Power level.

**Examples:**

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

**11.1.2.719 #define OutputMode 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 OutputMode value. Use [UF\\_UPDATE\\_MODE](#) with [UpdateFlags](#) to commit changes to this field.

**11.1.2.720 #define OutputModuleID 0x00020001**

The output module ID

**11.1.2.721 #define OutputModuleName "Output.mod"**

The output module name

**11.1.2.722 #define OutputOffsetActualSpeed(p) (((p)\*32)+21)**

R - Holds the current motor speed (1 byte) sbyte

**11.1.2.723 #define OutputOffsetBlockTachoCount(p) (((p)\*32)+4)**

R - Holds current number of counts for the current output block (4 bytes) slong

**11.1.2.724 #define OutputOffsetFlags(p) (((p)\*32)+18)**

RW - Holds flags for which data should be updated (1 byte) ubyte

**11.1.2.725 #define OutputOffsetMode(p) (((p)\*32)+19)**

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

**11.1.2.726 #define OutputOffsetMotorRPM(p) (((p)\*32)+16)**

Not updated, will be removed later !! (2 bytes) sword

**11.1.2.727 #define OutputOffsetOptions(p) (((p)\*32)+29)**

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

**11.1.2.728 #define OutputOffsetOverloaded(p) (((p)\*32)+27)**

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

**11.1.2.729 #define OutputOffsetPwnFreq 96**

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

**11.1.2.730 #define OutputOffsetRegDParameter(p) (((p)\*32)+24)**

RW - Holds the D-constant used in the regulation (1 byte) ubyte

**11.1.2.731 #define OutputOffsetRegIParameter(p) (((p)\*32)+23)**

RW - Holds the I-constant used in the regulation (1 byte) ubyte

**11.1.2.732 #define OutputOffsetRegMode(p) (((p)\*32)+26)**

RW - Tells which regulation mode should be used (1 byte) ubyte

**11.1.2.733 #define OutputOffsetRegPParameter(p) (((p)\*32)+22)**

RW - Holds the P-constant used in the regulation (1 byte) ubyte

**11.1.2.734 #define OutputOffsetRotationCount(p) (((p)\*32)+8)**

R - Holds current number of counts for the rotation counter to the output (4 bytes) slong

**11.1.2.735 #define OutputOffsetRunState(p) (((p)\*32)+25)**

RW - Holds the current RunState in the output module (1 byte) ubyte

**11.1.2.736 #define OutputOffsetSpeed(p) (((p)\*32)+20)**

RW - Holds the wanted speed (1 byte) sbyte

**11.1.2.737 #define OutputOffsetSyncTurnParameter(p) (((p)\*32)+28)**

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

**11.1.2.738 #define OutputOffsetTachoCount(p) (((p)\*32)+0)**

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes) slong

**11.1.2.739 #define OutputOffsetTachoLimit(p) (((p)\*32)+12)**

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

**11.1.2.740 #define OutputOptions 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 (not yet implemented).

**11.1.2.741 #define Overload 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 [RunState](#), an [OutputMode](#) which includes [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), and its [RegMode](#) must be set to [OUT\\_REGMODE\\_SPEED](#).

**11.1.2.742 #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\\_MSPFComboDirectEx.nxc](#), [ex\\_MSPFComboPWM.nxc](#), [ex\\_-](#)  
[MSPFComboPWMEEx.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputCSTEx.nxc](#),  
[ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSingleOutputPWMEEx.nxc](#), [ex\\_MSPFSinglePin.nxc](#),  
[ex\\_MSPFSinglePinEx.nxc](#), [ex\\_MSPFTrain.nxc](#), and [ex\\_MSPFTrainEx.nxc](#).

**11.1.2.743 #define PF\_CHANNEL\_2 1**

Power function channel 2

**11.1.2.744 #define PF\_CHANNEL\_3 2**

Power function channel 3

**11.1.2.745 #define PF\_CHANNEL\_4 3**

Power function channel 4

**11.1.2.746 #define PF\_CMD\_BRAKE 3**

Power function command brake

**11.1.2.747 #define PF\_CMD\_FWD 1**

Power function command forward

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_MSPFComboDirectEx.nxc](#).

**11.1.2.748 #define PF\_CMD\_REV 2**

Power function command reverse

**11.1.2.749 #define PF\_CMD\_STOP 0**

Power function command stop

**Examples:**

[ex\\_HTPFComboDirect.nxc](#), [ex\\_MSPFComboDirect.nxc](#), and [ex\\_MSPFComboDirectEx.nxc](#).

**11.1.2.750 #define PF\_CST\_CLEAR1\_CLEAR2 0**

Power function CST clear 1 and clear 2

**11.1.2.751 #define PF\_CST\_CLEAR1\_SET2 2**

Power function CST clear 1 and set 2

**11.1.2.752 #define PF\_CST\_DECREMENT\_PWM 5**

Power function CST decrement PWM

**11.1.2.753 #define PF\_CST\_FULL\_FWD 6**

Power function CST full forward

**11.1.2.754 #define PF\_CST\_FULL\_REV 7**

Power function CST full reverse

**11.1.2.755 #define PF\_CST\_INCREMENT\_PWM 4**

Power function CST increment PWM

**11.1.2.756 #define PF\_CST\_SET1\_CLEAR2 1**

Power function CST set 1 and clear 2

**11.1.2.757 #define PF\_CST\_SET1\_SET2 3**

Power function CST set 1 and set 2

**Examples:**

[ex\\_HTTPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), and [ex\\_MSPFSingleOutputCSTEx.nxc](#).

**11.1.2.758 #define PF\_CST\_TOGGLE\_DIR 8**

Power function CST toggle direction

**11.1.2.759 #define PF\_FUNC\_CLEAR 1**

Power function single pin - clear

**11.1.2.760 #define PF\_FUNC\_NOCHANGE 0**

Power function single pin - no change

**11.1.2.761 #define PF\_FUNC\_SET 2**

Power function single pin - set

**Examples:**

[ex\\_HTTPFSinglePin.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFSinglePinEx.nxc](#).

**11.1.2.762 #define PF\_FUNC\_TOGGLE 3**

Power function single pin - toggle

**11.1.2.763 #define PF\_MODE\_COMBO\_DIRECT 1**

Power function mode combo direct

**11.1.2.764 #define PF\_MODE\_COMBO\_PWM 4**

Power function mode combo pulse width modulation (PWM)

**11.1.2.765 #define PF\_MODE\_SINGLE\_OUTPUT\_CST 6**

Power function mode single output clear, set, toggle (CST)

**11.1.2.766 #define PF\_MODE\_SINGLE\_OUTPUT\_PWM 4**

Power function mode single output pulse width modulation (PWM)

**11.1.2.767 #define PF\_MODE\_SINGLE\_PIN\_CONT 2**

Power function mode single pin continuous

**11.1.2.768 #define PF\_MODE\_SINGLE\_PIN\_TIME 3**

Power function mode single pin timed

**11.1.2.769 #define PF\_MODE\_TRAIN 0**

Power function mode IR Train

**11.1.2.770 #define PF\_OUT\_A 0**

Power function output A

**Examples:**

[ex\\_HTPFSingleOutputCST.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#), [ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSingleOutputCST.nxc](#), [ex\\_MSPFSingleOutputCSTEx.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), [ex\\_MSPFSingleOutputPWMEEx.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFSinglePinEx.nxc](#).

**11.1.2.771 #define PF\_OUT\_B 1**

Power function output B

**11.1.2.772 #define PF\_PIN\_C1 0**

Power function pin C1

**Examples:**

[ex\\_HTPFSinglePin.nxc](#), [ex\\_MSPFSinglePin.nxc](#), and [ex\\_MSPFSinglePinEx.nxc](#).

**11.1.2.773 #define PF\_PIN\_C2 1**

Power function pin C2

**11.1.2.774 #define PF\_PWM\_BRAKE 8**

Power function PWM brake

**11.1.2.775 #define PF\_PWM\_FLOAT 0**

Power function PWM float

**11.1.2.776 #define PF\_PWM\_FWD1 1**

Power function PWM foward level 1

**11.1.2.777 #define PF\_PWM\_FWD2 2**

Power function PWM foward level 2

**11.1.2.778 #define PF\_PWM\_FWD3 3**

Power function PWM foward level 3

**11.1.2.779 #define PF\_PWM\_FWD4 4**

Power function PWM foward level 4

**11.1.2.780 #define PF\_PWM\_FWD5 5**

Power function PWM forward level 5

**Examples:**

[ex\\_HTPFComboPWM.nxc](#), [ex\\_HTPFSingleOutputPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#),  
[ex\\_MSPFComboPWMEEx.nxc](#), [ex\\_MSPFSingleOutputPWM.nxc](#), and [ex\\_-  
MSPFSingleOutputPWMEEx.nxc](#).

**11.1.2.781 #define PF\_PWM\_FWD6 6**

Power function PWM forward level 6

**11.1.2.782 #define PF\_PWM\_FWD7 7**

Power function PWM forward level 7

**11.1.2.783 #define PF\_PWM\_REV1 15**

Power function PWM reverse level 1

**11.1.2.784 #define PF\_PWM\_REV2 14**

Power function PWM reverse level 2

**11.1.2.785 #define PF\_PWM\_REV3 13**

Power function PWM reverse level 3

**11.1.2.786 #define PF\_PWM\_REV4 12**

Power function PWM reverse level 4

**Examples:**

[ex\\_HTPFComboPWM.nxc](#), [ex\\_MSPFComboPWM.nxc](#), and [ex\\_MSPFComboPWMEEx.nxc](#).

**11.1.2.787 #define PF\_PWM\_REV5 11**

Power function PWM reverse level 5

**11.1.2.788 #define PF\_PWM\_REV6 10**

Power function PWM reverse level 6



**11.1.2.789 #define PF\_PWM\_REV7 9**

Power function PWM reverse level 7

**11.1.2.790 #define PI 3.141593**

A constant for PI

**Examples:**

[ex\\_dispfnout.nxc](#), and [ex\\_string.nxc](#).

**11.1.2.791 #define PID\_0 0**

PID zero

**11.1.2.792 #define PID\_1 32**

PID one

**11.1.2.793 #define PID\_2 64**

PID two

**11.1.2.794 #define PID\_3 96**

PID three

**11.1.2.795 #define PID\_4 128**

PID four

**11.1.2.796 #define PID\_5 160**

PID five

**11.1.2.797 #define PID\_6 192**

PID six

**11.1.2.798 #define PID\_7 224**

PID seven

**11.1.2.799 #define Power 2**

Power field. Contains the desired power level (-100 to 100). Read/write. Specify the power level of the output. The absolute value of Power is a percentage of the full power of the motor. The sign of Power 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 [UpdateFlags](#) to commit changes to this field.

**11.1.2.800 #define PROG\_ABORT 4**

Program has been aborted

**11.1.2.801 #define PROG\_ERROR 3**

A program error has occurred

**11.1.2.802 #define PROG\_IDLE 0**

Program state is idle

**11.1.2.803 #define PROG\_OK 1**

Program state is okay

**11.1.2.804 #define PROG\_RESET 5**

Program has been reset

**11.1.2.805 #define PROG\_RUNNING 2**

Program is running

**11.1.2.806 #define PSP\_BTNSET1\_DOWN 0x02**

The PSP-Nx button set 1 down arrow

**11.1.2.807 #define PSP\_BTNSET1\_L3 0x40**

The PSP-Nx button set 1 L3

**11.1.2.808 #define PSP\_BTNSET1\_LEFT 0x01**

The PSP-Nx button set 1 left arrow

**11.1.2.809 #define PSP\_BTNSET1\_R3 0x20**

The PSP-Nx button set 1 R3

**11.1.2.810 #define PSP\_BTNSET1\_RIGHT 0x04**

The PSP-Nx button set 1 right arrow

**11.1.2.811 #define PSP\_BTNSET1\_UP 0x08**

The PSP-Nx button set 1 up arrow

**11.1.2.812 #define PSP\_BTNSET2\_CIRCLE 0x04**

The PSP-Nx button set 2 circle

**11.1.2.813 #define PSP\_BTNSET2\_CROSS 0x02**

The PSP-Nx button set 2 cross

**11.1.2.814 #define PSP\_BTNSET2\_L1 0x20**

The PSP-Nx button set 2 L1

**11.1.2.815 #define PSP\_BTNSET2\_L2 0x80**

The PSP-Nx button set 2 L2

**11.1.2.816 #define PSP\_BTNSET2\_R1 0x10**

The PSP-Nx button set 2 R1

**11.1.2.817 #define PSP\_BTNSET2\_R2 0x40**

The PSP-Nx button set 2 R2

**11.1.2.818 #define PSP\_BTNSET2\_SQUARE 0x01**

The PSP-Nx button set 2 square

**11.1.2.819 #define PSP\_BTNSET2\_TRIANGLE 0x08**

The PSP-Nx button set 2 triangle

**11.1.2.820 #define PSP\_CMD\_ANALOG 0x73**

Set the PSP-Nx to analog mode

**11.1.2.821 #define PSP\_CMD\_DIGITAL 0x41**

Set the PSP-Nx to digital mode

**11.1.2.822 #define PSP\_REG\_BTNSET1 0x42**

The PSP-Nx button set 1 register

**11.1.2.823 #define PSP\_REG\_BTNSET2 0x43**

The PSP-Nx button set 2 register

**11.1.2.824 #define PSP\_REG\_XLEFT 0x44**

The PSP-Nx X left register

**11.1.2.825 #define PSP\_REG\_XRIGHT 0x46**

The PSP-Nx X right register

**11.1.2.826 #define PSP\_REG\_YLEFT 0x45**

The PSP-Nx Y left register

**11.1.2.827 #define PSP\_REG\_YRIGHT 0x47**

The PSP-Nx Y right register

**11.1.2.828 #define RADIANS\_PER\_DEGREE PI/180**

Used for converting from degrees to radians

**11.1.2.829 #define RAND\_MAX 32768**

The maximum unsigned int random number returned by [rand](#)

**11.1.2.830 #define RandomNumber 24**

Generate a random number

**11.1.2.831 #define RawValue 2**

Raw value field. Contains the current raw analog sensor value. Read only.

**11.1.2.832 #define RCX\_AbsVarOp 0x74**

Absolute value function

**11.1.2.833 #define RCX\_AndVarOp 0x84**

AND function

**11.1.2.834 #define RCX\_AutoOffOp 0xb1**

Set auto off timer

**11.1.2.835 #define RCX\_BatteryLevelOp 0x30**

Read the battery level

**11.1.2.836 #define RCX\_BatteryLevelSrc 34**

The RCX battery level source

**11.1.2.837 #define RCX\_BootModeOp 0x65**

Set into book mode

**11.1.2.838 #define RCX\_CalibrateEventOp 0x04**

Calibrate event

**11.1.2.839 #define RCX\_ClearAllEventsOp 0x06**

Clear all events

**11.1.2.840 #define RCX\_ClearCounterOp 0xb7**

Clear a counter

**11.1.2.841 #define RCX\_ClearMsgOp 0x90**

Clear message

**11.1.2.842 #define RCX\_ClearSensorOp 0xd1**

Clear a sensor

**11.1.2.843 #define RCX\_ClearSoundOp 0x80**

Clear sound

**11.1.2.844 #define RCX\_ClearTimerOp 0xa1**

Clear a timer

**11.1.2.845 #define RCX\_ClickCounterSrc 27**

The RCX event click counter source

**11.1.2.846 #define RCX\_ConstantSrc 2**

The RCX constant value source

**Examples:**

```
ex_HTRCXEvent.nxc,    ex_HTRCXSetEvent.nxc,    ex_HTRCXSetMaxPower.nxc,    ex_
HTRCXSetPower.nxc,    ex_HTScoutSendVLL.nxc,    ex_HTScoutSetEventFeedback.nxc,
ex_HTScoutSetSensorClickTime.nxc,    ex_HTScoutSetSensorHysteresis.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.
```

**11.1.2.847 #define RCX\_CounterSrc 21**

The RCX counter source

**11.1.2.848 #define RCX\_DatalogOp 0x62**

Datalog the specified source/value

**11.1.2.849 #define RCX\_DatalogRawDirectSrc 42**

The RCX direct datalog raw source

**11.1.2.850 #define RCX\_DatalogRawIndirectSrc 41**

The RCX indirect datalog raw source

**11.1.2.851 #define RCX\_DatalogSrcDirectSrc 38**

The RCX direct datalog source source

**11.1.2.852 #define RCX\_DatalogSrcIndirectSrc 37**

The RCX indirect datalog source source

**11.1.2.853 #define RCX\_DatalogValueDirectSrc 40**

The RCX direct datalog value source

**11.1.2.854 #define RCX\_DatalogValueIndirectSrc 39**

The RCX indirect datalog value source

**11.1.2.855 #define RCX\_DecCounterOp 0xa7**

Decrement a counter

**11.1.2.856 #define RCX\_DeleteSubOp 0xc1**

Delete a subroutine

**11.1.2.857 #define RCX\_DeleteSubsOp 0x70**

Delete subroutines

**11.1.2.858 #define RCX\_DeleteTaskOp 0x61**

Delete a task

**11.1.2.859 #define RCX\_DeleteTasksOp 0x40**

Delete tasks

**11.1.2.860 #define RCX\_DirectEventOp 0x03**

Fire an event

**11.1.2.861 #define RCX\_DisplayOp 0x33**

Set LCD display value

**11.1.2.862 #define RCX\_DivVarOp 0x44**

Divide function

**11.1.2.863 #define RCX\_DurationSrc 31**

The RCX event duration source

**11.1.2.864 #define RCX\_EventStateSrc 25**

The RCX event static source

**11.1.2.865 #define RCX\_FirmwareVersionSrc 35**

The RCX firmware version source

**11.1.2.866 #define RCX\_GlobalMotorStatusSrc 17**

The RCX global motor status source

**11.1.2.867 #define RCX\_GOutputDirOp 0x77**

Set global motor direction

**11.1.2.868 #define RCX\_GOutputModeOp 0x67**

Set global motor mode

**11.1.2.869 #define RCX\_GOutputPowerOp 0xa3**

Set global motor power levels

**11.1.2.870 #define RCX\_HysteresisSrc 30**

The RCX event hysteresis source

**11.1.2.871 #define RCX\_IncCounterOp 0x97**

Increment a counter

**11.1.2.872 #define RCX\_IndirectVarSrc 36**

The RCX indirect variable source

**11.1.2.873 #define RCX\_InputBooleanSrc 13**

The RCX input boolean source



**11.1.2.874 #define RCX\_InputModeOp 0x42**

Set the input mode

**11.1.2.875 #define RCX\_InputModeSrc 11**

The RCX input mode source

**11.1.2.876 #define RCX\_InputRawSrc 12**

The RCX input raw source

**11.1.2.877 #define RCX\_InputTypeOp 0x32**

Set the input type

**11.1.2.878 #define RCX\_InputTypeSrc 10**

The RCX input type source

**11.1.2.879 #define RCX\_InputValueSrc 9**

The RCX input value source

**Examples:**

[ex\\_HTRCXAddToDatalog.nxc](#), [ex\\_MSRCXAddToDatalog.nxc](#), and [ex\\_MSRCXSumVar.nxc](#).

**11.1.2.880 #define RCX\_IRModeOp 0x31**

Set the IR transmit mode

**11.1.2.881 #define RCX\_LightOp 0x87**

Light opcode

**11.1.2.882 #define RCX\_LowerThresholdSrc 29**

The RCX event lower threshold source

**11.1.2.883 #define RCX\_LSblinkTimeOp 0xe3**

Set the light sensor blink time

**11.1.2.884 #define RCX\_LSCalibrateOp 0xc0**

Calibrate the light sensor

**11.1.2.885 #define RCX\_LSHysteresisOp 0xd3**

Set the light sensor hysteresis

**11.1.2.886 #define RCX\_LSLowerThreshOp 0xc3**

Set the light sensor lower threshold

**11.1.2.887 #define RCX\_LSupperThreshOp 0xb3**

Set the light sensor upper threshold

**11.1.2.888 #define RCX\_MessageOp 0xf7**

Set message

**11.1.2.889 #define RCX\_MessageSrc 15**

The RCX message source

**11.1.2.890 #define RCX\_MulVarOp 0x54**

Multiply function

**11.1.2.891 #define RCX\_MuteSoundOp 0xd0**

Mute sound

**11.1.2.892 #define RCX\_OnOffFloatOp 0x21**

Control motor state - on, off, float

**11.1.2.893 #define RCX\_OrVarOp 0x94**

OR function

**11.1.2.894 #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](#).

#### 11.1.2.895 `#define RCX_OUT_AB 0x03`

RCX Outputs A and B

#### 11.1.2.896 `#define RCX_OUT_ABC 0x07`

RCX Outputs A, B, and C

#### 11.1.2.897 `#define RCX_OUT_AC 0x05`

RCX Outputs A and C

#### 11.1.2.898 `#define RCX_OUT_B 0x02`

RCX Output B

#### 11.1.2.899 `#define RCX_OUT_BC 0x06`

RCX Outputs B and C

#### 11.1.2.900 `#define RCX_OUT_C 0x04`

RCX Output C

#### 11.1.2.901 `#define RCX_OUT_FLOAT 0`

Set RCX output to float

#### 11.1.2.902 `#define RCX_OUT_FULL 7`

Set RCX output power level to full

#### Examples:

[ex\\_HTRCXSetPower.nxc](#), and [ex\\_MSRCXSetPower.nxc](#).

**11.1.2.903 #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](#).

**11.1.2.904 #define RCX\_OUT\_HALF 3**

Set RCX output power level to half

**11.1.2.905 #define RCX\_OUT\_LOW 0**

Set RCX output power level to low

**11.1.2.906 #define RCX\_OUT\_OFF 0x40**

Set RCX output to off

**11.1.2.907 #define RCX\_OUT\_ON 0x80**

Set RCX output to on

**Examples:**

[ex\\_HTRCXSetGlobalOutput.nxc](#), [ex\\_HTRCXSetOutput.nxc](#), [ex\\_MSRCXSetGlobalOutput.nxc](#), and [ex\\_MSRCXSetOutput.nxc](#).

**11.1.2.908 #define RCX\_OUT\_REV 0**

Set RCX output direction to reverse

**11.1.2.909 #define RCX\_OUT\_TOGGLE 0x40**

Set RCX output direction to toggle

**11.1.2.910 #define RCX\_OutputDirOp 0xe1**

Set the motor direction

**11.1.2.911 #define RCX\_OutputPowerOp 0x13**

Set the motor power level

**11.1.2.912 #define RCX\_OutputStatusSrc 3**

The RCX output status source

**11.1.2.913 #define RCX\_PBTurnOffOp 0x60**

Turn off the brick

**11.1.2.914 #define RCX\_PingOp 0x10**

Ping the brick

**11.1.2.915 #define RCX\_PlaySoundOp 0x51**

Play a sound

**11.1.2.916 #define RCX\_PlayToneOp 0x23**

Play a tone

**11.1.2.917 #define RCX\_PlayToneVarOp 0x02**

Play a tone using a variable

**11.1.2.918 #define RCX\_PollMemoryOp 0x63**

Poll a memory location

**11.1.2.919 #define RCX\_PollOp 0x12**

Poll a source/value combination

**11.1.2.920 #define RCX\_ProgramSlotSrc 8**

The RCX program slot source

**11.1.2.921 #define RCX\_RandomSrc 4**

The RCX random number source

**Examples:**

[ex\\_MSRCXSet.nxc](#), and [ex\\_MSRCXSubVar.nxc](#).

**11.1.2.922 #define RCX\_RemoteKeysReleased 0x0000**

All remote keys have been released

**11.1.2.923 #define RCX\_RemoteOp 0xd2**

Execute simulated remote control buttons

**11.1.2.924 #define RCX\_RemoteOutABackward 0x4000**

Set output A backward

**11.1.2.925 #define RCX\_RemoteOutAForward 0x0800**

Set output A forward

**11.1.2.926 #define RCX\_RemoteOutBBackward 0x8000**

Set output B backward

**11.1.2.927 #define RCX\_RemoteOutBForward 0x1000**

Set output B forward

**11.1.2.928 #define RCX\_RemoteOutCBackward 0x0001**

Set output C backward

**11.1.2.929 #define RCX\_RemoteOutCForward 0x2000**

Set output C forward

**11.1.2.930 #define RCX\_RemotePBMessage1 0x0100**

Send PB message 1

**11.1.2.931 #define RCX\_RemotePBMessage2 0x0200**

Send PB message 2

**11.1.2.932 #define RCX\_RemotePBMessage3 0x0400**

Send PB message 3

**11.1.2.933 #define RCX\_RemotePlayASound 0x0080**

Play a sound

**Examples:**

[ex\\_HTRCXRemote.nxc](#), and [ex\\_MSRCXRemote.nxc](#).

**11.1.2.934 #define RCX\_RemoteSelProgram1 0x0002**

Select program 1

**11.1.2.935 #define RCX\_RemoteSelProgram2 0x0004**

Select program 2

**11.1.2.936 #define RCX\_RemoteSelProgram3 0x0008**

Select program 3

**11.1.2.937 #define RCX\_RemoteSelProgram4 0x0010**

Select program 4

**11.1.2.938 #define RCX\_RemoteSelProgram5 0x0020**

Select program 5

**11.1.2.939 #define RCX\_RemoteStopOutOff 0x0040**

Stop and turn off outputs

**11.1.2.940 #define RCX\_ScoutCounterLimitSrc 22**

The Scout counter limit source

**11.1.2.941 #define RCX\_ScoutEventFBSrc 24**

The Scout event feedback source

**11.1.2.942 #define RCX\_ScoutLightParamsSrc 19**

The Scout light parameters source

**11.1.2.943 #define RCX\_ScoutOp 0x47**

Scout opcode

**11.1.2.944 #define RCX\_ScoutRulesOp 0xd5**

Set Scout rules

**11.1.2.945 #define RCX\_ScoutRulesSrc 18**

The Scout rules source

**11.1.2.946 #define RCX\_ScoutTimerLimitSrc 20**

The Scout timer limit source

**11.1.2.947 #define RCX\_SelectProgramOp 0x91**

Select a program slot

**11.1.2.948 #define RCX\_SendUARTDataOp 0xc2**

Send data via IR using UART settings

**11.1.2.949 #define RCX\_SetCounterOp 0xd4**

Set counter value

**11.1.2.950 #define RCX\_SetDatalogOp 0x52**

Set the datalog size

**11.1.2.951 #define RCX\_SetEventOp 0x93**

Set an event

**11.1.2.952 #define RCX\_SetFeedbackOp 0x83**

Set Scout feedback

**11.1.2.953 #define RCX\_SetPriorityOp 0xd7**

Set task priority

**11.1.2.954 #define RCX\_SetSourceValueOp 0x05**

Set a source/value

**11.1.2.955 #define RCX\_SetTimerLimitOp 0xc4**

Set timer limit



**11.1.2.956 #define RCX\_SetVarOp 0x14**

Set function

**11.1.2.957 #define RCX\_SetWatchOp 0x22**

Set the watch source/value

**11.1.2.958 #define RCX\_SgnVarOp 0x64**

Sign function

**11.1.2.959 #define RCX\_SoundOp 0x57**

Sound opcode

**11.1.2.960 #define RCX\_StartTaskOp 0x71**

Start a task

**11.1.2.961 #define RCX\_StopAllTasksOp 0x50**

Stop all tasks

**11.1.2.962 #define RCX\_StopTaskOp 0x81**

Stop a task

**11.1.2.963 #define RCX\_SubVarOp 0x34**

Subtract function

**11.1.2.964 #define RCX\_SumVarOp 0x24**

Sum function

**11.1.2.965 #define RCX\_TaskEventsSrc 23**

The RCX task events source

**11.1.2.966 #define RCX\_TenMSTimerSrc 26**

The RCX 10ms timer source

**11.1.2.967 #define RCX\_TimerSrc 1**

The RCX timer source

**11.1.2.968 #define RCX\_UARTSetupSrc 33**

The RCX UART setup source

**11.1.2.969 #define RCX\_UnlockFirmOp 0xa5**

Unlock the firmware

**11.1.2.970 #define RCX\_UnlockOp 0x15**

Unlock the brick

**11.1.2.971 #define RCX\_UnmuteSoundOp 0xe0**

Unmute sound

**11.1.2.972 #define RCX\_UploadDatalogOp 0xa4**

Upload datalog contents

**11.1.2.973 #define RCX\_UpperThresholdSrc 28**

The RCX event upper threshold source

**11.1.2.974 #define RCX\_VariableSrc 0**

The RCX variable source

**Examples:**

[ex\\_HTRCXPoll.nxc](#), [ex\\_HTRCXSelectDisplay.nxc](#), [ex\\_HTSoutSetSensorLowerLimit.nxc](#),  
[ex\\_HTSoutSetSensorUpperLimit.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](#).

**11.1.2.975 #define RCX\_ViewSourceValOp 0xe5**

View a source/value

**11.1.2.976 #define RCX\_VLLOp 0xe2**

Send visual light link (VLL) data

**11.1.2.977 #define RCX\_WatchSrc 14**

The RCX watch source

**11.1.2.978 #define ReadButton 20**

Read the current button state

**11.1.2.979 #define ReadSemData 40**

Read motor semaphore data

**11.1.2.980 #define RegDValue 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.

**11.1.2.981 #define RegIValue 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.

**11.1.2.982 #define RegMode 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 [OutputMode](#) field. Unlike the [OutputMode](#) field, RegMode is not a bitfield. Only one RegMode value can be set at a time. Speed regulation means that the firmware tries to maintain a certain speed based on the [Power](#) 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 [ActualSpeed](#) property. When using speed regulation, do not set Power 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 synch regardless of physical loads. Use this mode to maintain a straight path for a mobile robot automatically. Also use this mode with the [TurnRatio](#) 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.

**11.1.2.983 #define RegPValue 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.

**11.1.2.984 #define RESET\_ALL 0x68**

Reset all three tachometer counters

**11.1.2.985 #define RESET\_BLOCK\_COUNT 0x20**

Reset the NXT-G block tachometer counter

**11.1.2.986 #define RESET\_BLOCKANDTACHO 0x28**

Reset both the internal counter and the NXT-G block counter

**11.1.2.987 #define RESET\_COUNT 0x08**

Reset the internal tachometer counter

**11.1.2.988 #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\\_onfwdsyncex.nxc](#), [ex\\_onfwdsyncexpid.nxc](#), [ex\\_onrevex.nxc](#), [ex\\_onrevregex.nxc](#), [ex\\_onrevregexpid.nxc](#), [ex\\_onrevsyncex.nxc](#), and [ex\\_onrevsyncexpid.nxc](#).

**11.1.2.989 #define RESET\_ROTATION\_COUNT 0x40**

Reset the rotation counter

**11.1.2.990 #define RICArg(\_arg) ((\_arg)|0x1000)**

Output an RIC parameterized argument.

**Parameters:**

***\_arg*** The argument that you want to parameterize.

**Examples:**

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

**11.1.2.991 #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](#).

**11.1.2.992** `#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](#).

**11.1.2.993** `#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.

**11.1.2.994** `#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.

**11.1.2.995** `#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](#).

**11.1.2.996** `#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.

**11.1.2.997** `#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](#).

**11.1.2.998** `#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](#).

**11.1.2.999** `#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.

**11.1.2.1000** `#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](#).

**11.1.2.1001** `#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.

**11.1.2.1002** `#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).

**11.1.2.1003** `#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](#).

```
11.1.2.1004 #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.

```
11.1.2.1005 #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\\_dispgayout.nxc](#), [ex\\_dispgayoutex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

```
11.1.2.1006 #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](#).

```
11.1.2.1007 #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](#).

**11.1.2.1008 #define RICSpriteData( ...) \_\_VA\_ARGS\_\_**

Output RIC sprite data.

**Examples:**

[ex\\_dispgayout.nxc](#), [ex\\_dispgautex.nxc](#), and [ex\\_sysdrawgraphicarray.nxc](#).

**11.1.2.1009 #define ROTATE\_QUEUE 5**

VM should rotate queue

**11.1.2.1010 #define RotationCount 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 [UpdateFlags](#) description for information about how to use program-relative position counts. Set the [UF\\_UPDATE\\_RESET\\_ROTATION\\_COUNT](#) flag in [UpdateFlags](#) to request that the firmware reset the [RotationCount](#). The sign of [RotationCount](#) 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.

**11.1.2.1011 #define RunState 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 [RunState](#) to [OUT\\_RUNSTATE\\_RUNNING](#) to enable power to any output. Use [OUT\\_RUNSTATE\\_RAMPUP](#) to enable automatic ramping to a new [Power](#) level greater than the current [Power](#) level. Use [OUT\\_RUNSTATE\\_RAMPDOWN](#) to enable automatic ramping to a new [Power](#) level less than the current [Power](#) level. Both the rampup and rampdown bits must be used in conjunction with appropriate [TachoLimit](#) and [Power](#) values. In this case the firmware smoothly increases or decreases the actual power to the new [Power](#) level over the total number of degrees of rotation specified in [TachoLimit](#).

**11.1.2.1012 #define SAMPLERATE\_DEFAULT 8000**

Default sample rate [sps]

**11.1.2.1013 #define SAMPLERATE\_MAX 16000**

Max sample rate [sps]

**11.1.2.1014 #define SAMPLERATE\_MIN 2000**

Min sample rate [sps]

**11.1.2.1015 #define ScaledValue 4**

Scaled value field. Contains the current scaled analog sensor value. Read/write.

**11.1.2.1016 #define SCHAR\_MAX 127**

The maximum value of the signed char type

**11.1.2.1017 #define SCHAR\_MIN -127**

The minimum value of the signed char type

**11.1.2.1018 #define SCOUT\_FXR\_ALARM 2**

Alarm special effects

**11.1.2.1019 #define SCOUT\_FXR\_BUG 1**

Bug special effects

**Examples:**

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

**11.1.2.1020 #define SCOUT\_FXR\_NONE 0**

No special effects

**11.1.2.1021 #define SCOUT\_FXR\_RANDOM 3**

Random special effects

**11.1.2.1022 #define SCOUT\_FXR\_SCIENCE 4**

Science special effects

**11.1.2.1023 #define SCOUT\_LIGHT\_OFF 0**

Turn off the scout light

**11.1.2.1024 #define SCOUT\_LIGHT\_ON 0x80**

Turn on the scout light

**Examples:**

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

**11.1.2.1025 #define SCOUT\_LR\_AVOID 3**

Light rule avoid

**11.1.2.1026 #define SCOUT\_LR\_IGNORE 0**

Light rule ignore

**Examples:**

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

**11.1.2.1027 #define SCOUT\_LR\_OFF\_WHEN 5**

Light rule off when

**11.1.2.1028 #define SCOUT\_LR\_SEEK\_DARK 2**

Light rule seek dark

**11.1.2.1029 #define SCOUT\_LR\_SEEK\_LIGHT 1**

Light rule seek light

**11.1.2.1030 #define SCOUT\_LR\_WAIT\_FOR 4**

Light rule wait for

**11.1.2.1031 #define SCOUT\_MODE\_POWER 1**

Enter power mode

**Examples:**

[ex\\_HTScoutSetScoutMode.nxc](#), and [ex\\_MSScoutSetScoutMode.nxc](#).

**11.1.2.1032 #define SCOUT\_MODE\_STANDALONE 0**

Enter stand alone mode

**11.1.2.1033 #define SCOUT\_MR\_CIRCLE\_LEFT 4**

Motion rule circle left

**11.1.2.1034 #define SCOUT\_MR\_CIRCLE\_RIGHT 3**

Motion rule circle right

**11.1.2.1035 #define SCOUT\_MR\_FORWARD 1**

Motion rule forward

**Examples:**

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

**11.1.2.1036 #define SCOUT\_MR\_LOOP\_A 5**

Motion rule loop A

**11.1.2.1037 #define SCOUT\_MR\_LOOP\_AB 7**

Motion rule loop A then B

**11.1.2.1038 #define SCOUT\_MR\_LOOP\_B 6**

Motion rule loop B

**11.1.2.1039 #define SCOUT\_MR\_NO\_MOTION 0**

Motion rule none

**11.1.2.1040 #define SCOUT\_MR\_ZIGZAG 2**

Motion rule zigzag

**11.1.2.1041 #define SCOUT\_SNDSET\_ALARM 3**

Set sound set to alarm

**11.1.2.1042 #define SCOUT\_SNDSET\_BASIC 1**

Set sound set to basic

**11.1.2.1043 #define SCOUT\_SNDSET\_BUG 2**

Set sound set to bug

**11.1.2.1044 #define SCOUT\_SNDSET\_NONE 0**

Set sound set to none

**11.1.2.1045 #define SCOUT\_SNDSET\_RANDOM 4**

Set sound set to random

**11.1.2.1046 #define SCOUT\_SNDSET\_SCIENCE 5**

Set sound set to science

**11.1.2.1047 #define SCOUT\_SOUND\_1\_BLINK 17**

Play the Scout 1 blink sound

**11.1.2.1048 #define SCOUT\_SOUND\_2\_BLINK 18**

Play the Scout 2 blink sound

**11.1.2.1049 #define SCOUT\_SOUND\_COUNTER1 19**

Play the Scout counter 1 sound

**11.1.2.1050 #define SCOUT\_SOUND\_COUNTER2 20**

Play the Scout counter 2 sound

**11.1.2.1051 #define SCOUT\_SOUND\_ENTER\_BRIGHT 14**

Play the Scout enter bright sound

**11.1.2.1052 #define SCOUT\_SOUND\_ENTER\_DARK 16**

Play the Scout enter dark sound

**11.1.2.1053 #define SCOUT\_SOUND\_ENTER\_NORMAL 15**

Play the Scout enter normal sound

**11.1.2.1054 #define SCOUT\_SOUND\_ENTERSA 7**

Play the Scout enter standalone sound

**11.1.2.1055 #define SCOUT\_SOUND\_KEYERROR 8**

Play the Scout key error sound

**11.1.2.1056 #define SCOUT\_SOUND\_MAIL\_RECEIVED 24**

Play the Scout mail received sound

**11.1.2.1057 #define SCOUT\_SOUND\_NONE 9**

Play the Scout none sound

**11.1.2.1058 #define SCOUT\_SOUND\_REMOTE 6**

Play the Scout remote sound

**11.1.2.1059 #define SCOUT\_SOUND\_SPECIAL1 25**

Play the Scout special 1 sound

**11.1.2.1060 #define SCOUT\_SOUND\_SPECIAL2 26**

Play the Scout special 2 sound

**11.1.2.1061 #define SCOUT\_SOUND\_SPECIAL3 27**

Play the Scout special 3 sound

**11.1.2.1062 #define SCOUT\_SOUND\_TIMER1 21**

Play the Scout timer 1 sound

**11.1.2.1063 #define SCOUT\_SOUND\_TIMER2 22**

Play the Scout timer 2 sound

**11.1.2.1064 #define SCOUT\_SOUND\_TIMER3 23**

Play the Scout timer 3 sound

**11.1.2.1065 #define SCOUT\_SOUND\_TOUCH1\_PRES 10**

Play the Scout touch 1 pressed sound

**11.1.2.1066 #define SCOUT\_SOUND\_TOUCH1\_REL 11**

Play the Scout touch 1 released sound

**11.1.2.1067 #define SCOUT\_SOUND\_TOUCH2\_PRES 12**

Play the Scout touch 2 pressed sound

**11.1.2.1068 #define SCOUT\_SOUND\_TOUCH2\_REL 13**

Play the Scout touch 2 released sound

**11.1.2.1069 #define SCOUT\_TGS\_LONG 2**

Transmit level long

**11.1.2.1070 #define SCOUT\_TGS\_MEDIUM 1**

Transmit level medium

**11.1.2.1071 #define SCOUT\_TGS\_SHORT 0**

Transmit level short

**Examples:**

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

**11.1.2.1072 #define SCOUT\_TR\_AVOID 2**

Touch rule avoid

**11.1.2.1073 #define SCOUT\_TR\_IGNORE 0**

Touch rule ignore

**11.1.2.1074 #define SCOUT\_TR\_OFF\_WHEN 4**

Touch rule off when

**11.1.2.1075 #define SCOUT\_TR\_REVERSE 1**

Touch rule reverse

**Examples:**

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

**11.1.2.1076 #define SCOUT\_TR\_WAIT\_FOR 3**

Touch rule wait for

**11.1.2.1077 #define SCREEN\_BACKGROUND 0**

Entire screen

**11.1.2.1078 #define SCREEN\_LARGE 1**

Entire screen except status line

**11.1.2.1079 #define SCREEN\_MODE\_CLEAR 0x01**

Clear the screen

See also:

[SetScreenMode\(\)](#)

**11.1.2.1080 #define SCREEN\_MODE\_RESTORE 0x00**

Restore the screen

See also:

[SetScreenMode\(\)](#)

**11.1.2.1081 #define SCREEN\_SMALL 2**

Screen between menu icons and status line

**11.1.2.1082 #define SCREENS 3**

The number of screen bits

**11.1.2.1083 #define SEC\_1 1000**

1 second

Examples:

[alternating\\_tasks.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_SysCommHSRead.nxc](#), [ex\\_sysdrawpolygon.nxc](#), [ex\\_wait.nxc](#), and [ex\\_yield.nxc](#).

**11.1.2.1084 #define SEC\_10 10000**

10 seconds

Examples:

[ex\\_addr.nxc](#), [ex\\_addrx.nxc](#), [ex\\_ClearScreen.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [ex\\_SysCommBTConnection.nxc](#), and [ex\\_SysCommHSControl.nxc](#).



**11.1.2.1085 #define SEC\_15 15000**

15 seconds

**Examples:**

[ex\\_dispfunc.nxc](#), and [ex\\_memcmp.nxc](#).

**11.1.2.1086 #define SEC\_2 2000**

2 seconds

**Examples:**

[ex\\_CircleOut.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_PolyOut.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**11.1.2.1087 #define SEC\_20 20000**

20 seconds

**11.1.2.1088 #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](#).

**11.1.2.1089 #define SEC\_30 30000**

30 seconds

**11.1.2.1090 #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](#).

**11.1.2.1091 #define SEC\_5 5000**

5 seconds

**Examples:**

[ex\\_atof.nxc](#), [ex\\_atoi.nxc](#), [ex\\_atol.nxc](#), [ex\\_ClearLine.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispftout.nxc](#), [ex\\_dispout.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.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](#), and [ex\\_wait.nxc](#).

**11.1.2.1092   #define SEC\_6 6000**

6 seconds

**Examples:**

[ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), and [ex\\_strtoul.nxc](#).

**11.1.2.1093   #define SEC\_7 7000**

7 seconds

**11.1.2.1094   #define SEC\_8 8000**

8 seconds

**11.1.2.1095   #define SEC\_9 9000**

9 seconds

**Examples:**

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

**11.1.2.1096   #define SetScreenMode 19**

Set the screen mode

**11.1.2.1097   #define SetSleepTimeoutVal 46**

Set the NXT sleep timeout value

**11.1.2.1098   #define SHRT\_MAX 32767**

The maximum value of the short type

**11.1.2.1099   #define SHRT\_MIN -32767**

The minimum value of the short type

**11.1.2.1100   #define SIZE\_OF\_BDADDR 7**

Size of Bluetooth Address

**11.1.2.1101   #define SIZE\_OF\_BRICK\_NAME 8**

Size of NXT Brick name

**11.1.2.1102 #define SIZE\_OF\_BT\_CONNECT\_TABLE 4**

Size of Bluetooth connection table -- Index 0 is always incoming connection

**11.1.2.1103 #define SIZE\_OF\_BT\_DEVICE\_TABLE 30**

Size of Bluetooth device table

**11.1.2.1104 #define SIZE\_OF\_BT\_NAME 16**

Size of Bluetooth name

**11.1.2.1105 #define SIZE\_OF\_BT\_PINCODE 16**

Size of Bluetooth PIN

**11.1.2.1106 #define SIZE\_OF\_BTBUF 128**

Size of Bluetooth buffer

**11.1.2.1107 #define SIZE\_OF\_CLASS\_OF\_DEVICE 4**

Size of class of device

**11.1.2.1108 #define SIZE\_OF\_HSBUF 128**

Size of High Speed Port 4 buffer

**11.1.2.1109 #define SIZE\_OF\_USBBUF 64**

Size of USB Buffer in bytes

**11.1.2.1110 #define SIZE\_OF\_USBDATA 62**

Size of USB Buffer available for data

**11.1.2.1111 #define SOUND\_CLICK 0**

Play the standard key click sound

**11.1.2.1112 #define SOUND\_DOUBLE\_BEEP 1**

Play the standard double beep sound

**11.1.2.1113 #define SOUND\_DOWN 2**

Play the standard sweep down sound

**Examples:**

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

**11.1.2.1114 #define SOUND\_FAST\_UP 5**

Play the standard fast up sound

**Examples:**

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

**11.1.2.1115 #define SOUND\_FLAGS\_IDLE 0x00**

R - Sound is idle

**11.1.2.1116 #define SOUND\_FLAGS\_RUNNING 0x02**

R - Currently processing a tone or file

**11.1.2.1117 #define SOUND\_FLAGS\_UPDATE 0x01**

W - Make changes take effect

**Examples:**

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

**11.1.2.1118 #define SOUND\_LOW\_BEEP 4**

Play the standard low beep sound

**Examples:**

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

**11.1.2.1119 #define SOUND\_MODE\_LOOP 0x01**

W - Play file until writing SOUND\_STATE\_STOP into SoundState

**11.1.2.1120 #define SOUND\_MODE\_ONCE 0x00**

W - Only play file once

**Examples:**

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

**11.1.2.1121 #define SOUND\_MODE\_TONE 0x02**

W - Play tone specified in Frequency for Duration ms

**11.1.2.1122 #define SOUND\_STATE\_FILE 0x02**

R - Processing a file of sound/melody data

**11.1.2.1123 #define SOUND\_STATE\_IDLE 0x00**

R - Idle, ready for start sound (SOUND\_UPDATE)

**Examples:**

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

**11.1.2.1124 #define SOUND\_STATE\_STOP 0x04**

W - Stop sound immediately and close hardware

**Examples:**

[ex\\_SetSoundModuleState.nxc](#), and [ex\\_syssoundsetstate.nxc](#).

**11.1.2.1125 #define SOUND\_STATE\_TONE 0x03**

R - Processing a play tone request

**11.1.2.1126 #define SOUND\_UP 3**

Play the standard sweep up sound

**Examples:**

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

**11.1.2.1127 #define SoundGetState 11**

Get the current sound module state

**11.1.2.1128 #define SoundModuleID 0x00080001**

The sound module ID

**Examples:**

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

**11.1.2.1129 #define SoundModuleName "Sound.mod"**

The sound module name

**Examples:**

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

**11.1.2.1130 #define SoundOffsetDuration 2**

RW - [Tone](#) duration [mS] (2 bytes)

**11.1.2.1131 #define SoundOffsetFlags 26**

RW - Play flag - described above (1 byte) [SoundFlags constants](#)

**11.1.2.1132 #define SoundOffsetFreq 0**

RW - [Tone](#) frequency [Hz] (2 bytes)

**11.1.2.1133 #define SoundOffsetMode 28**

RW - Play mode - described above (1 byte) [SoundMode constants](#)

**11.1.2.1134 #define SoundOffsetSampleRate 4**

RW - Sound file sample rate [2000..16000] (2 bytes)

**Examples:**

[ex\\_sysiomapwrite.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**11.1.2.1135 #define SoundOffsetSoundFilename 6**

RW - Sound/melody filename (20 bytes)

**11.1.2.1136 #define SoundOffsetState 27**

RW - Play state - described above (1 byte) [SoundState constants](#)

**11.1.2.1137 #define SoundOffsetVolume 29**

RW - Sound/melody volume [0..4] 0 = off (1 byte)

**11.1.2.1138 #define SoundPlayFile 9**

Play a sound or melody file

**11.1.2.1139 #define SoundPlayTone 10**

Play a simple tone with the specified frequency and duration

**11.1.2.1140 #define SoundSetState 12**

Set the sound module state

**11.1.2.1141 #define SPECIALS 5**

The number of special bit values

**11.1.2.1142 #define STAT\_COMM\_PENDING 32**

Pending setup operation in progress

**11.1.2.1143 #define STAT\_MSG\_EMPTY\_MAILBOX 64**

Specified mailbox contains no new messages

**11.1.2.1144 #define STATUSICON\_BATTERY 3**

Battery status icon collection

**11.1.2.1145 #define STATUSICON\_BLUETOOTH 0**

BlueTooth status icon collection

**11.1.2.1146 #define STATUSICON\_USB 1**

USB status icon collection

**11.1.2.1147 #define STATUSICON\_VM 2**

VM status icon collection

**11.1.2.1148 #define STATUSICONS 4**

The number of status icons

**11.1.2.1149 #define STATUSTEXT 1**

Status text (BT name)

**11.1.2.1150 #define STEPICON\_1 0**

Left most step icon

**11.1.2.1151 #define STEPICON\_2 1**

**11.1.2.1152 #define STEPICON\_3 2**

**11.1.2.1153 #define STEPICON\_4 3**

**11.1.2.1154 #define STEPICON\_5 4**

Right most step icon

**11.1.2.1155 #define STEPICONS 5**

**11.1.2.1156 #define STEPLINE 3**

Step collection lines

**11.1.2.1157 #define STOP\_REQ 4**

VM should stop executing program

**11.1.2.1158 #define TachoCount 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 TachoLimit and the [UF\\_UPDATE\\_TACHO\\_LIMIT](#) flag. Set the [UF\\_UPDATE\\_RESET\\_COUNT](#) flag in [UpdateFlags](#) to reset TachoCount and cancel any [TachoLimit](#). The sign of TachoCount indicates the motor rotation direction.

**11.1.2.1159 #define TachoLimit 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 [UpdateFlags](#) field to commit changes to the TachoLimit. 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.

**Examples:**

[ex\\_getoutput.nxc](#), and [ex\\_setoutput.nxc](#).

**11.1.2.1160 #define TEMP\_FQ\_1 0x00**

Set fault queue 1



**11.1.2.1161 #define TEMP\_FQ\_2 0x08**

Set fault queue 2

**11.1.2.1162 #define TEMP\_FQ\_4 0x10**

Set fault queue 4

**11.1.2.1163 #define TEMP\_FQ\_6 0x18**

Set fault queue 6

**11.1.2.1164 #define TEMP\_I2C\_ADDRESS 0x98**

The temperature sensor's I2C address

**11.1.2.1165 #define TEMP\_OS\_ONESHOT 0x80**

Set the sensor into oneshot mode

**11.1.2.1166 #define TEMP\_POL\_HIGH 0x04**

Set polarity to high

**11.1.2.1167 #define TEMP\_POL\_LOW 0x00**

Set polarity to low

**11.1.2.1168 #define TEMP\_REG\_CONFIG 0x01**

The register for reading/writing sensor configuration values

**11.1.2.1169 #define TEMP\_REG\_TEMP 0x00**

The register where temperature values can be read

**11.1.2.1170 #define TEMP\_REG\_THIGH 0x03**

The register where temperature high values can be read

**11.1.2.1171 #define TEMP\_REG\_TLOW 0x02**

The register where temperature low values can be read

**11.1.2.1172 #define TEMP\_RES\_10BIT 0x20**

Set the resolution to 10 bit

**11.1.2.1173 #define TEMP\_RES\_11BIT 0x40**

Set the resolution to 11 bit

**11.1.2.1174 #define TEMP\_RES\_12BIT 0x60**

Set the resolution to 12 bit

**11.1.2.1175 #define TEMP\_RES\_9BIT 0x00**

Set the resolution to 9 bit

**11.1.2.1176 #define TEMP\_SD\_CONTINUOUS 0x00**

Set the mode to continuous

**11.1.2.1177 #define TEMP\_SD\_SHUTDOWN 0x01**

Set the mode to shutdown

**11.1.2.1178 #define TEMP\_TM\_COMPARATOR 0x00**

Set the thermostat mode to comparator

**11.1.2.1179 #define TEMP\_TM\_INTERRUPT 0x02**

Set the thermostat mode to interrupt

**11.1.2.1180 #define TEXTLINE\_1 0**

Text line 1

**Examples:**

[ex\\_GetDisplayNormal.nxc](#), [ex\\_GetDisplayPopup.nxc](#), [ex\\_SetDisplayNormal.nxc](#), and [ex\\_SetDisplayPopup.nxc](#).

**11.1.2.1181 #define TEXTLINE\_2 1**

Text line 2

**11.1.2.1182   #define TEXTLINE\_3 2**

Text line 3

**11.1.2.1183   #define TEXTLINE\_4 3**

Text line 4

**11.1.2.1184   #define TEXTLINE\_5 4**

Text line 5

**11.1.2.1185   #define TEXTLINE\_6 5**

Text line 6

**11.1.2.1186   #define TEXTLINE\_7 6**

Text line 7

**11.1.2.1187   #define TEXTLINE\_8 7**

Text line 8

**11.1.2.1188   #define TEXTLINES 8**

The number of text lines on the LCD

**11.1.2.1189   #define TIMES\_UP 6**

VM time is up

**11.1.2.1190   #define TONE\_A3 220**

Third octave A

**11.1.2.1191   #define TONE\_A4 440**

Fourth octave A

**Examples:**

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

**11.1.2.1192   #define TONE\_A5 880**

Fifth octave A

**11.1.2.1193   #define TONE\_A6 1760**

Sixth octave A

**11.1.2.1194   #define TONE\_A7 3520**

Seventh octave A

**11.1.2.1195   #define TONE\_AS3 233**

Third octave A sharp

**11.1.2.1196   #define TONE\_AS4 466**

Fourth octave A sharp

**11.1.2.1197   #define TONE\_AS5 932**

Fifth octave A sharp

**11.1.2.1198   #define TONE\_AS6 1865**

Sixth octave A sharp

**11.1.2.1199   #define TONE\_AS7 3729**

Seventh octave A sharp

**11.1.2.1200   #define TONE\_B3 247**

Third octave B

**11.1.2.1201   #define TONE\_B4 494**

Fourth octave B

**11.1.2.1202   #define TONE\_B5 988**

Fifth octave B

**11.1.2.1203   #define TONE\_B6 1976**

Sixth octave B

**11.1.2.1204 #define TONE\_B7 3951**

Seventh octave B

**11.1.2.1205 #define TONE\_C4 262**

Fourth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**11.1.2.1206 #define TONE\_C5 523**

Fifth octave C

**Examples:**

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

**11.1.2.1207 #define TONE\_C6 1047**

Sixth octave C

**Examples:**

[alternating\\_tasks.nxc](#), and [ex\\_playtones.nxc](#).

**11.1.2.1208 #define TONE\_C7 2093**

Seventh octave C

**11.1.2.1209 #define TONE\_CS4 277**

Fourth octave C sharp

**11.1.2.1210 #define TONE\_CS5 554**

Fifth octave C sharp

**11.1.2.1211 #define TONE\_CS6 1109**

Sixth octave C sharp

**11.1.2.1212 #define TONE\_CS7 2217**

Seventh octave C sharp

**11.1.2.1213 #define TONE\_D4 294**

Fourth octave D

**11.1.2.1214 #define TONE\_D5 587**

Fifth octave D

**11.1.2.1215 #define TONE\_D6 1175**

Sixth octave D

**11.1.2.1216 #define TONE\_D7 2349**

Seventh octave D

**11.1.2.1217 #define TONE\_DS4 311**

Fourth octave D sharp

**11.1.2.1218 #define TONE\_DS5 622**

Fifth octave D sharp

**11.1.2.1219 #define TONE\_DS6 1245**

Sixth octave D sharp

**11.1.2.1220 #define TONE\_DS7 2489**

Seventh octave D sharp

**11.1.2.1221 #define TONE\_E4 330**

Fourth octave E

**Examples:**

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

**11.1.2.1222 #define TONE\_E5 659**

Fifth octave E

**Examples:**

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

**11.1.2.1223   #define TONE\_E6 1319**

Sixth octave E

**11.1.2.1224   #define TONE\_E7 2637**

Seventh octave E

**11.1.2.1225   #define TONE\_F4 349**

Fourth octave F

**11.1.2.1226   #define TONE\_F5 698**

Fifth octave F

**11.1.2.1227   #define TONE\_F6 1397**

Sixth octave F

**11.1.2.1228   #define TONE\_F7 2794**

Seventh octave F

**11.1.2.1229   #define TONE\_FS4 370**

Fourth octave F sharp

**11.1.2.1230   #define TONE\_FS5 740**

Fifth octave F sharp

**11.1.2.1231   #define TONE\_FS6 1480**

Sixth octave F sharp

**11.1.2.1232   #define TONE\_FS7 2960**

Seventh octave F sharp

**11.1.2.1233   #define TONE\_G4 392**

Fourth octave G

**Examples:**

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

**11.1.2.1234 #define TONE\_G5 784**

Fifth octave G

**Examples:**

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

**11.1.2.1235 #define TONE\_G6 1568**

Sixth octave G

**11.1.2.1236 #define TONE\_G7 3136**

Seventh octave G

**11.1.2.1237 #define TONE\_GS4 415**

Fourth octave G sharp

**11.1.2.1238 #define TONE\_GS5 831**

Fifth octave G sharp

**11.1.2.1239 #define TONE\_GS6 1661**

Sixth octave G sharp

**11.1.2.1240 #define TONE\_GS7 3322**

Seventh octave G sharp

**11.1.2.1241 #define TOPLINE 4**

Top status underline

**11.1.2.1242 #define TRAIN\_CHANNEL\_1 0**

IR Train channel 1

**Examples:**

[ex\\_HTIRTrain.nxc](#), [ex\\_MSIRTrain.nxc](#), and [ex\\_MSIRTrainEx.nxc](#).

**11.1.2.1243 #define TRAIN\_CHANNEL\_2 1**

IR Train channel 2



**11.1.2.1244 #define TRAIN\_CHANNEL\_3 2**

IR Train channel 3

**11.1.2.1245 #define TRAIN\_CHANNEL\_ALL 3**

IR Train channel all

**11.1.2.1246 #define TRAIN\_FUNC\_DECR\_SPEED 2**

PF/IR Train function decrement speed

**11.1.2.1247 #define TRAIN\_FUNC\_INCR\_SPEED 1**

PF/IR Train function increment speed

**Examples:**

[ex\\_HTIRTrain.nxc](#), [ex\\_HTPFTrain.nxc](#), [ex\\_MSIRTrain.nxc](#), [ex\\_MSIRTrainEx.nxc](#), [ex\\_MSPFTrain.nxc](#), and [ex\\_MSPFTrainEx.nxc](#).

**11.1.2.1248 #define TRAIN\_FUNC\_STOP 0**

PF/IR Train function stop

**11.1.2.1249 #define TRAIN\_FUNC\_TOGGLE\_LIGHT 4**

PF/IR Train function toggle light

**11.1.2.1250 #define TRUE 1**

A true value

**Examples:**

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

**11.1.2.1251 #define TurnRatio 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: [OutputMode](#) must include [OUT\\_MODE\\_MOTORON](#) and [OUT\\_MODE\\_REGULATED](#), [RegMode](#) must be set to [OUT\\_REGMODE\\_SYNC](#), [RunState](#) must not be [OUT\\_RUNSTATE\\_IDLE](#), and [Power](#) must be non-zero. There are only three valid combinations of left and right motors for use with [TurnRatio](#): [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 [TurnRatio](#) 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.

**11.1.2.1252 #define Type 0**

Type field. Contains one of the sensor type constants. Read/write.

**Examples:**

[ex\\_GetInput.nxc](#), and [ex\\_SetInput.nxc](#).

**11.1.2.1253 #define UCHAR\_MAX 255**

The maximum value of the unsigned char type

**11.1.2.1254 #define UF\_PENDING\_UPDATES 0x80**

Are there any pending motor updates?

**11.1.2.1255 #define UF\_UPDATE\_MODE 0x01**

Commits changes to the OutputMode output property

**11.1.2.1256 #define UF\_UPDATE\_PID\_VALUES 0x10**

Commits changes to the PID motor regulation properties

**11.1.2.1257 #define UF\_UPDATE\_RESET\_BLOCK\_COUNT 0x20**

Resets the NXT-G block-relative rotation counter

**11.1.2.1258 #define UF\_UPDATE\_RESET\_COUNT 0x08**

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

**11.1.2.1259 #define UF\_UPDATE\_RESET\_ROTATION\_COUNT 0x40**

Resets the program-relative (user) rotation counter

**11.1.2.1260 #define UF\_UPDATE\_SPEED 0x02**

Commits changes to the Power output property

**11.1.2.1261 #define UF\_UPDATE\_TACHO\_LIMIT 0x04**

Commits changes to the TachoLimit output property

**11.1.2.1262 #define UI\_BT\_CONNECT\_REQUEST 0x40**

RW - BT get connect accept in progress

**11.1.2.1263 #define UI\_BT\_ERROR\_ATTENTION 0x08**

W - BT error attention

**11.1.2.1264 #define UI\_BT\_PIN\_REQUEST 0x80**

RW - BT get pin code

**11.1.2.1265 #define UI\_BT\_STATE\_CONNECTED 0x02**

RW - BT connected to something

**11.1.2.1266 #define UI\_BT\_STATE\_OFF 0x04**

RW - BT power off

**Examples:**

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

**11.1.2.1267 #define UI\_BT\_STATE\_VISIBLE 0x01**

RW - BT visible

**11.1.2.1268 #define UI\_BUTTON\_ENTER 2**

W - Insert enter button

**Examples:**

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

**11.1.2.1269 #define UI\_BUTTON\_EXIT 4**

W - Insert exit button

**11.1.2.1270 #define UI\_BUTTON\_LEFT 1**

W - Insert left arrow button

**11.1.2.1271 #define UI\_BUTTON\_NONE 0**

R - Button inserted are executed

**11.1.2.1272 #define UI\_BUTTON\_RIGHT 3**

W - Insert right arrow button

**11.1.2.1273 #define UI\_FLAGS\_BUSY 0x40**

R - UI busy running or datalogging (popup disabled)

**11.1.2.1274 #define UI\_FLAGS\_DISABLE\_EXIT 0x04**

RW - Disable exit button

**11.1.2.1275 #define UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER 0x02**

RW - Disable left, right and enter button

**11.1.2.1276 #define UI\_FLAGS\_ENABLE\_STATUS\_UPDATE 0x80**

W - Enable status line to be updated

**11.1.2.1277 #define UI\_FLAGS\_EXECUTE\_LMS\_FILE 0x20**

W - Execute LMS file in "LMSfilename" (Try It)

**11.1.2.1278 #define UI\_FLAGS\_REDRAW\_STATUS 0x08**

W - Redraw entire status line

**Examples:**

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

**11.1.2.1279 #define UI\_FLAGS\_RESET\_SLEEP\_TIMER 0x10**

W - Reset sleep timeout timer

**11.1.2.1280 #define UI\_FLAGS\_UPDATE 0x01**

W - Make changes take effect

**11.1.2.1281 #define UI\_STATE\_BT\_ERROR 16**

R - BT error

**11.1.2.1282 #define UI\_STATE\_CONNECT\_REQUEST 12**

RW - Request for connection accept

**11.1.2.1283 #define UI\_STATE\_DRAW\_MENU 6**

RW - Execute function and draw menu icons

**11.1.2.1284 #define UI\_STATE\_ENTER\_PRESSED 10**

RW - Load selected function and next menu id

**11.1.2.1285 #define UI\_STATE\_EXECUTE\_FILE 13**

RW - Execute file in "LMSfilename"

**11.1.2.1286 #define UI\_STATE\_EXECUTING\_FILE 14**

R - Executing file in "LMSfilename"

**11.1.2.1287 #define UI\_STATE\_EXIT\_PRESSED 11**

RW - Load selected function and next menu id

**11.1.2.1288 #define UI\_STATE\_INIT\_DISPLAY 0**

RW - Init display and load font, menu etc.

**11.1.2.1289 #define UI\_STATE\_INIT\_INTRO 2**

R - Display intro

**11.1.2.1290 #define UI\_STATE\_INIT\_LOW\_BATTERY 1**

R - Low battery voltage at power on

**11.1.2.1291 #define UI\_STATE\_INIT\_MENU 4**

RW - Init menu system

**11.1.2.1292 #define UI\_STATE\_INIT\_WAIT 3**

RW - Wait for initialization end

**11.1.2.1293 #define UI\_STATE\_LEFT\_PRESSED 8**

RW - Load selected function and next menu id

**11.1.2.1294 #define UI\_STATE\_LOW\_BATTERY 15**

R - Low battery at runtime

**Examples:**

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

**11.1.2.1295 #define UI\_STATE\_NEXT\_MENU 5**

RW - Next menu icons ready for drawing

**11.1.2.1296 #define UI\_STATE\_RIGHT\_PRESSED 9**

RW - Load selected function and next menu id

**11.1.2.1297 #define UI\_STATE\_TEST\_BUTTONS 7**

RW - Wait for buttons to be pressed

**11.1.2.1298 #define UI\_VM\_IDLE 0**

VM\_IDLE: Just sitting around. Request to run program will lead to ONE of the VM\_RUN\* states.

**11.1.2.1299 #define UI\_VM\_RESET1 4**

VM\_RESET2: Final clean up and return to IDLE

**11.1.2.1300 #define UI\_VM\_RESET2 5**

VM\_RESET1: Initialize state variables and some I/O devices -- executed when programs end

**11.1.2.1301 #define UI\_VM\_RUN\_FREE 1**

VM\_RUN\_FREE: Attempt to run as many instructions as possible within our timeslice

**11.1.2.1302 #define UI\_VM\_RUN\_PAUSE 3**

VM\_RUN\_PAUSE: Program still "active", but someone has asked us to pause

**11.1.2.1303 #define UI\_VM\_RUN\_SINGLE 2**

VM\_RUN\_SINGLE: Run exactly one instruction per timeslice

**11.1.2.1304 #define UIModuleID 0x000C0001**

The Ui module ID

**11.1.2.1305 #define UIModuleName "Ui.mod"**

The Ui module name

**11.1.2.1306 #define UINT\_MAX 65535**

The maximum value of the unsigned int type

**11.1.2.1307 #define UIOffsetAbortFlag 40**

RW - Long Abort (true == use long press to abort) (1 byte)

**11.1.2.1308 #define UIOffsetBatteryState 30**

W - Battery state (0..4 capacity) (1 byte)

**11.1.2.1309 #define UIOffsetBatteryVoltage 4**

R - Battery voltage in millivolts (2 bytes)

**11.1.2.1310 #define UIOffsetBluetoothState 31**

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)

**11.1.2.1311 #define UIOffsetButton 28**

RW - Insert button (buttons enumerated above) (1 byte)

**11.1.2.1312 #define UIOffsetError 37**

W - Error code (1 byte)

**11.1.2.1313 #define UIOffsetFlags 26**

RW - Update command flags (flags enumerated above) (1 byte)

**11.1.2.1314 #define UIOffsetForceOff 39**

W - Force off (> 0 = off) (1 byte)

**11.1.2.1315 #define UIOffsetLMSfilename 6**

W - LMS filename to execute (Try It) (20 bytes)

**11.1.2.1316 #define UIOffsetOBPPointer 38**

W - Actual OBP step (0 - 4) (1 byte)

**11.1.2.1317 #define UIOffsetPMenu 0**

W - Pointer to menu file (4 bytes)

**11.1.2.1318 #define UIOffsetRechargeable 35**

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

**11.1.2.1319 #define UIOffsetRunState 29**

W - VM Run state (0 = stopped, 1 = running) (1 byte)

**11.1.2.1320 #define UIOffsetSleepTimeout 33**

RW - Sleep timeout time (min) (1 byte)

**11.1.2.1321 #define UIOffsetSleepTimer 34**

RW - Sleep timer (min) (1 byte)

**11.1.2.1322 #define UIOffsetState 27**

RW - UI state (states enumerated above) (1 byte)

**11.1.2.1323 #define UIOffsetUsbState 32**

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

**11.1.2.1324 #define UIOffsetVolume 36**

RW - Volume used in UI (0 - 4) (1 byte)

**11.1.2.1325 #define ULONG\_MAX 4294967295**

The maximum value of the unsigned long type

**11.1.2.1326 #define UpdateCalibCacheInfo 43**

Update sensor calibration cache information

**11.1.2.1327 #define UpdateFlags 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.



**11.1.2.1328 #define US\_CMD\_CONTINUOUS 0x02**

Command to put the ultrasonic sensor into continuous polling mode (default)

**11.1.2.1329 #define US\_CMD\_EVENTCAPTURE 0x03**

Command to put the ultrasonic sensor into event capture mode

**11.1.2.1330 #define US\_CMD\_OFF 0x00**

Command to turn off the ultrasonic sensor

**Examples:**

[ex\\_writei2cregister.nxc](#).

**11.1.2.1331 #define US\_CMD\_SINGLESHOT 0x01**

Command to put the ultrasonic sensor into single shot mode

**11.1.2.1332 #define US\_CMD\_WARMRESET 0x04**

Command to warm reset the ultrasonic sensor

**11.1.2.1333 #define US\_REG\_ACTUAL\_ZERO 0x50**

The register address used to store the actual zero value

**11.1.2.1334 #define US\_REG\_CM\_INTERVAL 0x40**

The register address used to store the CM interval

**11.1.2.1335 #define US\_REG\_FACTORY\_ACTUAL\_ZERO 0x11**

The register address containing the factory setting for the actual zero value

**11.1.2.1336 #define US\_REG\_FACTORY\_SCALE\_DIVISOR 0x13**

The register address containing the factory setting for the scale divisor value

**11.1.2.1337 #define US\_REG\_FACTORY\_SCALE\_FACTOR 0x12**

The register address containing the factory setting for the scale factor value

**11.1.2.1338 #define US\_REG\_MEASUREMENT\_UNITS 0x14**

The register address containing the measurement units (degrees C or F)

**11.1.2.1339 #define US\_REG\_SCALE\_DIVISOR 0x52**

The register address used to store the scale divisor value

**11.1.2.1340 #define US\_REG\_SCALE\_FACTOR 0x51**

The register address used to store the scale factor value

**11.1.2.1341 #define USB\_CMD\_READY 0x01**

A constant representing usb direct command

**11.1.2.1342 #define USB\_PROTOCOL\_OVERHEAD 2**

Size of USB Overhead in bytes -- Command type byte + Command

**11.1.2.1343 #define USHRT\_MAX 65535**

The maximum value of the unsigned short type

**11.1.2.1344 #define WriteSemData 41**

Write motor semaphore data

## 11.2 NXCAPIDocs.h File Reference

Additional documentation for the NXC API. `#include "NXCDefs.h"`

### 11.2.1 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) 2009-2010 John Hansen. All Rights Reserved.

-----  
**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2010-04-30

**Version:**

2

## 11.3 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 [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 [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 [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 [ListFilesType](#)  
*Parameters for the ListFiles system call.*
- struct [RandomNumberType](#)  
*Parameters for the RandomNumber 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 [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 { sinh \_\_FLTRETVAL\_\_, \_X }

*Compute hyperbolic sine (degrees).*

- #define [Atan2D](#)(\_Y, \_X) asm { atan2 \_\_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 [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](#) (const byte port)  
*Get the custom sensor active status.*
- byte [CustomSensorPercentFullScale](#) (const byte port)  
*Get the custom sensor percent full scale.*
- unsigned int [CustomSensorZeroOffset](#) (const byte port)  
*Get the custom sensor zero offset.*
- void [SetCustomSensorActiveStatus](#) (const byte port, const byte &activeStatus)  
*Set active status.*
- void [SetCustomSensorPercentFullScale](#) (const byte port, const byte &pctFullScale)  
*Set percent full scale.*
- void [SetCustomSensorZeroOffset](#) (const byte port, const int &zeroOffset)  
*Set percent zero offset.*
- void [SetSensorBoolean](#) (const byte port, const bool &value)  
*Set sensor boolean value.*
- void [SetSensorDigiPinsDirection](#) (const byte port, const byte &direction)  
*Set digital pins direction.*
- void [SetSensorDigiPinsOutputLevel](#) (const byte port, const byte &outputLevel)  
*Set digital pins output level.*
- void [SetSensorDigiPinsStatus](#) (const byte port, const 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](#) (const byte port, const byte color)   
*Read a LEGO color sensor AD raw value.*
- bool [ColorBoolean](#) (const byte port, const byte color)   
*Read a LEGO color sensor boolean value.*
- long [ColorCalibration](#) (const byte port, const byte point, const byte color)   
*Read a LEGO color sensor calibration point value.*
- byte [ColorCalibrationState](#) (const byte port)   
*Read LEGO color sensor calibration state.*
- unsigned int [ColorCalLimits](#) (const byte port, const byte point)   
*Read a LEGO color sensor calibration limit value.*
- unsigned int [ColorSensorRaw](#) (const byte port, const byte color)   
*Read a LEGO color sensor raw value.*
- unsigned int [ColorSensorValue](#) (const byte port, const byte color)   
*Read a LEGO color sensor scaled value.*
- void [SetMotorPwnFreq](#) (byte n)   
*Set motor PWM frequency.*
- 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 [MotorPwnFreq](#) ()  
*Get motor PWN frequency.*
- 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 addr)  
*Set the display font memory address.*
- void [SetDisplayDisplay](#) (unsigned long addr)  
*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 [ReadI2CRegister](#) (byte port, byte reg, byte &out)   
*Read I2C register.*
- char [WriteI2CRegister](#) (byte port, byte reg, byte val)   
*Write I2C register.*
- 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.*
- string [I2CDeviceInfo](#) (byte port, byte info)

*Read I2C device information.*

- string [I2CDeviceInfoEx](#) (byte port, byte addr, byte info)

*Read I2C device information extra.*

- string [I2CVersion](#) (byte port)

*Read I2C device version.*

- string [I2CVersionEx](#) (byte port, byte addr)

*Read I2C device version extra.*

- string [I2CVendorId](#) (byte port)

*Read I2C device vendor.*

- string [I2CVendorIdEx](#) (byte port, byte addr)

*Read I2C device vendor extra.*

- string [I2CDeviceId](#) (byte port)

*Read I2C device identifier.*

- string [I2CDeviceIdEx](#) (byte port, byte addr)

*Read I2C device identifier extra.*

- long [I2CSendCommand](#) (byte port, byte cmd)

*Send an I2C command.*

- long [I2CSendCommandEx](#) (byte port, byte addr, byte cmd)

*Send an I2C command extra.*

- 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 [LSState](#) ()  
*Get I2C state.*
- byte [LSSpeed](#) ()  
*Get I2C speed.*
- byte [LSNoRestartOnRead](#) ()  
*Get I2C no restart on read setting.*
- 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).*
- void [PowerDown](#) ()  
*Power down the NXT.*
- void [SleepNow](#) ()  
*Put the brick to sleep immediately.*
- void [RebootInFirmwareMode](#) ()  
*Reboot the NXT in firmware download mode.*
- 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.*
- 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 [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.*
- 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.*
- 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](#) (const byte conn)  
*Check bluetooth status.*
- char [BluetoothWrite](#) (const byte conn, byte buffer[ ])  
*Write to a bluetooth connection.*
- 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 [RS485Control](#) (byte cmd, byte baud, unsigned int mode)  
*Control the RS485 port.*
- bool [RS485DataAvailable](#) (void)  
*Check for RS485 available data.*
- char [RS485Exit](#) (void)  
*Exit RS485.*
- char [RS485Init](#) (void)  
*Initialize RS485.*

- char [RS485Read](#) (byte &buffer[ ])   
*Read RS485 data.*
- bool [RS485SendingData](#) (void)   
*Is RS485 sending data.*
- void [RS485Status](#) (bool &sendingData, bool &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 [HSSState](#) (void)  
*Get hi-speed port state.*
- int [HSMMode](#) (void)  
*Get hi-speed port mode.*
- 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](#) (const byte hsFlags)  
*Set hi-speed port flags.*
- void [SetHSSpeed](#) (const byte hsSpeed)  
*Set hi-speed port speed.*
- void [SetHSState](#) (const byte hsState)  
*Set hi-speed port state.*
- void [SetHSMode](#) (const unsigned int hsMode)  
*Set hi-speed port 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 [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 [SysListFiles](#) ([ListFilesType](#) &args)  
*List files.*
- int [SensorHTGyro](#) (const byte &port, int offset=0)  
*Read HiTechnic Gyro 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.*

- 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 addr)  
*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.*
- 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.*
- int [SensorMSPressure](#) (const byte &port)  
*Read mindsensors pressure sensor.*
- int [SensorMSCompass](#) (const byte &port)  
*Read mindsensors compass value.*
- int [SensorMSCompassEx](#) (const byte &port, const byte addr)  
*Read mindsensors compass value.*
- int [SensorMSDROD](#) (const byte &port)  
*Read mindsensors DROD value.*
- int [SensorMSPressureRaw](#) (const byte &port)  
*Read mindsensors raw pressure value.*
- bool [ReadSensorMSAccel](#) (const byte port, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSAccelEx](#) (const byte port, const byte addr, int &x, int &y, int &z)  
*Read mindsensors acceleration values.*
- bool [ReadSensorMSPlayStation](#) (const byte port, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)  
*Read mindsensors playstation controller values.*
- bool [ReadSensorMSPlayStationEx](#) (const byte port, const byte addr, 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, byte &x, byte &y, byte &z)

*Read mindsensors tilt values.*

- bool [ReadSensorMSTiltEx](#) (const byte port, const byte addr, byte &x, byte &y, byte &z)

*Read mindsensors tilt values.*

- int [MSReadValue](#) (const byte port, const byte reg, const byte numbytes)

*Read a mindsensors device value.*

- int [MSReadValueEx](#) (const byte port, const byte addr, const byte reg, const byte numbytes)

*Read a mindsensors device value.*

- char [MSEnergize](#) (const byte port)

*Turn on power to device.*

- char [MSEnergizeEx](#) (const byte port, const byte addr)

*Turn on power to device.*

- char [MSDeenergize](#) (const byte port)

*Turn off power to device.*

- char [MSDeenergizeEx](#) (const byte port, const byte addr)

*Turn off power to device.*

- char [MSADPAOn](#) (const byte port)

*Turn on mindsensors ADPA mode.*

- char [MSADPAOnEx](#) (const byte port, const byte addr)

*Turn on mindsensors ADPA mode.*

- char [MSADPAOff](#) (const byte port)

*Turn off mindsensors ADPA mode.*

- char [MSADPAOffEx](#) (const byte port, const byte addr)

*Turn off mindsensors ADPA mode.*

- char [DISTNxGP2D12](#) (const byte port)

*Configure DISTNx as GP2D12.*

- char [DISTNxGP2D12Ex](#) (const byte port, const byte addr)

*Configure DISTNx as GP2D12.*

- char [DISTNxGP2D120](#) (const byte port)

*Configure DISTNx as GP2D120.*

- char [DISTNxGP2D120Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2D120.*
- char [DISTNxGP2YA02](#) (const byte port)  
*Configure DISTNx as GP2YA02.*
- char [DISTNxGP2YA02Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2YA02.*
- char [DISTNxGP2YA21](#) (const byte port)  
*Configure DISTNx as GP2YA21.*
- char [DISTNxGP2YA21Ex](#) (const byte port, const byte addr)  
*Configure DISTNx as GP2YA21.*
- int [DISTNxDistance](#) (const byte port)  
*Read DISTNx distance value.*
- int [DISTNxDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx distance value.*
- int [DISTNxMaxDistance](#) (const byte port)  
*Read DISTNx maximum distance value.*
- int [DISTNxMaxDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx maximum distance value.*
- int [DISTNxMinDistance](#) (const byte port)  
*Read DISTNx minimum distance value.*
- int [DISTNxMinDistanceEx](#) (const byte port, const byte addr)  
*Read DISTNx minimum distance value.*
- byte [DISTNxModuleType](#) (const byte port)  
*Read DISTNx module type value.*
- byte [DISTNxModuleTypeEx](#) (const byte port, const byte addr)  
*Read DISTNx module type value.*
- byte [DISTNxNumPoints](#) (const byte port)  
*Read DISTNx num points value.*
- byte [DISTNxNumPointsEx](#) (const byte port, const byte addr)  
*Read DISTNx num points value.*
- int [DISTNxVoltage](#) (const byte port)  
*Read DISTNx voltage value.*
- int [DISTNxVoltageEx](#) (const byte port, const byte addr)



*Read DISTNx voltage value.*

- char [PSPNxDigital](#) (const byte &port)  
*Configure PSPNx in digital mode.*
- char [PSPNxDigitalEx](#) (const byte &port, const byte &addr)  
*Configure PSPNx in digital mode.*
- char [PSPNxAnalog](#) (const byte &port)  
*Configure PSPNx in analog mode.*
- char [PSPNxAnalogEx](#) (const byte &port, const byte &addr)  
*Configure PSPNx in analog mode.*
- char [NRLink2400](#) (const byte port)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink2400Ex](#) (const byte port, const byte addr)  
*Configure NRLink in 2400 baud mode.*
- char [NRLink4800](#) (const byte port)  
*Configure NRLink in 4800 baud mode.*
- char [NRLink4800Ex](#) (const byte port, const byte addr)  
*Configure NRLink in 4800 baud mode.*
- char [NRLinkFlush](#) (const byte port)  
*Flush NRLink buffers.*
- char [NRLinkFlushEx](#) (const byte port, const byte addr)  
*Flush NRLink buffers.*
- char [NRLinkIRLong](#) (const byte port)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRLongEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR long mode.*
- char [NRLinkIRShort](#) (const byte port)  
*Configure NRLink in IR short mode.*
- char [NRLinkIRShortEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR short mode.*
- char [NRLinkSetPF](#) (const byte port)  
*Configure NRLink in power function mode.*
- char [NRLinkSetPFEx](#) (const byte port, const byte addr)  
*Configure NRLink in power function mode.*

- char [NRLinkSetRCX](#) (const byte port)  
*Configure NRLink in RCX mode.*
- char [NRLinkSetRCXEx](#) (const byte port, const byte addr)  
*Configure NRLink in RCX mode.*
- char [NRLinkSetTrain](#) (const byte port)  
*Configure NRLink in IR train mode.*
- char [NRLinkSetTrainEx](#) (const byte port, const byte addr)  
*Configure NRLink in IR train mode.*
- char [NRLinkTxRaw](#) (const byte port)  
*Configure NRLink in raw IR transmit mode.*
- char [NRLinkTxRawEx](#) (const byte port, const byte addr)  
*Configure NRLink in raw IR transmit mode.*
- byte [NRLinkStatus](#) (const byte port)  
*Read NRLink status.*
- byte [NRLinkStatusEx](#) (const byte port, const byte addr)  
*Read NRLink status.*
- char [RunNRLinkMacro](#) (const byte port, const byte macro)  
*Run NRLink macro.*
- char [RunNRLinkMacroEx](#) (const byte port, const byte addr, const byte macro)  
*Run NRLink macro.*
- char [WriteNRLinkBytes](#) (const byte port, const byte data[ ])  
*Write data to NRLink.*
- char [WriteNRLinkBytesEx](#) (const byte port, const byte addr, const byte data[ ])  
*Write data to NRLink.*
- bool [ReadNRLinkBytes](#) (const byte port, byte &data[ ])  
*Read data from NRLink.*
- bool [ReadNRLinkBytesEx](#) (const byte port, const byte addr, byte &data[ ])  
*Read data from NRLink.*
- char [MSIRTrain](#) (const byte port, const byte channel, const byte func)  
*MSIRTrain function.*
- char [MSIRTrainEx](#) (const byte port, const byte addr, const byte channel, const byte func)  
*MSIRTrainEx function.*
- char [MSPFComboDirect](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirect function.*

- char [MSPFComboDirectEx](#) (const byte port, const byte addr, const byte channel, const byte outa, const byte outb)  
*MSPFComboDirectEx function.*
- char [MSPFComboPWM](#) (const byte port, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWM function.*
- char [MSPFComboPWMEEx](#) (const byte port, const byte addr, const byte channel, const byte outa, const byte outb)  
*MSPFComboPWMEEx function.*
- char [MSPFRawOutput](#) (const byte port, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutput function.*
- char [MSPFRawOutputEx](#) (const byte port, const byte addr, const byte nibble0, const byte nibble1, const byte nibble2)  
*MSPFRawOutputEx function.*
- char [MSPFRepeat](#) (const byte port, const byte count, const unsigned int delay)  
*MSPFRepeat function.*
- char [MSPFRepeatEx](#) (const byte port, const byte addr, const byte count, const unsigned int delay)  
*MSPFRepeatEx function.*
- char [MSPFSingleOutputCST](#) (const byte port, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCST function.*
- char [MSPFSingleOutputCSTEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputCSTEx function.*
- char [MSPFSingleOutputPWM](#) (const byte port, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWM function.*
- char [MSPFSingleOutputPWMEEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte func)  
*MSPFSingleOutputPWMEEx function.*
- char [MSPFSinglePin](#) (const byte port, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePin function.*
- char [MSPFSinglePinEx](#) (const byte port, const byte addr, const byte channel, const byte out, const byte pin, const byte func, bool cont)  
*MSPFSinglePinEx function.*
- char [MSPFTrain](#) (const byte port, const byte channel, const byte func)  
*MSPFTrain function.*

- char [MSPFTrainEx](#) (const byte port, const byte addr, const byte channel, const byte func)  
*MSPFTrainEx function.*
- void [MSRCXSetNRLinkPort](#) (const byte port)  
*MSRCXSetIRLinkPort function.*
- void [MSRCXSetNRLinkPortEx](#) (const byte port, const byte addr)  
*MSRCXSetIRLinkPortEx 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 pwrsrc, 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 pwrsrc, 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.*
- 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.*
- 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.*
- byte [fopen](#) (string filename, const string mode)  
*Open file.*
- int [fflush](#) (byte handle)  
*Flush file.*
- 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.*
- unsigned int [rand](#) ()  
*Generate random number.*
- int [Random](#) (unsigned int n=0)  
*Generate random number.*
- void [SysRandomNumber](#) ([RandomNumberType](#) &args)  
*Draw a random number.*
- 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.*

- 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 [addr](#) (variant data)  
*Get the absolute address of a variable.*
- unsigned long [reladdr](#) (variant data)  
*Get the relative address of a variable.*
- unsigned long [addrex](#) (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.*

### 11.3.1 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) 2009-2010 John Hansen. All Rights Reserved.

-----  
**Author:**

John Hansen (bricxcc\_at\_comcast.net)

**Date:**

2010-05-22

**Version:**

72

### 11.3.2 Define Documentation

#### 11.3.2.1 `#define _SENSOR_CFG(_type, _mode) (((_type)<<8)+(_mode))`

Macro for defining [SetSensor](#) combined type and mode constants

**11.3.2.2 #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.

**11.3.2.3 #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.

**11.3.2.4 #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.

### 11.3.2.5 #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.

### 11.3.2.6 #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.

### 11.3.2.7 #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.

**11.3.2.8 #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  $\_Y/\_X$ .

**11.3.2.9 #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  $\_X$ .

**11.3.2.10 #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  $\_X$ .

**11.3.2.11 #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.

**11.3.2.12 #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.

**11.3.2.13 #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.

**11.3.2.14 #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.

**11.3.2.15 #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.

**11.3.2.16 #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.

**11.3.2.17 #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.

**11.3.2.18 #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](#).

**11.3.2.19 #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.

**11.3.2.20 #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*.

**11.3.2.21 #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*.

**11.3.2.22 #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*.



**11.3.2.23 #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](#).

**11.3.2.24 #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

**11.3.2.25 #define S1 0**

Input port 1

**Examples:**

[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\\_CustomSensorActiveStatus.nxc](#), [ex\\_CustomSensorPercentFullScale.nxc](#),  
[ex\\_CustomSensorZeroOffset.nxc](#), [ex\\_DISTNxDistance.nxc](#), [ex\\_DISTNxDistanceEx.nxc](#),  
[ex\\_DISTNxGP2D12.nxc](#), [ex\\_DISTNxGP2D120.nxc](#), [ex\\_DISTNxGP2D120Ex.nxc](#), [ex\\_-](#)  
[DISTNxGP2D12Ex.nxc](#), [ex\\_DISTNxGP2YA02.nxc](#), [ex\\_DISTNxGP2YA02Ex.nxc](#), [ex\\_-](#)  
[DISTNxGP2YA21.nxc](#), [ex\\_DISTNxGP2YA21Ex.nxc](#), [ex\\_DISTNxMaxDistance.nxc](#), [ex\\_-](#)  
[DISTNxMaxDistanceEx.nxc](#), [ex\\_DISTNxMinDistance.nxc](#), [ex\\_DISTNxMinDistanceEx.nxc](#),  
[ex\\_DISTNxModuleType.nxc](#), [ex\\_DISTNxModuleTypeEx.nxc](#), [ex\\_DISTNxNumPoints.nxc](#),  
[ex\\_DISTNxNumPointsEx.nxc](#), [ex\\_DISTNxVoltage.nxc](#), [ex\\_DISTNxVoltageEx.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\_I2CDeviceIdEx.nxc, ex\_I2CDeviceInfo.nxc, ex\_I2CDeviceInfoEx.nxc, ex\_I2CRead.nxc, ex\_I2CSendCommand.nxc, ex\_I2CSendCommandEx.nxc, ex\_I2CStatus.nxc, ex\_I2CVendorId.nxc, ex\_I2CVendorIdEx.nxc, ex\_I2CVersion.nxc, ex\_I2CVersionEx.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\_LSPutBufferBytesToRx.nxc, ex\_LSPutBufferInPtr.nxc, ex\_LSPutBufferOutPtr.nxc, ex\_MSADPAOff.nxc, ex\_MSADPAOffEx.nxc, ex\_MSADPAOn.nxc, ex\_MSADPAOnEx.nxc, ex\_MSDeenergize.nxc, ex\_MSDeenergizeEx.nxc, ex\_MSEnergize.nxc, ex\_MSEnergizeEx.nxc, ex\_MSIRTrain.nxc, ex\_MSIRTrainEx.nxc, ex\_MSPFComboDirect.nxc, ex\_MSPFComboDirectEx.nxc, ex\_MSPFComboPWM.nxc, ex\_MSPFComboPWMEEx.nxc, ex\_MSPFRawOutput.nxc, ex\_MSPFRawOutputEx.nxc, ex\_MSPFRepeat.nxc, ex\_MSPFRepeatEx.nxc, ex\_MSPFSingleOutputCST.nxc, ex\_MSPFSingleOutputCSTEx.nxc, ex\_MSPFSingleOutputPWM.nxc, ex\_MSPFSingleOutputPWMEEx.nxc, ex\_MSPFSinglePin.nxc, ex\_MSPFSinglePinEx.nxc, ex\_MSPFTrain.nxc, ex\_MSPFTrainEx.nxc, ex\_MSRCXAddToDatalog.nxc, ex\_MSRCXClearSensor.nxc, ex\_MSRCXSetNRLinkPort.nxc, ex\_MSRCXSetNRLinkPortEx.nxc, ex\_MSRCXSetSensorMode.nxc, ex\_MSRCXSetSensorType.nxc, ex\_MSRCXSumVar.nxc, ex\_MSReadValue.nxc, ex\_MSReadValueEx.nxc, ex\_NRLink2400.nxc, ex\_NRLink2400Ex.nxc, ex\_NRLink4800.nxc, ex\_NRLink4800Ex.nxc, ex\_NRLinkFlush.nxc, ex\_NRLinkFlushEx.nxc, ex\_NRLinkIRLong.nxc, ex\_NRLinkIRLongEx.nxc, ex\_NRLinkIRShort.nxc, ex\_NRLinkIRShortEx.nxc, ex\_NRLinkSetPF.nxc, ex\_NRLinkSetPFEx.nxc, ex\_NRLinkSetRCX.nxc, ex\_NRLinkSetRCXEx.nxc, ex\_NRLinkSetTrain.nxc, ex\_NRLinkSetTrainEx.nxc, ex\_NRLinkStatus.nxc, ex\_NRLinkStatusEx.nxc, ex\_NRLinkTxRaw.nxc, ex\_NRLinkTxRawEx.nxc, ex\_PSPNxAnalog.nxc, ex\_PSPNxAnalogEx.nxc, ex\_PSPNxDigital.nxc, ex\_PSPNxDigitalEx.nxc, ex\_readi2cregister.nxc, ex\_ReadNRLinkBytes.nxc, ex\_ReadNRLinkBytesEx.nxc, ex\_ReadSensorColorEx.nxc, ex\_ReadSensorColorRaw.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\_ReadSensorMSAccelEx.nxc, ex\_ReadSensorMSPlayStation.nxc, ex\_ReadSensorMSPlayStationEx.nxc, ex\_ReadSensorMSRTClock.nxc, ex\_ReadSensorMSTilt.nxc, ex\_ReadSensorMSTiltEx.nxc, ex\_readsensorex.nxc, ex\_RemoteResetScaledValue.nxc, ex\_RemoteSetInputMode.nxc, ex\_RunNRLinkMacro.nxc, ex\_RunNRLinkMacroEx.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\_SensorHTIRSeeker2Dir.nxc, ex\_SensorInvalid.nxc, ex\_SensorMode.nxc, ex\_SensorMSCompass.nxc, ex\_SensorMSCompassEx.nxc, ex\_SensorMSDROD.nxc, ex\_SensorMSPressure.nxc, ex\_SensorMSPressureRaw.nxc, ex\_SensorNormalized.nxc, ex\_SensorRaw.nxc, ex\_SensorScaled.nxc, ex\_SensorType.nxc, ex\_SensorValue.nxc, ex\_SensorValueBool.nxc, ex\_SensorValueRaw.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\_SetSensorHTEOPD.nxc, ex\_SetSensorHTGyro.nxc, ex\_SetSensorLight.nxc, ex\_SetSensorLowspeed.nxc, ex\_SetSensorMode.nxc, ex\_SetSensorMSDROD.nxc, ex\_SetSensorMSPressure.nxc, ex\_SetSensorSound.nxc, ex\_SetSensorTouch.nxc, ex\_SetSensorType.nxc, ex\_SysColorSensorRead.nxc, ex\_syscommlscheckstatus.nxc, ex\_syscommlsread.nxc, ex\_syscommlswrite.nxc, ex\_

[syscommmlswriteex.nxc](#), [ex\\_SysComputeCalibValue.nxc](#), [ex\\_writei2cregister.nxc](#), [ex\\_WriteNRLinkBytes.nxc](#), and [ex\\_WriteNRLinkBytesEx.nxc](#).

#### 11.3.2.26 **#define s16 int**

Signed 16 bit type

#### 11.3.2.27 **#define S2 1**

Input port 2

#### 11.3.2.28 **#define S3 2**

Input port 3

#### 11.3.2.29 **#define s32 long**

Signed 32 bit type

#### 11.3.2.30 **#define S4 3**

Input port 4

#### Examples:

[ex\\_I2CBytes.nxc](#), and [ex\\_sensorus.nxc](#).

#### 11.3.2.31 **#define s8 char**

Signed 8 bit type

#### 11.3.2.32 **#define SEEK\_CUR 1**

Seek from the current file position

#### Examples:

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

#### 11.3.2.33 **#define SEEK\_END 2**

Seek from the end of the file

**11.3.2.34 #define SEEK\_SET 0**

Seek from the beginning of the file

**Examples:**

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

**11.3.2.35 #define SENSOR\_1 Sensor(S1)**

Read the value of the analog sensor on port S1

**11.3.2.36 #define SENSOR\_2 Sensor(S2)**

Read the value of the analog sensor on port S2

**11.3.2.37 #define SENSOR\_3 Sensor(S3)**

Read the value of the analog sensor on port S3

**11.3.2.38 #define SENSOR\_4 Sensor(S4)**

Read the value of the analog sensor on port S4

**11.3.2.39 #define SENSOR\_CELSIUS \_SENSOR\_CFG(SENSOR\_TYPE\_TEMPERATURE,  
SENSOR\_MODE\_CELSIUS)**

RCX temperature sensor in celcius mode

**11.3.2.40 #define SENSOR\_COLORBLUE \_SENSOR\_CFG(SENSOR\_TYPE\_COLORBLUE,  
SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (blue) in percent mode

**11.3.2.41 #define SENSOR\_COLORFULL \_SENSOR\_CFG(SENSOR\_TYPE\_COLORFULL,  
SENSOR\_MODE\_RAW)**

NXT 2.0 color sensor (full) in raw mode

**11.3.2.42 #define SENSOR\_COLORGREEN \_SENSOR\_CFG(SENSOR\_TYPE\_-  
COLORGREEN, SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

**11.3.2.43 #define SENSOR\_COLORNONE \_SENSOR\_CFG(SENSOR\_TYPE\_COLORNONE,  
SENSOR\_MODE\_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

**11.3.2.44** `#define SENSOR_COLORRED _SENSOR_CFG(SENSOR_TYPE_COLORRED, SENSOR_MODE_PERCENT)`

NXT 2.0 color sensor (red) in percent mode

**11.3.2.45** `#define SENSOR_EDGE _SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_EDGE)`

Touch sensor in edge mode

**11.3.2.46** `#define SENSOR_FAHRENHEIT _SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_FAHRENHEIT)`

RCX temperature sensor in fahrenheit mode

**11.3.2.47** `#define SENSOR_LIGHT _SENSOR_CFG(SENSOR_TYPE_LIGHT, SENSOR_MODE_PERCENT)`

Light sensor in percent mode

**11.3.2.48** `#define SENSOR_LOWSPEED _SENSOR_CFG(SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW)`

NXT I2C sensor without 9V power in raw mode

**11.3.2.49** `#define SENSOR_LOWSPEED_9V _SENSOR_CFG(SENSOR_TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)`

NXT I2C sensor with 9V power in raw mode

**11.3.2.50** `#define SENSOR_MODE_BOOL IN_MODE_BOOLEAN`

Boolean value (0 or 1)

**Examples:**

[ex\\_HTRCXSetSensorMode.nxc](#), and [ex\\_MSRCXSetSensorMode.nxc](#).

**11.3.2.51** `#define SENSOR_MODE_CELSIUS IN_MODE_CELSIUS`

RCX temperature sensor value in degrees celcius

**11.3.2.52** `#define SENSOR_MODE_EDGE IN_MODE_TRANSITIONCNT`

Counts the number of boolean transitions

**11.3.2.53 #define SENSOR\_MODE\_FAHRENHEIT IN\_MODE\_FAHRENHEIT**

RCX temperature sensor value in degrees fahrenheit

**11.3.2.54 #define SENSOR\_MODE\_PERCENT IN\_MODE\_PCTFULLSCALE**

Scaled value from 0 to 100

**11.3.2.55 #define SENSOR\_MODE\_PULSE IN\_MODE\_PERIODCOUNTER**

Counts the number of boolean periods

**11.3.2.56 #define SENSOR\_MODE\_RAW IN\_MODE\_RAW**

Raw value from 0 to 1023

**Examples:**

[ex\\_RemoteSetInputMode.nxc](#), and [ex\\_SetSensorMode.nxc](#).

**11.3.2.57 #define SENSOR\_MODE\_ROTATION IN\_MODE\_ANGLESTEP**

RCX rotation sensor (16 ticks per revolution)

**11.3.2.58 #define SENSOR\_NXTLIGHT \_SENSOR\_CFG(SENSOR\_TYPE\_LIGHT\_ACTIVE, SENSOR\_MODE\_PERCENT)**

NXT light sensor in active mode

**11.3.2.59 #define SENSOR\_PULSE \_SENSOR\_CFG(SENSOR\_TYPE\_TOUCH, SENSOR\_MODE\_PULSE)**

Touch sensor in pulse mode

**11.3.2.60 #define SENSOR\_ROTATION \_SENSOR\_CFG(SENSOR\_TYPE\_ROTATION, SENSOR\_MODE\_ROTATION)**

RCX rotation sensor in rotation mode

**11.3.2.61 #define SENSOR\_SOUND \_SENSOR\_CFG(SENSOR\_TYPE\_SOUND\_DB, SENSOR\_MODE\_PERCENT)**

NXT sound sensor (dB) in percent mode

**11.3.2.62 #define SENSOR\_TOUCH \_SENSOR\_CFG(SENSOR\_TYPE\_TOUCH, SENSOR\_MODE\_BOOL)**

Touch sensor in boolean mode

**Examples:**

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

**11.3.2.63 #define SENSOR\_TYPE\_COLORBLUE IN\_TYPE\_COLORBLUE**

NXT 2.0 color sensor with blue light

**11.3.2.64 #define SENSOR\_TYPE\_COLORFULL IN\_TYPE\_COLORFULL**

NXT 2.0 color sensor in full color mode

**11.3.2.65 #define SENSOR\_TYPE\_COLORGREEN IN\_TYPE\_COLORGREEN**

NXT 2.0 color sensor with green light

**11.3.2.66 #define SENSOR\_TYPE\_COLORNONE IN\_TYPE\_COLORNONE**

NXT 2.0 color sensor with no light

**11.3.2.67 #define SENSOR\_TYPE\_COLORRED IN\_TYPE\_COLORRED**

NXT 2.0 color sensor with red light

**11.3.2.68 #define SENSOR\_TYPE\_CUSTOM IN\_TYPE\_CUSTOM**

NXT custom sensor

**11.3.2.69 #define SENSOR\_TYPE\_HIGHSPEED IN\_TYPE\_HISPEED**

NXT Hi-speed port (only S4)

**11.3.2.70 #define SENSOR\_TYPE\_LIGHT IN\_TYPE\_REFLECTION**

RCX light sensor

**11.3.2.71 #define SENSOR\_TYPE\_LIGHT\_ACTIVE IN\_TYPE\_LIGHT\_ACTIVE**

NXT light sensor with light

**11.3.2.72 #define SENSOR\_TYPE\_LIGHT\_INACTIVE IN\_TYPE\_LIGHT\_INACTIVE**

NXT light sensor without light

**11.3.2.73 #define SENSOR\_TYPE\_LOWSPEED IN\_TYPE\_LOWSPEED**

NXT I2C digital sensor

**Examples:**

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

**11.3.2.74 #define SENSOR\_TYPE\_LOWSPEED\_9V IN\_TYPE\_LOWSPEED\_9V**

NXT I2C digital sensor with 9V power

**11.3.2.75 #define SENSOR\_TYPE\_NONE IN\_TYPE\_NO\_SENSOR**

No sensor configured

**11.3.2.76 #define SENSOR\_TYPE\_ROTATION IN\_TYPE\_ANGLE**

RCX rotation sensor

**11.3.2.77 #define SENSOR\_TYPE\_SOUND\_DB IN\_TYPE\_SOUND\_DB**

NXT sound sensor with dB scaling

**Examples:**

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

**11.3.2.78 #define SENSOR\_TYPE\_SOUND\_DBA IN\_TYPE\_SOUND\_DBA**

NXT sound sensor with dBA scaling

**11.3.2.79 #define SENSOR\_TYPE\_TEMPERATURE IN\_TYPE\_TEMPERATURE**

RCX temperature sensor

**11.3.2.80 #define SENSOR\_TYPE\_TOUCH IN\_TYPE\_SWITCH**

NXT or RCX touch sensor

**Examples:**

[ex\\_HTRCXSetSensorType.nxc](#), [ex\\_MSRCXSetSensorType.nxc](#), and [ex\\_SetSensorType.nxc](#).



**11.3.2.81 #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.

**11.3.2.82 #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.

**11.3.2.83 #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.

**11.3.2.84 #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 [sinh\(\)](#) instead.

**Parameters:**

\_X Floating point value.

**Returns:**

Hyperbolic sine of \_X.

**11.3.2.85 #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.

**11.3.2.86 #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.

**11.3.2.87 #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.

**11.3.2.88 #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.

**11.3.2.89 #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.

**11.3.2.90 #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.

**11.3.2.91 #define u16 unsigned int**

Unsigned 16 bit type

**11.3.2.92 #define u32 unsigned long**

Unsigned 32 bit type

**11.3.2.93 #define u8 unsigned char**

Unsigned 8 bit type

**11.3.3 Function Documentation****11.3.3.1 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](#).

**11.3.3.2 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](#).

**11.3.3.3 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](#).

**11.3.3.4 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](#).

**11.3.3.5 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](#).

**11.3.3.6 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](#).

**11.3.3.7 unsigned long addr (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\\_addr.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

**11.3.3.8 unsigned long address (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\\_addrx.nxc](#).

**11.3.3.9 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.

**Examples:**

[ex\\_arraybuild.nxc](#), [ex\\_SysCommHSWrite.nxc](#), and [ex\\_SysDatalogWrite.nxc](#).

**11.3.3.10 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.

**Examples:**

[ex\\_arrayinit.nxc](#), and [ex\\_sysdrawgraphic.nxc](#).

### 11.3.3.11 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.

#### Examples:

[ex\\_arraylen.nxc](#), and [ex\\_syslistfiles.nxc](#).

### 11.3.3.12 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.

#### 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*).

#### Examples:

[ex\\_ArrayMax.nxc](#), and [ex\\_ArraySort.nxc](#).

### 11.3.3.13 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.

#### 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*).



**Examples:**

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

**11.3.3.14    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.

**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).

**Examples:**

[ex\\_ArrayMin.nxc](#), and [ex\\_ArraySort.nxc](#).

**11.3.3.15    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.

**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).

**Examples:**

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

**11.3.3.16    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.

**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).

**Examples:**

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

**11.3.3.17 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.

**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).

**Examples:**

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

**11.3.3.18 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.

**Examples:**

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

### 11.3.3.19 **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.

#### **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*).

#### **Examples:**

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

### 11.3.3.20 **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.

#### **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*).

#### **Examples:**

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

### 11.3.3.21 **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](#).

**11.3.3.22 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](#).

**11.3.3.23 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](#).

#### 11.3.3.24 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](#).

#### 11.3.3.25 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 [-90,+90] degrees.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_atan2d.nxc](#).

#### 11.3.3.26 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](#).

**11.3.3.27 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](#).

### 11.3.3.28 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](#).

### 11.3.3.29 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](#).

**11.3.3.30 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](#).

**11.3.3.31 byte BatteryState (void) [inline]**

Get battery state. Return battery state information (0..4).

**Returns:**

The battery state (0..4)

**Examples:**

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

**11.3.3.32 byte bcd2dec (byte *bcd*) [inline]**

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

**Returns:**

The decimal equivalent of the binary coded decimal byte

**Examples:**

[ex\\_bcd2dec.nxc](#).

**11.3.3.33 byte BluetoothState (void) [inline]**

Get bluetooth state. Return the bluetooth state.

**Returns:**

The bluetooth state. See [BluetoothState constants](#).

**Examples:**

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



**11.3.3.34 char BluetoothStatus (const 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).

**Returns:**

The bluetooth status for the specified connection.

**Examples:**

[ex\\_bluetoothstatus.nxc](#), and [ex\\_SysCommBTConnection.nxc](#).

**11.3.3.35 char BluetoothWrite (const 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).

*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](#).

**11.3.3.36 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](#).

**11.3.3.37 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.](#)

**11.3.3.38 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.](#)

**11.3.3.39 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.](#)

**11.3.3.40 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.](#)

**11.3.3.41 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.](#)

**11.3.3.42 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](#).

**11.3.3.43 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](#).

**11.3.3.44 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](#).

#### 11.3.3.45 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](#).

#### 11.3.3.46 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](#).

#### 11.3.3.47 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](#).

**11.3.3.48 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](#).

**11.3.3.49 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](#).

**11.3.3.50 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](#).

**11.3.3.51 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](#).

**11.3.3.52 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](#).

**11.3.3.53 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](#).

**11.3.3.54 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](#).

**11.3.3.55 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](#).

**11.3.3.56 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](#).

**11.3.3.57 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](#).

**11.3.3.58 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](#).

### 11.3.3.59 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](#).

### 11.3.3.60 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](#).

### 11.3.3.61 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](#).



### 11.3.3.62 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](#).

### 11.3.3.63 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](#).

### 11.3.3.64 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](#).

**11.3.3.65 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](#).

**11.3.3.66 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](#).

**11.3.3.67 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](#).

**11.3.3.68 void ClearLine (byte *line*) [inline]**

Clear a line on the LCD screen. This function lets you clear a single line on the NXT LCD.

**Examples:**

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

**11.3.3.69 void ClearScreen () [inline]**

Clear LCD screen. This function lets you clear the NXT LCD to a blank screen.

**Examples:**

[ex\\_ClearScreen.nxc](#), [ex\\_dispfout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_string.nxc](#), and [ex\\_sysdrawpolygon.nxc](#).

**11.3.3.70 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](#).

**11.3.3.71 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](#).

**11.3.3.72 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](#).

**11.3.3.73 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](#).

**11.3.3.74 unsigned int ColorADRaw (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The AD raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**11.3.3.75 bool ColorBoolean (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The boolean value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**11.3.3.76 long ColorCalibration (const byte *port*, const byte *point*, const 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](#). Must be a constant.

*point* The calibration point. See [Color calibration constants](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The calibration point value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

### 11.3.3.77 `byte ColorCalibrationState (const 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](#). Must be a constant.

**Returns:**

The calibration state.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

### 11.3.3.78 `unsigned int ColorCalLimits (const byte port, const 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](#). Must be a constant.

*point* The calibration point. See [Color calibration constants](#). Must be a constant.

**Returns:**

The calibration limit value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

### 11.3.3.79 `unsigned int ColorSensorRaw (const byte port, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The raw value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**11.3.3.80 unsigned int ColorSensorValue (const byte *port*, const 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](#). Must be a constant.

*color* The color index. See [Color sensor array indices](#). Must be a constant.

**Returns:**

The scaled value.

**Warning:**

This function requires an NXT 2.0 compatible firmware.

**Examples:**

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

**11.3.3.81 byte CommandFlags (void) [inline]**

Get command flags. Return the command flags.

**Returns:**

Command flags. See [CommandFlags constants](#)

**Examples:**

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

**11.3.3.82 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](#).

### 11.3.3.83 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\\_cos.nxc](#).

### 11.3.3.84 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\\_cosd.nxc](#).



**11.3.3.85 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](#).

**11.3.3.86 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.

**11.3.3.87 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](#).

### 11.3.3.88 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](#).

### 11.3.3.89 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](#).

**11.3.3.90 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](#).

**11.3.3.91 byte CustomSensorActiveStatus (const 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](#).

**11.3.3.92 byte CustomSensorPercentFullScale (const 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](#).

**11.3.3.93 unsigned int CustomSensorZeroOffset (const 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](#).

**11.3.3.94 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\\_DeleteFile.nxc](#).

**11.3.3.95 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](#).

**11.3.3.96 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](#).

**11.3.3.97 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](#).

**11.3.3.98 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](#).

**11.3.3.99 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\\_addr.nxc](#), [ex\\_addrx.nxc](#), [ex\\_DisplayFont.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

**11.3.3.100 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](#).

**11.3.3.101 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](#).

**11.3.3.102 int DISTNxDistance (const byte *port*) [inline]**

Read DISTNx distance value. Read the mindsensors DISTNx distance 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 distance value.

**Examples:**

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

**11.3.3.103 int DISTNxDistanceEx (const byte *port*, const byte *addr*) [inline]**

Read DISTNx distance value. Read the mindsensors DISTNx sensor's distance value. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

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

**11.3.3.104 char DISTNxGP2D12 (const byte *port*) [inline]**

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. 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 function call result.

**Examples:**

[ex\\_DISTNxGP2D12.nxc](#).

**11.3.3.105 char DISTNxGP2D120 (const byte *port*) [inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. 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 function call result.

**Examples:**

[ex\\_DISTNxGP2D120.nxc](#).

**11.3.3.106 char DISTNxGP2D120Ex (const byte *port*, const byte *addr*) [inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2D120Ex.nxc](#).

**11.3.3.107 char DISTNxGP2D12Ex (const byte *port*, const byte *addr*) [inline]**

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2D12Ex.nxc](#).

**11.3.3.108 char DISTNxGP2YA02 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA02.nxc](#).

**11.3.3.109 char DISTNxGP2YA02Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA02Ex.nxc](#).

**11.3.3.110 char DISTNxGP2YA21 (const byte *port*) [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](#).



**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA21.nxc](#).

**11.3.3.111 char DISTNxGP2YA21Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_DISTNxGP2YA21Ex.nxc](#).

**11.3.3.112 int DISTNxMaxDistance (const byte *port*) [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](#).

**Returns:**

The maximum distance value.

**Examples:**

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

**11.3.3.113 int DISTNxMaxDistanceEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The maximum distance value.

**Examples:**

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

**11.3.3.114 int DISTNxMinDistance (const byte *port*) [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](#).

**Returns:**

The distance value.

**Examples:**

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

**11.3.3.115 int DISTNxMinDistanceEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The distance value.

**Examples:**

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

**11.3.3.116 byte DISTNxModuleType (const byte *port*) [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](#).

**Returns:**

The module type value.

**Examples:**

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

**11.3.3.117 byte DISTNxModuleTypeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The module type value.

**Examples:**

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

**11.3.3.118 byte DISTNxNumPoints (const byte *port*) [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](#).

**Returns:**

The num points value.

**Examples:**

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

**11.3.3.119 byte DISTNxNumPointsEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The num points value.

**Examples:**

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

**11.3.3.120 int DISTNxVoltage (const byte *port*) [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](#).

**Returns:**

The voltage value.

**Examples:**

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

**11.3.3.121 int DISTNxVoltageEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The voltage value.

**Examples:**

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

**11.3.3.122 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](#).

**11.3.3.123 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](#).

**11.3.3.124 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](#).

**11.3.3.125 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](#).

**11.3.3.126 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](#).

**11.3.3.127 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](#).

**11.3.3.128 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](#).

**11.3.3.129 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](#).

**11.3.3.130 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](#).

**11.3.3.131 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](#).

**11.3.3.132 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\\_FindNextFile.nxc](#).

**11.3.3.133 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](#).



**11.3.3.134 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](#).

**11.3.3.135 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](#), and [ex\\_string.nxc](#).

**11.3.3.136 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](#).

**11.3.3.137 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](#).

**11.3.3.138 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](#).

**11.3.3.139 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](#).

#### 11.3.3.140 `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](#).

### 11.3.3.141 **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](#).

### 11.3.3.142 **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](#).

### 11.3.3.143 **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\\_addr.nxc](#), [ex\\_addrex.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_FormatNum.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SetDisplayFont.nxc](#), [ex\\_string.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**11.3.3.144 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](#).

**11.3.3.145 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](#).

**11.3.3.146 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](#).

**11.3.3.147 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](#).

**11.3.3.148 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](#).

**11.3.3.149 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](#).

**11.3.3.150 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:**

Currently always returns -1.

**Examples:**

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

**11.3.3.151 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](#).

**11.3.3.152 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](#).

**11.3.3.153 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](#).

**11.3.3.154 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](#).

**11.3.3.155 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](#).

**11.3.3.156 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](#).

**11.3.3.157 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](#).

**11.3.3.158 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](#).

**11.3.3.159 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](#).

**11.3.3.160 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](#).

**11.3.3.161 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](#).

**11.3.3.162 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](#).

**11.3.3.163 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](#).

**11.3.3.164 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](#).

**Examples:**

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

**11.3.3.165 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](#).

**11.3.3.166 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](#).

**11.3.3.167 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](#).

**11.3.3.168 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](#).

**11.3.3.169 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](#).

**11.3.3.170 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.

**11.3.3.171 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.

**11.3.3.172 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](#).

**11.3.3.173 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.

**11.3.3.174 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](#).

**11.3.3.175 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](#).

**11.3.3.176 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](#).

**11.3.3.177 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.

**11.3.3.178 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](#).

**11.3.3.179 void glEnd () [inline]**

Finish a polygon for the current object. Stop defining a polygon surface for the current graphics object.

**11.3.3.180 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.

**11.3.3.181 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](#).

**11.3.3.182 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](#).

**11.3.3.183 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](#).

**11.3.3.184 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).

**11.3.3.185 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](#).

**11.3.3.186 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](#).

**11.3.3.187 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.



**11.3.3.188 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.

**11.3.3.189 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.

**11.3.3.190 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](#).

**11.3.3.191** `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](#).

**11.3.3.192** `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](#).

**11.3.3.193 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](#).

**11.3.3.194 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](#).

**11.3.3.195 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](#).

**11.3.3.196 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](#).

**11.3.3.197 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](#).

**Examples:**

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

**11.3.3.198 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](#).

**11.3.3.199 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](#).

**11.3.3.200 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](#).

**11.3.3.201 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](#).

**11.3.3.202 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](#).

**11.3.3.203 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 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 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](#).

### 11.3.3.204 **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](#).

### 11.3.3.205 **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](#).

### 11.3.3.206 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](#).

### 11.3.3.207 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](#).

### 11.3.3.208 **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 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 PWM function. See [Power Function PWM option constants](#).

**Returns:**

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

**Examples:**

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

### 11.3.3.209 **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](#).

**11.3.3.210 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 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](#).

*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](#).

**11.3.3.211 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](#).

**11.3.3.212 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](#).

**11.3.3.213 void HTRCXCleaAllEvents (void) [inline]**

HTRCXCleaAllEvents function. Send the ClearAllEvents command to an RCX.

**Examples:**

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

**11.3.3.214 void HTRCXCleaCounter (const byte counter) [inline]**

HTRCXCleaCounter function. Send the ClearCounter command to an RCX.

**Parameters:**

*counter* The counter to clear.

**Examples:**

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

**11.3.3.215 void HTRCXCleaMsg (void) [inline]**

HTRCXCleaMsg function. Send the ClearMsg command to an RCX.

**Examples:**

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

**11.3.3.216 void HTRCXCleaSensor (const byte port) [inline]**

HTRCXCleaSensor function. Send the ClearSensor command to an RCX.

**Parameters:**

*port* The RCX port number.

**Examples:**

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

**11.3.3.217 void HTRCXCleaSound (void) [inline]**

HTRCXCleaSound function. Send the ClearSound command to an RCX.

**Examples:**

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

**11.3.3.218 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](#).

**11.3.3.219 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](#).

**11.3.3.220 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](#).

**11.3.3.221 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](#).

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

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

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

**11.3.3.223 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](#).

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

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

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

**11.3.3.225 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](#).

**11.3.3.226 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](#).

**11.3.3.227 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](#).

**11.3.3.228 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](#).

**11.3.3.229 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](#).

**11.3.3.230 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](#).

**11.3.3.231 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](#).

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

HTRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

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

**11.3.3.233 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](#).

**11.3.3.234 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](#).

**11.3.3.235 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](#).

**11.3.3.236 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](#).

**11.3.3.237 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](#).

**11.3.3.238 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](#).

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

HTRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

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

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

HTRCXPing function. Send the Ping command to an RCX.

**Examples:**

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

**11.3.3.241 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](#).

**11.3.3.242 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](#).

**11.3.3.243 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](#).

**11.3.3.244 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](#).



**11.3.3.245 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](#).

**11.3.3.246 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](#).

**11.3.3.247 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](#).

**11.3.3.248 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](#).

**11.3.3.249 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](#).

**11.3.3.250 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](#).

**11.3.3.251 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](#).

**11.3.3.252 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](#).

**11.3.3.253 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](#).

**11.3.3.254 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](#).

**11.3.3.255 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 Lowspeed port before using any of the HiTechnic RCX and Scout iRLink functions.

**Parameters:**

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

**11.3.3.256 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](#).

**11.3.3.257 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](#).

**11.3.3.258 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](#).

**11.3.3.259 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](#).

**11.3.3.260 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](#).

**11.3.3.261 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.](#)

**11.3.3.262 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.](#)

**11.3.3.263 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.](#)

**11.3.3.264 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.](#)

**11.3.3.265 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](#).

**11.3.3.266 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](#).

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

HTRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

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

**11.3.3.268 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](#).

**11.3.3.269 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](#).

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

HTRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

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

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

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

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

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

HTScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

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

**11.3.3.273 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](#).

**11.3.3.274 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](#).

**11.3.3.275 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](#).

**11.3.3.276 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](#).

**11.3.3.277 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](#).

**11.3.3.278 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](#).



**11.3.3.279 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](#).

**11.3.3.280 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](#).

**11.3.3.281 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](#).

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

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

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

### 11.3.3.283 **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](#).

### 11.3.3.284 **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](#).

**11.3.3.285 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](#).

**11.3.3.286 string I2CDeviceId (byte *port*) [inline]**

Read I2C device identifier. Read standard I2C device identifier. The I2C device must use address 0x02.

**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 string containing the device identifier.

**See also:**

[I2CDeviceIdEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

**11.3.3.287 string I2CDeviceIdEx (byte *port*, byte *addr*) [inline]**

Read I2C device identifier extra. 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.

*addr* The I2C device address.

**Returns:**

A string containing the device identifier.

**See also:**

[I2CDeviceId](#)

**Examples:**

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

**11.3.3.288 string I2CDeviceInfo (byte *port*, byte *info*) [inline]**

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device must use address 0x02.

**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.

*info* A value indicating the type of device information you are requesting. See [Standard I2C constants](#).

**Returns:**

A string containing the requested device information.

**See also:**

[I2CDeviceInfoEx](#)

**Examples:**

[ex\\_I2CDeviceInfo.nxc](#).

**11.3.3.289 string I2CDeviceInfoEx (byte *port*, byte *addr*, byte *info*) [inline]**

Read I2C device information extra. 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.

*addr* 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.

**See also:**

[I2CDeviceInfo](#)

**Examples:**

[ex\\_I2CDeviceInfoEx.nxc](#).

**11.3.3.290 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](#).

**11.3.3.291 long I2CSendCommand (byte port, 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 must use address 0x02.

**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.

*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](#).

**11.3.3.292 long I2CSendCommandEx (byte port, byte addr, byte cmd) [inline]**

Send an I2C command extra. 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.

*addr* 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\\_I2CSendCommandEx.nxc](#).

**11.3.3.293 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](#).

**11.3.3.294 string I2CVendorId (byte *port*) [inline]**

Read I2C device vendor. Read standard I2C device vendor. The I2C device must use address 0x02.

**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 string containing the device vendor.

**See also:**

[I2CVendorIdEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

**11.3.3.295 string I2CVendorIdEx (byte *port*, byte *addr*) [inline]**

Read I2C device vendor extra. 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.

***addr*** The I2C device address.

**Returns:**

A string containing the device vendor.

**See also:**

[I2CVendorId](#)

**Examples:**

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

**11.3.3.296 string I2CVersion (byte *port*) [inline]**

Read I2C device version. Read standard I2C device version. The I2C device must use address 0x02.

**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 string containing the device version.

**See also:**

[I2CVersionEx](#)

**Examples:**

[ex\\_I2CDeviceId.nxc](#), [ex\\_I2CVendorId.nxc](#), and [ex\\_I2CVersion.nxc](#).

**11.3.3.297 string I2CVersionEx (byte *port*, byte *addr*) [inline]**

Read I2C device version extra. 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.

*addr* The I2C device address.

**Returns:**

A string containing the device version.

**See also:**

[I2CVersion](#)

**Examples:**

[ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), and [ex\\_I2CVersionEx.nxc](#).

**11.3.3.298 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](#).

### 11.3.3.299 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](#).

### 11.3.3.300 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](#).

**11.3.3.301 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](#).

**11.3.3.302 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](#).

**11.3.3.303 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](#).

**11.3.3.304 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](#).

**11.3.3.305 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](#).

**11.3.3.306 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](#).

**11.3.3.307 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](#).

**11.3.3.308 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](#).

**11.3.3.309 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](#).

**11.3.3.310 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](#).

### 11.3.3.311 `long labs (long n) [inline]`

Absolute value. Return the absolute value of parameter *n*.

**Parameters:**

*n* Integral value.

**Returns:**

The absolute value of *n*.

### 11.3.3.312 `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](#).

### 11.3.3.313 `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](#).

**11.3.3.314 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](#).

**11.3.3.315 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](#).

**11.3.3.316 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](#).

**11.3.3.317 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](#).

**11.3.3.318 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](#).

**11.3.3.319 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](#).

**11.3.3.320 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](#).

**11.3.3.321 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](#).

**11.3.3.322 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](#).

### 11.3.3.323 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](#).

### 11.3.3.324 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](#).

**11.3.3.325** 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](#).

**11.3.3.326** 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](#).

**11.3.3.327** 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](#).

**11.3.3.328 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](#).

**11.3.3.329 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](#).

**11.3.3.330 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](#).

**11.3.3.331 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](#).

**11.3.3.332 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](#).

**11.3.3.333 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](#).

**11.3.3.334 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](#).

**11.3.3.335 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](#).

**11.3.3.336 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](#).

**11.3.3.337 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](#).

**11.3.3.338 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](#).

### 11.3.3.339 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](#).

**Examples:**

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

### 11.3.3.340 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](#).

**Examples:**

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

### 11.3.3.341 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](#).

**Examples:**

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

**11.3.3.342 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](#).

**Examples:**

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

**11.3.3.343 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](#).

**Examples:**

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

**11.3.3.344 byte MotorPwnFreq () [inline]**

Get motor PWN frequency. Get the current motor pulse width modulation frequency.

**Examples:**

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

**11.3.3.345 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](#).

**Examples:**

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



**11.3.3.346 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](#).

**Examples:**

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

**11.3.3.347 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](#).

**Examples:**

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

**11.3.3.348 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](#).

**Examples:**

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

**11.3.3.349 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](#).

**Examples:**

[ex\\_motorrotationcount.nxc](#), and [util\\_rpm.nxc](#).

**11.3.3.350 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](#).

**Examples:**

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

**11.3.3.351 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](#).

**Examples:**

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

**11.3.3.352 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](#).

**Examples:**

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

**11.3.3.353 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](#).

**Examples:**

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

**11.3.3.354 char MSADPAOff (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.355 char MSADPAOffEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.356 char MSADPAOn (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.357 char MSADPAOnEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.358 char MSDeenergize (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.359 char MSDeenergizeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.360 char MSEnergize (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.361 char MSEnergizeEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.362 char MSIRTrain (const byte *port*, 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 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 Lowspeed 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\\_MSIRTrain.nxc](#).

### 11.3.3.363 `char MSIRTrainEx (const byte port, const byte addr, const byte channel, const byte func) [inline]`

MSIRTrainEx 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](#).  
*addr* 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\\_MSIRTrainEx.nxc](#).

### 11.3.3.364 `char MSPFComboDirect (const byte port, 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](#).  
*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](#).

### 11.3.3.365 `char MSPFComboDirectEx (const byte port, const byte addr, const byte channel, const byte outa, const byte outb) [inline]`

MSPFComboDirectEx 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](#).
- addr* 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\\_MSPFComboDirectEx.nxc](#).

**11.3.3.366 char MSPFComboPWM (const byte *port*, 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](#).
- 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](#).

**11.3.3.367 char MSPFComboPWMEEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]**

MSPFComboPWMEEx 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](#).
- addr* 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\\_MSPFComboPWMEEx.nxc](#).

### 11.3.3.368 `char MSPFRawOutput (const byte port, const byte nibble0, const byte nibble1, const byte nibble2) [inline]`

MSPFRawOutput function. Control a Power Function receiver set to the specified channel using the mind-sensors 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](#).
- 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](#).

### 11.3.3.369 `char MSPFRawOutputEx (const byte port, const byte addr, const byte nibble0, const byte nibble1, const byte nibble2) [inline]`

MSPFRawOutputEx 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](#).
- addr* 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\\_MSPFRawOutputEx.nxc](#).

### 11.3.3.370 **char MSPFRepeat (const byte *port*, 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 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\\_MSPFRepeat.nxc](#).

### 11.3.3.371 **char MSPFRepeatEx (const byte *port*, const byte *addr*, const byte *count*, const unsigned int *delay*) [inline]**

MSPFRepeatEx function. Repeat sending the last Power Function command using the mindsensors NR-Link 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](#).  
*addr* 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\\_MSPFRepeatEx.nxc](#).

### 11.3.3.372 `char MSPFSingleOutputCST (const byte port, 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](#).  
*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](#).

### 11.3.3.373 `char MSPFSingleOutputCSTEx (const byte port, const byte addr, const byte channel, const byte out, const byte func) [inline]`

MSPFSingleOutputCSTEx 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](#).  
*addr* 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\\_MSPFSingleOutputCSTEx.nxc](#).

### 11.3.3.374 `char MSPFSingleOutputPWM (const byte port, 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](#).  
*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](#).

### 11.3.3.375 `char MSPFSingleOutputPWMLEx (const byte port, const byte addr, const byte channel, const byte out, const byte func) [inline]`

MSPFSingleOutputPWMLEx 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](#).  
*addr* 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\\_MSPFSingleOutputPWMLEx.nxc](#).

### 11.3.3.376 **char MSPFSinglePin (const byte *port*, 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](#).  
*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](#).

### 11.3.3.377 **char MSPFSinglePinEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *out*, const byte *pin*, const byte *func*, bool *cont*) [inline]**

MSPFSinglePinEx 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](#).  
*addr* 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\\_MSPFSinglePinEx.nxc](#).

**11.3.3.378 char MSPFTrain (const byte *port*, 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](#).  
*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](#).

**11.3.3.379 char MSPFTrainEx (const byte *port*, const byte *addr*, const byte *channel*, const byte *func*) [inline]**

MSPFTrainEx 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](#).  
*addr* 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\\_MSPFTrainEx.nxc](#).

**11.3.3.380 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](#).

**11.3.3.381 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](#).

**11.3.3.382 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](#).

**11.3.3.383 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](#).

**11.3.3.384 void MSRCXBoot(void) [inline]**

MSRCXBoot function. Send the Boot command to an RCX.

**Examples:**

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

**11.3.3.385 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](#).

**11.3.3.386 void MSRCXClearAllEvents(void) [inline]**

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

**Examples:**

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

**11.3.3.387 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](#).

**11.3.3.388 void MSRCXClearMsg(void) [inline]**

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

**Examples:**

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

**11.3.3.389 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](#).

**11.3.3.390 void MSRCXClearSound (void) [inline]**

MSRCXClearSound function. Send the ClearSound command to an RCX.

**Examples:**

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

**11.3.3.391 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](#).

**11.3.3.392 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](#).

**11.3.3.393 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](#).



**11.3.3.394 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](#).

**11.3.3.395 void MSRCXDeleteSubs (void) [inline]**

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

**Examples:**

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

**11.3.3.396 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](#).

**11.3.3.397 void MSRCXDeleteTasks (void) [inline]**

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

**Examples:**

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

**11.3.3.398 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](#).

**11.3.3.399 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](#).

**11.3.3.400 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](#).

**11.3.3.401 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](#).

**11.3.3.402 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](#).

**11.3.3.403 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](#).

**11.3.3.404 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](#).

**11.3.3.405 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](#).

**11.3.3.406 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](#).

**11.3.3.407 void MSRCXMuteSound (void) [inline]**

MSRCXMuteSound function. Send the MuteSound command to an RCX.

**Examples:**

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

**11.3.3.408 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](#).

**11.3.3.409 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](#).

**11.3.3.410 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](#).

**11.3.3.411 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](#).

**11.3.3.412 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](#).

**11.3.3.413 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](#).

**11.3.3.414 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](#).

**11.3.3.415 void MSRCXPBTurnOff (void) [inline]**

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

**Examples:**

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

**11.3.3.416 void MSRCXPing (void) [inline]**

MSRCXPing function. Send the Ping command to an RCX.

**Examples:**

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

**11.3.3.417 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](#).

**11.3.3.418 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](#).

**11.3.3.419 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](#).

**11.3.3.420 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](#).

**11.3.3.421 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](#).

**11.3.3.422 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](#).

**11.3.3.423 void MSRCXReset (void) [inline]**

MSRCXReset function. Send the Reset command to an RCX.

**Examples:**

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

**11.3.3.424 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](#).

**11.3.3.425 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](#).

**11.3.3.426 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](#).

**11.3.3.427 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](#).



**11.3.3.428 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](#).

**11.3.3.429 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](#).

**11.3.3.430 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](#).

**11.3.3.431 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](#).

**11.3.3.432 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](#).

**11.3.3.433 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](#).

**11.3.3.434 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](#).

**11.3.3.435 void MSRCXSetNRLinkPort (const byte *port*) [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](#).

**Examples:**

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

**11.3.3.436 void MSRCXSetNRLinkPortEx (const byte *port*, const byte *addr*) [inline]**

MSRCXSetNRLinkPortEx 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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Examples:**

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

**11.3.3.437 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](#).

**11.3.3.438 void MSRCXSetPower (const byte *outputs*, const byte *pwsrc*, 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](#).

### 11.3.3.439 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](#).

### 11.3.3.440 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](#).

### 11.3.3.441 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](#).

**11.3.3.442 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](#).

**11.3.3.443 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](#).

**11.3.3.444 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](#).

**11.3.3.445 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](#).

**11.3.3.446 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](#).

**11.3.3.447 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](#).

**11.3.3.448 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](#).

**11.3.3.449 void MSRCXStopAllTasks (void) [inline]**

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

**Examples:**

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

**11.3.3.450 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](#).

**11.3.3.451 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](#).

**11.3.3.452 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](#).

**11.3.3.453 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](#).

**11.3.3.454 void MSRCXUnlock (void) [inline]**

MSRCXUnlock function. Send the Unlock command to an RCX.

**Examples:**

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

**11.3.3.455 void MSRCXUnmuteSound (void) [inline]**

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

**Examples:**

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

**11.3.3.456 int MSReadValue (const byte *port*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one or two byte value from a mindsensors sensor. The value must be stored with the least significant byte (LSB) first. 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](#).

*reg* The device register to read.

*numbytes* The number of bytes to read. Only 1 or 2 byte values are supported.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.457 int MSReadValueEx (const byte *port*, const byte *addr*, const byte *reg*, const byte *numbytes*) [inline]**

Read a mindsensors device value. Read a one or two byte value from a mindsensors sensor. The value must be stored with the least significant byte (LSB) first. 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](#).

*addr* 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 or 2 byte values are supported.



**Returns:**

The function call result.

**Examples:**

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

**11.3.3.458 void MSScoutCalibrateSensor (void) [inline]**

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

**Examples:**

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

**11.3.3.459 void MSScoutMuteSound (void) [inline]**

MSScoutMuteSound function. Send the MuteSound command to a Scout.

**Examples:**

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

**11.3.3.460 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](#).

**11.3.3.461 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](#).

**11.3.3.462 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](#).

**11.3.3.463 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](#).

**11.3.3.464 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](#).

**11.3.3.465 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](#).

**11.3.3.466 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](#).

**11.3.3.467 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](#).

**11.3.3.468 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](#).

**11.3.3.469 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](#).

**11.3.3.470 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](#).

**11.3.3.471 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](#).

**11.3.3.472 void MSScoutUnmuteSound (void) [inline]**

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

**Examples:**

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

**11.3.3.473 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](#).

**11.3.3.474 char NRLink2400 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400.nxc](#).

**11.3.3.475 char NRLink2400Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink2400Ex.nxc](#).

**11.3.3.476 char NRLink4800 (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800.nxc](#).

**11.3.3.477 char NRLink4800Ex (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

[ex\\_NRLink4800Ex.nxc](#).

**11.3.3.478 char NRLinkFlush (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.479 char NRLinkFlushEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.480 char NRLinkIRLong (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.481 char NRLinkIRLongEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.482 char NRLinkIRShort (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.483 char NRLinkIRShortEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.484 char NRLinkSetPF (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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



**11.3.3.485** `char NRLinkSetPFX (const byte port, const byte addr) [inline]`

Configure NRLink in power function mode. Configure the mindsensors NRLink device in power function mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.486** `char NRLinkSetRCX (const byte port) [inline]`

Configure NRLink in RCX mode. Configure the mindsensors NRLink device in RCX mode. 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 function call result.

**Examples:**

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

**11.3.3.487** `char NRLinkSetRCXEx (const byte port, const byte addr) [inline]`

Configure NRLink in RCX mode. Configure the mindsensors NRLink device in RCX mode. The port must be configured as a Low-speed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.488 char NRLinkSetTrain (const byte *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.489 char NRLinkSetTrainEx (const byte *port*, const byte *addr*) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.490 byte NRLinkStatus (const byte *port*) [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](#).

**Returns:**

The mindsensors NRLink status.

**Examples:**

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

**11.3.3.491** `byte NRLinkStatusEx (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors NRLink status.

**Examples:**

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

**11.3.3.492** `char NRLinkTxRaw (const byte port) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.493** `char NRLinkTxRawEx (const byte port, const byte addr) [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](#).

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

### 11.3.3.494 **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\\_dispgout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_isNAN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_NumOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_StrIndex.nxc](#), [ex\\_string.nxc](#), [ex\\_StrLenOld.nxc](#), [ex\\_strtod.nxc](#), [ex\\_strtol.nxc](#), [ex\\_strtoul.nxc](#), [ex\\_SysColorSensorRead.nxc](#), [ex\\_SysCommBTConnection.nxc](#), [ex\\_SysDatalogGetTimes.nxc](#), [ex\\_sysfileread.nxc](#), [ex\\_sysfilewrite.nxc](#), [ex\\_SysReadSemData.nxc](#), [ex\\_SysUpdateCalibCacheInfo.nxc](#), and [ex\\_SysWriteSemData.nxc](#).

### 11.3.3.495 **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](#), and [ex\\_string.nxc](#).

**11.3.3.496 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](#).

**11.3.3.497 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](#).

**11.3.3.498 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](#).

**11.3.3.499 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](#).

**11.3.3.500 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](#).

**11.3.3.501 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](#).

**11.3.3.502 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](#).

**11.3.3.503 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](#).

**11.3.3.504 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](#).

**11.3.3.505 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](#).

**11.3.3.506 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](#).

**11.3.3.507 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](#).



**11.3.3.508 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](#).

**11.3.3.509 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](#).

**11.3.3.510 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](#).

**11.3.3.511 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](#).

**11.3.3.512 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](#).

**11.3.3.513 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](#).

**11.3.3.514 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](#).

**11.3.3.515 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](#).

### 11.3.3.516 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](#).

### 11.3.3.517 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](#).

### 11.3.3.518 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](#).

**11.3.3.519 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\\_OpenFileAppend.nxc](#).

**11.3.3.520 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\\_OpenFileRead.nxc](#).

**11.3.3.521 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](#).

**11.3.3.522 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](#).

**11.3.3.523 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](#).

**11.3.3.524 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](#).

**11.3.3.525 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\\_PlayTone.nxc](#), and [ex\\_yield.nxc](#).

**11.3.3.526 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](#).

**11.3.3.527 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](#).

**11.3.3.528 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](#).



**11.3.3.529 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](#).

**11.3.3.530 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](#).

**11.3.3.531 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](#).

**11.3.3.532 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](#).

**11.3.3.533 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](#).

**11.3.3.534 char PSPNxAnalog (const byte & *port*) [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](#).

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.535 char PSPNxAnalogEx (const byte & *port*, const byte & *addr*) [inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.536 char PSPNxDigital (const byte & *port*) [inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. 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.

**Examples:**

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

**11.3.3.537 char PSPNxDigitalEx (const byte & *port*, const byte & *addr*) [inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. The port must be configured as a Lowspeed port before using this function.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.538 unsigned int rand () [inline]**

Generate random number. Returns a pseudo-random integral number in the range 0 to [RAND\\_MAX](#).

**Returns:**

An integer value between 0 and RAND\_MAX.

**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](#).

**11.3.3.539 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\\_Random.nxc](#), [ex\\_string.nxc](#), [ex\\_sysdrawellipse.nxc](#), and [ex\\_wait.nxc](#).

**11.3.3.540 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\\_Read.nxc](#).

**11.3.3.541 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](#).

**11.3.3.542 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](#).

**11.3.3.543 char ReadI2CRegister (byte *port*, 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.

*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](#).

**11.3.3.544 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](#).

**11.3.3.545 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](#).

**11.3.3.546 bool ReadNRLinkBytes (const byte *port*, 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](#).
- data* A byte array that will contain the data read from the device on output.

**Returns:**

The function call result.

**Examples:**

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

### 11.3.3.547 **bool ReadNRLinkBytesEx (const byte *port*, const byte *addr*, 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](#).
- addr* 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\\_ReadNRLinkBytesEx.nxc](#).

### 11.3.3.548 **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](#).

**11.3.3.549 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](#).

**11.3.3.550 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](#).



### 11.3.3.551 **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](#).

### 11.3.3.552 **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](#).

**11.3.3.553 bool ReadSensorHTIRReceiver (const byte *port*, char & *pfdata*[ ]) [inline]**

Read HiTechnic IRReceiver Power Function bytes. Read Power Function bytes from the HiTechnic IR-Receiver 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](#).

**11.3.3.554 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 IR-Receiver 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](#).

**11.3.3.555 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](#).

**11.3.3.556 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](#).

**11.3.3.557 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](#).

**11.3.3.558 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](#).

**11.3.3.559 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](#).

**11.3.3.560 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](#).

**11.3.3.561 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](#).

**11.3.3.562 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](#).

**11.3.3.563 bool ReadSensorMSAccel (const byte *port*, 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](#).

*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](#).

**11.3.3.564 bool ReadSensorMSAccelEx (const byte *port*, const byte *addr*, 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](#).

*addr* 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\\_ReadSensorMSAccelEx.nxc](#).

### 11.3.3.565 **bool ReadSensorMSPlayStation (const byte *port*, 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 Lowspeed port before using this function.

**Parameters:**

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

*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](#).

### 11.3.3.566 **bool ReadSensorMSPlayStationEx (const byte *port*, const byte *addr*, 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 Lowspeed port before using this function.

**Parameters:**

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

*addr* 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\\_ReadSensorMSPlayStationEx.nxc](#).

**11.3.3.567 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](#).

**11.3.3.568 bool ReadSensorMSTilt (const byte *port*, 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](#).

*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](#).

### 11.3.3.569 **bool ReadSensorMSTiltEx (const byte *port*, const byte *addr*, 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](#).  
*addr* 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\\_ReadSensorMSTiltEx.nxc](#).

### 11.3.3.570 **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\\_readsensorsex.nxc](#).

**11.3.3.571 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](#).

**11.3.3.572 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.

**11.3.3.573 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](#).

**11.3.3.574 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](#).

**11.3.3.575 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.

**11.3.3.576 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](#).

**11.3.3.577 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](#).

**11.3.3.578 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](#).

**11.3.3.579 unsigned long reladdr (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\\_reladdr.nxc](#).

**11.3.3.580 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](#).

**11.3.3.581 char RemoteKeepAlive (byte *conn*) [inline]**

Send a KeepAlive message. This method sends a KeepAlive direct command to the device on the specified connection. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**11.3.3.582 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*queue* The mailbox to read. See [Mailbox constants](#).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**11.3.3.583 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- 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](#).

**11.3.3.584 char RemotePlaySoundFile (byte *conn*, string *filename*, bool *bloop*) [inline]**

Send a PlaySoundFile message. Send the PlaySoundFile direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- 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](#).

**11.3.3.585 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

- conn* The connection slot (0..3).
- 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](#).

**11.3.3.586 char RemoteResetMotorPosition (byte *conn*, byte *port*, bool *brelative*) [inline]**

Send a ResetMotorPosition message. Send the ResetMotorPosition direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

**11.3.3.587 char RemoteResetScaledValue (byte *conn*, byte *port*) [inline]**

Send a ResetScaledValue message. Send the ResetScaledValue direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).  
*port* The input port to reset.

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**11.3.3.588 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

### 11.3.3.589 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 [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).

### 11.3.3.590 char RemoteStartProgram (byte *conn*, string *filename*) [inline]

Send a StartProgram message. Send the StartProgram direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

*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](#).



**11.3.3.591 char RemoteStopProgram (byte *conn*) [inline]**

Send a StopProgram message. Send the StopProgram direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**11.3.3.592 char RemoteStopSound (byte *conn*) [inline]**

Send a StopSound message. Send the StopSound direct command on the specified connection slot. Use [BluetoothStatus](#) to determine when this write request is completed.

**Parameters:**

*conn* The connection slot (0..3).

**Returns:**

A char value indicating whether the function call succeeded or not.

**Examples:**

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

**11.3.3.593 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.

**11.3.3.594 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](#).

**11.3.3.595 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.

**Examples:**

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

**11.3.3.596 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](#).

**11.3.3.597 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](#).

**11.3.3.598 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](#).

**11.3.3.599 void ResetScreen () [inline]**

Reset LCD screen. This function lets you restore the standard NXT running program screen.

**Examples:**

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

**11.3.3.600 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](#).

**11.3.3.601 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](#).

### 11.3.3.602 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](#).

### 11.3.3.603 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.

*newsize* The new size for the file.

#### Returns:

The function call result. See [Loader module error codes](#).

#### Examples:

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

### 11.3.3.604 unsigned int ResolveHandle (string *filename*, byte & *handle*, bool & *writeable*) [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.

*writeable* 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](#).

**11.3.3.605 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](#).

**11.3.3.606 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](#).

**11.3.3.607 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](#).

### 11.3.3.608 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](#).

### 11.3.3.609 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](#).

**11.3.3.610 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](#).

**11.3.3.611 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.

**Examples:**

[ex\\_RS485Control.nxc](#).

**11.3.3.612 bool RS485DataAvailable (void) [inline]**

Check for RS485 available data. Check the RS485 hi-speed port for available data.

**Returns:**

A value indicating whether data is available or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485DataAvailable.nxc](#).

**11.3.3.613 char RS485Exit (void) [inline]**

Exit 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\\_RS485Exit.nxc](#).

**11.3.3.614 char RS485Init (void) [inline]**

Initialize RS485. Initialize 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\\_RS485Init.nxc](#).

**11.3.3.615 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\\_RS485Read.nxc](#).

**11.3.3.616 bool RS485SendingData (void) [inline]**

Is RS485 sending data. Check whether the RS485 is actively sending data.

**Returns:**

A value indicating whether data is being sent or not.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485SendingData.nxc](#).

**11.3.3.617 void RS485Status (bool & *sendingData*, bool & *dataAvail*) [inline]**

Check RS485 status. Check the status of the RS485 hi-speed port.

**Parameters:**

*sendingData* A boolean value set to true on output if data is being sent.

*dataAvail* A boolean value set to true on output if data is available to be read.

**Warning:**

This function requires the enhanced NBC/NXC firmware.

**Examples:**

[ex\\_RS485Status.nxc](#).

**11.3.3.618 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\\_RS485Uart.nxc](#).

**11.3.3.619 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\\_RS485Write.nxc](#).

**11.3.3.620 char RunNRLinkMacro (const byte *port*, 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](#).

*macro* The address of the macro to execute.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.621 char RunNRLinkMacroEx (const byte *port*, const byte *addr*, 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](#).  
*addr* 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\\_RunNRLinkMacroEx.nxc](#).

**11.3.3.622 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](#).

**11.3.3.623 char SendRemoteBool (byte *conn*, byte *queue*, bool *bval*) [inline]**

Send a boolean value to a remote mailbox. Send a boolean value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

**11.3.3.624 char SendRemoteNumber (byte *conn*, byte *queue*, long *val*) [inline]**

Send a numeric value to a remote mailbox. Send a numeric value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

**11.3.3.625 char SendRemoteString (byte *conn*, byte *queue*, string *str*) [inline]**

Send a string value to a remote mailbox. Send a string value via bluetooth on the specified connection to the specified remote mailbox number.

**Parameters:**

*conn* The connection slot (0..3).  
*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](#).

**11.3.3.626 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](#).

**11.3.3.627 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](#).

**11.3.3.628 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](#).

**11.3.3.629 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.

**Examples:**

[ex\\_SendRS485Bool.nxc](#).

**11.3.3.630 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\\_SendRS485Number.nxc](#).

**11.3.3.631 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\\_SendRS485String.nxc](#).

**11.3.3.632 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](#).

**11.3.3.633 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](#).

**11.3.3.634 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](#).

**11.3.3.635 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](#).

**11.3.3.636 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](#).

**11.3.3.637 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](#).

**11.3.3.638 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](#).



**11.3.3.639 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](#).

**11.3.3.640 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\\_SensorHTGyro.nxc](#).

**11.3.3.641 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](#).

**11.3.3.642 int SensorHTIRSeeker2Addr (const byte & *port*, const byte *addr*) [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](#).

*addr* The register address. See [HiTechnic IRSeeker2 constants](#).

**Returns:**

The IRSeeker2 register value.

**Examples:**

[ex\\_SensorHTIRSeeker2Addr.nxc](#).

**11.3.3.643 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](#).

**11.3.3.644 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](#).

**11.3.3.645 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](#).

**11.3.3.646 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](#).

**11.3.3.647 int SensorMSCompass (const byte & *port*) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

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

**Returns:**

The mindsensors compass value

**Examples:**

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

**11.3.3.648 int SensorMSCompassEx (const byte & *port*, const byte *addr*) [inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

**Parameters:**

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

*addr* The sensor I2C address. See sensor documentation for this value.

**Returns:**

The mindsensors compass value

**Examples:**

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

**11.3.3.649 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](#).

**11.3.3.650 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](#).

**11.3.3.651 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](#).

**11.3.3.652 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](#).

**11.3.3.653 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](#).

**11.3.3.654 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](#).

**11.3.3.655 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](#).

**11.3.3.656 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 Low-speed 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](#).

**11.3.3.657 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](#).

**11.3.3.658 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](#).

**11.3.3.659 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](#).

**11.3.3.660 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](#).

**11.3.3.661 void SetBatteryState (byte *state*) [inline]**

Set battery state. Set battery state information.

**Parameters:**

*state* The desired battery state (0..4).

**Examples:**

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

**11.3.3.662 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](#).

**11.3.3.663 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](#).



**11.3.3.664 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](#).

**11.3.3.665 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](#).

**11.3.3.666 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](#).

**11.3.3.667 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](#).

**11.3.3.668 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](#).

**11.3.3.669 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](#).

**11.3.3.670 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](#).

**11.3.3.671 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](#).

**11.3.3.672 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](#).

**11.3.3.673 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](#).

**11.3.3.674 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](#).

**11.3.3.675 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](#).

**11.3.3.676 void SetCustomSensorActiveStatus (const byte *port*, const byte & *activeStatus*)  
[inline]**

Set active status. Sets the active status value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*activeStatus* The new active status value.

**Examples:**

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

**11.3.3.677 void SetCustomSensorPercentFullScale (const byte *port*, const 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](#). Must be a constant.

*pctFullScale* The new percent full scale value.

**Examples:**

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

**11.3.3.678 void SetCustomSensorZeroOffset (const byte *port*, const int & *zeroOffset*) [inline]**

Set percent zero offset. Sets the zero offset value of a custom sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*zeroOffset* The new zero offset value.

**Examples:**

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

**11.3.3.679 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](#).

**11.3.3.680 void SetDisplayDisplay (unsigned long *addr*) [inline]**

Set the display memory address. This function lets you set the current display memory address.

**Parameters:**

*addr* The new display memory address.

**Examples:**

[ex\\_dispmisc.nxc](#), and [ex\\_SetDisplayDisplay.nxc](#).

**11.3.3.681 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](#).

**11.3.3.682 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](#).

**11.3.3.683 void SetDisplayFont (unsigned long *addr*) [inline]**

Set the display font memory address. This function lets you set the current display font memory address.

**Parameters:**

*addr* The new display font memory address.

**Examples:**

[ex\\_addr.nxc](#), [ex\\_addrhex.nxc](#), [ex\\_DisplayFont.nxc](#), and [ex\\_SetDisplayFont.nxc](#).

**11.3.3.684 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](#).

**11.3.3.685 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](#).

**11.3.3.686 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](#).

**11.3.3.687 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](#).

**11.3.3.688 void SetHSFlags (const 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](#).

**11.3.3.689 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](#).

**11.3.3.690 void SetHSInputBufferInPtr (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\\_SetHSInputBufferInPtr.nxc](#).

**11.3.3.691 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](#).

**11.3.3.692 void SetHSMode (const 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](#).

**Examples:**

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

**11.3.3.693 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](#).

**11.3.3.694 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](#).



**11.3.3.695 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](#).

**11.3.3.696 void SetHSSpeed (const 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](#).

**11.3.3.697 void SetHSState (const 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](#).

**11.3.3.698 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](#).

**11.3.3.699 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](#).

**11.3.3.700 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](#).

**11.3.3.701 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](#), and [ex\\_SetLongAbort.nxc](#).

**11.3.3.702 void SetMotorPwnFreq (byte *n*) [inline]**

Set motor PWM frequency. Set the motor pulse width modulation frequency.

**Examples:**

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

**11.3.3.703 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](#).

**11.3.3.704 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](#).

**11.3.3.705 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](#).

### 11.3.3.706 void SetSensorBoolean (const byte *port*, const bool & *value*) [inline]

Set sensor boolean value. Sets the boolean value of a sensor.

**Parameters:**

*port* The sensor port. See [Input port constants](#). Must be a constant.

*value* The new boolean value.

### 11.3.3.707 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](#).

### 11.3.3.708 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](#).

**11.3.3.709 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](#).

**11.3.3.710 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](#).

**11.3.3.711 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](#).

**11.3.3.712 void SetSensorDigiPinsDirection (const byte *port*, const 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](#). Must be a constant.

*direction* The new digital pins direction value.

**Examples:**

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

**11.3.3.713 void SetSensorDigiPinsOutputLevel (const byte *port*, const 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](#). Must be a constant.

*outputLevel* The new digital pins output level value.

**Examples:**

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

**11.3.3.714 void SetSensorDigiPinsStatus (const byte *port*, const 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](#). Must be a constant.

*status* The new digital pins status value.

**Examples:**

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

**11.3.3.715 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](#).

**11.3.3.716 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\\_SensorHTGyro.nxc](#), and [ex\\_SetSensorHTGyro.nxc](#).

**11.3.3.717 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](#).

**11.3.3.718 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\\_HTRCXSetIRLinkPort.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), and [ex\\_SetSensorLowspeed.nxc](#).

**11.3.3.719 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](#).

#### 11.3.3.720 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](#).

#### 11.3.3.721 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](#).

#### 11.3.3.722 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](#).



**11.3.3.723 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\\_SetSensorTouch.nxc](#).

**11.3.3.724 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](#).

**11.3.3.725 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](#).

**11.3.3.726 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](#).

**11.3.3.727 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](#).

**11.3.3.728 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](#).

**11.3.3.729 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](#).

**11.3.3.730 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](#).

**11.3.3.731 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](#).

**11.3.3.732 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](#).

**11.3.3.733 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.](#)

#### 11.3.3.734 void SetSoundVolume (byte *volume*) [inline]

Set sound volume. Set the sound volume.

See also:

[SoundVolume\(\)](#)

Parameters:

*volume* The new volume

Examples:

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

#### 11.3.3.735 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.](#)

#### 11.3.3.736 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.](#)

**11.3.3.737 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](#).

**11.3.3.738 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](#).

**11.3.3.739 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](#).

**11.3.3.740 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](#).

**11.3.3.741 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](#).

**11.3.3.742 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](#).

**11.3.3.743 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](#).

**11.3.3.744 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](#).

**11.3.3.745 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](#).

**11.3.3.746 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](#).

**11.3.3.747 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](#).

**Examples:**

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

**11.3.3.748 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](#).

**11.3.3.749 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](#).

**11.3.3.750 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.nxc](#).

**11.3.3.751 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.nxc](#).



**11.3.3.752 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](#).

**11.3.3.753 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.

**11.3.3.754 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](#).

**11.3.3.755 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.](#)**11.3.3.756 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.](#)**11.3.3.757 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.](#)**11.3.3.758 unsigned int SoundDuration () [inline]**

Get sound duration. Return the current sound duration.

**See also:**[SetSoundDuration\(\)](#)**Returns:**

The current sound duration.

**Examples:**[ex\\_SoundDuration.nxc.](#)

**11.3.3.759 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](#).

**11.3.3.760 unsigned int SoundFrequency () [inline]**

Get sound frequency. Return the current sound frequency.

**See also:**

[SetSoundFrequency\(\)](#)

**Returns:**

The current sound frequency.

**Examples:**

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

**11.3.3.761 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](#).

**11.3.3.762 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](#).

**11.3.3.763 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](#).

**11.3.3.764 byte SoundVolume () [inline]**

Get volume. Return the current sound volume.

**See also:**

[SetSoundVolume\(\)](#)

**Returns:**

The current sound volume.

**Examples:**

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

**11.3.3.765 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](#).

**11.3.3.766 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](#).

**11.3.3.767 void StartTask (task *t*) [inline]**

Start a task. Start the specified task.

**Parameters:**

*t* The task to start.

**Examples:**

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

**11.3.3.768 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\\_stop.nxc](#).

**11.3.3.769 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](#).

**11.3.3.770 byte StopSound () [inline]**

Stop sound. Stop playing of the current tone or file.

**Returns:**

The result

**Todo**

?

**Examples:**

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

**11.3.3.771 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](#).

**11.3.3.772 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](#).

**11.3.3.773 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\\_StrCatOld.nxc](#), [ex\\_string.nxc](#), [ex\\_StrReplace.nxc](#), and [util\\_battery\\_1.nxc](#).

**11.3.3.774 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](#).

**11.3.3.775 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](#).

**11.3.3.776 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](#).

**11.3.3.777 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](#).



**11.3.3.778 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](#).

**11.3.3.779 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](#).

**11.3.3.780 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](#).

**11.3.3.781 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](#).

**11.3.3.782 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](#).

**11.3.3.783 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](#).

**11.3.3.784 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](#).

**11.3.3.785 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](#).

### 11.3.3.786 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](#).

### 11.3.3.787 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](#).

**11.3.3.788 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](#).

**11.3.3.789 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](#).

**11.3.3.790 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](#).

**11.3.3.791 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](#).

**11.3.3.792 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](#).

**11.3.3.793 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](#).

**11.3.3.794 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](#).

**11.3.3.795 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](#).

**11.3.3.796 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](#).

**11.3.3.797 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](#).

**11.3.3.798 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](#).

**11.3.3.799 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](#).

**11.3.3.800 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](#).

**11.3.3.801 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](#).

**11.3.3.802 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](#).

**11.3.3.803 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](#).

**11.3.3.804 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](#).

**11.3.3.805 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](#).

**11.3.3.806 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](#).

**11.3.3.807 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](#).

**11.3.3.808 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](#).

**11.3.3.809 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](#).

**11.3.3.810 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](#).

**11.3.3.811 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](#).

**11.3.3.812 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](#).

**11.3.3.813 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](#).

**11.3.3.814 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](#).

**11.3.3.815 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](#).

**11.3.3.816 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](#).

**11.3.3.817 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](#).

**11.3.3.818 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](#).

**11.3.3.819 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](#).

**11.3.3.820 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](#).

**11.3.3.821 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](#).

**11.3.3.822 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](#).

**11.3.3.823 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](#).

**11.3.3.824 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](#).

**11.3.3.825 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](#).

**11.3.3.826 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](#).



**11.3.3.827 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](#).

**11.3.3.828 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](#).

**11.3.3.829 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](#).

**11.3.3.830 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\\_sysfilesize.nxc](#).

**11.3.3.831 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](#).

**11.3.3.832 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](#).

**11.3.3.833 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](#).

**11.3.3.834 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](#).

**11.3.3.835 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](#).

**11.3.3.836 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\\_reladdr.nxc](#), and [ex\\_sysiomapreadbyid.nxc](#).

**11.3.3.837 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](#).

**11.3.3.838 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\\_reladdr.nxc](#), and [ex\\_sysiomapwritebyid.nxc](#).

**11.3.3.839 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](#).

**11.3.3.840 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](#).

**11.3.3.841 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](#).

**11.3.3.842 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](#).

**11.3.3.843 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](#).

**11.3.3.844 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](#).

**11.3.3.845 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](#).

**11.3.3.846 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](#).

**11.3.3.847 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](#).

**11.3.3.848 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](#).

**11.3.3.849 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](#).

**11.3.3.850 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](#).

**11.3.3.851 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](#).

**11.3.3.852 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](#).

**11.3.3.853 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](#).

**11.3.3.854 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](#).



**11.3.3.855 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](#).

**11.3.3.856 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](#).

**11.3.3.857 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](#).

**11.3.3.858 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.

**11.3.3.859 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\\_addr.nxc](#), [ex\\_addrex.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\\_ClearLine.nxc](#), [ex\\_Copy.nxc](#), [ex\\_ctype.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_I2CDeviceId.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_I2CVendorId.nxc](#), [ex\\_I2CVendorIdEx.nxc](#), [ex\\_I2CVersion.nxc](#), [ex\\_I2CVersionEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_midstr.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SetDisplayFont.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\\_SysCommBTOnOff.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\\_TextOut.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**11.3.3.860 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](#).

**11.3.3.861 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](#).

**11.3.3.862 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\\_trunc.nxc](#).

**11.3.3.863 byte UIButton (void) [inline]**

Read UI button. Return user interface button information.

**Returns:**

A UI button value. See [UIButton constants](#).

**Examples:**

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

**11.3.3.864 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](#).

**11.3.3.865 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\\_string.nxc](#), and [ex\\_UnflattenVar.nxc](#).

**11.3.3.866 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](#).

**11.3.3.867 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](#).

**11.3.3.868 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](#).

**11.3.3.869 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](#).

**11.3.3.870 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](#).

**11.3.3.871 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](#).

**11.3.3.872 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](#).

**11.3.3.873 byte USBState (void) [inline]**

Get USB state. This method returns the value of the USB state.

**Returns:**

The USB state.

**11.3.3.874 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](#).

**11.3.3.875 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](#).

**11.3.3.876 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\\_addr.nxc](#), [ex\\_addrex.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\\_dispfout.nxc](#), [ex\\_dispfunc.nxc](#), [ex\\_dispgaout.nxc](#), [ex\\_dispgout.nxc](#), [ex\\_dispgoutex.nxc](#), [ex\\_DisplayFont.nxc](#), [ex\\_dispmisc.nxc](#), [ex\\_div.nxc](#), [ex\\_getchar.nxc](#), [ex\\_I2CDeviceIdEx.nxc](#), [ex\\_I2CDeviceInfo.nxc](#), [ex\\_I2CDeviceInfoEx.nxc](#), [ex\\_isNaN.nxc](#), [ex\\_labs.nxc](#), [ex\\_ldiv.nxc](#), [ex\\_leftstr.nxc](#), [ex\\_LineOut.nxc](#), [ex\\_memcmp.nxc](#), [ex\\_midstr.nxc](#), [ex\\_playsound.nxc](#), [ex\\_playtones.nxc](#), [ex\\_PolyOut.nxc](#), [ex\\_reladdr.nxc](#), [ex\\_rightstr.nxc](#), [ex\\_SensorHTGyro.nxc](#), [ex\\_SetDisplayFont.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\\_wait.nxc](#), [ex\\_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util\\_battery\\_1.nxc](#), [util\\_battery\\_2.nxc](#), and [util\\_rpm.nxc](#).

**11.3.3.877 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\\_Write.nxc](#).

**11.3.3.878 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](#).

**11.3.3.879 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](#).

**11.3.3.880 char WriteI2CRegister (byte *port*, 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.  
*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](#).



**11.3.3.881 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](#).

**11.3.3.882 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](#).

**11.3.3.883 char WriteNRLinkBytes (const byte *port*, 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](#).

*data* A byte array containing the data to write.

**Returns:**

The function call result.

**Examples:**

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

**11.3.3.884 char WriteNRLinkBytesEx (const byte *port*, const byte *addr*, 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](#).

*addr* 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\\_WriteNRLinkBytesEx.nxc](#).

**11.3.3.885 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](#).

**11.3.3.886 void Yield () [inline]**

Yield to another task. Make a task yield to another concurrently running task.

**Examples:**

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

## Chapter 12

# Example Documentation

### 12.1 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);
}
```

## 12.2 ex\_abort.nxc

This is an example of how to use the [abort](#) function.

```
abort(); // stop the program
```

## 12.3 ex\_AbortFlag.nxc

This is an example of how to use the [AbortFlag](#) function.

```
byte af = AbortFlag();
```

## 12.4 ex\_abs.nxc

This is an example of how to use the [abs](#) function.

```
float val = abs(x); // return the absolute value of x
```

## 12.5 ex\_acos.nxc

This is an example of how to use the [acos](#) function.

```
float x = acos(0); // radians
```

## 12.6 ex\_acosd.nxc

This is an example of how to use the [acosd](#) function.

```
float x = acosd(0); // degrees
```



## 12.7 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);
```

## 12.8 ex\_addr.nxc

This is an example of how to use the [addr](#) 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,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x0
    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,
    0x00,0x02,0x05,0x02,0x00,0x00,0x20,0x54,0x54,0x78,0x00,0x7F,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 = 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);
}
```

## 12.9 ex\_addrx.nxc

This is an example of how to use the [addrx](#) 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,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x0
    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 = addrx(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);
}
```

## 12.10 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]);
}
```

## 12.11 ex\_arrayinit.nxc

This is an example of how to use the [ArrayInit](#) function.

```
ArrayInit(myArray, 0, 10); // 10 elements == zero
```

## 12.12 ex\_arraylen.nxc

This is an example of how to use the [ArrayLen](#) function.

```
x = ArrayLen(myArray);
```

## 12.13 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);
}
```

## 12.14 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);
}
```



## 12.15 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);
}
```

## 12.16 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);
}
```

## 12.17 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);
}
```

## 12.18 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);
}
```

## 12.19 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);
```

## 12.20 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);
}
```

## 12.21 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);
}
```

## 12.22 ex\_asin.nxc

This is an example of how to use the [asin](#) function.

```
float x = asin(0.80); // radians
```



## 12.23 ex\_asind.nxc

This is an example of how to use the [asind](#) function.

```
float x = asind(0.80); // degrees
```

## 12.24 ex\_atan.nxc

This is an example of how to use the [atan](#) function.

```
float x = atan(0); // radians
```

## 12.25 ex\_atan2.nxc

This is an example of how to use the [atan2](#) function.

```
z = atan2(y, x);
```

## 12.26 ex\_atan2d.nxc

This is an example of how to use the [atan2d](#) function.

```
z = atan2d(y, x);
```

## 12.27 ex\_atand.nxc

This is an example of how to use the [atand](#) function.

```
float x = atand(0); // degrees
```

## 12.28 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);
}
```

## 12.29 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);
}
```

## 12.30 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);
}
```



## 12.31 ex\_BatteryState.nxc

This is an example of how to use the [BatteryState](#) function.

```
x = BatteryState();
```

## 12.32 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);
```

## 12.33 ex\_BluetoothState.nxc

This is an example of how to use the [BluetoothState](#) function.

```
x = BluetoothState();
```

## 12.34 ex\_bluetoothstatus.nxc

This is an example of how to use the [BluetoothStatus](#) function.

```
x = BluetoothStatus(1);
```

## 12.35 ex\_bluetoothwrite.nxc

This is an example of how to use the [BluetoothWrite](#) function.

```
x = BluetoothWrite(1, data);
```

## 12.36 ex\_BrickDataBluecoreVersion.nxc

This is an example of how to use the [BrickDataBluecoreVersion](#) function.

```
int bv = BrickDataBluecoreVersion();
```

## 12.37 ex\_BrickDataBtHardwareStatus.nxc

This is an example of how to use the [BrickDataBtHardwareStatus](#) function.

```
int x = BrickDataBtHardwareStatus();
```

## 12.38 ex\_BrickDataBtStateStatus.nxc

This is an example of how to use the [BrickDataBtStateStatus](#) function.

```
int x = BrickDataBtStateStatus();
```



## 12.39 ex\_BrickDataName.nxc

This is an example of how to use the [BrickDataName](#) function.

```
string name = BrickDataName();
```

## 12.40 ex\_BrickDataTimeoutValue.nxc

This is an example of how to use the [BrickDataTimeoutValue](#) function.

```
int x = BrickDataTimeoutValue();
```

## 12.41 ex\_BTConnectionClass.nxc

This is an example of how to use the [BTConnectionClass](#) function.

```
long class = BTConnectionClass(1);
```

## 12.42 ex\_BTConnectionHandleNum.nxc

This is an example of how to use the [BTConnectionHandleNum](#) function.

```
byte handlenum = BTConnectionHandleNum(idx);
```

## 12.43 ex\_BTConnectionLinkQuality.nxc

This is an example of how to use the [BTConnectionLinkQuality](#) function.

```
byte linkquality = BTConnectionLinkQuality(1);
```

## 12.44 ex\_BTConnectionName.nxc

This is an example of how to use the [BTConnectionName](#) function.

```
string name = BTConnectionName(0);
```

## 12.45 ex\_BTConnectionPinCode.nxc

This is an example of how to use the [BTConnectionPinCode](#) function.

```
string pincode = BTConnectionPinCode(0);
```

## 12.46 ex\_BTConnectionStreamStatus.nxc

This is an example of how to use the [BTConnectionStreamStatus](#) function.

```
byte streamstatus = BTConnectionStreamStatus(1);
```



## 12.47 ex\_BTDeviceClass.nxc

This is an example of how to use the [BTDeviceClass](#) function.

```
long class = BTDeviceClass(1);
```

## 12.48 ex\_BTDeviceCount.nxc

This is an example of how to use the [BTDeviceCount](#) function.

```
byte x = BTDeviceCount();
```

## 12.49 ex\_BTDeviceName.nxc

This is an example of how to use the [BTDeviceName](#) function.

```
string name = BTDeviceName(0);
```

## 12.50 ex\_BTDeviceNameCount.nxc

This is an example of how to use the [BTDeviceNameCount](#) function.

```
byte x = BTDeviceNameCount();
```

## 12.51 ex\_BTDeviceStatus.nxc

This is an example of how to use the [BTDeviceStatus](#) function.

```
byte status = BTDeviceStatus(1);
```

## 12.52 ex\_BTInputBufferInPtr.nxc

This is an example of how to use the [BTInputBufferInPtr](#) function.

```
byte x = BTInputBufferInPtr();
```

## 12.53 ex\_BTInputBufferOutPtr.nxc

This is an example of how to use the [BTInputBufferOutPtr](#) function.

```
byte x = BTInputBufferOutPtr();
```

## 12.54 ex\_BTOutputBufferInPtr.nxc

This is an example of how to use the [BTOutputBufferInPtr](#) function.

```
byte x = BTOutputBufferInPtr();
```



## 12.55 ex\_BTOutputBufferOutPtr.nxc

This is an example of how to use the [BTOutputBufferOutPtr](#) function.

```
byte x = BTOutputBufferOutPtr();
```

## 12.56 ex\_ButtonCount.nxc

This is an example of how to use the [ButtonCount](#) function.

```
value = ButtonCount (BTN1, true);
```

## 12.57 ex\_ButtonLongPressCount.nxc

This is an example of how to use the [ButtonLongPressCount](#) function.

```
value = ButtonLongPressCount (BTN1);
```

## 12.58 ex\_ButtonLongReleaseCount.nxc

This is an example of how to use the [ButtonLongReleaseCount](#) function.

```
value = ButtonLongReleaseCount (BTN1);
```

## 12.59 ex\_ButtonPressCount.nxc

This is an example of how to use the [ButtonPressCount](#) function.

```
value = ButtonPressCount (BTN1);
```

## 12.60 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));
    }
}
```

## 12.61 ex\_ButtonReleaseCount.nxc

This is an example of how to use the [ButtonReleaseCount](#) function.

```
value = ButtonReleaseCount (BTN1);
```

## 12.62 ex\_ButtonShortReleaseCount.nxc

This is an example of how to use the [ButtonShortReleaseCount](#) function.

```
value = ButtonShortReleaseCount (BTN1);
```



## 12.63 ex\_ButtonState.nxc

This is an example of how to use the [ButtonState](#) function.

```
value = ButtonState(BTN1);
```

## 12.64 ex\_ByteArrayToStr.nxc

This is an example of how to use the [ByteArrayToStr](#) function.

```
myStr = ByteArrayToStr(myArray);
```

## 12.65 ex\_ByteArrayToStrEx.nxc

This is an example of how to use the [ByteArrayToStrEx](#) function.

```
ByteArrayToStrEx(myArray, myStr);
```

## 12.66 ex\_ceil.nxc

This is an example of how to use the [ceil](#) function.

```
y = ceil(x);
```

## 12.67 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);
}
```

## 12.68 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);
}
```

## 12.69 ex\_ClearScreen.nxc

This is an example of how to use the [ClearScreen](#) and [Wait](#) functions.

```
task main()
{
    ClearScreen();
    Wait(SEC_10);
}
```

## 12.70 ex\_ClearSensor.nxc

This is an example of how to use the [ClearSensor](#) function.

```
ClearSensor(S1);
```



## 12.71 ex\_CloseFile.nxc

This is an example of how to use the [CloseFile](#) function.

```
result = CloseFile(handle);
```

## 12.72 ex\_coast.nxc

This is an example of how to use the [Coast](#) function.

```
Coast(OUT_A); // coast output A
```

## 12.73 ex\_coastex.nxc

This is an example of how to use the [CoastEx](#) function.

```
CoastEx(OUT_A, RESET_NONE); // coast output A
```

## 12.74 ex\_ColorADRaw.nxc

This is an example of how to use the [ColorADRaw](#) function.

```
unsigned int rawRed = ColorADRaw(S1, INPUT_RED);
```

## 12.75 ex\_ColorBoolean.nxc

This is an example of how to use the [ColorBoolean](#) function.

```
bool bRed = ColorBoolean(S1, INPUT_RED);
```

## 12.76 ex\_ColorCalibration.nxc

This is an example of how to use the [ColorCalibration](#) function.

```
long value = ColorCalibration(S1, INPUT_CAL_POINT_0, INPUT_RED);
```

## 12.77 ex\_ColorCalibrationState.nxc

This is an example of how to use the [ColorCalibrationState](#) function.

```
byte value = ColorCalibrationState(S1);
```

## 12.78 ex\_ColorCalLimits.nxc

This is an example of how to use the [ColorCalLimits](#) function.

```
unsigned int limit = ColorCalLimits(S1, INPUT_CAL_POINT_0);
```



## 12.79 ex\_ColorSensorRaw.nxc

This is an example of how to use the [ColorSensorRaw](#) function.

```
unsigned int rawRed = ColorSensorRaw(S1, INPUT_RED);
```

## 12.80 ex\_ColorSensorValue.nxc

This is an example of how to use the [ColorSensorValue](#) function.

```
unsigned int valRed = ColorSensorValue(S1, INPUT_RED);
```

## 12.81 ex\_CommandFlags.nxc

This is an example of how to use the [CommandFlags](#) function.

```
x = CommandFlags();
```

## 12.82 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);
}
```

## 12.83 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);
}
```

## 12.84 ex\_cos.nxc

This is an example of how to use the `cos` function.

```
float x = cos(y); // radians
```

## 12.85 ex\_cosd.nxc

This is an example of how to use the `cosd` function.

```
float x = cosd(y); // degrees
```

## 12.86 ex\_cosh.nxc

This is an example of how to use the [cosh](#) function.

```
x = cosh(y);
```



## 12.87 ex\_CreateFile.nxc

This is an example of how to use the [CreateFile](#) function.

```
result = CreateFile("data.txt", 1024, handle);
```

## 12.88 ex\_CreateFileLinear.nxc

This is an example of how to use the [CreateFileLinear](#) function.

```
result = CreateFileLinear("data.txt", 1024, handle);
```

## 12.89 ex\_CreateFileNonLinear.nxc

This is an example of how to use the [CreateFileNonLinear](#) function.

```
result = CreateFileNonLinear("data.txt", 1024, handle);
```

## 12.90 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)

    */
}
```

## 12.91 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)
    */
}
```

## 12.92 ex\_ctype.nxc

This is an example of how to use the ctype API functions: [isupper](#), [islower](#), [isalpha](#), [isdigit](#), [isalnum](#), [isspace](#), [isctrl](#), [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, isctrl(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);
}
```

## 12.93 ex\_CurrentTick.nxc

This is an example of how to use the [CurrentTick](#) function.

```
unsigned int x = CurrentTick();
```

## 12.94 ex\_CustomSensorActiveStatus.nxc

This is an example of how to use the [CustomSensorActiveStatus](#) function.

```
x = CustomSensorActiveStatus(S1);
```



## 12.95 ex\_CustomSensorPercentFullScale.nxc

This is an example of how to use the [CustomSensorPercentFullScale](#) function.

```
x = CustomSensorPercentFullScale(S1);
```

## 12.96 ex\_CustomSensorZeroOffset.nxc

This is an example of how to use the [CustomSensorZeroOffset](#) function.

```
x = CustomSensorZeroOffset(S1);
```

## 12.97 ex\_DeleteFile.nxc

This is an example of how to use the [DeleteFile](#) function.

```
result = DeleteFile("data.txt");
```

## 12.98 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 ) ;
}
```

## 12.99 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);
}
```

## 12.100 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);
}
```

## 12.101 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 );
    }
}
```

## 12.102 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),
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 );
}
```

## 12.103 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);
}
```

## 12.104 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 );
    }
}
```

## 12.105 ex\_DisplayDisplay.nxc

This is an example of how to use the [DisplayDisplay](#) function.

```
x = DisplayDisplay();
```

## 12.106 ex\_DisplayEraseMask.nxc

This is an example of how to use the [DisplayEraseMask](#) function.

```
x = DisplayEraseMask();
```



## 12.107 ex\_DisplayFlags.nxc

This is an example of how to use the [DisplayFlags](#) function.

```
x = DisplayFlags();
```

## 12.108 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
    0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,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,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x0
    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
    ,0x66,0x00,0x26,0x49,0x49,0x32,0x00,0x01,0x01,0x7F,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,0x10,0x28,0x44,0x00,0x00,0x00,0x00,0x00,0x7F,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;
    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);
}
```

## 12.109 ex\_DisplayTextLinesCenterFlags.nxc

This is an example of how to use the [DisplayTextLinesCenterFlags](#) function.

```
x = DisplayTextLinesCenterFlags();
```

## 12.110 ex\_DisplayUpdateMask.nxc

This is an example of how to use the [DisplayUpdateMask](#) function.

```
x = DisplayUpdateMask();
```

## 12.111 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);
}
```

## 12.112 ex\_DISTNxDistance.nxc

This is an example of how to use the [DISTNxDistance](#) function.

```
int dist = DISTNxDistance(S1);
```

## 12.113 ex\_DISTNxDistanceEx.nxc

This is an example of how to use the [DISTNxDistanceEx](#) function.

```
int dist = DISTNxDistanceEx(S1, 0x02);
```

## 12.114 ex\_DISTNxGP2D12.nxc

This is an example of how to use the [DISTNxGP2D12](#) function.

```
char result = DISTNxGP2D12(S1);
```



## 12.115 ex\_DISTNxGP2D120.nxc

This is an example of how to use the [DISTNxGP2D120](#) function.

```
char result = DISTNxGP2D120(S1);
```

## 12.116 ex\_DISTNxGP2D120Ex.nxc

This is an example of how to use the [DISTNxGP2D120Ex](#) function.

```
char result = DISTNxGP2D120Ex(S1, 0x02);
```

## 12.117 ex\_DISTNxGP2D12Ex.nxc

This is an example of how to use the [DISTNxGP2D12Ex](#) function.

```
char result = DISTNxGP2D12Ex(S1, 0x02);
```

## 12.118 ex\_DISTNxGP2YA02.nxc

This is an example of how to use the [DISTNxGP2YA02](#) function.

```
char result = DISTNxGP2YA02(S1);
```

## 12.119 ex\_DISTNxGP2YA02Ex.nxc

This is an example of how to use the [DISTNxGP2YA02Ex](#) function.

```
char result = DISTNxGP2YA02Ex(S1, 0x02);
```

## 12.120 ex\_DISTNxGP2YA21.nxc

This is an example of how to use the [DISTNxGP2YA21](#) function.

```
char result = DISTNxGP2YA21(S1);
```

## 12.121 ex\_DISTNxGP2YA21Ex.nxc

This is an example of how to use the [DISTNxGP2YA21Ex](#) function.

```
char result = DISTNxGP2YA21Ex(S1, 0x02);
```

## 12.122 ex\_DISTNxMaxDistance.nxc

This is an example of how to use the [DISTNxMaxDistance](#) function.

```
int dist = DISTNxMaxDistance(S1);
```



## 12.123 ex\_DISTNxMaxDistanceEx.nxc

This is an example of how to use the [DISTNxMaxDistanceEx](#) function.

```
int dist = DISTNxMaxDistanceEx(S1, 0x02);
```

## 12.124 ex\_DISTNxMinDistance.nxc

This is an example of how to use the [DISTNxMinDistance](#) function.

```
int dist = DISTNxMinDistance(S1);
```

## 12.125 ex\_DISTNxMinDistanceEx.nxc

This is an example of how to use the [DISTNxMinDistanceEx](#) function.

```
int dist = DISTNxMinDistanceEx(S1, 0x02);
```

## 12.126 ex\_DISTNxModuleType.nxc

This is an example of how to use the [DISTNxModuleType](#) function.

```
int dist = DISTNxModuleType(S1);
```

## 12.127 ex\_DISTNxModuleTypeEx.nxc

This is an example of how to use the [DISTNxModuleTypeEx](#) function.

```
int dist = DISTNxModuleTypeEx(S1, 0x02);
```

## 12.128 ex\_DISTNxNumPoints.nxc

This is an example of how to use the [DISTNxNumPoints](#) function.

```
int dist = DISTNxNumPoints(S1);
```

## 12.129 ex\_DISTNxNumPointsEx.nxc

This is an example of how to use the [DISTNxNumPointsEx](#) function.

```
int dist = DISTNxNumPointsEx(S1, 0x02);
```

## 12.130 ex\_DISTNxVoltage.nxc

This is an example of how to use the [DISTNxVoltage](#) function.

```
int dist = DISTNxVoltage(S1);
```



## 12.131 ex\_DISTNxVoltageEx.nxc

This is an example of how to use the [DISTNxVoltageEx](#) function.

```
int dist = DISTNxVoltageEx(S1, 0x02);
```

## 12.132 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);
}
```

## 12.133 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);
}
```

## 12.134 ex\_exp.nxc

This is an example of how to use the [exp](#) function.

```
y = exp(x);
```

## 12.135 ex\_fclose.nxc

This is an example of how to use the [fclose](#) function.

```
result = fclose(handle);
```

## 12.136 ex\_feof.nxc

This is an example of how to use the [feof](#) function.

```
int i = feof(handle);
```

## 12.137 ex\_fflush.nxc

This is an example of how to use the [fflush](#) function.

```
int i = fflush(handle);
```

## 12.138 ex\_fgetc.nxc

This is an example of how to use the [fgetc](#) function.

```
char val = fgetc(handle);
```



## 12.139 ex\_fgets.nxc

This is an example of how to use the [fgets](#) function.

```
fgets(msg, 10, handle);
```

## 12.140 ex\_FindFirstFile.nxc

This is an example of how to use the [FindFirstFile](#) function.

```
result = FindFirstFile(fname, handle);
```

## 12.141 ex\_FindNextFile.nxc

This is an example of how to use the [FindNextFile](#) function.

```
result = FindNextFile(fname, handle);
```

## 12.142 ex\_FirstTick.nxc

This is an example of how to use the [FirstTick](#) function.

```
unsigned int x = FirstTick();
```

## 12.143 ex\_Flatten.nxc

This is an example of how to use the [Flatten](#) function.

```
msg = Flatten(48); // returns "0" since 48 == ascii("0")
```

## 12.144 ex\_FlattenVar.nxc

This is an example of how to use the [FlattenVar](#) function.

```
stringValue = FlattenVar(myStruct);
```

## 12.145 ex\_float.nxc

This is an example of how to use the [Float](#) function.

```
Float(OUT_A); // float output A
```

## 12.146 ex\_floor.nxc

This is an example of how to use the [floor](#) function.

```
y = floor(x);
```



## 12.147 ex\_Follows.nxc

This is an example of how to use the [Follows](#) statement.

```
Follows (main) ;
```

## 12.148 ex\_fopen.nxc

This is an example of how to use the [fopen](#) function.

```
byte handle = fopen("test.txt", "r");
```

## 12.149 ex\_ForceOff.nxc

This is an example of how to use the [ForceOff](#) function.

```
ForceOff(true);
```

## 12.150 ex\_FormatNum.nxc

This is an example of how to use the [FormatNum](#) function.

```
msg = FormatNum("value = %d", x);
```

## 12.151 ex\_fprintf.nxc

This is an example of how to use the [fprintf](#) function.

```
fprintf(handle, "value = %d", value);
```

## 12.152 ex\_fputc.nxc

This is an example of how to use the `fputc` function.

```
fputc(ch, handle);
```

## 12.153 ex\_fputs.nxc

This is an example of how to use the `fputs` function.

```
fputs(msg, handle);
```

## 12.154 ex\_frac.nxc

This is an example of how to use the [frac](#) function.

```
y = frac(x);
```



## 12.155 ex\_FreeMemory.nxc

This is an example of how to use the [FreeMemory](#) function.

```
x = FreeMemory();
```

## 12.156 ex\_fseek.nxc

This is an example of how to use the [fseek](#) function.

```
fseek(handle, 10, SEEK_CUR);
```

## 12.157 ex\_ftell.nxc

This is an example of how to use the [ftell](#) function.

```
long i = ftell(handle);
```

## 12.158 ex\_GetBrickDataAddress.nxc

This is an example of how to use the [GetBrickDataAddress](#) function.

```
GetBrickDataAddress (buffer) ;
```

## 12.159 ex\_GetBTConnectionAddress.nxc

This is an example of how to use the [GetBTConnectionAddress](#) function.

```
GetBTConnectionAddress(0, buffer);
```

## 12.160 ex\_GetBTDeviceAddress.nxc

This is an example of how to use the [GetBTDeviceAddress](#) function.

```
GetBTDeviceAddress (0, buffer);
```

## 12.161 ex\_GetBTInputBuffer.nxc

This is an example of how to use the [GetBTInputBuffer](#) function.

```
GetBTInputBuffer(0, 10, buffer);
```

## 12.162 ex\_GetBTOutputBuffer.nxc

This is an example of how to use the [GetBTOutputBuffer](#) function.

```
GetBTOutputBuffer(0, 10, buffer);
```



## 12.163 ex\_getc.nxc

This is an example of how to use the [getc](#) function.

```
char val = getc(handle);
```

## 12.164 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);
    }
}
```

## 12.165 ex\_GetDisplayNormal.nxc

This is an example of how to use the [GetDisplayNormal](#) function.

```
GetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

## 12.166 ex\_GetDisplayPopup.nxc

This is an example of how to use the [GetDisplayPopup](#) function.

```
GetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

## 12.167 ex\_GetHSInputBuffer.nxc

This is an example of how to use the [GetHSInputBuffer](#) function.

```
GetHSInputBuffer(0, 10, buffer);
```

## 12.168 ex\_GetHSOutputBuffer.nxc

This is an example of how to use the [GetHSOutputBuffer](#) function.

```
GetHSOutputBuffer(0, 10, buffer);
```

## 12.169 ex\_GetInput.nxc

This is an example of how to use the [GetInput](#) function.

```
x = GetInput (S1, Type);
```

## 12.170 ex\_GetLSInputBuffer.nxc

This is an example of how to use the [GetLSInputBuffer](#) function.

```
GetLSInputBuffer(S1, 0, 8, buffer);
```



## 12.171 ex\_GetLSOutputBuffer.nxc

This is an example of how to use the [GetLSOutputBuffer](#) function.

```
GetLSOutputBuffer(S1, 0, 8, outbuffer);
```

## 12.172 ex\_getoutput.nxc

This is an example of how to use the [GetOutput](#) function.

```
x = GetOutput(OUT_A, TachoLimit);
```

## 12.173 ex\_GetUSBInputBuffer.nxc

This is an example of how to use the [GetUSBInputBuffer](#) function.

```
GetUSBInputBuffer(0, 10, buffer);
```

## 12.174 ex\_GetUSBOutputBuffer.nxc

This is an example of how to use the [GetUSBOutputBuffer](#) function.

```
GetUSBOutputBuffer(0, 10, buffer);
```

## 12.175 ex\_GetUSBPollBuffer.nxc

This is an example of how to use the [GetUSBPollBuffer](#) function.

```
GetUSBPollBuffer(0, 10, buffer);
```

## 12.176 ex\_GraphicOut.nxc

This is an example of how to use the [GraphicOut](#) function.

```
GraphicOut(40, 40, "image.ric");
```

## 12.177 ex\_GraphicOutEx.nxc

This is an example of how to use the [GraphicOutEx](#) function.

```
GraphicOutEx(40, 40, "image.ric", variables);
```

## 12.178 ex\_HSFlags.nxc

This is an example of how to use the [HSFlags](#) function.

```
byte x = HSFlags();
```



## 12.179 ex\_HSInputBufferInPtr.nxc

This is an example of how to use the [HSInputBufferInPtr](#) function.

```
byte x = HSInputBufferInPtr();
```

## 12.180 ex\_HSInputBufferOutPtr.nxc

This is an example of how to use the [HSInputBufferOutPtr](#) function.

```
byte x = HSInputBufferOutPtr();
```

## 12.181 ex\_HSMMode.nxc

This is an example of how to use the [HSMMode](#) function.

```
int mode = HSMMode();
```

## 12.182 ex\_HSOutputBufferInPtr.nxc

This is an example of how to use the [HSOutputBufferInPtr](#) function.

```
byte x = HSOutputBufferInPtr();
```

## 12.183 ex\_HSOutputBufferOutPtr.nxc

This is an example of how to use the [HSOutputBufferOutPtr](#) function.

```
byte x = HSOutputBufferOutPtr();
```

## 12.184 ex\_HSSpeed.nxc

This is an example of how to use the [HSSpeed](#) function.

```
byte x = HSSpeed();
```

## 12.185 ex\_HSState.nxc

This is an example of how to use the [HSState](#) function.

```
byte x = HSState();
```

## 12.186 ex\_HTIRTrain.nxc

This is an example of how to use the [HTIRTrain](#) function.

```
HTIRTrain(S1, TRAIN_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```



## 12.187 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);
```

## 12.188 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);
```

## 12.189 ex\_HTTPFRawOutput.nxc

This is an example of how to use the [HTTPFRawOutput](#) function.

```
HTTPFRawOutput(S1, 0x0a, 0x01, 0x02);
```

## 12.190 ex\_HTPFRepeat.nxc

This is an example of how to use the [HTPFRepeat](#) function.

```
HTPFRepeat(S1, 5, 100);
```

## 12.191 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);
```

## 12.192 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);
```

## 12.193 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);
```

## 12.194 ex\_HTPFTrain.nxc

This is an example of how to use the [HTPFTrain](#) function.

```
HTPFTrain(S1, PF_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```



## 12.195 ex\_HTRCXAddToDatalog.nxc

This is an example of how to use the [HTRCXAddToDatalog](#) function.

```
HTRCXAddToDatalog(RCX_InputValueSrc, S1);
```

## 12.196 ex\_HTRCXBatteryLevel.nxc

This is an example of how to use the [HTRCXBatteryLevel](#) function.

```
x = HTRCXBatteryLevel();
```

## 12.197 ex\_HTRCXCleaAllEvents.nxc

This is an example of how to use the [HTRCXCleaAllEvents](#) function.

```
HTRCXCleaAllEvents();
```

## 12.198 ex\_HTRCXCleaCounter.nxc

This is an example of how to use the [HTRCXCleaCounter](#) function.

```
HTRCXCleaCounter(0);
```

## 12.199 ex\_HTRCXCleArMsg.nxc

This is an example of how to use the [HTRCXCleArMsg](#) function.

```
HTRCXCleArMsg();
```

## 12.200 ex\_HTRCXCleatSensor.nxc

This is an example of how to use the [HTRCXCleatSensor](#) function.

```
HTRCXCleatSensor(S1);
```

## 12.201 ex\_HTRCXCleatSound.nxc

This is an example of how to use the [HTRCXCleatSound](#) function.

```
HTRCXCleatSound();
```

## 12.202 ex\_HTRCXCleartimer.nxc

This is an example of how to use the [HTRCXCleartimer](#) function.

```
HTRCXCleartimer(0);
```



## 12.203 ex\_HTRCXCreateDatalog.nxc

This is an example of how to use the [HTRCXCreateDatalog](#) function.

```
HTRCXCreateDatalog(50);
```

## 12.204 ex\_HTRCXDecCounter.nxc

This is an example of how to use the [HTRCXDecCounter](#) function.

```
HTRCXDecCounter(0);
```

## 12.205 ex\_HTRCXDeleteSub.nxc

This is an example of how to use the [HTRCXDeleteSub](#) function.

```
HTRCXDeleteSub(2);
```

## 12.206 ex\_HTRCXDeleteSubs.nxc

This is an example of how to use the [HTRCXDeleteSubs](#) function.

```
HTRCXDeleteSubs ( ) ;
```

## 12.207 ex\_HTRCXDeleteTask.nxc

This is an example of how to use the [HTRCXDeleteTask](#) function.

```
HTRCXDeleteTask(3);
```

## 12.208 ex\_HTRCXDeleteTasks.nxc

This is an example of how to use the [HTRCXDeleteTasks](#) function.

```
HTRCXDeleteTasks();
```

## 12.209 ex\_HTRCXDisableOutput.nxc

This is an example of how to use the [HTRCXDisableOutput](#) function.

```
HTRCXDisableOutput(RCX_OUT_A);
```

## 12.210 ex\_HTRCXEnableOutput.nxc

This is an example of how to use the [HTRCXEnableOutput](#) function.

```
HTRCXEnableOutput (RCX_OUT_A);
```



## 12.211 ex\_HTRCXEvent.nxc

This is an example of how to use the [HTRCXEvent](#) function.

```
HTRCXEvent(RCX_ConstantSrc, 2);
```

## 12.212 ex\_HTRCFloat.nxc

This is an example of how to use the [HTRCFloat](#) function.

```
HTRCFloat(RCX_OUT_A);
```

## 12.213 ex\_HTRCXFwd.nxc

This is an example of how to use the [HTRCXFwd](#) function.

```
HTRCXFwd(RCX_OUT_A);
```

## 12.214 ex\_HTRCXIncCounter.nxc

This is an example of how to use the [HTRCXIncCounter](#) function.

```
HTRCXIncCounter(0);
```

## 12.215 ex\_HTRCXInvertOutput.nxc

This is an example of how to use the [HTRCXInvertOutput](#) function.

```
HTRCXInvertOutput (RCX_OUT_A);
```

## 12.216 ex\_HTRCXMuteSound.nxc

This is an example of how to use the [HTRCXMuteSound](#) function.

```
HTRCXMuteSound();
```

## 12.217 ex\_HTRCXObvertOutput.nxc

This is an example of how to use the [HTRCXObvertOutput](#) function.

```
HTRCXObvertOutput(RCX_OUT_A);
```

## 12.218 ex\_HTRCXOff.nxc

This is an example of how to use the [HTRCXOff](#) function.

```
HTRCXOff(RCX_OUT_A);
```



## 12.219 ex\_HTRCXOn.nxc

This is an example of how to use the [HTRCXOn](#) function.

```
HTRCXOn (RCX_OUT_A) ;
```

## 12.220 ex\_HTRCXOnFor.nxc

This is an example of how to use the [HTRCXOnFor](#) function.

```
HTRCXOnFor(RCX_OUT_A, 100);
```

## 12.221 ex\_HTRCXOnFwd.nxc

This is an example of how to use the [HTRCXOnFwd](#) function.

```
HTRCXOnFwd(RCX_OUT_A);
```

## 12.222 ex\_HTRCXOnRev.nxc

This is an example of how to use the [HTRCXOnRev](#) function.

```
HTRCXOnRev(RCX_OUT_A);
```

## 12.223 ex\_HTRCXPBTurnOff.nxc

This is an example of how to use the [HTRCXPBTurnOff](#) function.

```
HTRCXPBTurnOff ( ) ;
```

## 12.224 ex\_HTRCXPing.nxc

This is an example of how to use the [HTRCXPing](#) function.

```
HTRCXPing();
```

## 12.225 ex\_HTRCXPlaySound.nxc

This is an example of how to use the [HTRCXPlaySound](#) function.

```
HTRCXPlaySound(RCX_SOUND_UP);
```

## 12.226 ex\_HTRCXPlayTone.nxc

This is an example of how to use the [HTRCXPlayTone](#) function.

```
HTRCXPlayTone(440, 100);
```



## 12.227 ex\_HTRCXPlayToneVar.nxc

This is an example of how to use the [HTRCXPlayToneVar](#) function.

```
HTRCXPlayToneVar(0, 50);
```

## 12.228 ex\_HTRCXPoll.nxc

This is an example of how to use the [HTRCXPoll](#) function.

```
x = HTRCXPoll(RCX_VariableSrc, 0);
```

## 12.229 ex\_HTRCXPollMemory.nxc

This is an example of how to use the [HTRCXPollMemory](#) function.

```
HTRCXPollMemory(0, 10);
```

## 12.230 ex\_HTRCXRemote.nxc

This is an example of how to use the [HTRCXRemote](#) function.

```
HTRCXRemote(RCX_RemotePlayASound);
```

## 12.231 ex\_HTRCXRev.nxc

This is an example of how to use the [HTRCXRev](#) function.

```
HTRCXRev(RCX_OUT_A);
```

## 12.232 ex\_HTRCXSelectDisplay.nxc

This is an example of how to use the [HTRCXSelectDisplay](#) function.

```
HTRCXSelectDisplay(RCX_VariableSrc, 2);
```

## 12.233 ex\_HTRCXSelectProgram.nxc

This is an example of how to use the [HTRCXSelectProgram](#) function.

```
HTRCXSelectProgram(3);
```

## 12.234 ex\_HTRCXSndSerial.nxc

This is an example of how to use the [HTRCXSndSerial](#) function.

```
HTRCXSndSerial(0, 10);
```



## 12.235 ex\_HTRCXSetDirection.nxc

This is an example of how to use the [HTRCXSetDirection](#) function.

```
HTRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

## 12.236 ex\_HTRCXSetEvent.nxc

This is an example of how to use the [HTRCXSetEvent](#) function.

```
HTRCXSetEvent(0, RCX_ConstantSrc, 5);
```

## 12.237 ex\_HTRCXSetGlobalDirection.nxc

This is an example of how to use the [HTRCXSetGlobalDirection](#) function.

```
HTRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

## 12.238 ex\_HTRCXSetGlobalOutput.nxc

This is an example of how to use the [HTRCXSetGlobalOutput](#) function.

```
HTRCXSetGlobalOutput (RCX_OUT_A, RCX_OUT_ON);
```

## 12.239 ex\_HTRCXSetIRLinkPort.nxc

This is an example of how to use the [HTRCXSetIRLinkPort](#) function.

```
SetSensorLowspeed(S1);
```

## 12.240 ex\_HTRCXSetMaxPower.nxc

This is an example of how to use the [HTRCXSetMaxPower](#) function.

```
HTRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

## 12.241 ex\_HTRCXSetMessage.nxc

This is an example of how to use the [HTRCXSetMessage](#) function.

```
HTRCXSetMessage(20);
```

## 12.242 ex\_HTRCXSetOutput.nxc

This is an example of how to use the [HTRCXSetOutput](#) function.

```
HTRCXSetOutput(RCX_OUT_A, RCX_OUT_ON);
```



## 12.243 ex\_HTRCXSetPower.nxc

This is an example of how to use the [HTRCXSetPower](#) function.

```
HTRCXSetPower(RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

## 12.244 ex\_HTRCXSetPriority.nxc

This is an example of how to use the [HTRCXSetPriority](#) function.

```
HTRCXSetPriority(2);
```

## 12.245 ex\_HTRCXSetSensorMode.nxc

This is an example of how to use the [HTRCXSetSensorMode](#) function.

```
HTRCXSetSensorMode (S1, SENSOR_MODE_BOOL);
```

## 12.246 ex\_HTRCXSetSensorType.nxc

This is an example of how to use the [HTRCXSetSensorType](#) function.

```
HTRCXSetSensorType (S1, SENSOR_TYPE_TOUCH);
```

## 12.247 ex\_HTRCXSetSleepTime.nxc

This is an example of how to use the [HTRCXSetSleepTime](#) function.

```
HTRCXSetSleepTime(4);
```

## 12.248 ex\_HTRCXSetTxPower.nxc

This is an example of how to use the [HTRCXSetTxPower](#) function.

```
HTRCXSetTxPower (0) ;
```

## 12.249 ex\_HTRCXSetWatch.nxc

This is an example of how to use the [HTRCXSetWatch](#) function.

```
HTRCXSetWatch(3, 30);
```

## 12.250 ex\_HTRCXStartTask.nxc

This is an example of how to use the [HTRCXStartTask](#) function.

```
HTRCXStartTask (2) ;
```



## 12.251 ex\_HTRCXStopAllTasks.nxc

This is an example of how to use the [HTRCXStopAllTasks](#) function.

```
HTRCXStopAllTasks();
```

## 12.252 ex\_HTRCXStopTask.nxc

This is an example of how to use the [HTRCXStopTask](#) function.

```
HTRCXStopTask (1) ;
```

## 12.253 ex\_HTRCXToggle.nxc

This is an example of how to use the [HTRCXToggle](#) function.

```
HTRCXToggle(RCX_OUT_A);
```

## 12.254 ex\_HTRCXUnmuteSound.nxc

This is an example of how to use the [HTRCXUnmuteSound](#) function.

```
HTRCXUnmuteSound();
```

## 12.255 ex\_HTScoutCalibrateSensor.nxc

This is an example of how to use the [HTScoutCalibrateSensor](#) function.

```
HTScoutCalibrateSensor();
```

## 12.256 ex\_HTScoutMuteSound.nxc

This is an example of how to use the [HTScoutMuteSound](#) function.

```
HTScoutMuteSound();
```

## 12.257 ex\_HTScoutSelectSounds.nxc

This is an example of how to use the [HTScoutSelectSounds](#) function.

```
HTScoutSelectSounds (0) ;
```

## 12.258 ex\_HTScoutSendVLL.nxc

This is an example of how to use the [HTScoutSendVLL](#) function.

```
HTScoutSendVLL(RCX_ConstantSrc, 0x30);
```



## 12.259 ex\_HTScoutSetEventFeedback.nxc

This is an example of how to use the [HTScoutSetEventFeedback](#) function.

```
HTScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

## 12.260 ex\_HTScoutSetLight.nxc

This is an example of how to use the [HTScoutSetLight](#) function.

```
HTScoutSetLight (SCOUT_LIGHT_ON);
```

## 12.261 ex\_HTScoutSetScoutMode.nxc

This is an example of how to use the [HTScoutSetScoutMode](#) function.

```
HTScoutSetScoutMode (SCOUT_MODE_POWER) ;
```

## 12.262 ex\_HTScoutSetSensorClickTime.nxc

This is an example of how to use the [HTScoutSetSensorClickTime](#) function.

```
HTScoutSetSensorClickTime(RCX_ConstantSrc, 200);
```

## 12.263 ex\_HTScoutSetSensorHysteresis.nxc

This is an example of how to use the [HTScoutSetSensorHysteresis](#) function.

```
HTScoutSetSensorHysteresis(RCX_ConstantSrc, 50);
```

## 12.264 ex\_HTScoutSetSensorLowerLimit.nxc

This is an example of how to use the [HTScoutSetSensorLowerLimit](#) function.

```
HTScoutSetSensorLowerLimit(RCX_VariableSrc, 0);
```

## 12.265 ex\_HTScoutSetSensorUpperLimit.nxc

This is an example of how to use the [HTScoutSetSensorUpperLimit](#) function.

```
HTScoutSetSensorUpperLimit (RCX_VariableSrc, 0);
```

## 12.266 ex\_HTScoutUnmuteSound.nxc

This is an example of how to use the [HTScoutUnmuteSound](#) function.

```
HTScoutUnmuteSound ();
```



## 12.267 ex\_I2CBytes.nxc

This is an example of how to use the [I2CBytes](#) function.

```
x = I2CBytes(S4, writebuf, cnt, readbuf);
```

## 12.268 ex\_I2CBytesReady.nxc

This is an example of how to use the [I2CBytesReady](#) function.

```
x = I2CBytesReady(S1);
```

## 12.269 ex\_I2CCheckStatus.nxc

This is an example of how to use the [I2CCheckStatus](#) function.

```
x = I2CCheckStatus (S1) ;
```

## 12.270 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));
        TextOut(0, LCD_LINE2, I2CDeviceId(S1));
        TextOut(0, LCD_LINE3, I2CVersion(S1));
    }
}
```

## 12.271 ex\_I2CDeviceIdEx.nxc

This is an example of how to use the [I2CDeviceIdEx](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    TextOut(0, LCD_LINE1, I2CVendorIdEx(S1, 0x10));
    TextOut(0, LCD_LINE2, I2CDeviceIdEx(S1, 0x10));
    TextOut(0, LCD_LINE3, I2CVersionEx(S1, 0x10));
    Wait(SEC_10);
}
```

## 12.272 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_REG_DEVICE_ID));
    TextOut(0, LCD_LINE2, I2CDeviceInfo(S1, I2C_REG_VENDOR_ID));
    TextOut(0, LCD_LINE3, I2CDeviceInfo(S1, I2C_REG_VERSION));
    Wait(SEC_10);
}
```

## 12.273 ex\_I2CDeviceInfoEx.nxc

This is an example of how to use the [I2CDeviceInfoEx](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    TextOut(0, LCD_LINE1, I2CDeviceInfoEx(S1, 0x10, I2C_REG_DEVICE_ID));
    TextOut(0, LCD_LINE2, I2CDeviceInfoEx(S1, 0x10, I2C_REG_VENDOR_ID));
    TextOut(0, LCD_LINE3, I2CDeviceInfoEx(S1, 0x10, I2C_REG_VERSION));
    Wait(SEC_10);
}
```

## 12.274 ex\_I2CRead.nxc

This is an example of how to use the [I2CRead](#) function.

```
x = I2CRead(S1, 1, outbuffer);
```



## 12.275 ex\_I2CSendCommand.nxc

This is an example of how to use the [I2CSendCommand](#) function.

```
long result = I2CSendCommand(S1, HT_CMD_COLOR2_ACTIVE);
```

## 12.276 ex\_I2CSendCommandEx.nxc

This is an example of how to use the [I2CSendCommandEx](#) function.

```
long result = I2CSendCommandEx(S1, 0x10, HT_CMD_COLOR2_ACTIVE);
```

## 12.277 ex\_I2CStatus.nxc

This is an example of how to use the [I2CStatus](#) function.

```
x = I2CStatus(S1, nRead);
```

## 12.278 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));
        TextOut (0, LCD_LINE2, I2CDeviceId(S1));
        TextOut (0, LCD_LINE3, I2CVersion(S1));
    }
}
```

## 12.279 ex\_I2CVendorIdEx.nxc

This is an example of how to use the [I2CVendorIdEx](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    while (true) {
        TextOut(0, LCD_LINE1, I2CVendorIdEx(S1, 0x10));
        TextOut(0, LCD_LINE2, I2CDeviceIdEx(S1, 0x10));
        TextOut(0, LCD_LINE3, I2CVersionEx(S1, 0x10));
    }
}
```

## 12.280 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));
        TextOut(0, LCD_LINE2, I2CDeviceId(S1));
        TextOut(0, LCD_LINE3, I2CVersion(S1));
    }
}
```

## 12.281 ex\_I2CVersionEx.nxc

This is an example of how to use the [I2CVersionEx](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    while (true) {
        TextOut(0, LCD_LINE1, I2CVendorIdEx(S1, 0x10));
        TextOut(0, LCD_LINE2, I2CDeviceIdEx(S1, 0x10));
        TextOut(0, LCD_LINE3, I2CVersionEx(S1, 0x10));
    }
}
```

## 12.282 ex\_i2cwrite.nxc

This is an example of how to use the [I2CWrite](#) function.

```
x = I2CWrite(S1, 1, inbuffer);
```



## 12.283 ex\_isalnum.nxc

This is an example of how to use the [isalnum](#) function.

```
i = isalnum(x);
```

## 12.284 ex\_isalpha.nxc

This is an example of how to use the [isalpha](#) function.

```
i = isalpha(x);
```

## 12.285 ex\_isctrl.nxc

This is an example of how to use the [isctrl](#) function.

```
i = isctrl(x);
```

## 12.286 ex\_isdigit.nxc

This is an example of how to use the [isdigit](#) function.

```
i = isdigit(x);
```

## 12.287 ex\_isgraph.nxc

This is an example of how to use the [isgraph](#) function.

```
i = isgraph(x);
```

## 12.288 ex\_islower.nxc

This is an example of how to use the [islower](#) function.

```
i = islower(x);
```

## 12.289 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);
}
```

## 12.290 ex\_isprint.nxc

This is an example of how to use the [isprint](#) function.

```
i = isprint(x);
```



## 12.291 ex\_ispunct.nxc

This is an example of how to use the [ispunct](#) function.

```
i = ispunct(x);
```

## 12.292 ex\_isspace.nxc

This is an example of how to use the [isspace](#) function.

```
i = isspace(x);
```

## 12.293 ex\_isupper.nxc

This is an example of how to use the [isupper](#) function.

```
i = isupper(x);
```

## 12.294 ex\_isxdigit.nxc

This is an example of how to use the [isxdigit](#) function.

```
i = isxdigit(x);
```

## 12.295 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);
}
```

## 12.296 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);
}
```

## 12.297 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);
}
```

## 12.298 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);
    }
}
```



## 12.299 ex\_log.nxc

This is an example of how to use the [log](#) function.

```
y = log(x);
```

## 12.300 ex\_log10.nxc

This is an example of how to use the [log10](#) function.

```
y = log10(x);
```

## 12.301 ex\_LongAbort.nxc

This is an example of how to use the [LongAbort](#) function.

```
x = LongAbort();
```

## 12.302 ex\_LowspeedBytesReady.nxc

This is an example of how to use the [LowspeedBytesReady](#) function.

```
x = LowspeedBytesReady(S1);
```

## 12.303 ex\_LowspeedCheckStatus.nxc

This is an example of how to use the [LowspeedCheckStatus](#) function.

```
x = LowspeedCheckStatus(S1);
```

## 12.304 ex\_LowspeedRead.nxc

This is an example of how to use the [LowspeedRead](#) function.

```
x = LowspeedRead(S1, 1, outbuffer);
```

## 12.305 ex\_lowspeedstatus.nxc

This is an example of how to use the [LowSpeedStatus](#) function.

```
x = LowSpeedStatus (S1, nRead);
```

## 12.306 ex\_LowspeedWrite.nxc

This is an example of how to use the [LowspeedWrite](#) function.

```
x = LowspeedWrite(S1, 1, inbuffer);
```



## 12.307 ex\_LSChannelState.nxc

This is an example of how to use the [LSChannelState](#) function.

```
x = LSChannelState(S1);
```

## 12.308 ex\_LSErrType.nxc

This is an example of how to use the [LSErrType](#) function.

```
x = LSErrType(S1);
```

## 12.309 ex\_LSInputBufferBytesToRx.nxc

This is an example of how to use the [LSInputBufferBytesToRx](#) function.

```
x = LSInputBufferBytesToRx(S1);
```

## 12.310 ex\_LSInputBufferInPtr.nxc

This is an example of how to use the [LSInputBufferInPtr](#) function.

```
x = LSInputBufferInPtr(S1);
```

## 12.311 ex\_LSInputBufferOutPtr.nxc

This is an example of how to use the [LSInputBufferOutPtr](#) function.

```
x = LSInputBufferOutPtr(S1);
```

## 12.312 ex\_LSMODE.nxc

This is an example of how to use the [LSMode](#) function.

```
x = LSMODE(S1);
```

## 12.313 ex\_LSNorRestartOnRead.nxc

This is an example of how to use the [LSNorRestartOnRead](#) function.

```
byte val = LSNorRestartOnRead();
```

## 12.314 ex\_LSOutputBufferBytesToRx.nxc

This is an example of how to use the [LSOutputBufferBytesToRx](#) function.

```
x = LSOutputBufferBytesToRx(S1);
```



## 12.315 ex\_LSOutputBufferInPtr.nxc

This is an example of how to use the [LSOutputBufferInPtr](#) function.

```
x = LSOutputBufferInPtr(S1);
```

## 12.316 ex\_LSOutputBufferOutPtr.nxc

This is an example of how to use the [LSOutputBufferOutPtr](#) function.

```
x = LSOutputBufferOutPtr(S1);
```

## 12.317 ex\_LSSpeed.nxc

This is an example of how to use the [LSSpeed](#) function.

```
x = LSSpeed();
```

## 12.318 ex\_LSSState.nxc

This is an example of how to use the [LSState](#) function.

```
x = LSState();
```

## 12.319 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);
}
```

## 12.320 ex\_memcpy.nxc

This is an example of how to use the [memcpy](#) function.

```
memcpy(myArray, anotherArray, 1);
```

## 12.321 ex\_memmove.nxc

This is an example of how to use the [memmove](#) function.

```
memmove(myArray, anotherArray, 1);
```

## 12.322 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);
}
```



## 12.323 ex\_motoractualspeed.nxc

This is an example of how to use the [MotorActualSpeed](#) function.

```
x = MotorActualSpeed (OUT_A) ;
```

## 12.324 ex\_motorblocktachocount.nxc

This is an example of how to use the [MotorBlockTachoCount](#) function.

```
x = MotorBlockTachoCount (OUT_A);
```

## 12.325 ex\_motormode.nxc

This is an example of how to use the [MotorMode](#) function.

```
x = MotorMode (OUT_A);
```

## 12.326 ex\_motoroverload.nxc

This is an example of how to use the [MotorOverload](#) function.

```
x = MotorOverload(OUT_A);
```

## 12.327 ex\_motorpower.nxc

This is an example of how to use the [MotorPower](#) function.

```
x = MotorPower (OUT_A);
```

## 12.328 ex\_motorpwnfreq.nxc

This is an example of how to use the [MotorPwnFreq](#) function.

```
x = MotorPwnFreq();
```

## 12.329 ex\_motorregdvalue.nxc

This is an example of how to use the [MotorRegDValue](#) function.

```
x = MotorRegDValue (OUT_A);
```

## 12.330 ex\_motorregivalue.nxc

This is an example of how to use the [MotorRegIValue](#) function.

```
x = MotorRegIValue (OUT_A);
```



## 12.331 ex\_motorregpvalue.nxc

This is an example of how to use the [MotorRegPValue](#) function.

```
x = MotorRegPValue (OUT_A);
```

## 12.332 ex\_motorregulation.nxc

This is an example of how to use the [MotorRegulation](#) function.

```
x = MotorRegulation(OUT_A);
```

## 12.333 ex\_motorrotationcount.nxc

This is an example of how to use the [MotorRotationCount](#) function.

```
x = MotorRotationCount (OUT_A);
```

## 12.334 ex\_motorrunstate.nxc

This is an example of how to use the [MotorRunState](#) function.

```
x = MotorRunState (OUT_A);
```

## 12.335 ex\_motortachocount.nxc

This is an example of how to use the [MotorTachoCount](#) function.

```
x = MotorTachoCount (OUT_A);
```

## 12.336 ex\_motortacholimit.nxc

This is an example of how to use the [MotorTachoLimit](#) function.

```
x = MotorTachoLimit (OUT_A);
```

## 12.337 ex\_motorturnratio.nxc

This is an example of how to use the [MotorTurnRatio](#) function.

```
x = MotorTurnRatio(OUT_A);
```

## 12.338 ex\_MSADPAOff.nxc

This is an example of how to use the [MSADPAOff](#) function.

```
char result = MSADPAOff(S1);
```



## 12.339 ex\_MSADPAOffEx.nxc

This is an example of how to use the [MSADPAOffEx](#) function.

```
char result = MSADPAOffEx(S1, 0x02);
```

## 12.340 ex\_MSADPAOn.nxc

This is an example of how to use the [MSADPAOn](#) function.

```
char result = MSADPAOn(S1);
```

## 12.341 ex\_MSADPAnEx.nxc

This is an example of how to use the [MSADPAnEx](#) function.

```
char result = MSADPAnEx(S1, 0x02);
```

## 12.342 ex\_MSDeenergize.nxc

This is an example of how to use the [MSDeenergize](#) function.

```
char result = MSDeenergize(S1);
```

## 12.343 ex\_MSDeenergizeEx.nxc

This is an example of how to use the [MSDeenergizeEx](#) function.

```
char result = MSDeenergizeEx(S1, 0x02);
```

## 12.344 ex\_MSEnergize.nxc

This is an example of how to use the [MSEnergize](#) function.

```
char result = MSEnergize(S1);
```

## 12.345 ex\_MSEnergizeEx.nxc

This is an example of how to use the [MSEnergizeEx](#) function.

```
char result = MSEnergizeEx(S1, 0x02);
```

## 12.346 ex\_MSIRTrain.nxc

This is an example of how to use the [MSIRTrain](#) function.

```
char result = MSIRTrain(S1, TRAIN_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```



## 12.347 ex\_MSIRTrainEx.nxc

This is an example of how to use the [MSIRTrainEx](#) function.

```
char result = MSIRTrainEx(S1, 0x02, TRAIN_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

## 12.348 ex\_MSPFComboDirect.nxc

This is an example of how to use the [MSPFComboDirect](#) function.

```
char result = MSPFComboDirect(S1, PF_CHANNEL_1, PF_CMD_STOP, PF_CMD_FWD);
```

## 12.349 ex\_MSPFComboDirectEx.nxc

This is an example of how to use the [MSPFComboDirectEx](#) function.

```
char result = MSPFComboDirectEx(S1, 0x02, PF_CHANNEL_1, PF_CMD_STOP, PF_CMD_FWD);
```

## 12.350 ex\_MSPFComboPWM.nxc

This is an example of how to use the [MSPFComboPWM](#) function.

```
char result = MSPFComboPWM(S1, PF_CHANNEL_1, PF_PWM_REV4, PF_PWM_FWD5);
```

## 12.351 ex\_MSPFComboPWMEEx.nxc

This is an example of how to use the [MSPFComboPWMEEx](#) function.

```
char result = MSPFComboPWMEEx(S1, 0x02, PF_CHANNEL_1, PF_PWM_REV4, PF_PWM_FWD5);
```

## 12.352 ex\_MSPFRawOutput.nxc

This is an example of how to use the [MSPFRawOutput](#) function.

```
char result = MSPFRawOutput(S1, 0x0a, 0x01, 0x02);
```

## 12.353 ex\_MSPFRawOutputEx.nxc

This is an example of how to use the [MSPFRawOutputEx](#) function.

```
char result = MSPFRawOutputEx(S1, 0x02, 0x0a, 0x01, 0x02);
```

## 12.354 ex\_MSPFRepeat.nxc

This is an example of how to use the [MSPFRepeat](#) function.

```
char result = MSPFRepeat(S1, 5, 100);
```



## 12.355 ex\_MSPFRepeatEx.nxc

This is an example of how to use the [MSPFRepeatEx](#) function.

```
char result = MSPFRepeatEx(S1, 0x02, 5, 100);
```

## 12.356 ex\_MSPFSingleOutputCST.nxc

This is an example of how to use the [MSPFSingleOutputCST](#) function.

```
char result = MSPFSingleOutputCST(S1, PF_CHANNEL_1, PF_OUT_A, PF_CST_SET1_SET2);
```

## 12.357 ex\_MSPFSingleOutputCSTEx.nxc

This is an example of how to use the [MSPFSingleOutputCSTEx](#) function.

```
char result = MSPFSingleOutputCSTEx(S1, 0x02, PF_CHANNEL_1, PF_OUT_A,  
    PF_CST_SET1_SET2);
```

## 12.358 ex\_MSPFSingleOutputPWM.nxc

This is an example of how to use the [MSPFSingleOutputPWM](#) function.

```
char result = MSPFSingleOutputPWM(S1, PF_CHANNEL_1, PF_OUT_A, PF_PWM_FWD5);
```

## 12.359 ex\_MSPFSingleOutputPWMEEx.nxc

This is an example of how to use the [MSPFSingleOutputPWMEEx](#) function.

```
char result = MSPFSingleOutputPWMEEx(S1, 0x02, PF_CHANNEL_1, PF_OUT_A,  
    PF_PWM_FWD5);
```

## 12.360 ex\_MSPFSinglePin.nxc

This is an example of how to use the [MSPFSinglePin](#) function.

```
char result = MSPFSinglePin(S1, PF_CHANNEL_1, PF_OUT_A, PF_PIN_C1, PF_FUNC_SET, t  
    rue);
```

## 12.361 ex\_MSPFSinglePinEx.nxc

This is an example of how to use the [MSPFSinglePinEx](#) function.

```
char result = MSPFSinglePinEx(S1, 0x02, PF_CHANNEL_1, PF_OUT_A, PF_PIN_C1,  
    PF_FUNC_SET, true);
```

## 12.362 ex\_MSPFTrain.nxc

This is an example of how to use the [MSPFTrain](#) function.

```
char result = MSPFTrain(S1, PF_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```



## 12.363 ex\_MSPFTrainEx.nxc

This is an example of how to use the [MSPFTrainEx](#) function.

```
char result = MSPFTrainEx(S1, 0x02, PF_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

## 12.364 ex\_MSRCXAbsVar.nxc

This is an example of how to use the [MSRCXAbsVar](#) function.

```
MSRCXAbsVar(0, RCX_VariableSrc, 0);
```

## 12.365 ex\_MSRCXAddToDatalog.nxc

This is an example of how to use the [MSRCXAddToDatalog](#) function.

```
MSRCXAddToDatalog(RCX_InputValueSrc, S1);
```

## 12.366 ex\_MSRCXAndVar.nxc

This is an example of how to use the [MSRCXAndVar](#) function.

```
MSRCXAndVar(0, RCX_ConstantSrc, 0x7f);
```

## 12.367 ex\_MSRCXBatteryLevel.nxc

This is an example of how to use the [MSRCXBatteryLevel](#) function.

```
x = MSRCXBatteryLevel();
```

## 12.368 ex\_MSRCXBoot.nxc

This is an example of how to use the [MSRCXBoot](#) function.

```
MSRCXBoot ( ) ;
```

## 12.369 ex\_MSRCXCalibrateEvent.nxc

This is an example of how to use the [MSRCXCalibrateEvent](#) function.

```
MSRCXCalibrateEvent(0, 200, 500, 50);
```

## 12.370 ex\_MSRCXClearAllEvents.nxc

This is an example of how to use the [MSRCXClearAllEvents](#) function.

```
MSRCXClearAllEvents();
```



## 12.371 ex\_MSRCXClearCounter.nxc

This is an example of how to use the [MSRCXClearCounter](#) function.

```
MSRCXClearCounter(0);
```

## 12.372 ex\_MSRCXClearMsg.nxc

This is an example of how to use the [MSRCXClearMsg](#) function.

```
MSRCXClearMsg();
```

## 12.373 ex\_MSRCXClearSensor.nxc

This is an example of how to use the [MSRCXClearSensor](#) function.

```
MSRCXClearSensor(S1);
```

## 12.374 ex\_MSRCXClearSound.nxc

This is an example of how to use the [MSRCXClearSound](#) function.

```
MSRCXClearSound();
```

## 12.375 ex\_MSRCXClearTimer.nxc

This is an example of how to use the [MSRCXClearTimer](#) function.

```
MSRCXClearTimer(0);
```

## 12.376 ex\_MSRCXCreateDatalog.nxc

This is an example of how to use the [MSRCXCreateDatalog](#) function.

```
MSRCXCreateDatalog(50);
```

## 12.377 ex\_MSRCXDecCounter.nxc

This is an example of how to use the [MSRCXDecCounter](#) function.

```
MSRCXDecCounter (0) ;
```

## 12.378 ex\_MSRCXDeleteSub.nxc

This is an example of how to use the [MSRCXDeleteSub](#) function.

```
MSRCXDeleteSub(2);
```



## 12.379 ex\_MSRCXDeleteSubs.nxc

This is an example of how to use the [MSRCXDeleteSubs](#) function.

```
MSRCXDeleteSubs ( ) ;
```

## 12.380 ex\_MSRCXDeleteTask.nxc

This is an example of how to use the [MSRCXDeleteTask](#) function.

```
MSRCXDeleteTask(3);
```

## 12.381 ex\_MSRCXDeleteTasks.nxc

This is an example of how to use the [MSRCXDeleteTasks](#) function.

```
MSRCXDeleteTasks();
```

## 12.382 ex\_MSRCXDisableOutput.nxc

This is an example of how to use the [MSRCXDisableOutput](#) function.

```
MSRCXDisableOutput (RCX_OUT_A) ;
```

## 12.383 ex\_MSRCXDivVar.nxc

This is an example of how to use the [MSRCXDivVar](#) function.

```
MSRCXDivVar(0, RCX_ConstantSrc, 2);
```

## 12.384 ex\_MSRCXEnableOutput.nxc

This is an example of how to use the [MSRCXEnableOutput](#) function.

```
MSRCXEnableOutput (RCX_OUT_A);
```

## 12.385 ex\_MSRCXEvent.nxc

This is an example of how to use the [MSRCXEvent](#) function.

```
MSRCXEvent(RCX_ConstantSrc, 2);
```

## 12.386 ex\_MSRCXFloat.nxc

This is an example of how to use the [MSRCXFloat](#) function.

```
MSRCXFloat(RCX_OUT_A);
```



## 12.387 ex\_MSRCxFwd.nxc

This is an example of how to use the [MSRCxFwd](#) function.

```
MSRCxFwd(RCX_OUT_A);
```

## 12.388 ex\_MSRCXIncCounter.nxc

This is an example of how to use the [MSRCXIncCounter](#) function.

```
MSRCXIncCounter(0);
```

## 12.389 ex\_MSRCXInvertOutput.nxc

This is an example of how to use the [MSRCXInvertOutput](#) function.

```
MSRCXInvertOutput (RCX_OUT_A);
```

## 12.390 ex\_MSRCXMulVar.nxc

This is an example of how to use the [MSRCXMulVar](#) function.

```
MSRCXMulVar(0, RCX_VariableSrc, 4);
```

## 12.391 ex\_MSRCXMuteSound.nxc

This is an example of how to use the [MSRCXMuteSound](#) function.

```
MSRCXMuteSound();
```

## 12.392 ex\_MSRCXObvertOutput.nxc

This is an example of how to use the [MSRCXObvertOutput](#) function.

```
MSRCXObvertOutput (RCX_OUT_A);
```

## 12.393 ex\_MSRCXOff.nxc

This is an example of how to use the [MSRCXOff](#) function.

```
MSRCXOff(RCX_OUT_A);
```

## 12.394 ex\_MSRCXOn.nxc

This is an example of how to use the [MSRCXOn](#) function.

```
MSRCXOn (RCX_OUT_A) ;
```



## 12.395 ex\_MSRCXOnFor.nxc

This is an example of how to use the [MSRCXOnFor](#) function.

```
MSRCXOnFor(RCX_OUT_A, 100);
```

## 12.396 ex\_MSRCXOnFwd.nxc

This is an example of how to use the [MSRCXOnFwd](#) function.

```
MSRCXOnFwd(RCX_OUT_A);
```

## 12.397 ex\_MSRCXOnRev.nxc

This is an example of how to use the [MSRCXOnRev](#) function.

```
MSRCXOnRev(RCX_OUT_A);
```

## 12.398 ex\_MSRCXOrVar.nxc

This is an example of how to use the [MSRCXOrVar](#) function.

```
MSRCXOrVar(0, RCX_ConstantSrc, 0xCC);
```

## 12.399 ex\_MSRCXPBTurnOff.nxc

This is an example of how to use the [MSRCXPBTurnOff](#) function.

```
MSRCXPBTurnOff ( ) ;
```

## 12.400 ex\_MSRCXPing.nxc

This is an example of how to use the [MSRCXPing](#) function.

```
MSRCXPing();
```

## 12.401 ex\_MSRCXPlaySound.nxc

This is an example of how to use the [MSRCXPlaySound](#) function.

```
MSRCXPlaySound(RCX_SOUND_UP);
```

## 12.402 ex\_MSRCXPlayTone.nxc

This is an example of how to use the [MSRCXPlayTone](#) function.

```
MSRCXPlayTone(440, 100);
```



## 12.403 ex\_MSRCXPlayToneVar.nxc

This is an example of how to use the [MSRCXPlayToneVar](#) function.

```
MSRCXPlayToneVar(0, 50);
```

## 12.404 ex\_MSRCXPoll.nxc

This is an example of how to use the [MSRCXPoll](#) function.

```
x = MSRCXPoll(RCX_VariableSrc, 0);
```

## 12.405 ex\_MSRCXPollMemory.nxc

This is an example of how to use the [MSRCXPollMemory](#) function.

```
MSRCXPollMemory(0, 10);
```

## 12.406 ex\_MSRCXRemote.nxc

This is an example of how to use the [MSRCXRemote](#) function.

```
MSRCXRemote(RCX_RemotePlayASound);
```

## 12.407 ex\_MSRCXReset.nxc

This is an example of how to use the [MSRCXReset](#) function.

```
MSRCXReset ( ) ;
```

## 12.408 ex\_MSRCXRev.nxc

This is an example of how to use the [MSRCXRev](#) function.

```
MSRCXRev(RCX_OUT_A);
```

## 12.409 ex\_MSRCXSelectDisplay.nxc

This is an example of how to use the [MSRCXSelectDisplay](#) function.

```
MSRCXSelectDisplay(RCX_VariableSrc, 2);
```

## 12.410 ex\_MSRCXSelectProgram.nxc

This is an example of how to use the [MSRCXSelectProgram](#) function.

```
MSRCXSelectProgram(3);
```



## 12.411 ex\_MSRCXSendSerial.nxc

This is an example of how to use the [MSRCXSendSerial](#) function.

```
MSRCXSendSerial(0, 10);
```

## 12.412 ex\_MSRCXSet.nxc

This is an example of how to use the [MSRCXSet](#) function.

```
MSRCXSet(RCX_VariableSrc, 0, RCX_RandomSrc, 10000);
```

## 12.413 ex\_MSRCXSetDirection.nxc

This is an example of how to use the [MSRCXSetDirection](#) function.

```
MSRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

## 12.414 ex\_MSRCXSetEvent.nxc

This is an example of how to use the [MSRCXSetEvent](#) function.

```
MSRCXSetEvent(0, RCX_ConstantSrc, 5);
```

## 12.415 ex\_MSRCXSetGlobalDirection.nxc

This is an example of how to use the [MSRCXSetGlobalDirection](#) function.

```
MSRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

## 12.416 ex\_MSRCXSetGlobalOutput.nxc

This is an example of how to use the [MSRCXSetGlobalOutput](#) function.

```
MSRCXSetGlobalOutput (RCX_OUT_A, RCX_OUT_ON);
```

## 12.417 ex\_MSRCXSetMaxPower.nxc

This is an example of how to use the [MSRCXSetMaxPower](#) function.

```
MSRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

## 12.418 ex\_MSRCXSetMessage.nxc

This is an example of how to use the [MSRCXSetMessage](#) function.

```
MSRCXSetMessage(20);
```



## 12.419 ex\_MSRCXSetNRLinkPort.nxc

This is an example of how to use the [MSRCXSetNRLinkPort](#) function.

```
MSRCXSetNRLinkPort (S1) ;
```

## 12.420 ex\_MSRCXSetNRLinkPortEx.nxc

This is an example of how to use the [MSRCXSetNRLinkPortEx](#) function.

```
MSRCXSetNRLinkPortEx(S1, 0x02);
```

## 12.421 ex\_MSRCXSetOutput.nxc

This is an example of how to use the [MSRCXSetOutput](#) function.

```
MSRCXSetOutput (RCX_OUT_A, RCX_OUT_ON);
```

## 12.422 ex\_MSRCXSetPower.nxc

This is an example of how to use the [MSRCXSetPower](#) function.

```
MSRCXSetPower(RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

## 12.423 `ex_MSRCXSetPriority.nxc`

This is an example of how to use the [MSRCXSetPriority](#) function.

```
MSRCXSetPriority(2);
```

## 12.424 ex\_MSRCXSetSensorMode.nxc

This is an example of how to use the [MSRCXSetSensorMode](#) function.

```
MSRCXSetSensorMode(S1, SENSOR_MODE_BOOL);
```

## 12.425 ex\_MSRCXSetSensorType.nxc

This is an example of how to use the [MSRCXSetSensorType](#) function.

```
MSRCXSetSensorType (S1, SENSOR_TYPE_TOUCH);
```

## 12.426 ex\_MSRCXSetSleepTime.nxc

This is an example of how to use the [MSRCXSetSleepTime](#) function.

```
MSRCXSetSleepTime(4);
```



## 12.427 ex\_MSRCXSetTxPower.nxc

This is an example of how to use the [MSRCXSetTxPower](#) function.

```
MSRCXSetTxPower (0) ;
```

## 12.428 ex\_MSRCXSetUserDisplay.nxc

This is an example of how to use the [MSRCXSetUserDisplay](#) function.

```
MSRCXSetUserDisplay(RCX_VariableSrc, 0, 2);
```

## 12.429 ex\_MSRCXSetVar.nxc

This is an example of how to use the [MSRCXSetVar](#) function.

```
MSRCXSetVar(0, RCX_VariableSrc, 1);
```

## 12.430 ex\_MSRCXSetWatch.nxc

This is an example of how to use the [MSRCXSetWatch](#) function.

```
MSRCXSetWatch(3, 30);
```

## 12.431 ex\_MSRCXSgnVar.nxc

This is an example of how to use the [MSRCXSgnVar](#) function.

```
MSRCXSgnVar(0, RCX_VariableSrc, 0);
```

## 12.432 ex\_MSRCXStartTask.nxc

This is an example of how to use the [MSRCXStartTask](#) function.

```
MSRCXStartTask (2) ;
```

## 12.433 ex\_MSRCXStopAllTasks.nxc

This is an example of how to use the [MSRCXStopAllTasks](#) function.

```
MSRCXStopAllTasks();
```

## 12.434 ex\_MSRCXStopTask.nxc

This is an example of how to use the [MSRCXStopTask](#) function.

```
MSRCXStopTask (1);
```



## 12.435 ex\_MSRCXSubVar.nxc

This is an example of how to use the [MSRCXSubVar](#) function.

```
MSRCXSubVar(0, RCX_RandomSrc, 10);
```

## 12.436 ex\_MSRCXSumVar.nxc

This is an example of how to use the [MSRCXSumVar](#) function.

```
MSRCXSumVar(0, RCX_InputValueSrc, S1);
```

## 12.437 ex\_MSRCXToggle.nxc

This is an example of how to use the [MSRCXToggle](#) function.

```
MSRCXToggle(RCX_OUT_A);
```

## 12.438 ex\_MSRCXUnlock.nxc

This is an example of how to use the [MSRCXUnlock](#) function.

```
MSRCXUnlock();
```

## 12.439 ex\_MSRCXUnmuteSound.nxc

This is an example of how to use the [MSRCXUnmuteSound](#) function.

```
MSRCXUnmuteSound();
```

## 12.440 ex\_MSReadValue.nxc

This is an example of how to use the [MSReadValue](#) function.

```
byte value = MSReadValue(S1, I2C_REG_CMD, 1);
```

## 12.441 ex\_MSReadValueEx.nxc

This is an example of how to use the [MSReadValueEx](#) function.

```
byte value = MSReadValueEx(S1, 0x02, I2C_REG_CMD, 1);
```

## 12.442 ex\_MSScoutCalibrateSensor.nxc

This is an example of how to use the [MSScoutCalibrateSensor](#) function.

```
MSScoutCalibrateSensor();
```



## 12.443 ex\_MSScoutMuteSound.nxc

This is an example of how to use the [MSScoutMuteSound](#) function.

```
MSScoutMuteSound();
```

## 12.444 ex\_MSScoutSelectSounds.nxc

This is an example of how to use the [MSScoutSelectSounds](#) function.

```
MSScoutSelectSounds (0) ;
```

## 12.445 ex\_MSScoutSendVLL.nxc

This is an example of how to use the [MSScoutSendVLL](#) function.

```
MSScoutSendVLL(RCX_ConstantSrc, 0x30);
```

## 12.446 ex\_MSScoutSetCounterLimit.nxc

This is an example of how to use the [MSScoutSetCounterLimit](#) function.

```
MSScoutSetCounterLimit(0, RCX_ConstantSrc, 2000);
```

## 12.447 ex\_MSScoutSetEventFeedback.nxc

This is an example of how to use the [MSScoutSetEventFeedback](#) function.

```
MSScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

## 12.448 ex\_MSScoutSetLight.nxc

This is an example of how to use the [MSScoutSetLight](#) function.

```
MSScoutSetLight (SCOUT_LIGMS_ON);
```

## 12.449 ex\_MSScoutSetScoutMode.nxc

This is an example of how to use the [MSScoutSetScoutMode](#) function.

```
MSScoutSetScoutMode (SCOUT_MODE_POWER) ;
```

## 12.450 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);
```



## 12.451 ex\_MSScoutSetSensorClickTime.nxc

This is an example of how to use the [MSScoutSetSensorClickTime](#) function.

```
MSScoutSetSensorClickTime(RCX_ConstantSrc, 200);
```

## 12.452 ex\_MSScoutSetSensorHysteresis.nxc

This is an example of how to use the [MSScoutSetSensorHysteresis](#) function.

```
MSScoutSetSensorHysteresis(RCX_ConstantSrc, 50);
```

## 12.453 ex\_MSScoutSetSensorLowerLimit.nxc

This is an example of how to use the [MSScoutSetSensorLowerLimit](#) function.

```
MSScoutSetSensorLowerLimit(RCX_VariableSrc, 0);
```

## 12.454 ex\_MSScoutSetSensorUpperLimit.nxc

This is an example of how to use the [MSScoutSetSensorUpperLimit](#) function.

```
MSScoutSetSensorUpperLimit(RCX_VariableSrc, 0);
```

## 12.455 ex\_MSScoutSetTimerLimit.nxc

This is an example of how to use the [MSScoutSetTimerLimit](#) function.

```
MSScoutSetTimerLimit(0, RCX_ConstantSrc, 10000);
```

## 12.456 ex\_MSScoutUnmuteSound.nxc

This is an example of how to use the [MSScoutUnmuteSound](#) function.

```
MSScoutUnmuteSound ();
```

## 12.457 ex\_muldiv32.nxc

This is an example of how to use the [muldiv32](#) function.

```
y = muldiv32(a, b, c);
```

## 12.458 ex\_NRLink2400.nxc

This is an example of how to use the [NRLink2400](#) function.

```
char result = NRLink2400(S1);
```



## 12.459 ex\_NRLink2400Ex.nxc

This is an example of how to use the [NRLink2400Ex](#) function.

```
char result = NRLink2400Ex(S1, 0x02);
```

## 12.460 ex\_NRLink4800.nxc

This is an example of how to use the [NRLink4800](#) function.

```
char result = NRLink4800(S1);
```

## 12.461 ex\_NRLink4800Ex.nxc

This is an example of how to use the [NRLink4800Ex](#) function.

```
char result = NRLink4800Ex(S1, 0x02);
```

## 12.462 ex\_NRLinkFlush.nxc

This is an example of how to use the [NRLinkFlush](#) function.

```
char result = NRLinkFlush(S1);
```

## 12.463 ex\_NRLinkFlushEx.nxc

This is an example of how to use the [NRLinkFlushEx](#) function.

```
char result = NRLinkFlushEx(S1, 0x02);
```

## 12.464 ex\_NRLinkIRLong.nxc

This is an example of how to use the [NRLinkIRLong](#) function.

```
char result = NRLinkIRLong(S1);
```

## 12.465 ex\_NRLinkIRLongEx.nxc

This is an example of how to use the [NRLinkIRLongEx](#) function.

```
char result = NRLinkIRLongEx(S1, 0x02);
```

## 12.466 ex\_NRLinkIRShort.nxc

This is an example of how to use the [NRLinkIRShort](#) function.

```
char result = NRLinkIRShort(S1);
```



## 12.467 ex\_NRLinkIRShortEx.nxc

This is an example of how to use the [NRLinkIRShortEx](#) function.

```
char result = NRLinkIRShortEx(S1, 0x02);
```

## 12.468 ex\_NRLinkSetPF.nxc

This is an example of how to use the [NRLinkSetPF](#) function.

```
char result = NRLinkSetPF(S1);
```

## 12.469 ex\_NRLinkSetPFEEx.nxc

This is an example of how to use the [NRLinkSetPFEEx](#) function.

```
char result = NRLinkSetPFEEx(S1, 0x02);
```

## 12.470 ex\_NRLinkSetRCX.nxc

This is an example of how to use the [NRLinkSetRCX](#) function.

```
char result = NRLinkSetRCX(S1);
```

## 12.471 ex\_NRLinkSetRCXEx.nxc

This is an example of how to use the [NRLinkSetRCXEx](#) function.

```
char result = NRLinkSetRCXEx(S1, 0x02);
```

## 12.472 ex\_NRLinkSetTrain.nxc

This is an example of how to use the [NRLinkSetTrain](#) function.

```
char result = NRLinkSetTrain(S1);
```

## 12.473 ex\_NRLinkSetTrainEx.nxc

This is an example of how to use the [NRLinkSetTrainEx](#) function.

```
char result = NRLinkSetTrainEx(S1, 0x02);
```

## 12.474 ex\_NRLinkStatus.nxc

This is an example of how to use the [NRLinkStatus](#) function.

```
byte result = NRLinkStatus(S1);
```



## 12.475 ex\_NRLinkStatusEx.nxc

This is an example of how to use the [NRLinkStatusEx](#) function.

```
byte result = NRLinkStatusEx(S1, 0x02);
```

## 12.476 ex\_NRLinkTxRaw.nxc

This is an example of how to use the [NRLinkTxRaw](#) function.

```
byte result = NRLinkTxRaw(S1);
```

## 12.477 ex\_NRLinkTxRawEx.nxc

This is an example of how to use the [NRLinkTxRawEx](#) function.

```
byte result = NRLinkTxRawEx(S1, 0x02);
```

## 12.478 ex\_NumOut.nxc

This is an example of how to use the [NumOut](#) function.

```
NumOut (0, LCD_LINE1, x);
```

## 12.479 ex\_NumToStr.nxc

This is an example of how to use the [NumToStr](#) function.

```
msg = NumToStr(-2); // returns "-2" in a string
```

## 12.480 ex\_off.nxc

This is an example of how to use the [Off](#) function.

```
Off(OUT_A); // turn off output A
```

## 12.481 ex\_offex.nxc

This is an example of how to use the [OffEx](#) function.

```
OffEx(OUT_A, RESET_NONE); // turn off output A
```

## 12.482 ex\_OnBrickProgramPointer.nxc

This is an example of how to use the [OnBrickProgramPointer](#) function.

```
x = OnBrickProgramPointer();
```



## 12.483 ex\_onfwd.nxc

This is an example of how to use the [OnFwd](#) function.

```
OnFwd(OUT_A, 75);
```

## 12.484 ex\_onfwdex.nxc

This is an example of how to use the [OnFwdEx](#) function.

```
OnFwdEx (OUT_A, 75, RESET_NONE);
```

## 12.485 ex\_onfwdreg.nxc

This is an example of how to use the [OnFwdReg](#) function.

```
OnFwdReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

## 12.486 ex\_onfwdregex.nxc

This is an example of how to use the [OnFwdRegEx](#) function.

```
OnFwdRegEx (OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

## 12.487 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);
```

## 12.488 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
```

## 12.489 ex\_onfwdsync.nxc

This is an example of how to use the [OnFwdSync](#) function.

```
OnFwdSync(OUT_AB, 75, -100); // spin right
```

## 12.490 ex\_onfwdsyncex.nxc

This is an example of how to use the [OnFwdSyncEx](#) function.

```
OnFwdSyncEx(OUT_AB, 75, 0, RESET_NONE);
```



## 12.491 ex\_onfwdsyncexpid.nxc

This is an example of how to use the OnFwdSyncExPID function.

```
OnFwdSyncExPID(OUT_AB, 75, 0, RESET_NONE, 30, 50, 90);
```

## 12.492 ex\_onfwdsyncpid.nxc

This is an example of how to use the OnFwdSyncPID function.

```
OnFwdSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin right
```

## 12.493 ex\_onrev.nxc

This is an example of how to use the [OnRev](#) function.

```
OnRev (OUT_A, 75);
```

## 12.494 ex\_onrevex.nxc

This is an example of how to use the [OnRevEx](#) function.

```
OnRevEx(OUT_A, 75, RESET_NONE);
```

## 12.495 ex\_onrevreg.nxc

This is an example of how to use the [OnRevReg](#) function.

```
OnRevReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

## 12.496 ex\_onrevregex.nxc

This is an example of how to use the [OnRevRegEx](#) function.

```
OnRevRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

## 12.497 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);
```

## 12.498 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
```



## 12.499 ex\_onrevsync.nxc

This is an example of how to use the [OnRevSync](#) function.

```
OnRevSync(OUT_AB, 75, -100); // spin left
```

## 12.500 ex\_onrevsyncex.nxc

This is an example of how to use the [OnRevSyncEx](#) function.

```
OnRevSyncEx(OUT_AB, 75, -100, RESET_NONE); // spin left
```

## 12.501 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
```

## 12.502 ex\_onrevsyncpid.nxc

This is an example of how to use the OnRevSyncPID function.

```
OnRevSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin left
```

## 12.503 ex\_OpenFileAppend.nxc

This is an example of how to use the [OpenFileAppend](#) function.

```
result = OpenFileAppend("data.txt", fsize, handle);
```

## 12.504 ex\_OpenFileRead.nxc

This is an example of how to use the [OpenFileRead](#) function.

```
result = OpenFileRead("data.txt", fsize, handle);
```

## 12.505 ex\_OpenFileReadLinear.nxc

This is an example of how to use the [OpenFileReadLinear](#) function.

```
result = OpenFileReadLinear("data.txt", fsize, handle);
```

## 12.506 ex\_PlayFile.nxc

This is an example of how to use the [PlayFile](#) function.

```
PlayFile("startup.rso");
```



## 12.507 ex\_PlayFileEx.nxc

This is an example of how to use the [PlayFileEx](#) function.

```
PlayFileEx("startup.rso", 3, true);
```

## 12.508 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);
}
```

## 12.509 ex\_PlayTone.nxc

This is an example of how to use the [PlayTone](#) function.

```
PlayTone(440, 500);    // Play 'A' for one half second
```

## 12.510 ex\_PlayToneEx.nxc

This is an example of how to use the [PlayToneEx](#) function.

```
PlayToneEx(440, 500, 2, false);
```

## 12.511 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);
}
```

## 12.512 ex\_PointOut.nxc

This is an example of how to use the [PointOut](#) function.

```
PointOut(40, 40);
```

## 12.513 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);
}
```

## 12.514 ex\_pow.nxc

This is an example of how to use the `pow` function.

```
y = pow(x, 3);
```



## 12.515 ex\_powerdown.nxc

This is an example of how to use the [PowerDown](#) functions.

```
PowerDown();
```

## 12.516 ex\_Precedes.nxc

This is an example of how to use the [Precedes](#) statement.

```
Precedes(moving, drawing, playing);
```

## 12.517 ex\_printf.nxc

This is an example of how to use the [printf](#) function.

```
printf("value = %d", value);
```

## 12.518 ex\_PSPNxAnalog.nxc

This is an example of how to use the [PSPNxAnalog](#) function.

```
char result = PSPNxAnalog(S1);
```

## 12.519 ex\_PSPNxAnalogEx.nxc

This is an example of how to use the [PSPNxAnalogEx](#) function.

```
char result = PSPNxAnalogEx(S1, 0x02);
```

## 12.520 ex\_PSPNxDigital.nxc

This is an example of how to use the [PSPNxDigital](#) function.

```
char result = PSPNxDigital(S1);
```

## 12.521 ex\_PSPNxDigitalEx.nxc

This is an example of how to use the [PSPNxDigitalEx](#) function.

```
char result = PSPNxDigitalEx(S1, 0x02);
```

## 12.522 ex\_putc.nxc

This is an example of how to use the [putc](#) function.

```
putc(ch, handle);
```



## 12.523 ex\_rand.nxc

This is an example of how to use the [rand](#) function.

```
unsigned int x = rand(); // 0..RAND_MAX-1
```

## 12.524 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
```

## 12.525 ex\_Read.nxc

This is an example of how to use the [Read](#) function.

```
result = Read(handle, value);
```

## 12.526 ex\_ReadButtonEx.nxc

This is an example of how to use the [ReadButtonEx](#) function.

```
ReadButtonEx(BTN1, true, pressed, count);
```

## 12.527 ex\_ReadBytes.nxc

This is an example of how to use the [ReadBytes](#) function.

```
result = ReadBytes(handle, len, buffer);
```

## 12.528 ex\_readi2cregister.nxc

This is an example of how to use the [ReadI2CRegister](#) function.

```
char result = ReadI2CRegister(S1, I2C_REG_CMD, out);
```

## 12.529 ex\_ReadLn.nxc

This is an example of how to use the [ReadLn](#) function.

```
result = ReadLn(handle, value);
```

## 12.530 ex\_ReadNRLinkBytes.nxc

This is an example of how to use the [ReadNRLinkBytes](#) function.

```
bool result = ReadNRLinkBytes(S1, data);
```



## 12.531 ex\_ReadNRLinkBytesEx.nxc

This is an example of how to use the [ReadNRLinkBytesEx](#) function.

```
bool result = ReadNRLinkBytesEx(S1, 0x02, data);
```

## 12.532 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);
```

## 12.533 ex\_ReadSensorColorRaw.nxc

This is an example of how to use the [ReadSensorColorRaw](#) function.

```
unsigned int rawData[];  
int result = ReadSensorColorRaw(S1, rawData);
```

## 12.534 ex\_ReadSensorHTAccel.nxc

This is an example of how to use the [ReadSensorHTAccel](#) function.

```
bVal = ReadSensorHTAccel(S1, x, y, z);
```

## 12.535 ex\_ReadSensorHTColor.nxc

This is an example of how to use the [ReadSensorHTColor](#) function.

```
bVal = ReadSensorHTColor(S1, c, r, g, b);
```

## 12.536 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);
```

## 12.537 ex\_ReadSensorHTIRReceiver.nxc

This is an example of how to use the [ReadSensorHTIRReceiver](#) function.

```
char pfdata[];  
bool result = ReadSensorHTIRReceiver(S1, pfdata);
```

## 12.538 ex\_ReadSensorHTIRReceiverEx.nxc

This is an example of how to use the [ReadSensorHTIRReceiverEx](#) function.

```
char pfchar;  
bool result = ReadSensorHTIRReceiverEx(S1, HT_CH1_A, pfchar);
```



## 12.539 ex\_ReadSensorHTIRSeeker.nxc

This is an example of how to use the [ReadSensorHTIRSeeker](#) function.

```
bVal = ReadSensorHTIRSeeker(port, dir, s1, s3, s5, s7, s9);
```

## 12.540 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);
```

## 12.541 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);
```

## 12.542 ex\_ReadSensorHTNormalizedColor.nxc

This is an example of how to use the [ReadSensorHTNormalizedColor](#) function.

```
bVal = ReadSensorHTNormalizedColor(S1, c, r, g, b);
```

## 12.543 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);
```

## 12.544 ex\_ReadSensorHTRawColor.nxc

This is an example of how to use the [ReadSensorHTRawColor](#) function.

```
bVal = ReadSensorHTRawColor(S1, r, g, b);
```

## 12.545 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);
```

## 12.546 ex\_ReadSensorHTTouchMultiplexer.nxc

This is an example of how to use the [ReadSensorHTTouchMultiplexer](#) function.

```
byte t1, t2, t3, t4;  
ReadSensorHTTouchMultiplexer(S1, t1, t2, t3, t4);
```



## 12.547 ex\_ReadSensorMSAccel.nxc

This is an example of how to use the [ReadSensorMSAccel](#) function.

```
int x, y, z;  
bool result = ReadSensorMSAccel(S1, x, y, z);
```

## 12.548 ex\_ReadSensorMSAccelEx.nxc

This is an example of how to use the [ReadSensorMSAccelEx](#) function.

```
int x, y, z;  
bool result = ReadSensorMSAccelEx(S1, 0x02, x, y, z);
```

## 12.549 ex\_ReadSensorMSPlayStation.nxc

This is an example of how to use the [ReadSensorMSPlayStation](#) function.

```
byte btnset1, btnset2, xleft, yleft, xright, yright;  
bool result = ReadSensorMSPlayStation(S1, btnset1, btnset2, xleft, yleft, xright,  
    yright);
```

## 12.550 ex\_ReadSensorMSPlayStationEx.nxc

This is an example of how to use the [ReadSensorMSPlayStationEx](#) function.

```
byte btnset1, btnset2, xleft, yleft, xright, yright;  
bool result = ReadSensorMSPlayStationEx(S1, 0x02, btnset1, btnset2, xleft, yleft,  
    xright, yright);
```

## 12.551 ex\_ReadSensorMSRTClock.nxc

This is an example of how to use the [ReadSensorMSRTClock](#) function.

```
ReadSensorMSRTClock(S1, ss, mm, hh, dow, dd, mon, yy);
```

## 12.552 ex\_ReadSensorMSTilt.nxc

This is an example of how to use the [ReadSensorMSTilt](#) function.

```
byte x, y, z;  
bool result = ReadSensorMSTilt(S1, x, y, z);
```

## 12.553 ex\_ReadSensorMSTiltEx.nxc

This is an example of how to use the [ReadSensorMSTiltEx](#) function.

```
byte x, y, z;  
bool result = ReadSensorMSTiltEx(S1, 0x02, x, y, z);
```

## 12.554 ex\_readsensorsex.nxc

This is an example of how to use the [ReadSensorUSEx](#) function.

```
byte values[];  
char result = ReadSensorUSEx(S1, values);
```



## 12.555 ex\_rebootinfirmwaremode.nxc

This is an example of how to use the [RebootInFirmwareMode](#) functions.

```
RebootInFirmwareMode();
```

## 12.556 ex\_receivemessage.nxc

This is an example of how to use the [ReceiveMessage](#) function.

```
x = RecieveMessage(MAILBOX1, true, buffer);
```

## 12.557 ex\_ReceiveRemoteBool.nxc

This is an example of how to use the [ReceiveRemoteBool](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, bvalue);
```

## 12.558 ex\_ReceiveRemoteMessageEx.nxc

This is an example of how to use the [ReceiveRemoteMessageEx](#) function.

```
x = ReceiveRemoteMessageEx(MAILBOX1, true, strval, val, bval);
```

## 12.559 ex\_ReceiveRemoteNumber.nxc

This is an example of how to use the [ReceiveRemoteNumber](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, value);
```

## 12.560 ex\_ReceiveRemoteString.nxc

This is an example of how to use the [ReceiveRemoteString](#) function.

```
x = ReceiveRemoteString(queue, true, strval);
```

## 12.561 ex\_RechargeableBattery.nxc

This is an example of how to use the [RechargeableBattery](#) function.

```
x = RechargeableBattery();
```

## 12.562 ex\_RectOut.nxc

This is an example of how to use the [RectOut](#) function.

```
RectOut (40, 40, 30, 10);
```



## 12.563 ex\_reladdr.nxc

This is an example of how to use the [reladdr](#) function.

```
task main()
{
    char x[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    unsigned long ptr = reladdr(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);
}
```

## 12.564 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
```

## 12.565 ex\_RemoteKeepAlive.nxc

This is an example of how to use the [RemoteKeepAlive](#) function.

```
x = RemoteKeepAlive(1);
```

## 12.566 ex\_RemoteMessageRead.nxc

This is an example of how to use the [RemoteMessageRead](#) function.

```
x = RemoteMessageRead(1, 5);
```

## 12.567 ex\_RemoteMessageWrite.nxc

This is an example of how to use the [RemoteMessageWrite](#) function.

```
x = RemoteMessageWrite(1, 5, "test");
```

## 12.568 ex\_RemotePlaySoundFile.nxc

This is an example of how to use the [RemotePlaySoundFile](#) function.

```
x = RemotePlaySoundFile(1, "click.rso", false);
```

## 12.569 ex\_RemotePlayTone.nxc

This is an example of how to use the [RemotePlayTone](#) function.

```
x = RemotePlayTone(1, 440, 1000);
```

## 12.570 ex\_RemoteResetMotorPosition.nxc

This is an example of how to use the [RemoteResetMotorPosition](#) function.

```
x = RemoteResetMotorPosition(1, OUT_A, true);
```



## 12.571 ex\_RemoteResetScaledValue.nxc

This is an example of how to use the [RemoteResetScaledValue](#) function.

```
x = RemoteResetScaledValue(1, S1);
```

## 12.572 ex\_RemoteSetInputMode.nxc

This is an example of how to use the [RemoteSetInputMode](#) function.

```
x = RemoteSetInputMode(1, S1, SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW);
```

## 12.573 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);
```

## 12.574 ex\_RemoteStartProgram.nxc

This is an example of how to use the [RemoteStartProgram](#) function.

```
x = RemoteStartProgram(1, "myprog.rxe");
```

## 12.575 ex\_RemoteStopProgram.nxc

This is an example of how to use the [RemoteStopProgram](#) function.

```
x = RemoteStopProgram(1);
```

## 12.576 ex\_RemoteStopSound.nxc

This is an example of how to use the [RemoteStopSound](#) function.

```
x = RemoteStopSound(1);
```

## 12.577 ex\_remove.nxc

This is an example of how to use the [remove](#) function.

```
result = remove("data.txt");
```

## 12.578 ex\_rename.nxc

This is an example of how to use the [rename](#) function.

```
result = rename("data.txt", "mydata.txt");
```



## 12.579 ex\_RenameFile.nxc

This is an example of how to use the [RenameFile](#) function.

```
result = RenameFile("data.txt", "mydata.txt");
```

## 12.580 ex\_resetalltachocounts.nxc

This is an example of how to use the [ResetAllTachoCounts](#) function.

```
ResetAllTachoCounts (OUT_AB) ;
```

## 12.581 ex\_resetblocktachocount.nxc

This is an example of how to use the [ResetBlockTachoCount](#) function.

```
ResetBlockTachoCount (OUT_AB) ;
```

## 12.582 ex\_resetrotationcount.nxc

This is an example of how to use the [ResetRotationCount](#) function.

```
ResetRotationCount (OUT_AB);
```

## 12.583 ex\_ResetScreen.nxc

This is an example of how to use the [ResetScreen](#) function.

```
ResetScreen();
```

## 12.584 ex\_ResetSensor.nxc

This is an example of how to use the [ResetSensor](#) function.

```
ResetSensor(x); // x = S1
```

## 12.585 ex\_ResetSleepTimer.nxc

This is an example of how to use the [ResetSleepTimer](#) function.

```
ResetSleepTimer();
```

## 12.586 ex\_resettachocount.nxc

This is an example of how to use the [ResetTachoCount](#) function.

```
ResetTachoCount (OUT_AB) ;
```



## 12.587 ex\_ResizeFile.nxc

This is an example of how to use the [ResizeFile](#) function.

```
result = ResizeFile("data.txt", 2048);
```

## 12.588 ex\_ResolveHandle.nxc

This is an example of how to use the [ResolveHandle](#) function.

```
result = ResolveHandle("data.txt", handle, bCanWrite);
```

## 12.589 ex\_rewind.nxc

This is an example of how to use the [rewind](#) function.

```
rewind(handle);
```

## 12.590 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);
}
```

## 12.591 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
```

## 12.592 ex\_rotatemotorex.nxc

This is an example of how to use the [RotateMotorEx](#) function.

```
RotateMotorEx(OUT_AB, 75, 360, 50, true, true);
```

## 12.593 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);
```

## 12.594 ex\_rotatemotorpid.nxc

This is an example of how to use the [RotateMotorPID](#) function.

```
RotateMotorPID(OUT_A, 75, 45, 20, 40, 100);
```



## 12.595 ex\_RS485Control.nxc

This is an example of how to use the [RS485Control](#) function.

```
char result = RS485Control(HS_CTRL_UART, HS_BAUD_9600, HS_MODE_8N1);
```

## 12.596 ex\_RS485DataAvailable.nxc

This is an example of how to use the [RS485DataAvailable](#) function.

```
bool result = RS485DataAvailable();
```

## 12.597 ex\_RS485Exit.nxc

This is an example of how to use the [RS485Exit](#) function.

```
char result = RS485Exit();
```

## 12.598 ex\_RS485Init.nxc

This is an example of how to use the [RS485Init](#) function.

```
char result = RS485Init();
```

## 12.599 ex\_RS485Read.nxc

This is an example of how to use the [RS485Read](#) function.

```
byte buffer[];  
char result = RS485Read(buffer);
```

## 12.600 ex\_RS485SendingData.nxc

This is an example of how to use the [RS485SendingData](#) function.

```
bool result = RS485SendingData()
```

## 12.601 ex\_RS485Status.nxc

This is an example of how to use the [RS485Status](#) function.

```
bool sendingData, dataAvail;  
RS485Status(sendingData, dataAvail);
```

## 12.602 ex\_RS485Uart.nxc

This is an example of how to use the [RS485Uart](#) function.

```
char result = RS485Uart(HS_BAUD_115200, HS_MODE_8N1);
```



## 12.603 ex\_RS485Write.nxc

This is an example of how to use the [RS485Write](#) function.

```
byte buffer[] = {1, 2, 3, 4, 5};  
char result = RS485Write(buffer);
```

## 12.604 ex\_RunNRLinkMacro.nxc

This is an example of how to use the [RunNRLinkMacro](#) function.

```
char result = RunNRLinkMacro(S1, macro);
```

## 12.605 ex\_RunNRLinkMacroEx.nxc

This is an example of how to use the [RunNRLinkMacroEx](#) function.

```
char result = RunNRLinkMacroEx(S1, 0x02, macro);
```

## 12.606 ex\_sendmessage.nxc

This is an example of how to use the [SendMessage](#) function.

```
x = SendMessage(MAILBOX1, data);
```

## 12.607 ex\_sendremotebool.nxc

This is an example of how to use the [SendRemoteBool](#) function.

```
x = SendRemoteBool(1, MAILBOX1, false);
```

## 12.608 ex\_SendRemoteNumber.nxc

This is an example of how to use the [SendRemoteNumber](#) function.

```
x = SendRemoteNumber(1, MAILBOX1, 123);
```

## 12.609 ex\_SendRemoteString.nxc

This is an example of how to use the [SendRemoteString](#) function.

```
x = SendRemoteString(1, MAILBOX1, "hello world");
```

## 12.610 ex\_SendResponseBool.nxc

This is an example of how to use the [SendResponseBool](#) function.

```
x = SendResponseBool(MAILBOX1, false);
```



## 12.611 ex\_SendResponseNumber.nxc

This is an example of how to use the [SendResponseNumber](#) function.

```
x = SendResponseNumber (MAILBOX1, 123);
```

## 12.612 ex\_SendResponseString.nxc

This is an example of how to use the [SendResponseString](#) function.

```
x = SendResponseString(MAILBOX1, "hello world");
```

## 12.613 ex\_SendRS485Bool.nxc

This is an example of how to use the [SendRS485Bool](#) function.

```
bool bval = true;
char result = SendRS485Bool(bval);
```

## 12.614 ex\_SendRS485Number.nxc

This is an example of how to use the [SendRS485Number](#) function.

```
long val = 12345;  
char result = SendRS485Number(val);
```

## 12.615 ex\_SendRS485String.nxc

This is an example of how to use the [SendRS485String](#) function.

```
string msg = "123";  
char result = SendRS485String(msg);
```

## 12.616 ex\_Sensor.nxc

This is an example of how to use the [Sensor](#) function.

```
x = Sensor(S1); // read sensor 1
```

## 12.617 ex\_SensorBoolean.nxc

This is an example of how to use the [SensorBoolean](#) function.

```
x = SensorBoolean(S1);
```

## 12.618 ex\_SensorDigiPinsDirection.nxc

This is an example of how to use the [SensorDigiPinsDirection](#) function.

```
x = SensorDigiPinsDirection(S1);
```



## 12.619 ex\_SensorDigiPinsOutputLevel.nxc

This is an example of how to use the [SensorDigiPinsOutputLevel](#) function.

```
x = SensorDigiPinsOutputLevel (S1);
```

## 12.620 ex\_SensorDigiPinsStatus.nxc

This is an example of how to use the [SensorDigiPinsStatus](#) function.

```
x = SensorDigiPinsStatus(S1);
```

## 12.621 ex\_SensorHTColorNum.nxc

This is an example of how to use the [SensorHTColorNum](#) function.

```
x = SensorHTColorNum(S1);
```

## 12.622 ex\_SensorHTCompass.nxc

This is an example of how to use the [SensorHTCompass](#) function.

```
x = SensorHTCompass(S1);
```

## 12.623 ex\_SensorHTEOPD.nxc

This is an example of how to use the [SensorHTEOPD](#) function.

```
int val = SensorHTEOPD(S1);
```

## 12.624 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);
}
```

## 12.625 ex\_SensorHTIRSeeker2ACDir.nxc

This is an example of how to use the [SensorHTIRSeeker2ACDir](#) function.

```
int val = SensorHTIRSeeker2ACDir(S1);
```

## 12.626 ex\_SensorHTIRSeeker2Addr.nxc

This is an example of how to use the [SensorHTIRSeeker2Addr](#) function.

```
int val = SensorHTIRSeeker2Addr(S1, HTIR2_REG_DCAVG);
```



## 12.627 ex\_SensorHTIRSeeker2DCDir.nxc

This is an example of how to use the [SensorHTIRSeeker2DCDir](#) function.

```
int val = SensorHTIRSeeker2DCDir(S1);
```

## 12.628 ex\_SensorHTIRSeekerDir.nxc

This is an example of how to use the [SensorHTIRSeekerDir](#) function.

```
x = SensorHTIRSeekerDir(S1);
```

## 12.629 ex\_SensorInvalid.nxc

This is an example of how to use the [SensorInvalid](#) function.

```
x = SensorInvalid(S1);
```

## 12.630 ex\_SensorMode.nxc

This is an example of how to use the [SensorMode](#) function.

```
x = SensorMode(S1);
```

## 12.631 ex\_SensorMSCompass.nxc

This is an example of how to use the [SensorMSCompass](#) function.

```
x = SensorMSCompass (S1);
```

## 12.632 ex\_SensorMSCompassEx.nxc

This is an example of how to use the [SensorMSCompassEx](#) function.

```
int val = SensorMSCompassEx(S1, 0x02);
```

## 12.633 ex\_SensorMSDROD.nxc

This is an example of how to use the [SensorMSDROD](#) function.

```
x = SensorMSDROD(S1);
```

## 12.634 ex\_SensorMSPressure.nxc

This is an example of how to use the [SensorMSPressure](#) function.

```
int val = SensorMSPressure(S1);
```



## 12.635 ex\_SensorMSPressureRaw.nxc

This is an example of how to use the [SensorMSPressureRaw](#) function.

```
int val = SensorMSPressureRaw(S1);
```

## 12.636 ex\_SensorNormalized.nxc

This is an example of how to use the [SensorNormalized](#) function.

```
x = SensorNormalized(S1);
```

## 12.637 ex\_SensorRaw.nxc

This is an example of how to use the [SensorRaw](#) function.

```
x = SensorRaw(S1);
```

## 12.638 ex\_SensorScaled.nxc

This is an example of how to use the [SensorScaled](#) function.

```
x = SensorScaled(S1);
```

## 12.639 ex\_SensorType.nxc

This is an example of how to use the [SensorType](#) function.

```
x = SensorType(S1);
```

## 12.640 ex\_sensorus.nxc

This is an example of how to use the [SensorUS](#) function.

```
x = SensorUS(S4); // read sensor 4
```

## 12.641 ex\_SensorValue.nxc

This is an example of how to use the [SensorValue](#) function.

```
unsigned int val = SensorValue(S1);
```

## 12.642 ex\_SensorValueBool.nxc

This is an example of how to use the [SensorValueBool](#) function.

```
bool val = SensorValueBool(S1);
```



## 12.643 ex\_SensorValueRaw.nxc

This is an example of how to use the [SensorValueRaw](#) function.

```
unsigned int val = SensorValueRaw(S1);
```

## 12.644 ex\_SetAbortFlag.nxc

This is an example of how to use the [SetAbortFlag](#) function.

```
SetAbortFlag(BTNSTATE_LONG_PRESSED_EV);
```

## 12.645 ex\_SetBatteryState.nxc

This is an example of how to use the [SetBatteryState](#) function.

```
SetBatteryState(4);
```

## 12.646 ex\_SetBluetoothState.nxc

This is an example of how to use the [SetBluetoothState](#) function.

```
SetBluetoothState (UI_BT_STATE_OFF) ;
```

## 12.647 ex\_SetBTInputBuffer.nxc

This is an example of how to use the [SetBTInputBuffer](#) function.

```
SetBTInputBuffer(0, 10, buffer);
```

## 12.648 ex\_SetBTInputBufferInPtr.nxc

This is an example of how to use the [SetBTInputBufferInPtr](#) function.

```
SetBTInputBufferInPtr(0);
```

## 12.649 ex\_SetBTInputBufferOutPtr.nxc

This is an example of how to use the [SetBTInputBufferOutPtr](#) function.

```
SetBTInputBufferOutPtr(0);
```

## 12.650 ex\_SetBTOutputBuffer.nxc

This is an example of how to use the [SetBTOutputBuffer](#) function.

```
SetBTOutputBuffer(0, 10, buffer);
```



## 12.651 ex\_SetBTOutputBufferInPtr.nxc

This is an example of how to use the [SetBTOutputBufferInPtr](#) function.

```
SetBTOutputBufferInPtr(0);
```

## 12.652 ex\_SetBTOutputBufferOutPtr.nxc

This is an example of how to use the [SetBTOutputBufferOutPtr](#) function.

```
SetBTOutputBufferOutPtr(0);
```

## 12.653 ex\_SetButtonLongPressCount.nxc

This is an example of how to use the [SetButtonLongPressCount](#) function.

```
SetButtonLongPressCount (BTN1, value);
```

## 12.654 ex\_SetButtonLongReleaseCount.nxc

This is an example of how to use the [SetButtonLongReleaseCount](#) function.

```
SetButtonLongReleaseCount (BTN1, value);
```

## 12.655 ex\_SetButtonPressCount.nxc

This is an example of how to use the [SetButtonPressCount](#) function.

```
SetButtonPressCount (BTN1, value);
```

## 12.656 ex\_SetButtonReleaseCount.nxc

This is an example of how to use the [SetButtonReleaseCount](#) function.

```
SetButtonReleaseCount (BTN1, value);
```

## 12.657 ex\_SetButtonShortReleaseCount.nxc

This is an example of how to use the [SetButtonShortReleaseCount](#) function.

```
SetButtonShortReleaseCount (BTN1, value);
```

## 12.658 ex\_SetButtonState.nxc

This is an example of how to use the [SetButtonState](#) function.

```
SetButtonState (BTN1, BTNSTATE_PRESSED_EV);
```



## 12.659 ex\_SetCommandFlags.nxc

This is an example of how to use the [SetCommandFlags](#) function.

```
SetCommandFlags (UI_FLAGS_REDRAW_STATUS) ;
```

## 12.660 ex\_SetCustomSensorActiveStatus.nxc

This is an example of how to use the [SetCustomSensorActiveStatus](#) function.

```
SetCustomSensorActiveStatus(S1, true);
```

## 12.661 ex\_SetCustomSensorPercentFullScale.nxc

This is an example of how to use the [SetCustomSensorPercentFullScale](#) function.

```
SetCustomSensorPercentFullScale(S1, 100);
```

## 12.662 ex\_SetCustomSensorZeroOffset.nxc

This is an example of how to use the [SetCustomSensorZeroOffset](#) function.

```
SetCustomSensorZeroOffset (S1, 12);
```

## 12.663 ex\_SetDisplayContrast.nxc

This is an example of how to use the [SetDisplayContrast](#) function.

```
SetDisplayContrast (DISPLAY_CONTRAST_DEFAULT) ;
```

## 12.664 ex\_SetDisplayDisplay.nxc

This is an example of how to use the [SetDisplayDisplay](#) function.

```
SetDisplayDisplay(x);
```

## 12.665 ex\_SetDisplayEraseMask.nxc

This is an example of how to use the [SetDisplayEraseMask](#) function.

```
SetDisplayEraseMask (x) ;
```

## 12.666 ex\_SetDisplayFlags.nxc

This is an example of how to use the [SetDisplayFlags](#) function.

```
SetDisplayFlags(x);
```



## 12.667 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
    0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,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,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x08,0x08,0x0
    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
    ,0x66,0x00,0x26,0x49,0x49,0x32,0x00,0x01,0x01,0x7F,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,0x10,0x28,0x44,0x00,0x00,0x00,0x00,0x00,0x7F,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;
    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);
}
```

## 12.668 ex\_SetDisplayNormal.nxc

This is an example of how to use the [SetDisplayNormal](#) function.

```
SetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

## 12.669 ex\_SetDisplayPopup.nxc

This is an example of how to use the [SetDisplayPopup](#) function.

```
SetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

## 12.670 ex\_SetDisplayTextLinesCenterFlags.nxc

This is an example of how to use the [SetDisplayTextLinesCenterFlags](#) function.

```
SetDisplayTextLinesCenterFlags(x);
```

## 12.671 ex\_SetDisplayUpdateMask.nxc

This is an example of how to use the [SetDisplayUpdateMask](#) function.

```
SetDisplayUpdateMask(x);
```

## 12.672 ex\_SetHSFlags.nxc

This is an example of how to use the [SetHSFlags](#) function.

```
SetHSFlags(0);
```

## 12.673 ex\_SetHSInputBuffer.nxc

This is an example of how to use the [SetHSInputBuffer](#) function.

```
SetHSInputBuffer(0, 10, buffer);
```

## 12.674 ex\_SetHSInputBufferInPtr.nxc

This is an example of how to use the [SetHSInputBufferInPtr](#) function.

```
SetHSInputBufferInPtr(0);
```



## 12.675 ex\_SetHSInputBufferOutPtr.nxc

This is an example of how to use the [SetHSInputBufferOutPtr](#) function.

```
SetHSInputBufferOutPtr(0);
```

## 12.676 ex\_SetHSMMode.nxc

This is an example of how to use the [SetHSMMode](#) function.

```
SetHSMMode (HS_MODE_8N1) ;
```

## 12.677 ex\_SetHSOutputBuffer.nxc

This is an example of how to use the [SetHSOutputBuffer](#) function.

```
SetHSOutputBuffer(0, 10, buffer);
```

## 12.678 ex\_SetHSOutputBufferInPtr.nxc

This is an example of how to use the [SetHSOutputBufferInPtr](#) function.

```
SetHSOutputBufferInPtr(0);
```

## 12.679 ex\_SetHSOutputBufferOutPtr.nxc

This is an example of how to use the [SetHSOutputBufferOutPtr](#) function.

```
SetHSOutputBufferOutPtr(0);
```

## 12.680 ex\_SetHSSpeed.nxc

This is an example of how to use the [SetHSSpeed](#) function.

```
SetHSSpeed(1);
```

## 12.681 ex\_SetHSState.nxc

This is an example of how to use the [SetHSState](#) function.

```
SetHSState(1);
```

## 12.682 ex\_SetHTColor2Mode.nxc

This is an example of how to use the [SetHTColor2Mode](#) function.

```
SetHTColor2Mode(S1, HT_CMD_COLOR2_ACTIVE);
```



## 12.683 ex\_SetHTIRSeeker2Mode.nxc

This is an example of how to use the [SetHTIRSeeker2Mode](#) function.

```
SetHTIRSeeker2Mode (S1, HTIR2_MODE_1200);
```

## 12.684 ex\_SetInput.nxc

This is an example of how to use the [SetInput](#) function.

```
SetInput (S1, Type, SENSOR_TYPE_SOUND_DB);
```

## 12.685 ex\_SetLongAbort.nxc

This is an example of how to use the [SetLongAbort](#) function.

```
SetLongAbort (true) ;
```

## 12.686 ex\_setmotorpwnfreq.nxc

This is an example of how to use the SetMotorPwnFreq function.

```
SetMotorPwnFreq(x);
```

## 12.687 ex\_SetOnBrickProgramPointer.nxc

This is an example of how to use the [SetOnBrickProgramPointer](#) function.

```
SetOnBrickProgramPointer(2);
```

## 12.688 ex\_setoutput.nxc

This is an example of how to use the [SetOutput](#) function.

```
SetOutput(OUT_AB, TachoLimit, 720); // set tacho limit
```

## 12.689 ex\_SetSensor.nxc

This is an example of how to use the [SetSensor](#) function.

```
SetSensor (S1, SENSOR_TOUCH);
```

## 12.690 ex\_SetSensorBoolean.nxc

This is an example of how to use the [SetSensorBoolean](#) function.

```
SetHTIRSeeker2Mode (S1, HTIR2_MODE_1200);
```



## 12.691 ex\_SetSensorColorBlue.nxc

This is an example of how to use the [SetSensorColorBlue](#) function.

```
SetSensorColorBlue (S1) ;
```

## 12.692 ex\_SetSensorColorFull.nxc

This is an example of how to use the [SetSensorColorFull](#) function.

```
SetSensorColorFull (S1) ;
```

## 12.693 ex\_SetSensorColorGreen.nxc

This is an example of how to use the [SetSensorColorGreen](#) function.

```
SetSensorColorGreen(S1);
```

## 12.694 ex\_SetSensorColorNone.nxc

This is an example of how to use the [SetSensorColorNone](#) function.

```
SetSensorColorNone (S1) ;
```

## 12.695 ex\_SetSensorColorRed.nxc

This is an example of how to use the [SetSensorColorRed](#) function.

```
SetSensorColorRed(S1);
```

## 12.696 ex\_SetSensorDigiPinsDirection.nxc

This is an example of how to use the [SetSensorDigiPinsDirection](#) function.

```
SetSensorDigiPinsDirection(S1, 1);
```

## 12.697 ex\_SetSensorDigiPinsOutputLevel.nxc

This is an example of how to use the [SetSensorDigiPinsOutputLevel](#) function.

```
SetSensorDigiPinsOutputLevel(S1, 100);
```

## 12.698 ex\_SetSensorDigiPinsStatus.nxc

This is an example of how to use the [SetSensorDigiPinsStatus](#) function.

```
SetSensorDigiPinsStatus(S1, false);
```



## 12.699 ex\_SetSensorHTEOPD.nxc

This is an example of how to use the [SetSensorHTEOPD](#) function.

```
SetSensorHTEOPD (S1) ;
```

## 12.700 ex\_SetSensorHTGyro.nxc

This is an example of how to use the [SetSensorHTGyro](#) function.

```
SetSensorHTGyro(S1);
```

## 12.701 ex\_SetSensorLight.nxc

This is an example of how to use the [SetSensorLight](#) function.

```
SetSensorLight (S1) ;
```

## 12.702 ex\_SetSensorLowspeed.nxc

This is an example of how to use the [SetSensorLowspeed](#) function.

```
SetSensorLowspeed(S1);
```

## 12.703 ex\_SetSensorMode.nxc

This is an example of how to use the [SetSensorMode](#) function.

```
SetSensorMode(S1, SENSOR_MODE_RAW); // raw mode
```

## 12.704 ex\_SetSensorMSDROD.nxc

This is an example of how to use the [SetSensorMSDROD](#) function.

```
SetSensorMSDROD (S1) ;
```

## 12.705 ex\_SetSensorMSPressure.nxc

This is an example of how to use the [SetSensorMSPressure](#) function.

```
SetSensorMSPressure(S1);
```

## 12.706 ex\_SetSensorSound.nxc

This is an example of how to use the [SetSensorSound](#) function.

```
SetSensorSound(S1);
```



## 12.707 ex\_SetSensorTouch.nxc

This is an example of how to use the [SetSensorTouch](#) function.

```
SetSensorTouch (S1) ;
```

## 12.708 ex\_SetSensorType.nxc

This is an example of how to use the [SetSensorType](#) function.

```
SetSensorType(S1, SENSOR_TYPE_TOUCH);
```

## 12.709 ex\_SetSleepTime.nxc

This is an example of how to use the [SetSleepTime](#) function.

```
SetSleepTime(5); // sleep in 5 minutes
```

## 12.710 ex\_SetSleepTimeout.nxc

This is an example of how to use the [SetSleepTimeout](#) function.

```
SetSleepTimeout (8);
```

## 12.711 ex\_SetSleepTimer.nxc

This is an example of how to use the [SetSleepTimer](#) function.

```
SetSleepTimer(3);
```

## 12.712 ex\_SetSoundDuration.nxc

This is an example of how to use the [SetSoundDuration](#) function.

```
SetSoundDuration(500);
```

## 12.713 ex\_SetSoundFlags.nxc

This is an example of how to use the [SetSoundFlags](#) function.

```
SetSoundFlags (SOUND_FLAGS_UPDATE) ;
```

## 12.714 ex\_SetSoundFrequency.nxc

This is an example of how to use the [SetSoundFrequency](#) function.

```
SetSoundFrequency(440);
```



## 12.715 ex\_SetSoundMode.nxc

This is an example of how to use the [SetSoundMode](#) function.

```
SetSoundMode (SOUND_MODE_ONCE) ;
```

## 12.716 ex\_SetSoundModuleState.nxc

This is an example of how to use the [SetSoundModuleState](#) function.

```
SetSoundModuleState (SOUND_STATE_STOP) ;
```

## 12.717 ex\_SetSoundSampleRate.nxc

This is an example of how to use the [SetSoundSampleRate](#) function.

```
SetSoundSampleRate(4000);
```

## 12.718 ex\_SetSoundVolume.nxc

This is an example of how to use the [SetSoundVolume](#) function.

```
SetSoundVolume (3) ;
```

## 12.719 ex\_SetUIButton.nxc

This is an example of how to use the [SetUIButton](#) function.

```
SetUIButton (UI_BUTTON_ENTER) ;
```

## 12.720 ex\_SetUIState.nxc

This is an example of how to use the [SetUIState](#) function.

```
SetUIState (UI_STATE_LOW_BATTERY) ;
```

## 12.721 ex\_SetUSBInputBuffer.nxc

This is an example of how to use the [SetUSBInputBuffer](#) function.

```
SetUSBInputBuffer(0, 10, buffer);
```

## 12.722 ex\_SetUSBInputBufferInPtr.nxc

This is an example of how to use the [SetUSBInputBufferInPtr](#) function.

```
SetUSBInputBufferInPtr(0);
```



## 12.723 ex\_SetUSBInputBufferOutPtr.nxc

This is an example of how to use the [SetUSBInputBufferOutPtr](#) function.

```
SetUSBInputBufferOutPtr(0);
```

## 12.724 ex\_SetUSBOutputBuffer.nxc

This is an example of how to use the [SetUSBOutputBuffer](#) function.

```
SetUSBOutputBuffer(0, 10, buffer);
```

## 12.725 ex\_SetUSBOutputBufferInPtr.nxc

This is an example of how to use the [SetUSBOutputBufferInPtr](#) function.

```
SetUSBOutputBufferInPtr(0);
```

## 12.726 ex\_SetUSBOutputBufferOutPtr.nxc

This is an example of how to use the [SetUSBOutputBufferOutPtr](#) function.

```
SetUSBOutputBufferOutPtr(0);
```

## 12.727 ex\_SetUSBPollBuffer.nxc

This is an example of how to use the [SetUSBPollBuffer](#) function.

```
SetUSBPollBuffer(0, 10, buffer);
```

## 12.728 ex\_SetUSBPollBufferInPtr.nxc

This is an example of how to use the [SetUSBPollBufferInPtr](#) function.

```
SetUSBPollBufferInPtr(0);
```

## 12.729 ex\_SetUSBPollBufferOutPtr.nxc

This is an example of how to use the [SetUSBPollBufferOutPtr](#) function.

```
SetUSBPollBufferOutPtr(0);
```

## 12.730 ex\_SetUSBState.nxc

This is an example of how to use the [SetUSBState](#) function.

```
SetUSBState(0);
```



## 12.731 ex\_SetVMRunState.nxc

This is an example of how to use the [SetVMRunState](#) function.

```
SetVMRunState(0); // stopped
```

## 12.732 ex\_SetVolume.nxc

This is an example of how to use the [SetVolume](#) function.

```
SetVolume(3);
```

## 12.733 ex\_sign.nxc

This is an example of how to use the [sign](#) function.

```
char val = sign(x); // return -1, 0, or 1
```

## 12.734 ex\_sin.nxc

This is an example of how to use the [sin](#) function.

```
float x = sin(theta); // radians
```

## 12.735 ex\_sind.nxc

This is an example of how to use the [sind](#) function.

```
float x = sind(theta); // degrees
```

## 12.736 `ex_sinh.nxc`

This is an example of how to use the [sinh](#) function.

```
x = sinh(y);
```

## 12.737 ex\_sleepnow.nxc

This is an example of how to use the [SleepNow](#) functions.

```
SleepNow ( ) ;
```

## 12.738 ex\_SleepTime.nxc

This is an example of how to use the [SleepTime](#) function.

```
x = SleepTime(); // read sleep time
```



## 12.739 ex\_SleepTimeout.nxc

This is an example of how to use the [SleepTimeout](#) function.

```
byte x = SleepTimeout();
```

## 12.740 ex\_SleepTimer.nxc

This is an example of how to use the [SleepTimer](#) function.

```
byte x = SleepTimer();
```

## 12.741 ex\_SoundDuration.nxc

This is an example of how to use the [SoundDuration](#) function.

```
x = SoundDuration();
```

## 12.742 ex\_SoundFlags.nxc

This is an example of how to use the [SoundFlags](#) function.

```
x = SoundFlags();
```

## 12.743 ex\_SoundFrequency.nxc

This is an example of how to use the [SoundFrequency](#) function.

```
x = SoundFrequency();
```

## 12.744 ex\_SoundMode.nxc

This is an example of how to use the [SoundMode](#) function.

```
x = SoundMode();
```

## 12.745 ex\_SoundSampleRate.nxc

This is an example of how to use the [SoundSampleRate](#) function.

```
x = SoundSampleRate();
```

## 12.746 ex\_SoundState.nxc

This is an example of how to use the [SoundState](#) function.

```
x = SoundState();
```



## 12.747 ex\_SoundVolume.nxc

This is an example of how to use the [SoundVolume](#) function.

```
x = SoundVolume();
```

## 12.748 ex\_sprintf.nxc

This is an example of how to use the [sprintf](#) function.

```
sprintf(msg, "value = %d", value);
```

## 12.749 ex\_sqrt.nxc

This is an example of how to use the [sqrt](#) function.

```
x = sqrt(x);
```

## 12.750 ex\_starttask.nxc

This is an example of how to use the [StartTask](#) function.

```
StartTask(sound); // start the sound task
```

## 12.751 ex\_stop.nxc

This is an example of how to use the [Stop](#) function.

```
Stop(x == 24); // stop the program if x==24
```

## 12.752 ex\_stopalltasks.nxc

This is an example of how to use the [StopAllTasks](#) function.

```
StopAllTasks(); // stop the program
```

## 12.753 ex\_StopSound.nxc

This is an example of how to use the [StopSound](#) function.

```
StopSound();
```

## 12.754 ex\_stoptask.nxc

This is an example of how to use the [StopTask](#) function.

```
StopTask(sound); // stop the sound task
```



## 12.755 ex\_strcat.nxc

This is an example of how to use the [strcat](#) function.

```
strcat(msg, "foo"); // msg = msg+"foo"
```

## 12.756 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);
}
```

## 12.757 ex\_strcmp.nxc

This is an example of how to use the [strcmp](#) function.

```
int i = strcmp(msg, "foo"); // returns -1, 0, or 1
```

## 12.758 ex\_strcpy.nxc

This is an example of how to use the [strcpy](#) function.

```
strcpy(msg, "foo"); // msg = "foo"
```

## 12.759 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);
}
```

## 12.760 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);
}
```

## 12.761 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
}
```

## 12.762 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);
}
```



## 12.763 ex\_strncat.nxc

This is an example of how to use the [strncat](#) function.

```
strncat(msg, "foo", 2); // msg = msg+"fo"
```

## 12.764 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
```

## 12.765 ex\_strncpy.nxc

This is an example of how to use the [strncpy](#) function.

```
strncpy(msg, "foo", 2); // msg = "fo"
```

## 12.766 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);
}
```

## 12.767 ex\_StrToByteArray.nxc

This is an example of how to use the [StrToByteArray](#) function.

```
StrToByteArray(myStr, myArray);
```

## 12.768 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);
}
```

## 12.769 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);
}
```

## 12.770 ex\_StrToNum.nxc

This is an example of how to use the [StrToNum](#) function.

```
x = StrToNum(strVal);
```



## 12.771 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);
}
```

## 12.772 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);
}
```

## 12.773 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);
}
```

## 12.774 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);
    }
}
```

## 12.775 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 */ }
}
```

## 12.776 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);
}
```

## 12.777 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");
}
```

## 12.778 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);
}
```



## 12.779 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);
}
```

## 12.780 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");
}
```

## 12.781 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);
}
```

## 12.782 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);
}
```

## 12.783 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
    }
}
```

## 12.784 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 */ }
}
```

## 12.785 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
}
```

## 12.786 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
}
```



## 12.787 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
    }
}
```

## 12.788 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");
}
```

## 12.789 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);
}
```

## 12.790 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");
}
```

## 12.791 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);
}
```

## 12.792 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);
}
```

## 12.793 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);
}
```

## 12.794 ex\_sysdrawfont.nxc

This is an example of how to use the [SysDrawFont](#) function along with the [DrawFontType](#) 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);
}
```



## 12.795 ex\_sysdrawgraphic.nxc

This is an example of how to use the [SysDrawGraphic](#) function along with the [DrawGraphicType](#) 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);
}
```

## 12.796 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);
}
```

## 12.797 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);
}
```

## 12.798 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);
}
```

## 12.799 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);
}
```

## 12.800 ex\_sysdrawrect.nxc

This is an example of how to use the [SysDrawRect](#) function along with the [DrawRectType](#) 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);
}
```

## 12.801 ex\_sysdrawtext.nxc

This is an example of how to use the [SysDrawText](#) function along with the [DrawTextType](#) 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);
}
```

## 12.802 ex\_sysfileclose.nxc

This is an example of how to use the [SysFileClose](#) function along with the [FileCloseType](#) structure.

```
task main()  
{  
    FileCloseType fcArgs;  
    fcArgs.FileHandle = foArgs.FileHandle;  
    SysFileClose(fcArgs);  
}
```



## 12.803 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
}
```

## 12.804 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);
}
```

## 12.805 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);
}
```

## 12.806 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
    }
}
```

## 12.807 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
    }
}
```

## 12.808 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
    }
}
```

## 12.809 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
    }
}
```

## 12.810 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
    }
}
```



## 12.811 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
    }
}
```

## 12.812 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);
    }
}
```

## 12.813 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 */ }
}
```

## 12.814 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
    }
}
```

## 12.815 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
        }
    }
}
```

## 12.816 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
    }
}
```

## 12.817 ex\_sysfilewrite.nxc

This is an example of how to use the [SysFileWrite](#) function along with the [FileReadWriteType](#) 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);
    }
}
```

## 12.818 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;
}
```



## 12.819 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 */ }
}
```

## 12.820 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 */ }
}
```

## 12.821 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);
}
```

## 12.822 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);
}
```

## 12.823 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  
}
```

## 12.824 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 = "*.rxe";
    SysListFiles(args);
    if (args.Result == NO_ERR && ArrayLen(args.FileList) > 0)
    {
        TextOut(0, LCD_LINE6, args.FileList[0]);
    }
    Wait(SEC_4);
}
```

## 12.825 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);
}
```

## 12.826 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);
    }
}
```



## 12.827 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
}
```

## 12.828 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;  
}
```

## 12.829 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 */ }
}
```

## 12.830 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);
}
```

## 12.831 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);
}
```

## 12.832 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);  
}
```

## 12.833 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 */}
}
```

## 12.834 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);
}
```



## 12.835 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);
}
```

## 12.836 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);  
}
```

## 12.837 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);
}
```

## 12.838 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);
}
```

## 12.839 ex\_tan.nxc

This is an example of how to use the [tan](#) function.

```
float x = tan(y); // radians
```

## 12.840 ex\_tand.nxc

This is an example of how to use the [tand](#) function.

```
float x = tand(y); // degrees
```

## 12.841 ex\_tanh.nxc

This is an example of how to use the [tanh](#) function.

```
x = tanh(y);
```

## 12.842 ex\_TextOut.nxc

This is an example of how to use the [TextOut](#) function.

```
TextOut(0, LCD_LINE3, "Hello World!");
```



## 12.843 ex\_tolower.nxc

This is an example of how to use the [tolower](#) function.

```
i = tolower(x);
```

## 12.844 ex\_toupper.nxc

This is an example of how to use the [toupper](#) function.

```
i = toupper(x);
```

## 12.845 ex\_trunc.nxc

This is an example of how to use the [trunc](#) function.

```
y = trunc(x);
```

## 12.846 ex\_UIButton.nxc

This is an example of how to use the [UIButton](#) function.

```
x = UIButton();
```

## 12.847 ex\_UIState.nxc

This is an example of how to use the [UIState](#) function.

```
x = UIState();
```

## 12.848 ex\_UiUsbState.nxc

This is an example of how to use the [UsbState](#) function.

```
value = UsbState();
```

## 12.849 ex\_UnflattenVar.nxc

This is an example of how to use the [UnflattenVar](#) function.

```
UnflattenVar(stringValue, myStruct);
```

## 12.850 ex\_USBInputBufferInPtr.nxc

This is an example of how to use the [USBInputBufferInPtr](#) function.

```
byte x = USBInputBufferInPtr();
```



## 12.851 ex\_USBInputBufferOutPtr.nxc

This is an example of how to use the [USBInputBufferOutPtr](#) function.

```
byte x = USBInputBufferOutPtr();
```

## 12.852 ex\_USBOutputBufferInPtr.nxc

This is an example of how to use the [USBOutputBufferInPtr](#) function.

```
byte x = USBOutputBufferInPtr();
```

## 12.853 ex\_USBOutputBufferOutPtr.nxc

This is an example of how to use the [USBOutputBufferOutPtr](#) function.

```
byte x = USBOutputBufferOutPtr();
```

## 12.854 ex\_USBPollBufferInPtr.nxc

This is an example of how to use the [USBPollBufferInPtr](#) function.

```
byte x = USBPollBufferInPtr();
```

## 12.855 ex\_USBPollBufferOutPtr.nxc

This is an example of how to use the [USBPollBufferOutPtr](#) function.

```
byte x = USBPollBufferOutPtr();
```

## 12.856 ex\_USBState.nxc

This is an example of how to use the [USBState](#) function.

```
byte x = USBPollBufferOutPtr();
```

## 12.857 ex\_VMRunState.nxc

This is an example of how to use the [VMRunState](#) function.

```
x = VMRunState();
```

## 12.858 ex\_Volume.nxc

This is an example of how to use the [Volume](#) function.

```
x = Volume();
```



## 12.859 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
}
```

## 12.860 ex\_Write.nxc

This is an example of how to use the [Write](#) function.

```
result = Write(handle, value);
```

## 12.861 ex\_WriteBytes.nxc

This is an example of how to use the [WriteBytes](#) function.

```
result = WriteBytes(handle, buffer, count);
```

## 12.862 ex\_WriteBytesEx.nxc

This is an example of how to use the [WriteBytesEx](#) function.

```
result = WriteBytesEx(handle, len, buffer);
```

## 12.863 ex\_writei2cregister.nxc

This is an example of how to use the [WriteI2CRegister](#) function.

```
char result = WriteI2CRegister(S1, I2C_REG_CMD, US_CMD_OFF);
```

## 12.864 ex\_WriteLn.nxc

This is an example of how to use the [WriteLn](#) function.

```
result = WriteLn(handle, value);
```

## 12.865 ex\_WriteLnString.nxc

This is an example of how to use the [WriteLnString](#) function.

```
result = WriteLnString(handle, "testing", count);
```

## 12.866 ex\_WriteNRLinkBytes.nxc

This is an example of how to use the [WriteNRLinkBytes](#) function.

```
byte data[] = {0x01, 0x02, 0x03};  
char result = WriteNRLinkBytes(S1, data);
```



## 12.867 ex\_WriteNRLinkBytesEx.nxc

This is an example of how to use the [WriteNRLinkBytesEx](#) function.

```
byte data[] = {0x01, 0x02, 0x03};  
char result = WriteNRLinkBytesEx(S1, 0x02, data);
```

## 12.868 ex\_WriteString.nxc

This is an example of how to use the [WriteString](#) function.

```
result = WriteString(handle, "testing", count);
```

## 12.869 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);
}
```

## 12.870 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);  
    }  
}
```

## 12.871 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();  
    }  
}
```

## 12.872 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;  
    }  
}
```

## 12.873 glScaleDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glAddToAngleY](#), [glBeginRender](#), [glCallObject](#), [glObjectAction](#), and [glFinishRender](#) functions.

```
/*-----  
; File      : glScaleDemo.nbc  
; 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);  
    }  
}
```

## 12.874 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;
    }
}
```

## 12.875 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);
}
```

## 12.876 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);
}
```

## 12.877 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, 1045](#)
    - [SensorTypeModes, 202](#)
- [A simple 3D graphics library, 185](#)
- [abort](#)
  - [cstdlibAPI, 437](#)
  - [NXCDefs.h, 1064](#)
- [AbortFlag](#)
  - [NXCDefs.h, 1064](#)
  - [UiModuleFunctions, 374](#)
- [abs](#)
  - [cstdlibAPI, 437](#)
  - [NXCDefs.h, 1065](#)
- [Acos](#)
  - [cmathAPI, 404](#)
  - [NXCDefs.h, 1045](#)
- [acos](#)
  - [cmathAPI, 414](#)
  - [NXCDefs.h, 1065](#)
- [AcosD](#)
  - [cmathAPI, 404](#)
  - [NXCDefs.h, 1046](#)
- [acosd](#)
  - [cmathAPI, 414](#)
  - [NXCDefs.h, 1065](#)
- [Acquire](#)
  - [CommandModuleFunctions, 306](#)
  - [NXCDefs.h, 1066](#)
- [Action](#)
  - [CommBTConnectionType, 722](#)
- [ActualSpeed](#)
  - [NBCCCommon.h, 846](#)
  - [OutputFieldConstants, 570](#)
- [addr](#)
  - [cstringAPI, 447](#)
  - [NXCDefs.h, 1066](#)
- [address](#)
  - [cstringAPI, 447](#)
  - [NXCDefs.h, 1066](#)
- [Array API functions, 314](#)
- [Array operation constants, 465](#)
- [ArrayBuild](#)
  - [ArrayFunctions, 315](#)
  - [NXCDefs.h, 1067](#)
- [ArrayFunctions](#)
  - [ArrayBuild, 315](#)
  - [ArrayInit, 315](#)
  - [ArrayLen, 315](#)
  - [ArrayMax, 315](#)
  - [ArrayMean, 316](#)
  - [ArrayMin, 316](#)
  - [ArrayOp, 317](#)
  - [ArraySort, 317](#)
  - [ArrayStd, 317](#)
  - [ArraySubset, 318](#)
  - [ArraySum, 318](#)
  - [ArraySumSqr, 318](#)
- [ArrayInit](#)
  - [ArrayFunctions, 315](#)
  - [NXCDefs.h, 1067](#)
- [ArrayLen](#)
  - [ArrayFunctions, 315](#)
  - [NXCDefs.h, 1067](#)
- [ArrayMax](#)
  - [ArrayFunctions, 315](#)
  - [NXCDefs.h, 1068](#)
- [ArrayMean](#)
  - [ArrayFunctions, 316](#)
  - [NXCDefs.h, 1068](#)
- [ArrayMin](#)
  - [ArrayFunctions, 316](#)
  - [NXCDefs.h, 1069](#)
- [ArrayOp](#)
  - [ArrayFunctions, 317](#)
  - [NXCDefs.h, 1069](#)
- [ArrayOpConstants](#)
  - [OPARR\\_MAX, 465](#)
  - [OPARR\\_MEAN, 465](#)
  - [OPARR\\_MIN, 465](#)
  - [OPARR\\_SORT, 465](#)
  - [OPARR\\_STD, 465](#)
  - [OPARR\\_SUM, 465](#)
  - [OPARR\\_SUMSQR, 465](#)
- [ArraySort](#)
  - [ArrayFunctions, 317](#)
  - [NXCDefs.h, 1069](#)
- [ArrayStd](#)
  - [ArrayFunctions, 317](#)
  - [NXCDefs.h, 1070](#)

- ArraySubset
  - ArrayFunctions, 318
  - NXCDefs.h, 1070
- ArraySum
  - ArrayFunctions, 318
  - NXCDefs.h, 1070
- ArraySumSqr
  - ArrayFunctions, 318
  - NXCDefs.h, 1071
- Asin
  - cmathAPI, 404
  - NXCDefs.h, 1046
- asin
  - cmathAPI, 414
  - NXCDefs.h, 1071
- AsinD
  - cmathAPI, 404
  - NXCDefs.h, 1046
- asind
  - cmathAPI, 415
  - NXCDefs.h, 1072
- Atan
  - cmathAPI, 405
  - NXCDefs.h, 1047
- atan
  - cmathAPI, 415
  - NXCDefs.h, 1072
- Atan2
  - cmathAPI, 405
  - NXCDefs.h, 1047
- atan2
  - cmathAPI, 415
  - NXCDefs.h, 1072
- Atan2D
  - cmathAPI, 405
  - NXCDefs.h, 1047
- atan2d
  - cmathAPI, 416
  - NXCDefs.h, 1073
- AtanD
  - cmathAPI, 406
  - NXCDefs.h, 1048
- atand
  - cmathAPI, 416
  - NXCDefs.h, 1073
- atof
  - cstdlibAPI, 437
  - NXCDefs.h, 1074
- atoi
  - cstdlibAPI, 438
  - NXCDefs.h, 1074
- atol
  - cstdlibAPI, 438
  - NXCDefs.h, 1075
- Basic analog sensor value names, 225
- BasicSensorValues
  - SENSOR\_1, 225
  - SENSOR\_2, 225
  - SENSOR\_3, 225
  - SENSOR\_4, 225
- BatteryLevel
  - NXCDefs.h, 1075
  - UiModuleFunctions, 374
- BatteryState
  - NXCDefs.h, 1076
  - UiModuleFunctions, 374
- BaudRate
  - CommHSControlType, 729
- bcd2dec
  - cmathAPI, 417
  - NXCDefs.h, 1076
- BITMAP\_1
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 846
- BITMAP\_2
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 846
- BITMAP\_3
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 846
- BITMAP\_4
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 847
- BITMAPS
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 847
- BlockTachoCount
  - NBCCCommon.h, 847
  - OutputFieldConstants, 570
- Bluetooth hardware status constants, 617
- Bluetooth State constants, 615
- Bluetooth state status constants, 616
- BluetoothState
  - NXCDefs.h, 1076
  - UiModuleFunctions, 374
- BluetoothState constants, 537
- BluetoothStatus
  - CommModuleFunctions, 328
  - NXCDefs.h, 1076
- BluetoothWrite
  - CommModuleFunctions, 328
  - NXCDefs.h, 1077
- BREAKOUT\_REQ
  - CommandVMState, 487
  - NBCCCommon.h, 847
- BrickDataBluecoreVersion
  - CommModuleFunctions, 329
  - NXCDefs.h, 1077

- BrickDataBtHardwareStatus
  - CommModuleFunctions, 329
  - NXCDefs.h, 1077
- BrickDataBtStateStatus
  - CommModuleFunctions, 329
  - NXCDefs.h, 1078
- BrickDataName
  - CommModuleFunctions, 329
  - NXCDefs.h, 1078
- BrickDataTimeoutValue
  - CommModuleFunctions, 329
  - NXCDefs.h, 1078
- BT\_ARM\_CMD\_MODE
  - CommBtStateConstants, 615
  - NBCCCommon.h, 847
- BT\_ARM\_DATA\_MODE
  - CommBtStateConstants, 615
  - NBCCCommon.h, 847
- BT\_ARM\_OFF
  - CommBtStateConstants, 615
  - NBCCCommon.h, 847
- BT\_BRICK\_PORT\_OPEN
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 847
- BT\_BRICK\_VISIBILITY
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 847
- BT\_CMD\_BYTE
  - CommMiscConstants, 613
  - NBCCCommon.h, 847
- BT\_CMD\_READY
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 848
- BT\_CONNECTION\_0\_ENABLE
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 848
- BT\_CONNECTION\_1\_ENABLE
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 848
- BT\_CONNECTION\_2\_ENABLE
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 848
- BT\_CONNECTION\_3\_ENABLE
  - CommBtStateStatusConstants, 616
  - NBCCCommon.h, 848
- BT\_DEFAULT\_INQUIRY\_MAX
  - CommMiscConstants, 613
  - NBCCCommon.h, 848
- BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO
  - CommMiscConstants, 613
  - NBCCCommon.h, 848
- BT\_DEVICE\_AWAY
  - CommDeviceStatusConstants, 629
  - NBCCCommon.h, 848
- BT\_DEVICE\_EMPTY
  - CommDeviceStatusConstants, 629
  - NBCCCommon.h, 848
- BT\_DEVICE\_KNOWN
  - CommDeviceStatusConstants, 629
  - NBCCCommon.h, 848
- BT\_DEVICE\_NAME
  - CommDeviceStatusConstants, 629
  - NBCCCommon.h, 848
- BT\_DEVICE\_UNKNOWN
  - CommDeviceStatusConstants, 629
  - NBCCCommon.h, 849
- BT\_DISABLE
  - CommBtHwStatusConstants, 617
  - NBCCCommon.h, 849
- BT\_ENABLE
  - CommBtHwStatusConstants, 617
  - NBCCCommon.h, 849
- BTConnectionClass
  - CommModuleFunctions, 330
  - NXCDefs.h, 1078
- BTConnectionHandleNum
  - CommModuleFunctions, 330
  - NXCDefs.h, 1078
- BTConnectionLinkQuality
  - CommModuleFunctions, 330
  - NXCDefs.h, 1079
- BTConnectionName
  - CommModuleFunctions, 331
  - NXCDefs.h, 1079
- BTConnectionPinCode
  - CommModuleFunctions, 331
  - NXCDefs.h, 1079
- BTConnectionStreamStatus
  - CommModuleFunctions, 331
  - NXCDefs.h, 1080
- BTDeviceClass
  - CommModuleFunctions, 332
  - NXCDefs.h, 1080
- BTDeviceCount
  - CommModuleFunctions, 332
  - NXCDefs.h, 1080
- BTDeviceName
  - CommModuleFunctions, 332
  - NXCDefs.h, 1081
- BTDeviceNameCount
  - CommModuleFunctions, 332
  - NXCDefs.h, 1081
- BTDeviceStatus
  - CommModuleFunctions, 333
  - NXCDefs.h, 1081
- BTInputBufferInPtr
  - CommModuleFunctions, 333
  - NXCDefs.h, 1082

- BTInputBufferOutPtr
  - CommModuleFunctions, 333
  - NXCDefs.h, 1082
- BTN1
  - ButtonNameConstants, 525
  - NBCCCommon.h, 849
- BTN2
  - ButtonNameConstants, 525
  - NBCCCommon.h, 849
- BTN3
  - ButtonNameConstants, 525
  - NBCCCommon.h, 849
- BTN4
  - ButtonNameConstants, 525
  - NBCCCommon.h, 849
- BTNCENTER
  - ButtonNameConstants, 525
  - NBCCCommon.h, 849
- BTNEXIT
  - ButtonNameConstants, 526
  - NBCCCommon.h, 849
- BTNLEFT
  - ButtonNameConstants, 526
  - NBCCCommon.h, 850
- BTNRIGHT
  - ButtonNameConstants, 526
  - NBCCCommon.h, 850
- BTNSTATE\_LONG\_PRESSED\_EV
  - ButtonStateConstants, 527
  - NBCCCommon.h, 850
- BTNSTATE\_LONG\_RELEASED\_EV
  - ButtonStateConstants, 527
  - NBCCCommon.h, 850
- BTNSTATE\_NONE
  - ButtonStateConstants, 527
  - NBCCCommon.h, 850
- BTNSTATE\_PRESSED\_EV
  - ButtonStateConstants, 527
  - NBCCCommon.h, 850
- BTNSTATE\_PRESSED\_STATE
  - ButtonStateConstants, 527
  - NBCCCommon.h, 850
- BTNSTATE\_SHORT\_RELEASED\_EV
  - ButtonStateConstants, 528
  - NBCCCommon.h, 851
- BTOutputBufferInPtr
  - CommModuleFunctions, 333
  - NXCDefs.h, 1082
- BTOutputBufferOutPtr
  - CommModuleFunctions, 334
  - NXCDefs.h, 1082
- Buffer
  - CommBTWriteType, 725
  - CommHSReadWriteType, 731
  - CommLSReadType, 733
  - CommLSWriteExType, 735
  - CommLSWriteType, 737
  - FileReadWriteType, 770
  - IOMapReadByIDType, 780
  - IOMapReadType, 782
  - IOMapWriteByIDType, 784
  - IOMapWriteType, 786
  - LoaderExecuteFunctionType, 793
- BufferLen
  - CommLSReadType, 733
- Button module, 55
- Button module constants, 524
- Button module functions, 365
- Button module IOMAP offsets, 529
- Button module types, 364
- Button name constants, 525
- ButtonCount
  - ButtonModuleFunctions, 366
  - NXCDefs.h, 1083
- ButtonIOMAP
  - ButtonOffsetLongPressCnt, 529
  - ButtonOffsetLongRelCnt, 529
  - ButtonOffsetPressedCnt, 529
  - ButtonOffsetRelCnt, 529
  - ButtonOffsetShortRelCnt, 529
  - ButtonOffsetState, 529
- ButtonLongPressCount
  - ButtonModuleFunctions, 366
  - NXCDefs.h, 1083
- ButtonLongReleaseCount
  - ButtonModuleFunctions, 366
  - NXCDefs.h, 1083
- ButtonModuleFunctions
  - ButtonCount, 366
  - ButtonLongPressCount, 366
  - ButtonLongReleaseCount, 366
  - ButtonPressCount, 367
  - ButtonPressed, 367
  - ButtonReleaseCount, 367
  - ButtonShortReleaseCount, 368
  - ButtonState, 368
  - ReadButtonEx, 368
  - SetButtonLongPressCount, 369
  - SetButtonLongReleaseCount, 369
  - SetButtonPressCount, 369
  - SetButtonReleaseCount, 369
  - SetButtonShortReleaseCount, 370
  - SetButtonState, 370
  - SysReadButton, 370
- ButtonModuleID
  - ModuleIDConstants, 179
  - NBCCCommon.h, 851
- ButtonModuleName



- ModuleNameConstants, 177
- NBCCCommon.h, 851
- ButtonNameConstants
  - BTN1, 525
  - BTN2, 525
  - BTN3, 525
  - BTN4, 525
  - BTNCENTER, 525
  - BTNEXIT, 526
  - BTNLEFT, 526
  - BTNRIGHT, 526
  - NO\_OF\_BTNS, 526
- ButtonOffsetLongPressCnt
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonOffsetLongRelCnt
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonOffsetPressedCnt
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonOffsetRelCnt
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonOffsetShortRelCnt
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonOffsetState
  - ButtonIOMAP, 529
  - NBCCCommon.h, 851
- ButtonPressCount
  - ButtonModuleFunctions, 367
  - NXCDefs.h, 1083
- ButtonPressed
  - ButtonModuleFunctions, 367
  - NXCDefs.h, 1084
- ButtonReleaseCount
  - ButtonModuleFunctions, 367
  - NXCDefs.h, 1084
- ButtonShortReleaseCount
  - ButtonModuleFunctions, 368
  - NXCDefs.h, 1084
- ButtonState
  - ButtonModuleFunctions, 368
  - NXCDefs.h, 1085
- ButtonState constants, 527
- ButtonStateConstants
  - BTNSTATE\_LONG\_PRESSED\_EV, 527
  - BTNSTATE\_LONG\_RELEASED\_EV, 527
  - BTNSTATE\_NONE, 527
  - BTNSTATE\_PRESSED\_EV, 527
  - BTNSTATE\_PRESSED\_STATE, 527
  - BTNSTATE\_SHORT\_RELEASED\_EV, 528
- ByteArrayToStr
  - cstringAPI, 447
  - NXCDefs.h, 1085
- ByteArrayToStrEx
  - cstringAPI, 448
  - NXCDefs.h, 1085
- BytesReady
  - CommLSCheckStatusType, 732
- Ceil
  - cmathAPI, 406
  - NXCDefs.h, 1048
- ceil
  - cmathAPI, 417
  - NXCDefs.h, 1086
- Center
  - DrawCircleType, 746
  - DrawEllipseType, 748
- CHAR\_BIT
  - NBCCCommon.h, 851
  - NXTLimits, 711
- CHAR\_MAX
  - NBCCCommon.h, 851
  - NXTLimits, 711
- CHAR\_MIN
  - NBCCCommon.h, 852
  - NXTLimits, 711
- CircleOut
  - DisplayModuleFunctions, 251
  - NXCDefs.h, 1086
- ClearBits
  - WriteSemDataType, 814
- ClearLine
  - DisplayModuleFunctions, 251
  - NXCDefs.h, 1087
- ClearScreen
  - DisplayModuleFunctions, 251
  - NXCDefs.h, 1087
- ClearSensor
  - InputModuleFunctions, 208
  - NXCDefs.h, 1087
- CloseFile
  - LoaderModuleFunctions, 385
  - NXCDefs.h, 1087
- CLUMP\_DONE
  - CommandVMState, 487
  - NBCCCommon.h, 852
- CLUMP\_SUSPEND
  - CommandVMState, 487
  - NBCCCommon.h, 852
- cmath API, 400
- cmathAPI
  - Acos, 404
  - acos, 414
  - AcosD, 404

- acosd, [414](#)
- Asin, [404](#)
- asin, [414](#)
- AsinD, [404](#)
- asind, [415](#)
- Atan, [405](#)
- atan, [415](#)
- Atan2, [405](#)
- atan2, [415](#)
- Atan2D, [405](#)
- atan2d, [416](#)
- AtanD, [406](#)
- atand, [416](#)
- bcd2dec, [417](#)
- Ceil, [406](#)
- ceil, [417](#)
- Cos, [406](#)
- cos, [417](#)
- CosD, [407](#)
- cosd, [418](#)
- Cosh, [407](#)
- cosh, [418](#)
- CoshD, [407](#)
- coshd, [418](#)
- Exp, [408](#)
- exp, [419](#)
- Floor, [408](#)
- floor, [419](#)
- Frac, [408](#)
- frac, [419](#)
- isNAN, [420](#)
- Log, [409](#)
- log, [420](#)
- Log10, [409](#)
- log10, [421](#)
- MulDiv32, [409](#)
- muldiv32, [421](#)
- Pow, [410](#)
- pow, [421](#)
- sign, [422](#)
- Sin, [410](#)
- sin, [422](#)
- SinD, [410](#)
- sind, [422](#)
- Sinh, [411](#)
- sinh, [423](#)
- SinhD, [411](#)
- sinhd, [423](#)
- Sqrt, [411](#)
- sqrt, [423](#)
- Tan, [412](#)
- tan, [424](#)
- TanD, [412](#)
- tand, [424](#)
- Tanh, [412](#)
- tanh, [424](#)
- TanhD, [413](#)
- tanhd, [425](#)
- Trunc, [413](#)
- trunc, [425](#)
- Cmd
  - CommExecuteFunctionType, [727](#)
  - DisplayExecuteFunctionType, [743](#)
  - LoaderExecuteFunctionType, [793](#)
- Coast
  - NXCDefs.h, [1088](#)
  - OutputModuleFunctions, [229](#)
- CoastEx
  - NXCDefs.h, [1088](#)
  - OutputModuleFunctions, [229](#)
- Color calibration constants, [553](#)
- Color calibration state constants, [552](#)
- Color sensor array indices, [550](#)
- Color values, [551](#)
- ColorADRaw
  - InputModuleFunctions, [208](#)
  - NXCDefs.h, [1088](#)
- ColorBoolean
  - InputModuleFunctions, [209](#)
  - NXCDefs.h, [1089](#)
- ColorCalibration
  - InputModuleFunctions, [209](#)
  - NXCDefs.h, [1089](#)
- ColorCalibrationState
  - InputModuleFunctions, [209](#)
  - NXCDefs.h, [1089](#)
- ColorCalLimits
  - InputModuleFunctions, [210](#)
  - NXCDefs.h, [1090](#)
- ColorSensorRaw
  - InputModuleFunctions, [210](#)
  - NXCDefs.h, [1090](#)
- ColorSensorRead
  - NBCCCommon.h, [852](#)
  - SysCallConstants, [468](#)
- ColorSensorReadType, [719](#)
  - ColorValue, [719](#)
  - Invalid, [720](#)
  - NormalizedArray, [720](#)
  - Port, [720](#)
  - RawArray, [720](#)
  - Result, [720](#)
  - ScaledArray, [720](#)
- ColorSensorValue
  - InputModuleFunctions, [211](#)
  - NXCDefs.h, [1091](#)
- ColorValue
  - ColorSensorReadType, [719](#)

- COM\_CHANNEL\_FOUR\_ACTIVE
  - LowSpeedStateConstants, 578
  - NBCCCommon.h, 852
- COM\_CHANNEL\_NONE\_ACTIVE
  - LowSpeedStateConstants, 578
  - NBCCCommon.h, 852
- COM\_CHANNEL\_ONE\_ACTIVE
  - LowSpeedStateConstants, 578
  - NBCCCommon.h, 852
- COM\_CHANNEL\_THREE\_ACTIVE
  - LowSpeedStateConstants, 578
  - NBCCCommon.h, 852
- COM\_CHANNEL\_TWO\_ACTIVE
  - LowSpeedStateConstants, 578
  - NBCCCommon.h, 852
- Combined sensor type and mode constants, 201
- Comm module, 54
- Comm module constants, 612
- Comm module functions, 321
- Comm module interface function constants, 630
- Comm module IOMAP offsets, 635
- Comm module status code constants, 633
- Comm module types, 320
- Command
  - CommHSControlType, 729
- Command module, 51
- Command module constants, 52
- Command module functions, 304
- Command module IOMAP offsets, 495
- Command module types, 303
- CommandCommErrors
  - ERR\_COMM\_BUFFER\_FULL, 492
  - ERR\_COMM\_BUS\_ERR, 492
  - ERR\_COMM\_CHAN\_INVALID, 492
  - ERR\_COMM\_CHAN\_NOT\_READY, 492
- CommandFatalErrors
  - ERR\_ARG, 488
  - ERR\_BAD\_POOL\_SIZE, 488
  - ERR\_BAD\_PTR, 488
  - ERR\_CLUMP\_COUNT, 488
  - ERR\_DEFAULT\_OFFSETS, 488
  - ERR\_FILE, 489
  - ERR\_INSANE\_OFFSET, 489
  - ERR\_INSTR, 489
  - ERR\_LOADER\_ERR, 489
  - ERR\_MEM, 489
  - ERR\_MEMMGR\_FAIL, 489
  - ERR\_NO\_ACTIVE\_CLUMP, 489
  - ERR\_NO\_CODE, 489
  - ERR\_NON\_FATAL, 489
  - ERR\_SPOTCHECK\_FAIL, 489
  - ERR\_VER, 489
- CommandFlags
  - NXCDefs.h, 1091
- UiModuleFunctions, 375
- CommandFlags constants, 531
- CommandGenErrors
  - ERR\_INVALID\_FIELD, 491
  - ERR\_INVALID\_PORT, 491
  - ERR\_INVALID\_QUEUE, 491
  - ERR\_INVALID\_SIZE, 491
  - ERR\_NO\_PROG, 491
- CommandIOMAP
  - CommandOffsetActivateFlag, 495
  - CommandOffsetAwake, 495
  - CommandOffsetDeactivateFlag, 495
  - CommandOffsetFileName, 495
  - CommandOffsetFormatString, 495
  - CommandOffsetMemoryPool, 495
  - CommandOffsetOffsetDS, 496
  - CommandOffsetOffsetDVA, 496
  - CommandOffsetPRCHandler, 496
  - CommandOffsetProgStatus, 496
  - CommandOffsetSyncTick, 496
  - CommandOffsetSyncTime, 496
  - CommandOffsetTick, 496
- CommandModuleConstants
  - NO\_ERR, 53
  - STAT\_COMM\_PENDING, 53
  - STAT\_MSG\_EMPTY\_MAILBOX, 53
- CommandModuleFunctions
  - Acquire, 306
  - CurrentTick, 306
  - ExitTo, 306
  - FirstTick, 306
  - Follows, 306
  - Precedes, 307
  - Release, 307
  - ResetSleepTimer, 307
  - StartTask, 308
  - Stop, 308
  - StopAllTasks, 308
  - StopTask, 308
  - SysCall, 309
  - SysComputeCalibValue, 309
  - SysDatalogGetTimes, 309
  - SysDatalogWrite, 310
  - SysGetStartTick, 310
  - SysIOMapRead, 310
  - SysIOMapReadByID, 311
  - SysIOMapWrite, 311
  - SysIOMapWriteByID, 311
  - SysKeepAlive, 312
  - SysReadSemData, 312
  - SysUpdateCalibCacheInfo, 312
  - SysWriteSemData, 312
  - Wait, 313
  - Yield, 313

- CommandModuleID
  - ModuleIDConstants, [179](#)
  - NBCCCommon.h, [852](#)
- CommandModuleName
  - ModuleNameConstants, [177](#)
  - NBCCCommon.h, [852](#)
- CommandOffsetActivateFlag
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetAwake
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetDeactivateFlag
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetFileName
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetFormatString
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetMemoryPool
  - CommandIOMAP, [495](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetOffsetDS
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetOffsetDVA
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetPRCHandler
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetProgStatus
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [853](#)
- CommandOffsetSyncTick
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [854](#)
- CommandOffsetSyncTime
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [854](#)
- CommandOffsetTick
  - CommandIOMAP, [496](#)
  - NBCCCommon.h, [854](#)
- CommandProgStatus
  - PROG\_ABORT, [494](#)
  - PROG\_ERROR, [494](#)
  - PROG\_IDLE, [494](#)
  - PROG\_OK, [494](#)
  - PROG\_RESET, [494](#)
  - PROG\_RUNNING, [494](#)
- CommandRCErrors
  - ERR\_RC\_BAD\_PACKET, [493](#)
  - ERR\_RC\_FAILED, [493](#)
  - ERR\_RC\_ILLEGAL\_VAL, [493](#)
  - ERR\_RC\_UNKNOWN\_CMD, [493](#)
- CommandVMState
  - BREAKOUT\_REQ, [487](#)
  - CLUMP\_DONE, [487](#)
  - CLUMP\_SUSPEND, [487](#)
  - ROTATE\_QUEUE, [487](#)
  - STOP\_REQ, [487](#)
  - TIMES\_UP, [487](#)
- CommBTCheckStatus
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [468](#)
- CommBTCheckStatusType, [721](#)
  - Connection, [721](#)
  - Result, [721](#)
- CommBTConnection
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [468](#)
- CommBTConnectionType, [722](#)
  - Action, [722](#)
  - ConnectionSlot, [722](#)
  - Name, [722](#)
  - Result, [722](#)
- CommBtHwStatusConstants
  - BT\_DISABLE, [617](#)
  - BT\_ENABLE, [617](#)
- CommBTOnOff
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [468](#)
- CommBTOnOffType, [724](#)
  - PowerState, [724](#)
  - Result, [724](#)
- CommBTRead
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [468](#)
- CommBtStateConstants
  - BT\_ARM\_CMD\_MODE, [615](#)
  - BT\_ARM\_DATA\_MODE, [615](#)
  - BT\_ARM\_OFF, [615](#)
- CommBtStateStatusConstants
  - BT\_BRICK\_PORT\_OPEN, [616](#)
  - BT\_BRICK\_VISIBILITY, [616](#)
  - BT\_CONNECTION\_0\_ENABLE, [616](#)
  - BT\_CONNECTION\_1\_ENABLE, [616](#)
  - BT\_CONNECTION\_2\_ENABLE, [616](#)
  - BT\_CONNECTION\_3\_ENABLE, [616](#)
- CommBTWrite
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [469](#)
- CommBTWriteType, [725](#)
  - Buffer, [725](#)
  - Connection, [725](#)
  - Result, [725](#)

- CommDeviceStatusConstants
  - BT\_DEVICE\_AWAY, [629](#)
  - BT\_DEVICE\_EMPTY, [629](#)
  - BT\_DEVICE\_KNOWN, [629](#)
  - BT\_DEVICE\_NAME, [629](#)
  - BT\_DEVICE\_UNKNOWN, [629](#)
- CommExecuteFunction
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [469](#)
- CommExecuteFunctionType, [726](#)
  - Cmd, [727](#)
  - Name, [727](#)
  - Param1, [727](#)
  - Param2, [727](#)
  - Param3, [727](#)
  - Result, [727](#)
  - RetVal, [727](#)
- CommHiSpeedBaudConstants
  - HS\_BAUD\_115200, [622](#)
  - HS\_BAUD\_1200, [622](#)
  - HS\_BAUD\_14400, [622](#)
  - HS\_BAUD\_19200, [622](#)
  - HS\_BAUD\_230400, [622](#)
  - HS\_BAUD\_2400, [623](#)
  - HS\_BAUD\_28800, [623](#)
  - HS\_BAUD\_3600, [623](#)
  - HS\_BAUD\_38400, [623](#)
  - HS\_BAUD\_460800, [623](#)
  - HS\_BAUD\_4800, [623](#)
  - HS\_BAUD\_57600, [623](#)
  - HS\_BAUD\_7200, [623](#)
  - HS\_BAUD\_76800, [623](#)
  - HS\_BAUD\_921600, [623](#)
  - HS\_BAUD\_9600, [623](#)
- CommHiSpeedCombinedConstants
  - HS\_MODE\_7E1, [628](#)
  - HS\_MODE\_8N1, [628](#)
- CommHiSpeedCtrlConstants
  - HS\_CTRL\_EXIT, [621](#)
  - HS\_CTRL\_INIT, [621](#)
  - HS\_CTRL\_UART, [621](#)
- CommHiSpeedDataBitsConstants
  - HS\_MODE\_5\_DATA, [625](#)
  - HS\_MODE\_6\_DATA, [625](#)
  - HS\_MODE\_7\_DATA, [625](#)
  - HS\_MODE\_8\_DATA, [625](#)
- CommHiSpeedFlagsConstants
  - HS\_UPDATE, [619](#)
- CommHiSpeedParityConstants
  - HS\_MODE\_E\_PARITY, [627](#)
  - HS\_MODE\_M\_PARITY, [627](#)
  - HS\_MODE\_N\_PARITY, [627](#)
  - HS\_MODE\_O\_PARITY, [627](#)
  - HS\_MODE\_S\_PARITY, [627](#)
- CommHiSpeedStateConstants
  - HS\_DISABLE, [620](#)
  - HS\_ENABLE, [620](#)
  - HS\_INIT\_RECEIVER, [620](#)
  - HS\_INITIALISE, [620](#)
  - HS\_SEND\_DATA, [620](#)
- CommHiSpeedStopBitsConstants
  - HS\_MODE\_10\_STOP, [626](#)
  - HS\_MODE\_15\_STOP, [626](#)
  - HS\_MODE\_20\_STOP, [626](#)
- CommHSCheckStatus
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [469](#)
- CommHSCheckStatusType, [728](#)
  - DataAvailable, [728](#)
  - SendingData, [728](#)
- CommHSControl
  - NBCCCommon.h, [854](#)
  - SysCallConstants, [469](#)
- CommHSControlType, [729](#)
  - BaudRate, [729](#)
  - Command, [729](#)
  - Mode, [729](#)
  - Result, [729](#)
- CommHSRead
  - NBCCCommon.h, [855](#)
  - SysCallConstants, [469](#)
- CommHSReadWriteType, [731](#)
  - Buffer, [731](#)
  - Status, [731](#)
- CommHSWrite
  - NBCCCommon.h, [855](#)
  - SysCallConstants, [469](#)
- CommInterfaceConstants
  - INTF\_BTOFF, [630](#)
  - INTF\_BTON, [630](#)
  - INTF\_CONNECT, [630](#)
  - INTF\_CONNECTBYNAME, [631](#)
  - INTF\_CONNECTREQ, [631](#)
  - INTF\_DISCONNECT, [631](#)
  - INTF\_DISCONNECTALL, [631](#)
  - INTF\_EXTREAD, [631](#)
  - INTF\_FACTORYRESET, [631](#)
  - INTF\_OPENSTREAM, [631](#)
  - INTF\_PINREQ, [631](#)
  - INTF\_REMOVEDEVICE, [631](#)
  - INTF\_SEARCH, [631](#)
  - INTF\_SENDDATA, [631](#)
  - INTF\_SENDFILE, [632](#)
  - INTF\_SETBTNAME, [632](#)
  - INTF\_SETCMDMODE, [632](#)
  - INTF\_STOPSEARCH, [632](#)
  - INTF\_VISIBILITY, [632](#)
- CommIOMAP

- CommOffsetBrickDataBdAddr, 636
- CommOffsetBrickDataBluecoreVersion, 636
- CommOffsetBrickDataBtHwStatus, 636
- CommOffsetBrickDataBtStateStatus, 636
- CommOffsetBrickDataName, 636
- CommOffsetBrickDataTimeOutValue, 636
- CommOffsetBtConnectTableBdAddr, 636
- CommOffsetBtConnectTableClassOfDevice, 636
- CommOffsetBtConnectTableHandleNr, 636
- CommOffsetBtConnectTableLinkQuality, 636
- CommOffsetBtConnectTableName, 636
- CommOffsetBtConnectTablePinCode, 637
- CommOffsetBtConnectTableStreamStatus, 637
- CommOffsetBtDeviceCnt, 637
- CommOffsetBtDeviceNameCnt, 637
- CommOffsetBtDeviceTableBdAddr, 637
- CommOffsetBtDeviceTableClassOfDevice, 637
- CommOffsetBtDeviceTableDeviceStatus, 637
- CommOffsetBtDeviceTableName, 637
- CommOffsetBtInBufBuf, 637
- CommOffsetBtInBufInPtr, 637
- CommOffsetBtInBufOutPtr, 637
- CommOffsetBtOutBufBuf, 638
- CommOffsetBtOutBufInPtr, 638
- CommOffsetBtOutBufOutPtr, 638
- CommOffsetHsFlags, 638
- CommOffsetHsInBufBuf, 638
- CommOffsetHsInBufInPtr, 638
- CommOffsetHsInBufOutPtr, 638
- CommOffsetHsMode, 638
- CommOffsetHsOutBufBuf, 638
- CommOffsetHsOutBufInPtr, 638
- CommOffsetHsOutBufOutPtr, 638
- CommOffsetHsSpeed, 639
- CommOffsetHsState, 639
- CommOffsetPFunc, 639
- CommOffsetPFuncTwo, 639
- CommOffsetUsbInBufBuf, 639
- CommOffsetUsbInBufInPtr, 639
- CommOffsetUsbInBufOutPtr, 639
- CommOffsetUsbOutBufBuf, 639
- CommOffsetUsbOutBufInPtr, 639
- CommOffsetUsbOutBufOutPtr, 639
- CommOffsetUsbPollBufBuf, 639
- CommOffsetUsbPollBufInPtr, 640
- CommOffsetUsbPollBufOutPtr, 640
- CommOffsetUsbState, 640
- CommLSCheckStatus
  - NBCCCommon.h, 855
  - SysCallConstants, 469
- CommLSCheckStatusType, 732
- BytesReady, 732
- Port, 732
- Result, 732
- CommLSRead
  - NBCCCommon.h, 855
  - SysCallConstants, 469
- CommLSReadType, 733
  - Buffer, 733
  - BufferLen, 733
  - Port, 733
  - Result, 733
- CommLSWrite
  - NBCCCommon.h, 855
  - SysCallConstants, 469
- CommLSWriteEx
  - NBCCCommon.h, 855
  - SysCallConstants, 469
- CommLSWriteExType, 735
  - Buffer, 735
  - NoRestartOnRead, 735
  - Port, 735
  - Result, 735
  - ReturnLen, 736
- CommLSWriteType, 737
  - Buffer, 737
  - Port, 737
  - Result, 737
  - ReturnLen, 737
- CommMiscConstants
  - BT\_CMD\_BYTE, 613
  - BT\_DEFAULT\_INQUIRY\_MAX, 613
  - BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO, 613
  - MAX\_BT\_MSG\_SIZE, 613
  - SIZE\_OF\_BDADDR, 613
  - SIZE\_OF\_BRICK\_NAME, 613
  - SIZE\_OF\_BT\_CONNECT\_TABLE, 614
  - SIZE\_OF\_BT\_DEVICE\_TABLE, 614
  - SIZE\_OF\_BT\_NAME, 614
  - SIZE\_OF\_BT\_PINCODE, 614
  - SIZE\_OF\_BTBUF, 614
  - SIZE\_OF\_CLASS\_OF\_DEVICE, 614
  - SIZE\_OF\_HSBUFF, 614
  - SIZE\_OF\_USBBUFF, 614
  - SIZE\_OF\_USBDATA, 614
  - USB\_PROTOCOL\_OVERHEAD, 614
- CommModuleFunctions
  - BluetoothStatus, 328
  - BluetoothWrite, 328
  - BrickDataBluecoreVersion, 329
  - BrickDataBtHardwareStatus, 329
  - BrickDataBtStateStatus, 329
  - BrickDataName, 329
  - BrickDataTimeoutValue, 329



BTConnectionClass, 330  
BTConnectionHandleNum, 330  
BTConnectionLinkQuality, 330  
BTConnectionName, 331  
BTConnectionPinCode, 331  
BTConnectionStreamStatus, 331  
BTDeviceClass, 332  
BTDeviceCount, 332  
BTDeviceName, 332  
BTDeviceNameCount, 332  
BTDeviceStatus, 333  
BTInputBufferInPtr, 333  
BTInputBufferOutPtr, 333  
BTOutputBufferInPtr, 333  
BTOutputBufferOutPtr, 334  
GetBrickDataAddress, 334  
GetBTConnectionAddress, 334  
GetBTDeviceAddress, 334  
GetBTInputBuffer, 335  
GetBTOutputBuffer, 335  
GetHSInputBuffer, 335  
GetHSOutputBuffer, 336  
GetUSBInputBuffer, 336  
GetUSBOutputBuffer, 336  
GetUSBPollBuffer, 336  
HSFlags, 337  
HSInputBufferInPtr, 337  
HSInputBufferOutPtr, 337  
HSMode, 337  
HSOutputBufferInPtr, 338  
HSOutputBufferOutPtr, 338  
HSSpeed, 338  
HSState, 338  
ReceiveMessage, 339  
ReceiveRemoteBool, 339  
ReceiveRemoteMessageEx, 339  
ReceiveRemoteNumber, 340  
ReceiveRemoteString, 340  
RemoteKeepAlive, 341  
RemoteMessageRead, 341  
RemoteMessageWrite, 341  
RemotePlaySoundFile, 342  
RemotePlayTone, 342  
RemoteResetMotorPosition, 342  
RemoteResetScaledValue, 343  
RemoteSetInputMode, 343  
RemoteSetOutputState, 343  
RemoteStartProgram, 344  
RemoteStopProgram, 344  
RemoteStopSound, 345  
RS485Control, 345  
RS485DataAvailable, 345  
RS485Exit, 346  
RS485Init, 346  
RS485Read, 346  
RS485SendingData, 347  
RS485Status, 347  
RS485Uart, 347  
RS485Write, 348  
SendMessage, 348  
SendRemoteBool, 349  
SendRemoteNumber, 349  
SendRemoteString, 349  
SendResponseBool, 350  
SendResponseNumber, 350  
SendResponseString, 350  
SendRS485Bool, 351  
SendRS485Number, 351  
SendRS485String, 351  
SetBTInputBuffer, 352  
SetBTInputBufferInPtr, 352  
SetBTInputBufferOutPtr, 352  
SetBTOutputBuffer, 353  
SetBTOutputBufferInPtr, 353  
SetBTOutputBufferOutPtr, 353  
SetHSFlags, 353  
SetHSInputBuffer, 353  
SetHSInputBufferInPtr, 354  
SetHSInputBufferOutPtr, 354  
SetHSMode, 354  
SetHSOutputBuffer, 354  
SetHSOutputBufferInPtr, 355  
SetHSOutputBufferOutPtr, 355  
SetHSSpeed, 355  
SetHSState, 355  
SetUSBInputBuffer, 356  
SetUSBInputBufferInPtr, 356  
SetUSBInputBufferOutPtr, 356  
SetUSBOutputBuffer, 356  
SetUSBOutputBufferInPtr, 357  
SetUSBOutputBufferOutPtr, 357  
SetUSBPollBuffer, 357  
SetUSBPollBufferInPtr, 357  
SetUSBPollBufferOutPtr, 358  
SetUSBState, 358  
SysCommBTCheckStatus, 358  
SysCommBTConnection, 358  
SysCommBTOnOff, 359  
SysCommBTWrite, 359  
SysCommExecuteFunction, 359  
SysCommHSCheckStatus, 360  
SysCommHSControl, 360  
SysCommHSRead, 360  
SysCommHSWrite, 361  
SysMessageRead, 361  
SysMessageWrite, 361  
USBInputBufferInPtr, 361  
USBInputBufferOutPtr, 362

- USBOutputBufferInPtr, [362](#)
- USBOutputBufferOutPtr, [362](#)
- USBPollBufferInPtr, [362](#)
- USBPollBufferOutPtr, [363](#)
- USBState, [363](#)
- CommModuleID
  - ModuleIDConstants, [179](#)
  - NBCCCommon.h, [855](#)
- CommModuleName
  - ModuleNameConstants, [177](#)
  - NBCCCommon.h, [855](#)
- CommOffsetBrickDataBdAddr
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [855](#)
- CommOffsetBrickDataBluecoreVersion
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [855](#)
- CommOffsetBrickDataBtHwStatus
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [855](#)
- CommOffsetBrickDataBtStateStatus
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBrickDataName
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBrickDataTimeOutValue
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableBdAddr
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableClassOfDevice
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableHandleNr
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableLinkQuality
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableName
  - CommIOMAP, [636](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTablePinCode
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtConnectTableStreamStatus
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtDeviceCnt
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [856](#)
- CommOffsetBtDeviceNameCnt
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtDeviceTableBdAddr
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtDeviceTableClassOfDevice
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtDeviceTableDeviceStatus
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtDeviceTableName
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtInBufBuf
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtInBufInPtr
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtInBufOutPtr
  - CommIOMAP, [637](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtOutBufBuf
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtOutBufInPtr
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [857](#)
- CommOffsetBtOutBufOutPtr
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [857](#)
- CommOffsetHsFlags
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsInBufBuf
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsInBufInPtr
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsInBufOutPtr
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsMode
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsOutBufBuf
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsOutBufInPtr
  - CommIOMAP, [638](#)
  - NBCCCommon.h, [858](#)
- CommOffsetHsOutBufOutPtr



- CommIOMAP, 638
- NBCCCommon.h, 858
- CommOffsetHsSpeed
  - CommIOMAP, 639
  - NBCCCommon.h, 858
- CommOffsetHsState
  - CommIOMAP, 639
  - NBCCCommon.h, 858
- CommOffsetPFunc
  - CommIOMAP, 639
  - NBCCCommon.h, 858
- CommOffsetPFuncTwo
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbInBufBuf
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbInBufInPtr
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbInBufOutPtr
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbOutBufBuf
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbOutBufInPtr
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbOutBufOutPtr
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbPollBufBuf
  - CommIOMAP, 639
  - NBCCCommon.h, 859
- CommOffsetUsbPollBufInPtr
  - CommIOMAP, 640
  - NBCCCommon.h, 859
- CommOffsetUsbPollBufOutPtr
  - CommIOMAP, 640
  - NBCCCommon.h, 859
- CommOffsetUsbState
  - CommIOMAP, 640
  - NBCCCommon.h, 859
- CommStatusCodesConstants
  - BT\_CMD\_READY, 633
  - HS\_CMD\_READY, 633
  - LR\_COULD\_NOT\_SAVE, 633
  - LR\_ENTRY\_REMOVED, 633
  - LR\_STORE\_IS\_FULL, 633
  - LR\_SUCCESS, 633
  - LR\_UNKNOWN\_ADDR, 633
  - USB\_CMD\_READY, 634
- Communications specific errors, 492
- ComputeCalibValue
  - NBCCCommon.h, 860
  - SysCallConstants, 469
- ComputeCalibValueType, 739
  - Name, 739
  - RawVal, 739
  - Result, 739
- Connection
  - CommBTCheckStatusType, 721
  - CommBTWriteType, 725
- ConnectionSlot
  - CommBTConnectionType, 722
- Copy
  - cstringAPI, 448
  - NXCDefs.h, 1091
- Cos
  - cmathAPI, 406
  - NXCDefs.h, 1048
- cos
  - cmathAPI, 417
  - NXCDefs.h, 1092
- CosD
  - cmathAPI, 407
  - NXCDefs.h, 1049
- cosd
  - cmathAPI, 418
  - NXCDefs.h, 1092
- Cosh
  - cmathAPI, 407
  - NXCDefs.h, 1049
- cosh
  - cmathAPI, 418
  - NXCDefs.h, 1092
- CoshD
  - cmathAPI, 407
  - NXCDefs.h, 1049
- coshd
  - cmathAPI, 418
  - NXCDefs.h, 1093
- Count
  - IOMapReadByIDType, 780
  - IOMapReadType, 782
  - ReadButtonType, 799
- CreateFile
  - LoaderModuleFunctions, 386
  - NXCDefs.h, 1093
- CreateFileLinear
  - LoaderModuleFunctions, 386
  - NXCDefs.h, 1093
- CreateFileNonLinear
  - LoaderModuleFunctions, 386
  - NXCDefs.h, 1094
- cstdio API, 426
- cstdioAPI

- fclose, 428
- feof, 428
- fflush, 428
- fgetc, 429
- fgets, 429
- fopen, 429
- fprintf, 430
- fputc, 430
- fputs, 431
- fseek, 431
- ftell, 431
- getc, 427
- getchar, 432
- printf, 432
- putc, 427
- remove, 432
- rename, 433
- rewind, 433
- sprintf, 433
- cstdlib API, 436
- cstdlib API types, 444
- cstdlibAPI
  - abort, 437
  - abs, 437
  - atof, 437
  - atoi, 438
  - atol, 438
  - div, 439
  - labs, 439
  - ldiv, 439
  - rand, 440
  - Random, 440
  - strtod, 440
  - strtol, 441
  - strtoul, 442
  - SysRandomNumber, 442
- cstring API, 445
- cstringAPI
  - addr, 447
  - address, 447
  - ByteArrayToStr, 447
  - ByteArrayToStrEx, 448
  - Copy, 448
  - Flatten, 448
  - FlattenVar, 449
  - FormatNum, 449
  - LeftStr, 449
  - memcmp, 450
  - memcpy, 450
  - memmove, 450
  - MidStr, 451
  - NumToStr, 451
  - reladdr, 451
  - RightStr, 452
  - StrCat, 453
  - strcat, 452
  - strcmp, 453
  - strcpy, 453
  - StrIndex, 454
  - StrLen, 454
  - strlen, 454
  - strncat, 455
  - strncmp, 455
  - strncpy, 455
  - StrReplace, 456
  - StrToByteArray, 456
  - StrToNum, 456
  - SubStr, 457
  - UnflattenVar, 457
- cctype API, 459
- cctypeAPI
  - isalnum, 460
  - isalpha, 460
  - isctrl, 460
  - isdigit, 460
  - isgraph, 461
  - islower, 461
  - isprint, 461
  - ispunct, 462
  - isspace, 462
  - isupper, 462
  - isxdigit, 463
  - tolower, 463
  - toupper, 463
- CurrentTick
  - CommandModuleFunctions, 306
  - NXCDefs.h, 1094
- CustomSensorActiveStatus
  - InputModuleFunctions, 211
  - NXCDefs.h, 1095
- CustomSensorPercentFullScale
  - InputModuleFunctions, 211
  - NXCDefs.h, 1095
- CustomSensorZeroOffset
  - InputModuleFunctions, 212
  - NXCDefs.h, 1095
- Data
  - DrawGraphicArrayType, 752
- Data type limits, 711
- DataAvailable
  - CommHSCheckStatusType, 728
- DatalogGetTimes
  - NBCCCommon.h, 860
  - SysCallConstants, 470
- DatalogGetTimesType, 741
  - SyncTick, 741
  - SyncTime, 741

- DatalogWrite
  - NBCCCommon.h, 860
  - SysCallConstants, 470
- DatalogWriteType, 742
  - Message, 742
  - Result, 742
- DEGREES\_PER\_RADIAN
  - MiscConstants, 181
  - NBCCCommon.h, 860
- DeleteFile
  - LoaderModuleFunctions, 387
  - NXCDefs.h, 1096
- Device status constants, 629
- Display contrast constants, 606
- Display flags, 605
- Display module, 62
- Display module constants, 593
- Display module functions, 248
- Display module IOMAP offsets, 609
- Display module types, 246
- DISPLAY\_BUSY
  - DisplayFlagsGroup, 605
  - NBCCCommon.h, 860
- DISPLAY\_CHAR
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 860
- DISPLAY\_CONTRAST\_DEFAULT
  - DisplayContrastConstants, 606
  - NBCCCommon.h, 860
- DISPLAY\_CONTRAST\_MAX
  - DisplayContrastConstants, 606
  - NBCCCommon.h, 860
- DISPLAY\_ERASE\_ALL
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 860
- DISPLAY\_ERASE\_LINE
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 861
- DISPLAY\_FILL\_REGION
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 861
- DISPLAY\_FRAME
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 861
- DISPLAY\_HEIGHT
  - DisplayModuleConstants, 594
  - NBCCCommon.h, 861
- DISPLAY\_HORIZONTAL\_LINE
  - DisplayExecuteFunctionConstants, 598
  - NBCCCommon.h, 861
- DISPLAY\_MENUICONS\_X\_DIFF
  - DisplayModuleConstants, 595
  - NBCCCommon.h, 861
- DISPLAY\_MENUICONS\_X\_OFFS
  - DisplayModuleConstants, 595
  - NBCCCommon.h, 861
- DISPLAY\_MENUICONS\_Y
  - DisplayModuleConstants, 595
  - NBCCCommon.h, 861
- DISPLAY\_ON
  - DisplayFlagsGroup, 605
  - NBCCCommon.h, 861
- DISPLAY\_PIXEL
  - DisplayExecuteFunctionConstants, 599
  - NBCCCommon.h, 861
- DISPLAY\_POPUP
  - DisplayFlagsGroup, 605
  - NBCCCommon.h, 861
- DISPLAY\_REFRESH
  - DisplayFlagsGroup, 605
  - NBCCCommon.h, 862
- DISPLAY\_REFRESH\_DISABLED
  - DisplayFlagsGroup, 605
  - NBCCCommon.h, 862
- DISPLAY\_VERTICAL\_LINE
  - DisplayExecuteFunctionConstants, 599
  - NBCCCommon.h, 862
- DISPLAY\_WIDTH
  - DisplayModuleConstants, 595
  - NBCCCommon.h, 862
- DisplayContrast
  - DisplayModuleFunctions, 251
  - NXCDefs.h, 1096
- DisplayContrastConstants
  - DISPLAY\_CONTRAST\_DEFAULT, 606
  - DISPLAY\_CONTRAST\_MAX, 606
- DisplayDisplay
  - DisplayModuleFunctions, 252
  - NXCDefs.h, 1096
- DisplayDrawOptionConstants
  - DRAW\_OPT\_CLEAR, 600
  - DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_-  
SCREEN, 600
  - DRAW\_OPT\_CLEAR\_PIXELS, 601
  - DRAW\_OPT\_CLEAR\_SCREEN\_MODES,  
601
  - DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN,  
601
  - DRAW\_OPT\_FILL\_SHAPE, 601
  - DRAW\_OPT\_INVERT, 601
  - DRAW\_OPT\_LOGICAL\_AND, 601
  - DRAW\_OPT\_LOGICAL\_COPY, 601
  - DRAW\_OPT\_LOGICAL\_OPERATIONS,  
601
  - DRAW\_OPT\_LOGICAL\_OR, 602
  - DRAW\_OPT\_LOGICAL\_XOR, 602
  - DRAW\_OPT\_NORMAL, 602
  - DRAW\_OPT\_POLYGON\_POLYLINE, 602

- DisplayEraseMask
  - DisplayModuleFunctions, 252
  - NXCDefs.h, 1096
- DisplayExecuteFunction
  - NBCCCommon.h, 862
  - SysCallConstants, 470
- DisplayExecuteFunction constants, 598
- DisplayExecuteFunctionConstants
  - DISPLAY\_CHAR, 598
  - DISPLAY\_ERASE\_ALL, 598
  - DISPLAY\_ERASE\_LINE, 598
  - DISPLAY\_FILL\_REGION, 598
  - DISPLAY\_FRAME, 598
  - DISPLAY\_HORIZONTAL\_LINE, 598
  - DISPLAY\_PIXEL, 599
  - DISPLAY\_VERTICAL\_LINE, 599
- DisplayExecuteFunctionType, 743
  - Cmd, 743
  - On, 743
  - Status, 744
  - X1, 744
  - X2, 744
  - Y1, 744
  - Y2, 744
- DisplayFlags
  - DisplayModuleFunctions, 252
  - NXCDefs.h, 1097
- DisplayFlagsGroup
  - DISPLAY\_BUSY, 605
  - DISPLAY\_ON, 605
  - DISPLAY\_POPUP, 605
  - DISPLAY\_REFRESH, 605
  - DISPLAY\_REFRESH\_DISABLED, 605
- DisplayFont
  - DisplayModuleFunctions, 252
  - NXCDefs.h, 1097
- DisplayFontDrawOptionConstants
  - DRAW\_OPT\_FONT\_DIR\_B2TL, 603
  - DRAW\_OPT\_FONT\_DIR\_B2TR, 603
  - DRAW\_OPT\_FONT\_DIR\_L2RB, 603
  - DRAW\_OPT\_FONT\_DIR\_L2RT, 603
  - DRAW\_OPT\_FONT\_DIR\_R2LB, 604
  - DRAW\_OPT\_FONT\_DIR\_R2LT, 604
  - DRAW\_OPT\_FONT\_DIR\_T2BL, 604
  - DRAW\_OPT\_FONT\_DIR\_T2BR, 604
  - DRAW\_OPT\_FONT DIRECTIONS, 604
  - DRAW\_OPT\_FONT\_WRAP, 604
- DisplayIOMAP
  - DisplayOffsetContrast, 609
  - DisplayOffsetDisplay, 609
  - DisplayOffsetEraseMask, 609
  - DisplayOffsetFlags, 609
  - DisplayOffsetNormal, 609
  - DisplayOffsetPBitmaps, 610
  - DisplayOffsetPFont, 610
  - DisplayOffsetPFunc, 610
  - DisplayOffsetPMenuIcons, 610
  - DisplayOffsetPMenuText, 610
  - DisplayOffsetPopup, 610
  - DisplayOffsetPScreens, 610
  - DisplayOffsetPStatusIcons, 610
  - DisplayOffsetPStatusText, 610
  - DisplayOffsetPStepIcons, 610
  - DisplayOffsetPTextLines, 610
  - DisplayOffsetStatusIcons, 611
  - DisplayOffsetStepIcons, 611
  - DisplayOffsetTextLinesCenterFlags, 611
  - DisplayOffsetUpdateMask, 611
- DisplayModuleConstants
  - BITMAP\_1, 594
  - BITMAP\_2, 594
  - BITMAP\_3, 594
  - BITMAP\_4, 594
  - BITMAPS, 594
  - DISPLAY\_HEIGHT, 594
  - DISPLAY\_MENUICONS\_X\_DIFF, 595
  - DISPLAY\_MENUICONS\_X\_OFFS, 595
  - DISPLAY\_MENUICONS\_Y, 595
  - DISPLAY\_WIDTH, 595
  - FRAME\_SELECT, 595
  - MENUICON\_CENTER, 595
  - MENUICON\_LEFT, 595
  - MENUICON\_RIGHT, 595
  - MENUICONS, 595
  - MENUTEXT, 595
  - SCREEN\_BACKGROUND, 595
  - SCREEN\_LARGE, 596
  - SCREEN\_MODE\_CLEAR, 596
  - SCREEN\_MODE\_RESTORE, 596
  - SCREEN\_SMALL, 596
  - SCREENS, 596
  - SPECIALS, 596
  - STATUSICON\_BATTERY, 596
  - STATUSICON\_BLUETOOTH, 596
  - STATUSICON\_USB, 596
  - STATUSICON\_VM, 596
  - STATUSICONS, 597
  - STATUSTEXT, 597
  - STEPICON\_1, 597
  - STEPICON\_2, 597
  - STEPICON\_3, 597
  - STEPICON\_4, 597
  - STEPICON\_5, 597
  - STEPICONS, 597
  - STEPLINE, 597
  - TOPLINE, 597
- DisplayModuleFunctions
  - CircleOut, 251

- ClearLine, 251
- ClearScreen, 251
- DisplayContrast, 251
- DisplayDisplay, 252
- DisplayEraseMask, 252
- DisplayFlags, 252
- DisplayFont, 252
- DisplayTextLinesCenterFlags, 253
- DisplayUpdateMask, 253
- EllipseOut, 253
- FontNumOut, 254
- FontTextOut, 254
- GetDisplayNormal, 255
- GetDisplayPopup, 255
- GraphicArrayOut, 256
- GraphicArrayOutEx, 256
- GraphicOut, 257
- GraphicOutEx, 257
- LineOut, 258
- NumOut, 258
- PointOut, 259
- PolyOut, 259
- RectOut, 260
- ResetScreen, 260
- SetDisplayContrast, 261
- SetDisplayDisplay, 261
- SetDisplayEraseMask, 261
- SetDisplayFlags, 261
- SetDisplayFont, 262
- SetDisplayNormal, 262
- SetDisplayPopup, 262
- SetDisplayTextLinesCenterFlags, 263
- SetDisplayUpdateMask, 263
- SysDisplayExecuteFunction, 263
- SysDrawCircle, 263
- SysDrawEllipse, 264
- SysDrawFont, 264
- SysDrawGraphic, 264
- SysDrawGraphicArray, 265
- SysDrawLine, 265
- SysDrawPoint, 265
- SysDrawPolygon, 265
- SysDrawRect, 266
- SysDrawText, 266
- SysSetScreenMode, 266
- TextOut, 266
- DisplayModuleID
  - ModuleIDConstants, 179
  - NBCCCommon.h, 862
- DisplayModuleName
  - ModuleNameConstants, 177
  - NBCCCommon.h, 862
- DisplayOffsetContrast
  - DisplayIOMAP, 609
  - NBCCCommon.h, 862
- DisplayOffsetDisplay
  - DisplayIOMAP, 609
  - NBCCCommon.h, 862
- DisplayOffsetEraseMask
  - DisplayIOMAP, 609
  - NBCCCommon.h, 862
- DisplayOffsetFlags
  - DisplayIOMAP, 609
  - NBCCCommon.h, 863
- DisplayOffsetNormal
  - DisplayIOMAP, 609
  - NBCCCommon.h, 863
- DisplayOffsetPBitmaps
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPFont
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPFunc
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPMenuIcons
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPMenuText
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPopup
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPScreens
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPStatusIcons
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPStatusText
  - DisplayIOMAP, 610
  - NBCCCommon.h, 863
- DisplayOffsetPStepIcons
  - DisplayIOMAP, 610
  - NBCCCommon.h, 864
- DisplayOffsetPTextLines
  - DisplayIOMAP, 610
  - NBCCCommon.h, 864
- DisplayOffsetStatusIcons
  - DisplayIOMAP, 611
  - NBCCCommon.h, 864
- DisplayOffsetStepIcons
  - DisplayIOMAP, 611
  - NBCCCommon.h, 864
- DisplayOffsetTextLinesCenterFlags
  - DisplayIOMAP, 611

NBCCCommon.h, [864](#)  
 DisplayOffsetUpdateMask  
   DisplayIOMAP, [611](#)  
   NBCCCommon.h, [864](#)  
 DisplayTextLineConstants  
   TEXTLINE\_1, [607](#)  
   TEXTLINE\_2, [607](#)  
   TEXTLINE\_3, [607](#)  
   TEXTLINE\_4, [607](#)  
   TEXTLINE\_5, [607](#)  
   TEXTLINE\_6, [608](#)  
   TEXTLINE\_7, [608](#)  
   TEXTLINE\_8, [608](#)  
   TEXTLINES, [608](#)  
 DisplayTextLinesCenterFlags  
   DisplayModuleFunctions, [253](#)  
   NXCDefs.h, [1097](#)  
 DisplayUpdateMask  
   DisplayModuleFunctions, [253](#)  
   NXCDefs.h, [1097](#)  
 DIST\_CMD\_CUSTOM  
   MSDistNX, [702](#)  
   NBCCCommon.h, [864](#)  
 DIST\_CMD\_GP2D12  
   MSDistNX, [702](#)  
   NBCCCommon.h, [864](#)  
 DIST\_CMD\_GP2D120  
   MSDistNX, [702](#)  
   NBCCCommon.h, [864](#)  
 DIST\_CMD\_GP2YA02  
   MSDistNX, [702](#)  
   NBCCCommon.h, [864](#)  
 DIST\_CMD\_GP2YA21  
   MSDistNX, [702](#)  
   NBCCCommon.h, [864](#)  
 DIST\_REG\_DIST  
   MSDistNX, [702](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_DIST1  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_DIST\_MAX  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_DIST\_MIN  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_MODULE\_TYPE  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_NUM\_POINTS  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DIST\_REG\_VOLT

MSDistNX, [703](#)  
 NBCCCommon.h, [865](#)  
 DIST\_REG\_VOLT1  
   MSDistNX, [703](#)  
   NBCCCommon.h, [865](#)  
 DISTNxDistance  
   MindSensorsAPI, [112](#)  
   NXCDefs.h, [1098](#)  
 DISTNxDistanceEx  
   MindSensorsAPI, [112](#)  
   NXCDefs.h, [1098](#)  
 DISTNxGP2D12  
   MindSensorsAPI, [113](#)  
   NXCDefs.h, [1098](#)  
 DISTNxGP2D120  
   MindSensorsAPI, [113](#)  
   NXCDefs.h, [1099](#)  
 DISTNxGP2D120Ex  
   MindSensorsAPI, [113](#)  
   NXCDefs.h, [1099](#)  
 DISTNxGP2D12Ex  
   MindSensorsAPI, [114](#)  
   NXCDefs.h, [1099](#)  
 DISTNxGP2YA02  
   MindSensorsAPI, [114](#)  
   NXCDefs.h, [1100](#)  
 DISTNxGP2YA02Ex  
   MindSensorsAPI, [114](#)  
   NXCDefs.h, [1100](#)  
 DISTNxGP2YA21  
   MindSensorsAPI, [115](#)  
   NXCDefs.h, [1100](#)  
 DISTNxGP2YA21Ex  
   MindSensorsAPI, [115](#)  
   NXCDefs.h, [1101](#)  
 DISTNxMaxDistance  
   MindSensorsAPI, [115](#)  
   NXCDefs.h, [1101](#)  
 DISTNxMaxDistanceEx  
   MindSensorsAPI, [116](#)  
   NXCDefs.h, [1101](#)  
 DISTNxMinDistance  
   MindSensorsAPI, [116](#)  
   NXCDefs.h, [1102](#)  
 DISTNxMinDistanceEx  
   MindSensorsAPI, [116](#)  
   NXCDefs.h, [1102](#)  
 DISTNxModuleType  
   MindSensorsAPI, [117](#)  
   NXCDefs.h, [1102](#)  
 DISTNxModuleTypeEx  
   MindSensorsAPI, [117](#)  
   NXCDefs.h, [1103](#)  
 DISTNxNumPoints

- MindSensorsAPI, [117](#)
- NXCDefs.h, [1103](#)
- DISTNxNumPointsEx
  - MindSensorsAPI, [118](#)
  - NXCDefs.h, [1103](#)
- DISTNxVoltage
  - MindSensorsAPI, [118](#)
  - NXCDefs.h, [1104](#)
- DISTNxVoltageEx
  - MindSensorsAPI, [118](#)
  - NXCDefs.h, [1104](#)
- div
  - cstdlibAPI, [439](#)
  - NXCDefs.h, [1104](#)
- div\_t, [745](#)
  - quot, [745](#)
  - rem, [745](#)
- DRAW\_OPT\_CLEAR
  - DisplayDrawOptionConstants, [600](#)
  - NBCCCommon.h, [865](#)
- DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_-  
SCREEN
  - DisplayDrawOptionConstants, [600](#)
  - NBCCCommon.h, [865](#)
- DRAW\_OPT\_CLEAR\_PIXELS
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [865](#)
- DRAW\_OPT\_CLEAR\_SCREEN\_MODES
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FILL\_SHAPE
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_B2TL
  - DisplayFontDrawOptionConstants, [603](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_B2TR
  - DisplayFontDrawOptionConstants, [603](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_L2RB
  - DisplayFontDrawOptionConstants, [603](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_L2RT
  - DisplayFontDrawOptionConstants, [603](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_R2LB
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [866](#)
- DRAW\_OPT\_FONT\_DIR\_R2LT
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_FONT\_DIR\_T2BL
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_FONT\_DIR\_T2BR
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_FONT\_DIRECTIONS
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_FONT\_WRAP
  - DisplayFontDrawOptionConstants, [604](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_INVERT
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_LOGICAL\_AND
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_LOGICAL\_COPY
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [867](#)
- DRAW\_OPT\_LOGICAL\_OPERATIONS
  - DisplayDrawOptionConstants, [601](#)
  - NBCCCommon.h, [868](#)
- DRAW\_OPT\_LOGICAL\_OR
  - DisplayDrawOptionConstants, [602](#)
  - NBCCCommon.h, [868](#)
- DRAW\_OPT\_LOGICAL\_XOR
  - DisplayDrawOptionConstants, [602](#)
  - NBCCCommon.h, [868](#)
- DRAW\_OPT\_NORMAL
  - DisplayDrawOptionConstants, [602](#)
  - NBCCCommon.h, [868](#)
- DRAW\_OPT\_POLYGON\_POLYLINE
  - DisplayDrawOptionConstants, [602](#)
  - NBCCCommon.h, [868](#)
- DrawCircle
  - NBCCCommon.h, [868](#)
  - SysCallConstants, [470](#)
- DrawCircleType, [746](#)
  - Center, [746](#)
  - Options, [746](#)
  - Result, [746](#)
  - Size, [746](#)
- DrawEllipse
  - NBCCCommon.h, [868](#)
  - SysCallConstants, [470](#)
- DrawEllipseType, [748](#)
  - Center, [748](#)
  - Options, [748](#)
  - Result, [748](#)
  - SizeX, [748](#)
  - SizeY, [749](#)
- DrawFont



- NBCCCommon.h, 868
- SysCallConstants, 470
- DrawFontType, 750
  - Filename, 750
  - Location, 750
  - Options, 750
  - Result, 750
  - Text, 751
- DrawGraphic
  - NBCCCommon.h, 868
  - SysCallConstants, 470
- DrawGraphicArray
  - NBCCCommon.h, 869
  - SysCallConstants, 470
- DrawGraphicArrayType, 752
  - Data, 752
  - Location, 752
  - Options, 752
  - Result, 753
  - Variables, 753
- DrawGraphicType, 754
  - Filename, 754
  - Location, 754
  - Options, 754
  - Result, 755
  - Variables, 755
- Drawing option constants, 600
- DrawLine
  - NBCCCommon.h, 869
  - SysCallConstants, 470
- DrawLineType, 756
  - EndLoc, 756
  - Options, 756
  - Result, 756
  - StartLoc, 756
- DrawPoint
  - NBCCCommon.h, 869
  - SysCallConstants, 470
- DrawPointType, 758
  - Location, 758
  - Options, 758
  - Result, 758
- DrawPolygon
  - NBCCCommon.h, 869
  - SysCallConstants, 470
- DrawPolygonType, 759
  - Options, 759
  - Points, 759
  - Result, 759
- DrawRect
  - NBCCCommon.h, 869
  - SysCallConstants, 471
- DrawRectType, 760
  - Location, 760
  - Options, 760
  - Result, 760
  - Size, 760
- DrawText
  - NBCCCommon.h, 869
  - SysCallConstants, 471
- DrawTextType, 762
  - Location, 762
  - Options, 762
  - Result, 762
  - Text, 762
- Duration
  - SoundPlayToneType, 808
  - Tone, 811
- EllipseOut
  - DisplayModuleFunctions, 253
  - NXCDefs.h, 1105
- EndLoc
  - DrawLineType, 756
- EOF
  - LoaderModuleConstants, 500
  - NBCCCommon.h, 869
- ERR\_ARG
  - CommandFatalErrors, 488
  - NBCCCommon.h, 869
- ERR\_BAD\_POOL\_SIZE
  - CommandFatalErrors, 488
  - NBCCCommon.h, 869
- ERR\_BAD\_PTR
  - CommandFatalErrors, 488
  - NBCCCommon.h, 869
- ERR\_CLUMP\_COUNT
  - CommandFatalErrors, 488
  - NBCCCommon.h, 870
- ERR\_COMM\_BUFFER\_FULL
  - CommandCommErrors, 492
  - NBCCCommon.h, 870
- ERR\_COMM\_BUS\_ERR
  - CommandCommErrors, 492
  - NBCCCommon.h, 870
- ERR\_COMM\_CHAN\_INVALID
  - CommandCommErrors, 492
  - NBCCCommon.h, 870
- ERR\_COMM\_CHAN\_NOT\_READY
  - CommandCommErrors, 492
  - NBCCCommon.h, 870
- ERR\_DEFAULT\_OFFSETS
  - CommandFatalErrors, 488
  - NBCCCommon.h, 870
- ERR\_FILE
  - CommandFatalErrors, 489
  - NBCCCommon.h, 870
- ERR\_INSANE\_OFFSET



- CommandFatalErrors, [489](#)
- NBCCCommon.h, [870](#)
- ERR\_INSTR
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [870](#)
- ERR\_INVALID\_FIELD
  - CommandGenErrors, [491](#)
  - NBCCCommon.h, [870](#)
- ERR\_INVALID\_PORT
  - CommandGenErrors, [491](#)
  - NBCCCommon.h, [870](#)
- ERR\_INVALID\_QUEUE
  - CommandGenErrors, [491](#)
  - NBCCCommon.h, [871](#)
- ERR\_INVALID\_SIZE
  - CommandGenErrors, [491](#)
  - NBCCCommon.h, [871](#)
- ERR\_LOADER\_ERR
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_MEM
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_MEMMGR\_FAIL
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_NO\_ACTIVE\_CLUMP
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_NO\_CODE
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_NO\_PROG
  - CommandGenErrors, [491](#)
  - NBCCCommon.h, [871](#)
- ERR\_NON\_FATAL
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [871](#)
- ERR\_RC\_BAD\_PACKET
  - CommandRCErrors, [493](#)
  - NBCCCommon.h, [871](#)
- ERR\_RC\_FAILED
  - CommandRCErrors, [493](#)
  - NBCCCommon.h, [871](#)
- ERR\_RC\_ILLEGAL\_VAL
  - CommandRCErrors, [493](#)
  - NBCCCommon.h, [872](#)
- ERR\_RC\_UNKNOWN\_CMD
  - CommandRCErrors, [493](#)
  - NBCCCommon.h, [872](#)
- ERR\_SPOTCHECK\_FAIL
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [872](#)
- ERR\_VER
  - CommandFatalErrors, [489](#)
  - NBCCCommon.h, [872](#)
- ExitTo
  - CommandModuleFunctions, [306](#)
  - NXCDefs.h, [1105](#)
- Exp
  - cmathAPI, [408](#)
  - NXCDefs.h, [1050](#)
- exp
  - cmathAPI, [419](#)
  - NXCDefs.h, [1105](#)
- FALSE
  - MiscConstants, [181](#)
  - NBCCCommon.h, [872](#)
- Fatal errors, [488](#)
- fclose
  - cstdioAPI, [428](#)
  - NXCDefs.h, [1106](#)
- feof
  - cstdioAPI, [428](#)
  - NXCDefs.h, [1106](#)
- fflush
  - cstdioAPI, [428](#)
  - NXCDefs.h, [1106](#)
- fgetc
  - cstdioAPI, [429](#)
  - NXCDefs.h, [1107](#)
- fgets
  - cstdioAPI, [429](#)
  - NXCDefs.h, [1107](#)
- FileClose
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileCloseType, [764](#)
  - FileHandle, [764](#)
  - Result, [764](#)
- FileDelete
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileDeleteType, [765](#)
  - Filename, [765](#)
  - Result, [765](#)
- FileFindFirst
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileFindNext
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileFindType, [766](#)
  - FileHandle, [766](#)
  - Filename, [766](#)
  - Length, [766](#)
  - Result, [766](#)

- FileHandle
  - FileCloseType, [764](#)
  - FileFindType, [766](#)
  - FileOpenType, [768](#)
  - FileReadWriteType, [770](#)
  - FileResizeType, [774](#)
  - FileResolveHandleType, [775](#)
  - FileSeekType, [777](#)
- FileList
  - ListFilesType, [790](#)
- Filename
  - DrawFontType, [750](#)
  - DrawGraphicType, [754](#)
  - FileDeleteType, [765](#)
  - FileFindType, [766](#)
  - FileOpenType, [768](#)
  - FileResolveHandleType, [775](#)
  - LoaderExecuteFunctionType, [793](#)
  - SoundPlayFileType, [806](#)
- FileOpenAppend
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileOpenRead
  - NBCCCommon.h, [872](#)
  - SysCallConstants, [471](#)
- FileOpenReadLinear
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [471](#)
- FileOpenType, [768](#)
  - FileHandle, [768](#)
  - Filename, [768](#)
  - Length, [768](#)
  - Result, [769](#)
- FileOpenWrite
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [471](#)
- FileOpenWriteLinear
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [471](#)
- FileOpenWriteNonLinear
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileRead
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileReadWriteType, [770](#)
  - Buffer, [770](#)
  - FileHandle, [770](#)
  - Length, [770](#)
  - Result, [770](#)
- FileRename
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileRenameType, [772](#)
  - NewFilename, [772](#)
  - OldFilename, [772](#)
  - Result, [772](#)
- FileResize
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileResizeType, [774](#)
  - FileHandle, [774](#)
  - NewSize, [774](#)
  - Result, [774](#)
- FileResolveHandle
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileResolveHandleType, [775](#)
  - FileHandle, [775](#)
  - Filename, [775](#)
  - Result, [775](#)
  - WriteHandle, [775](#)
- FileSeek
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FileSeekType, [777](#)
  - FileHandle, [777](#)
  - Length, [777](#)
  - Origin, [777](#)
  - Result, [777](#)
- FileWrite
  - NBCCCommon.h, [873](#)
  - SysCallConstants, [472](#)
- FindFirstFile
  - LoaderModuleFunctions, [387](#)
  - NXCDefs.h, [1107](#)
- FindNextFile
  - LoaderModuleFunctions, [388](#)
  - NXCDefs.h, [1108](#)
- FirstTick
  - CommandModuleFunctions, [306](#)
  - NXCDefs.h, [1108](#)
- Flags
  - SoundGetStateType, [805](#)
  - SoundSetStateType, [810](#)
- Flatten
  - cstringAPI, [448](#)
  - NXCDefs.h, [1108](#)
- FlattenVar
  - cstringAPI, [449](#)
  - NXCDefs.h, [1109](#)
- Float
  - NXCDefs.h, [1109](#)
  - OutputModuleFunctions, [229](#)
- Floor
  - cmathAPI, [408](#)
  - NXCDefs.h, [1050](#)
- floor

- cmathAPI, 419
- NXCDefs.h, 1109
- Follows
  - CommandModuleFunctions, 306
  - NXCDefs.h, 1110
- Font drawing option constants, 603
- FontNumOut
  - DisplayModuleFunctions, 254
  - NXCDefs.h, 1110
- FontTextOut
  - DisplayModuleFunctions, 254
  - NXCDefs.h, 1111
- fopen
  - cstdioAPI, 429
  - NXCDefs.h, 1111
- ForceOff
  - NXCDefs.h, 1112
  - UiModuleFunctions, 375
- FormatNum
  - cstringAPI, 449
  - NXCDefs.h, 1112
- fprintf
  - cstdioAPI, 430
  - NXCDefs.h, 1112
- fputc
  - cstdioAPI, 430
  - NXCDefs.h, 1113
- fputs
  - cstdioAPI, 431
  - NXCDefs.h, 1113
- Frac
  - cmathAPI, 408
  - NXCDefs.h, 1050
- frac
  - cmathAPI, 419
  - NXCDefs.h, 1113
- FRAME\_SELECT
  - DisplayModuleConstants, 595
  - NBCCCommon.h, 873
- FreeMemory
  - LoaderModuleFunctions, 388
  - NXCDefs.h, 1114
- Frequency
  - SoundPlayToneType, 808
  - Tone, 811
- FREQUENCY\_MAX
  - NBCCCommon.h, 874
  - SoundMisc, 516
- FREQUENCY\_MIN
  - NBCCCommon.h, 874
  - SoundMisc, 516
- fseek
  - cstdioAPI, 431
  - NXCDefs.h, 1114
- fseek origin constants, 435
- fseekConstants
  - SEEK\_CUR, 435
  - SEEK\_END, 435
  - SEEK\_SET, 435
- ftell
  - cstdioAPI, 431
  - NXCDefs.h, 1114
- General errors, 491
- GenericI2CConstants
  - I2C\_REG\_CMD, 587
  - I2C\_REG\_DEVICE\_ID, 587
  - I2C\_REG\_VENDOR\_ID, 587
  - I2C\_REG\_VERSION, 587
- GetBrickDataAddress
  - CommModuleFunctions, 334
  - NXCDefs.h, 1115
- GetBTConnectionAddress
  - CommModuleFunctions, 334
  - NXCDefs.h, 1115
- GetBTDeviceAddress
  - CommModuleFunctions, 334
  - NXCDefs.h, 1115
- GetBTInputBuffer
  - CommModuleFunctions, 335
  - NXCDefs.h, 1115
- GetBTOutputBuffer
  - CommModuleFunctions, 335
  - NXCDefs.h, 1116
- getc
  - cstdioAPI, 427
  - NXCDefs.h, 1051
- getchar
  - cstdioAPI, 432
  - NXCDefs.h, 1116
- GetDisplayNormal
  - DisplayModuleFunctions, 255
  - NXCDefs.h, 1116
- GetDisplayPopup
  - DisplayModuleFunctions, 255
  - NXCDefs.h, 1117
- GetHSInputBuffer
  - CommModuleFunctions, 335
  - NXCDefs.h, 1117
- GetHSOutputBuffer
  - CommModuleFunctions, 336
  - NXCDefs.h, 1117
- GetInput
  - InputModuleFunctions, 212
  - NXCDefs.h, 1118
- GetLSInputBuffer
  - LowLevelLowSpeedModuleFunctions, 294
  - NXCDefs.h, 1118

- GetLSOutputBuffer
  - LowLevelLowSpeedModuleFunctions, 294
  - NXCDefs.h, 1118
- GetOutput
  - NXCDefs.h, 1119
  - OutputModuleFunctions, 230
- GetStartTick
  - NBCCCommon.h, 874
  - SysCallConstants, 472
- GetStartTickType, 779
  - Result, 779
- GetUSBInputBuffer
  - CommModuleFunctions, 336
  - NXCDefs.h, 1119
- GetUSBOutputBuffer
  - CommModuleFunctions, 336
  - NXCDefs.h, 1119
- GetUSBPollBuffer
  - CommModuleFunctions, 336
  - NXCDefs.h, 1120
- GL\_CAMERA\_DEPTH
  - GLConstantsSettings, 716
  - NBCCCommon.h, 874
- GL\_CIRCLE
  - GLConstantsBeginModes, 713
  - NBCCCommon.h, 874
- GL\_CIRCLE\_SIZE
  - GLConstantsSettings, 716
  - NBCCCommon.h, 874
- GL\_CULL\_BACK
  - GLConstantsCullMode, 717
  - NBCCCommon.h, 874
- GL\_CULL\_FRONT
  - GLConstantsCullMode, 717
  - NBCCCommon.h, 874
- GL\_CULL\_MODE
  - GLConstantsSettings, 716
  - NBCCCommon.h, 874
- GL\_CULL\_NONE
  - GLConstantsCullMode, 717
  - NBCCCommon.h, 874
- GL\_LINE
  - GLConstantsBeginModes, 713
  - NBCCCommon.h, 875
- GL\_POINT
  - GLConstantsBeginModes, 713
  - NBCCCommon.h, 875
- GL\_POLYGON
  - GLConstantsBeginModes, 713
  - NBCCCommon.h, 875
- GL\_ROTATE\_X
  - GLConstantsActions, 714
  - NBCCCommon.h, 875
- GL\_ROTATE\_Y
  - GLConstantsActions, 714
  - NBCCCommon.h, 875
- GL\_ROTATE\_Z
  - GLConstantsActions, 714
  - NBCCCommon.h, 875
- GL\_SCALE\_X
  - GLConstantsActions, 714
  - NBCCCommon.h, 875
- GL\_SCALE\_Y
  - GLConstantsActions, 714
  - NBCCCommon.h, 876
- GL\_SCALE\_Z
  - GLConstantsActions, 715
  - NBCCCommon.h, 876
- GL\_TRANSLATE\_X
  - GLConstantsActions, 715
  - NBCCCommon.h, 876
- GL\_TRANSLATE\_Y
  - GLConstantsActions, 715
  - NBCCCommon.h, 876
- GL\_TRANSLATE\_Z
  - GLConstantsActions, 715
  - NBCCCommon.h, 876
- GL\_ZOOM\_FACTOR
  - GLConstantsSettings, 716
  - NBCCCommon.h, 876
- glAddToAngleX
  - GraphicsLibrary, 186
  - NXCDefs.h, 1120
- glAddToAngleY
  - GraphicsLibrary, 187
  - NXCDefs.h, 1120
- glAddToAngleZ
  - GraphicsLibrary, 187
  - NXCDefs.h, 1120
- glAddVertex
  - GraphicsLibrary, 187
  - NXCDefs.h, 1121
- glBegin
  - GraphicsLibrary, 187
  - NXCDefs.h, 1121
- glBeginObject
  - GraphicsLibrary, 187
  - NXCDefs.h, 1121
- glBeginRender
  - GraphicsLibrary, 188
  - NXCDefs.h, 1121
- glBox
  - GraphicsLibrary, 188
  - NXCDefs.h, 1121
- glCallObject
  - GraphicsLibrary, 188
  - NXCDefs.h, 1122
- GLConstantsActions

- GL\_ROTATE\_X, [714](#)
- GL\_ROTATE\_Y, [714](#)
- GL\_ROTATE\_Z, [714](#)
- GL\_SCALE\_X, [714](#)
- GL\_SCALE\_Y, [714](#)
- GL\_SCALE\_Z, [715](#)
- GL\_TRANSLATE\_X, [715](#)
- GL\_TRANSLATE\_Y, [715](#)
- GL\_TRANSLATE\_Z, [715](#)
- GLConstantsBeginModes
  - GL\_CIRCLE, [713](#)
  - GL\_LINE, [713](#)
  - GL\_POINT, [713](#)
  - GL\_POLYGON, [713](#)
- GLConstantsCullMode
  - GL\_CULL\_BACK, [717](#)
  - GL\_CULL\_FRONT, [717](#)
  - GL\_CULL\_NONE, [717](#)
- GLConstantsSettings
  - GL\_CAMERA\_DEPTH, [716](#)
  - GL\_CIRCLE\_SIZE, [716](#)
  - GL\_CULL\_MODE, [716](#)
  - GL\_ZOOM\_FACTOR, [716](#)
- glCos32768
  - GraphicsLibrary, [188](#)
  - NXCDefs.h, [1122](#)
- glCube
  - GraphicsLibrary, [189](#)
  - NXCDefs.h, [1122](#)
- glEnd
  - GraphicsLibrary, [189](#)
  - NXCDefs.h, [1123](#)
- glEndObject
  - GraphicsLibrary, [189](#)
  - NXCDefs.h, [1123](#)
- glFinishRender
  - GraphicsLibrary, [189](#)
  - NXCDefs.h, [1123](#)
- glInit
  - GraphicsLibrary, [189](#)
  - NXCDefs.h, [1123](#)
- glObjectAction
  - GraphicsLibrary, [190](#)
  - NXCDefs.h, [1123](#)
- glPyramid
  - GraphicsLibrary, [190](#)
  - NXCDefs.h, [1123](#)
- glSet
  - GraphicsLibrary, [190](#)
  - NXCDefs.h, [1124](#)
- glSetAngleX
  - GraphicsLibrary, [190](#)
  - NXCDefs.h, [1124](#)
- glSetAngleY
  - GraphicsLibrary, [191](#)
  - NXCDefs.h, [1124](#)
- glSetAngleZ
  - GraphicsLibrary, [191](#)
  - NXCDefs.h, [1124](#)
- glSin32768
  - GraphicsLibrary, [191](#)
  - NXCDefs.h, [1125](#)
- GraphicArrayOut
  - DisplayModuleFunctions, [256](#)
  - NXCDefs.h, [1125](#)
- GraphicArrayOutEx
  - DisplayModuleFunctions, [256](#)
  - NXCDefs.h, [1125](#)
- GraphicOut
  - DisplayModuleFunctions, [257](#)
  - NXCDefs.h, [1126](#)
- GraphicOutEx
  - DisplayModuleFunctions, [257](#)
  - NXCDefs.h, [1126](#)
- Graphics library actions, [714](#)
- Graphics library begin modes, [713](#)
- Graphics library cull mode, [717](#)
- Graphics library settings, [716](#)
- GraphicsLibrary
  - glAddToAngleX, [186](#)
  - glAddToAngleY, [187](#)
  - glAddToAngleZ, [187](#)
  - glAddVertex, [187](#)
  - glBegin, [187](#)
  - glBeginObject, [187](#)
  - glBeginRender, [188](#)
  - glBox, [188](#)
  - glCallObject, [188](#)
  - glCos32768, [188](#)
  - glCube, [189](#)
  - glEnd, [189](#)
  - glEndObject, [189](#)
  - glFinishRender, [189](#)
  - glInit, [189](#)
  - glObjectAction, [190](#)
  - glPyramid, [190](#)
  - glSet, [190](#)
  - glSetAngleX, [190](#)
  - glSetAngleY, [191](#)
  - glSetAngleZ, [191](#)
  - glSin32768, [191](#)
- Height
  - SizeType, [804](#)
- Hi-speed port baud rate constants, [622](#)
- Hi-speed port combined UART constants, [628](#)
- Hi-speed port constants, [618](#)
- Hi-speed port data bits constants, [625](#)

- Hi-speed port flags constants, [619](#)
- Hi-speed port parity constants, [627](#)
- Hi-speed port state constants, [620](#)
- Hi-speed port stop bits constants, [626](#)
- Hi-speed port SysCommHSControl constants, [621](#)
- HiTechnic API Functions, [63](#)
- HiTechnic Color2 constants, [699](#)
- HiTechnic device constants, [693](#)
- HiTechnic IRReceiver constants, [697](#)
- HiTechnic IRSeeker2 constants, [694](#)
- HiTechnic/mindsensors Power Function/IR Train constants, [679](#)
- HiTechnicAPI
  - HTIRTrain, [70](#)
  - HTPFComboDirect, [70](#)
  - HTPFComboPWM, [70](#)
  - HTPFRawOutput, [71](#)
  - HTPFRepeat, [71](#)
  - HTPFSingleOutputCST, [72](#)
  - HTPFSingleOutputPWM, [72](#)
  - HTPFSinglePin, [73](#)
  - HTPFTrain, [73](#)
  - HTRCXAddToDatalog, [74](#)
  - HTRCXBatteryLevel, [74](#)
  - HTRCXCLEARAllEvents, [74](#)
  - HTRCXCLEARCounter, [74](#)
  - HTRCXCLEARMsg, [75](#)
  - HTRCXCLEARSensor, [75](#)
  - HTRCXCLEARSound, [75](#)
  - HTRCXCLEARTimer, [75](#)
  - HTRCXCreateDatalog, [75](#)
  - HTRCXDecCounter, [76](#)
  - HTRCXDeleteSub, [76](#)
  - HTRCXDeleteSubs, [76](#)
  - HTRCXDeleteTask, [76](#)
  - HTRCXDeleteTasks, [76](#)
  - HTRCXDisableOutput, [77](#)
  - HTRCXEnableOutput, [77](#)
  - HTRCXEvent, [77](#)
  - HTRCXFloat, [77](#)
  - HTRCXFwd, [78](#)
  - HTRCXIncCounter, [78](#)
  - HTRCXInvertOutput, [78](#)
  - HTRCXMuteSound, [78](#)
  - HTRCXObvertOutput, [78](#)
  - HTRCXOff, [79](#)
  - HTRCXOn, [79](#)
  - HTRCXOnFor, [79](#)
  - HTRCXOnFwd, [79](#)
  - HTRCXOnRev, [80](#)
  - HTRCXPBTurnOff, [80](#)
  - HTRCXPing, [80](#)
  - HTRCXPlaySound, [80](#)
  - HTRCXPlayTone, [80](#)
  - HTRCXPlayToneVar, [81](#)
  - HTRCXPoll, [81](#)
  - HTRCXPollMemory, [81](#)
  - HTRCXRemote, [82](#)
  - HTRCXRev, [82](#)
  - HTRCXSelectDisplay, [82](#)
  - HTRCXSelectProgram, [82](#)
  - HTRCXSendSerial, [82](#)
  - HTRCXSetDirection, [83](#)
  - HTRCXSetEvent, [83](#)
  - HTRCXSetGlobalDirection, [83](#)
  - HTRCXSetGlobalOutput, [83](#)
  - HTRCXSetIRLinkPort, [84](#)
  - HTRCXSetMaxPower, [84](#)
  - HTRCXSetMessage, [84](#)
  - HTRCXSetOutput, [84](#)
  - HTRCXSetPower, [85](#)
  - HTRCXSetPriority, [85](#)
  - HTRCXSetSensorMode, [85](#)
  - HTRCXSetSensorType, [85](#)
  - HTRCXSetSleepTime, [86](#)
  - HTRCXSetTxPower, [86](#)
  - HTRCXSetWatch, [86](#)
  - HTRCXStartTask, [86](#)
  - HTRCXStopAllTasks, [87](#)
  - HTRCXStopTask, [87](#)
  - HTRCXToggle, [87](#)
  - HTRCXUnmuteSound, [87](#)
  - HTScoutCalibrateSensor, [87](#)
  - HTScoutMuteSound, [88](#)
  - HTScoutSelectSounds, [88](#)
  - HTScoutSendVLL, [88](#)
  - HTScoutSetEventFeedback, [88](#)
  - HTScoutSetLight, [88](#)
  - HTScoutSetScoutMode, [89](#)
  - HTScoutSetSensorClickTime, [89](#)
  - HTScoutSetSensorHysteresis, [89](#)
  - HTScoutSetSensorLowerLimit, [89](#)
  - HTScoutSetSensorUpperLimit, [90](#)
  - HTScoutUnmuteSound, [90](#)
  - ReadSensorHTAccel, [90](#)
  - ReadSensorHTColor, [91](#)
  - ReadSensorHTColor2Active, [91](#)
  - ReadSensorHTIRReceiver, [91](#)
  - ReadSensorHTIRReceiverEx, [92](#)
  - ReadSensorHTIRSeeker, [92](#)
  - ReadSensorHTIRSeeker2AC, [93](#)
  - ReadSensorHTIRSeeker2DC, [93](#)
  - ReadSensorHTNormalizedColor, [94](#)
  - ReadSensorHTNormalizedColor2Active, [94](#)
  - ReadSensorHTRawColor, [95](#)
  - ReadSensorHTRawColor2, [95](#)
  - ReadSensorHTTouchMultiplexer, [95](#)
  - SensorHTColorNum, [96](#)

- SensorHTCompass, 96
- SensorHTEOPD, 96
- SensorHTGyro, 97
- SensorHTIRSeeker2ACDir, 97
- SensorHTIRSeeker2Addr, 97
- SensorHTIRSeeker2DCDir, 98
- SensorHTIRSeekerDir, 98
- SetHTColor2Mode, 98
- SetHTIRSeeker2Mode, 99
- SetSensorHTEOPD, 99
- SetSensorHTGyro, 99
- HS\_BAUD\_115200
  - CommHiSpeedBaudConstants, 622
  - NBCCCommon.h, 876
- HS\_BAUD\_1200
  - CommHiSpeedBaudConstants, 622
  - NBCCCommon.h, 876
- HS\_BAUD\_14400
  - CommHiSpeedBaudConstants, 622
  - NBCCCommon.h, 877
- HS\_BAUD\_19200
  - CommHiSpeedBaudConstants, 622
  - NBCCCommon.h, 877
- HS\_BAUD\_230400
  - CommHiSpeedBaudConstants, 622
  - NBCCCommon.h, 877
- HS\_BAUD\_2400
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_28800
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_3600
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_38400
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_460800
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_4800
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_57600
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_7200
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 877
- HS\_BAUD\_76800
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 878
- HS\_BAUD\_921600
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 878
- HS\_BAUD\_9600
  - CommHiSpeedBaudConstants, 623
  - NBCCCommon.h, 878
- HS\_CMD\_READY
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 878
- HS\_CTRL\_EXIT
  - CommHiSpeedCtrlConstants, 621
  - NBCCCommon.h, 878
- HS\_CTRL\_INIT
  - CommHiSpeedCtrlConstants, 621
  - NBCCCommon.h, 878
- HS\_CTRL\_UART
  - CommHiSpeedCtrlConstants, 621
  - NBCCCommon.h, 878
- HS\_DISABLE
  - CommHiSpeedStateConstants, 620
  - NBCCCommon.h, 878
- HS\_ENABLE
  - CommHiSpeedStateConstants, 620
  - NBCCCommon.h, 878
- HS\_INIT\_RECEIVER
  - CommHiSpeedStateConstants, 620
  - NBCCCommon.h, 879
- HS\_INITIALISE
  - CommHiSpeedStateConstants, 620
  - NBCCCommon.h, 879
- HS\_MODE\_10\_STOP
  - CommHiSpeedStopBitsConstants, 626
  - NBCCCommon.h, 879
- HS\_MODE\_15\_STOP
  - CommHiSpeedStopBitsConstants, 626
  - NBCCCommon.h, 879
- HS\_MODE\_20\_STOP
  - CommHiSpeedStopBitsConstants, 626
  - NBCCCommon.h, 879
- HS\_MODE\_5\_DATA
  - CommHiSpeedDataBitsConstants, 625
  - NBCCCommon.h, 879
- HS\_MODE\_6\_DATA
  - CommHiSpeedDataBitsConstants, 625
  - NBCCCommon.h, 879
- HS\_MODE\_7\_DATA
  - CommHiSpeedDataBitsConstants, 625
  - NBCCCommon.h, 879
- HS\_MODE\_7E1
  - CommHiSpeedCombinedConstants, 628
  - NBCCCommon.h, 879
- HS\_MODE\_8\_DATA
  - CommHiSpeedDataBitsConstants, 625
  - NBCCCommon.h, 879
- HS\_MODE\_8N1



- CommHiSpeedCombinedConstants, 628
- NBCCCommon.h, 879
- HS\_MODE\_E\_PARITY
  - CommHiSpeedParityConstants, 627
  - NBCCCommon.h, 880
- HS\_MODE\_M\_PARITY
  - CommHiSpeedParityConstants, 627
  - NBCCCommon.h, 880
- HS\_MODE\_N\_PARITY
  - CommHiSpeedParityConstants, 627
  - NBCCCommon.h, 880
- HS\_MODE\_O\_PARITY
  - CommHiSpeedParityConstants, 627
  - NBCCCommon.h, 880
- HS\_MODE\_S\_PARITY
  - CommHiSpeedParityConstants, 627
  - NBCCCommon.h, 880
- HS\_SEND\_DATA
  - CommHiSpeedStateConstants, 620
  - NBCCCommon.h, 880
- HS\_UPDATE
  - CommHiSpeedFlagsConstants, 619
  - NBCCCommon.h, 880
- HSFlags
  - CommModuleFunctions, 337
  - NXCDefs.h, 1127
- HSInputBufferInPtr
  - CommModuleFunctions, 337
  - NXCDefs.h, 1127
- HSInputBufferOutPtr
  - CommModuleFunctions, 337
  - NXCDefs.h, 1127
- HSMODE
  - CommModuleFunctions, 337
  - NXCDefs.h, 1128
- HSOutputBufferInPtr
  - CommModuleFunctions, 338
  - NXCDefs.h, 1128
- HSOutputBufferOutPtr
  - CommModuleFunctions, 338
  - NXCDefs.h, 1128
- HSSpeed
  - CommModuleFunctions, 338
  - NXCDefs.h, 1128
- HSSpeed
  - CommModuleFunctions, 338
  - NXCDefs.h, 1128
- HSSpeed
  - CommModuleFunctions, 338
  - NXCDefs.h, 1128
- HT\_CH1\_A
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 880
- HT\_CH1\_B
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 880
- HT\_CH2\_A
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 880
- HT\_CH2\_B
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 881
- HT\_CH3\_A
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 881
- HT\_CH3\_B
  - HTIRReceiverConstants, 697
  - NBCCCommon.h, 881
- HT\_CH4\_A
  - HTIRReceiverConstants, 698
  - NBCCCommon.h, 881
- HT\_CH4\_B
  - HTIRReceiverConstants, 698
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_50HZ
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_60HZ
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_ACTIVE
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_BLCAL
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_FAR
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_LED\_HI
  - HTColor2Constants, 699
  - NBCCCommon.h, 881
- HT\_CMD\_COLOR2\_LED\_LOW
  - HTColor2Constants, 700
  - NBCCCommon.h, 882
- HT\_CMD\_COLOR2\_NEAR
  - HTColor2Constants, 700
  - NBCCCommon.h, 882
- HT\_CMD\_COLOR2\_PASSIVE
  - HTColor2Constants, 700
  - NBCCCommon.h, 882
- HT\_CMD\_COLOR2\_RAW
  - HTColor2Constants, 700
  - NBCCCommon.h, 882
- HT\_CMD\_COLOR2\_WBCAL
  - HTColor2Constants, 700
  - NBCCCommon.h, 882
- HTColor2Constants
  - HT\_CMD\_COLOR2\_50HZ, 699
  - HT\_CMD\_COLOR2\_60HZ, 699
  - HT\_CMD\_COLOR2\_ACTIVE, 699



- HT\_CMD\_COLOR2\_BLCAL, 699
- HT\_CMD\_COLOR2\_FAR, 699
- HT\_CMD\_COLOR2\_LED\_HI, 699
- HT\_CMD\_COLOR2\_LED\_LOW, 700
- HT\_CMD\_COLOR2\_NEAR, 700
- HT\_CMD\_COLOR2\_PASSIVE, 700
- HT\_CMD\_COLOR2\_RAW, 700
- HT\_CMD\_COLOR2\_WBCAL, 700
- HTIR2\_MODE\_1200
  - HTIRSeeker2Constants, 694
  - NBCCCommon.h, 882
- HTIR2\_MODE\_600
  - HTIRSeeker2Constants, 694
  - NBCCCommon.h, 882
- HTIR2\_REG\_AC01
  - HTIRSeeker2Constants, 694
  - NBCCCommon.h, 882
- HTIR2\_REG\_AC02
  - HTIRSeeker2Constants, 694
  - NBCCCommon.h, 882
- HTIR2\_REG\_AC03
  - HTIRSeeker2Constants, 694
  - NBCCCommon.h, 882
- HTIR2\_REG\_AC04
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 882
- HTIR2\_REG\_AC05
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_ACDIR
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DC01
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DC02
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DC03
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DC04
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DC05
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DCAVG
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_DCDIR
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIR2\_REG\_MODE
  - HTIRSeeker2Constants, 695
  - NBCCCommon.h, 883
- HTIRSeeker2Constants, 695
- NBCCCommon.h, 883
- HTIRReceiverConstants
  - HT\_CH1\_A, 697
  - HT\_CH1\_B, 697
  - HT\_CH2\_A, 697
  - HT\_CH2\_B, 697
  - HT\_CH3\_A, 697
  - HT\_CH3\_B, 697
  - HT\_CH4\_A, 698
  - HT\_CH4\_B, 698
- HTIRSeeker2Constants
  - HTIR2\_MODE\_1200, 694
  - HTIR2\_MODE\_600, 694
  - HTIR2\_REG\_AC01, 694
  - HTIR2\_REG\_AC02, 694
  - HTIR2\_REG\_AC03, 694
  - HTIR2\_REG\_AC04, 695
  - HTIR2\_REG\_AC05, 695
  - HTIR2\_REG\_ACDIR, 695
  - HTIR2\_REG\_DC01, 695
  - HTIR2\_REG\_DC02, 695
  - HTIR2\_REG\_DC03, 695
  - HTIR2\_REG\_DC04, 695
  - HTIR2\_REG\_DC05, 695
  - HTIR2\_REG\_DCAVG, 695
  - HTIR2\_REG\_DCDIR, 695
  - HTIR2\_REG\_MODE, 695
- HTIRTrain
  - HiTechnicAPI, 70
  - NXCDefs.h, 1129
- HTPFComboDirect
  - HiTechnicAPI, 70
  - NXCDefs.h, 1129
- HTPFComboPWM
  - HiTechnicAPI, 70
  - NXCDefs.h, 1130
- HTPFRawOutput
  - HiTechnicAPI, 71
  - NXCDefs.h, 1130
- HTPFRepeat
  - HiTechnicAPI, 71
  - NXCDefs.h, 1131
- HTPFSingleOutputCST
  - HiTechnicAPI, 72
  - NXCDefs.h, 1131
- HTPFSingleOutputPWM
  - HiTechnicAPI, 72
  - NXCDefs.h, 1132
- HTPFSinglePin
  - HiTechnicAPI, 73
  - NXCDefs.h, 1132
- HTPFTrain
  - HiTechnicAPI, 73

- NXCDefs.h, [1133](#)
- HTRCXAddToDatalog
  - HiTechnicAPI, [74](#)
  - NXCDefs.h, [1133](#)
- HTRCXBatteryLevel
  - HiTechnicAPI, [74](#)
  - NXCDefs.h, [1133](#)
- HTRCXCLEARAllEvents
  - HiTechnicAPI, [74](#)
  - NXCDefs.h, [1133](#)
- HTRCXCLEARCounter
  - HiTechnicAPI, [74](#)
  - NXCDefs.h, [1134](#)
- HTRCXCLEARMsg
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1134](#)
- HTRCXCLEARSensor
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1134](#)
- HTRCXCLEARSound
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1134](#)
- HTRCXCLEARTimer
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1134](#)
- HTRCXCreateDatalog
  - HiTechnicAPI, [75](#)
  - NXCDefs.h, [1135](#)
- HTRCXDecCounter
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1135](#)
- HTRCXDeleteSub
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1135](#)
- HTRCXDeleteSubs
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1135](#)
- HTRCXDeleteTask
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1135](#)
- HTRCXDeleteTasks
  - HiTechnicAPI, [76](#)
  - NXCDefs.h, [1136](#)
- HTRCXDisableOutput
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1136](#)
- HTRCXEnableOutput
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1136](#)
- HTRCXEvent
  - HiTechnicAPI, [77](#)
  - NXCDefs.h, [1136](#)
- HTRCXFloat
  - HiTechnicAPI, [77](#)
- NXCDefs.h, [1136](#)
- HTRCXFwd
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1137](#)
- HTRCXIncCounter
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1137](#)
- HTRCXInvertOutput
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1137](#)
- HTRCXMuteSound
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1137](#)
- HTRCXObvertOutput
  - HiTechnicAPI, [78](#)
  - NXCDefs.h, [1138](#)
- HTRCXOff
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1138](#)
- HTRCXOn
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1138](#)
- HTRCXOnFor
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1138](#)
- HTRCXOnFwd
  - HiTechnicAPI, [79](#)
  - NXCDefs.h, [1139](#)
- HTRCXOnRev
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1139](#)
- HTRCXPBTurnOff
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1139](#)
- HTRCXPing
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1139](#)
- HTRCXPlaySound
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1139](#)
- HTRCXPlayTone
  - HiTechnicAPI, [80](#)
  - NXCDefs.h, [1140](#)
- HTRCXPlayToneVar
  - HiTechnicAPI, [81](#)
  - NXCDefs.h, [1140](#)
- HTRCXPoll
  - HiTechnicAPI, [81](#)
  - NXCDefs.h, [1140](#)
- HTRCXPollMemory
  - HiTechnicAPI, [81](#)
  - NXCDefs.h, [1140](#)
- HTRCXRemote
  - HiTechnicAPI, [82](#)

- NXCDefs.h, [1141](#)
- HTRCXRev
  - HiTechnicAPI, [82](#)
  - NXCDefs.h, [1141](#)
- HTRCXSelectDisplay
  - HiTechnicAPI, [82](#)
  - NXCDefs.h, [1141](#)
- HTRCXSelectProgram
  - HiTechnicAPI, [82](#)
  - NXCDefs.h, [1141](#)
- HTRCXSendSerial
  - HiTechnicAPI, [82](#)
  - NXCDefs.h, [1142](#)
- HTRCXSetDirection
  - HiTechnicAPI, [83](#)
  - NXCDefs.h, [1142](#)
- HTRCXSetEvent
  - HiTechnicAPI, [83](#)
  - NXCDefs.h, [1142](#)
- HTRCXSetGlobalDirection
  - HiTechnicAPI, [83](#)
  - NXCDefs.h, [1142](#)
- HTRCXSetGlobalOutput
  - HiTechnicAPI, [83](#)
  - NXCDefs.h, [1143](#)
- HTRCXSetIRLinkPort
  - HiTechnicAPI, [84](#)
  - NXCDefs.h, [1143](#)
- HTRCXSetMaxPower
  - HiTechnicAPI, [84](#)
  - NXCDefs.h, [1143](#)
- HTRCXSetMessage
  - HiTechnicAPI, [84](#)
  - NXCDefs.h, [1143](#)
- HTRCXSetOutput
  - HiTechnicAPI, [84](#)
  - NXCDefs.h, [1144](#)
- HTRCXSetPower
  - HiTechnicAPI, [85](#)
  - NXCDefs.h, [1144](#)
- HTRCXSetPriority
  - HiTechnicAPI, [85](#)
  - NXCDefs.h, [1144](#)
- HTRCXSetSensorMode
  - HiTechnicAPI, [85](#)
  - NXCDefs.h, [1144](#)
- HTRCXSetSensorType
  - HiTechnicAPI, [85](#)
  - NXCDefs.h, [1145](#)
- HTRCXSetSleepTime
  - HiTechnicAPI, [86](#)
  - NXCDefs.h, [1145](#)
- HTRCXSetTxPower
  - HiTechnicAPI, [86](#)
  - NXCDefs.h, [1145](#)
- HTRCXSetWatch
  - HiTechnicAPI, [86](#)
  - NXCDefs.h, [1145](#)
- HTRCXStartTask
  - HiTechnicAPI, [86](#)
  - NXCDefs.h, [1146](#)
- HTRCXStopAllTasks
  - HiTechnicAPI, [87](#)
  - NXCDefs.h, [1146](#)
- HTRCXStopTask
  - HiTechnicAPI, [87](#)
  - NXCDefs.h, [1146](#)
- HTRCXToggle
  - HiTechnicAPI, [87](#)
  - NXCDefs.h, [1146](#)
- HTRCXUnmuteSound
  - HiTechnicAPI, [87](#)
  - NXCDefs.h, [1147](#)
- HTScoutCalibrateSensor
  - HiTechnicAPI, [87](#)
  - NXCDefs.h, [1147](#)
- HTScoutMuteSound
  - HiTechnicAPI, [88](#)
  - NXCDefs.h, [1147](#)
- HTScoutSelectSounds
  - HiTechnicAPI, [88](#)
  - NXCDefs.h, [1147](#)
- HTScoutSendVLL
  - HiTechnicAPI, [88](#)
  - NXCDefs.h, [1147](#)
- HTScoutSetEventFeedback
  - HiTechnicAPI, [88](#)
  - NXCDefs.h, [1147](#)
- HTScoutSetLight
  - HiTechnicAPI, [88](#)
  - NXCDefs.h, [1148](#)
- HTScoutSetScoutMode
  - HiTechnicAPI, [89](#)
  - NXCDefs.h, [1148](#)
- HTScoutSetSensorClickTime
  - HiTechnicAPI, [89](#)
  - NXCDefs.h, [1148](#)
- HTScoutSetSensorHysteresis
  - HiTechnicAPI, [89](#)
  - NXCDefs.h, [1148](#)
- HTScoutSetSensorLowerLimit
  - HiTechnicAPI, [89](#)
  - NXCDefs.h, [1149](#)
- HTScoutSetSensorUpperLimit
  - HiTechnicAPI, [90](#)
  - NXCDefs.h, [1149](#)
- HTScoutUnmuteSound
  - HiTechnicAPI, [90](#)

- NXCDefs.h, 1149
- I2C\_REG\_CMD
  - GenericI2CConstants, 587
  - NBCCCommon.h, 883
- I2C\_REG\_DEVICE\_ID
  - GenericI2CConstants, 587
  - NBCCCommon.h, 884
- I2C\_REG\_VENDOR\_ID
  - GenericI2CConstants, 587
  - NBCCCommon.h, 884
- I2C\_REG\_VERSION
  - GenericI2CConstants, 587
  - NBCCCommon.h, 884
- I2CBytes
  - LowSpeedModuleFunctions, 281
  - NXCDefs.h, 1149
- I2CBytesReady
  - LowSpeedModuleFunctions, 281
  - NXCDefs.h, 1150
- I2CCheckStatus
  - LowSpeedModuleFunctions, 282
  - NXCDefs.h, 1151
- I2CDeviceId
  - LowSpeedModuleFunctions, 282
  - NXCDefs.h, 1151
- I2CDeviceIdEx
  - LowSpeedModuleFunctions, 282
  - NXCDefs.h, 1151
- I2CDeviceInfo
  - LowSpeedModuleFunctions, 283
  - NXCDefs.h, 1152
- I2CDeviceInfoEx
  - LowSpeedModuleFunctions, 283
  - NXCDefs.h, 1152
- I2CRead
  - LowSpeedModuleFunctions, 284
  - NXCDefs.h, 1153
- I2CSendCommand
  - LowSpeedModuleFunctions, 284
  - NXCDefs.h, 1153
- I2CSendCommandEx
  - LowSpeedModuleFunctions, 285
  - NXCDefs.h, 1154
- I2CStatus
  - LowSpeedModuleFunctions, 285
  - NXCDefs.h, 1154
- I2CVendorId
  - LowSpeedModuleFunctions, 286
  - NXCDefs.h, 1155
- I2CVendorIdEx
  - LowSpeedModuleFunctions, 286
  - NXCDefs.h, 1155
- I2CVersion
  - LowSpeedModuleFunctions, 286
  - NXCDefs.h, 1155
- I2CVersionEx
  - LowSpeedModuleFunctions, 287
  - NXCDefs.h, 1156
- I2CWrite
  - LowSpeedModuleFunctions, 287
  - NXCDefs.h, 1156
- IN\_1
  - NBCCCommon.h, 884
  - NBCInputPortConstants, 543
- IN\_2
  - NBCCCommon.h, 884
  - NBCInputPortConstants, 543
- IN\_3
  - NBCCCommon.h, 884
  - NBCInputPortConstants, 543
- IN\_4
  - NBCCCommon.h, 884
  - NBCInputPortConstants, 543
- IN\_MODE\_ANGLESTEP
  - NBCCCommon.h, 884
  - NBCSensorModeConstants, 547
- IN\_MODE\_BOOLEAN
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 547
- IN\_MODE\_CELSIUS
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 547
- IN\_MODE\_FAHRENHEIT
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 547
- IN\_MODE\_MODEMASK
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 547
- IN\_MODE\_PCTFULLSCALE
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 547
- IN\_MODE\_PERIODCOUNTER
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 548
- IN\_MODE\_RAW
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 548
- IN\_MODE\_SLOPEMASK
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 548
- IN\_MODE\_TRANSITIONCNT
  - NBCCCommon.h, 885
  - NBCSensorModeConstants, 548
- IN\_TYPE\_ANGLE
  - NBCCCommon.h, 885
  - NBCSensorTypeConstants, 544
- IN\_TYPE\_COLORBLUE

- NBCCCommon.h, 885
- NBCSensorTypeConstants, 544
- IN\_TYPE\_COLOREXIT
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 544
- IN\_TYPE\_COLORFULL
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 544
- IN\_TYPE\_COLORGREEN
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_COLORNONE
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_COLORRED
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_CUSTOM
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_HISPEED
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_LIGHT\_ACTIVE
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_LIGHT\_INACTIVE
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_LOWSPEED
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_LOWSPEED\_9V
  - NBCCCommon.h, 886
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_NO\_SENSOR
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_REFLECTION
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 545
- IN\_TYPE\_SOUND\_DB
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 546
- IN\_TYPE\_SOUND\_DBA
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 546
- IN\_TYPE\_SWITCH
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 546
- IN\_TYPE\_TEMPERATURE
  - NBCCCommon.h, 887
  - NBCSensorTypeConstants, 546
- Index
- ReadButtonType, 799
- InPorts
  - S1, 193
  - S2, 194
  - S3, 194
  - S4, 195
- Input field constants, 549
- Input module, 45
- Input module constants, 46
- Input module functions, 205
- Input module IOMAP offsets, 554
- Input module types, 204
- Input port constants, 193
- INPUT\_BLACKCOLOR
  - InputColorValueConstants, 551
  - NBCCCommon.h, 887
- INPUT\_BLANK
  - InputColorIdxConstants, 550
  - NBCCCommon.h, 887
- INPUT\_BLUE
  - InputColorIdxConstants, 550
  - NBCCCommon.h, 887
- INPUT\_BLUECOLOR
  - InputColorValueConstants, 551
  - NBCCCommon.h, 887
- INPUT\_CAL\_POINT\_0
  - InputColorCalibrationConstants, 553
  - NBCCCommon.h, 887
- INPUT\_CAL\_POINT\_1
  - InputColorCalibrationConstants, 553
  - NBCCCommon.h, 888
- INPUT\_CAL\_POINT\_2
  - InputColorCalibrationConstants, 553
  - NBCCCommon.h, 888
- INPUT\_CUSTOM9V
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_CUSTOMACTIVE
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_CUSTOMINACTIVE
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_DIGIO
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_DIGI1
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_GREEN
  - InputColorIdxConstants, 550
  - NBCCCommon.h, 888
- INPUT\_GREENCOLOR
  - InputColorValueConstants, 551

- NBCCCommon.h, 888
- INPUT\_INVALID\_DATA
  - InputModuleConstants, 47
  - NBCCCommon.h, 888
- INPUT\_NO\_OF\_COLORS
  - InputColorIdxConstants, 550
  - NBCCCommon.h, 888
- INPUT\_NO\_OF\_POINTS
  - InputColorCalibrationConstants, 553
  - NBCCCommon.h, 889
- INPUT\_RED
  - InputColorIdxConstants, 550
  - NBCCCommon.h, 889
- INPUT\_REDCOLOR
  - InputColorValueConstants, 551
  - NBCCCommon.h, 889
- INPUT\_RESETCAL
  - InputColorCalibrationStateConstants, 552
  - NBCCCommon.h, 889
- INPUT\_RUNNINGCAL
  - InputColorCalibrationStateConstants, 552
  - NBCCCommon.h, 889
- INPUT\_SENSORCAL
  - InputColorCalibrationStateConstants, 552
  - NBCCCommon.h, 889
- INPUT\_SENSOROFF
  - InputColorCalibrationStateConstants, 552
  - NBCCCommon.h, 889
- INPUT\_STARTCAL
  - InputColorCalibrationStateConstants, 552
  - NBCCCommon.h, 889
- INPUT\_WHITECOLOR
  - InputColorValueConstants, 551
  - NBCCCommon.h, 889
- INPUT\_YELLOWCOLOR
  - InputColorValueConstants, 551
  - NBCCCommon.h, 889
- InputColorCalibrationConstants
  - INPUT\_CAL\_POINT\_0, 553
  - INPUT\_CAL\_POINT\_1, 553
  - INPUT\_CAL\_POINT\_2, 553
  - INPUT\_NO\_OF\_POINTS, 553
- InputColorCalibrationStateConstants
  - INPUT\_RESETCAL, 552
  - INPUT\_RUNNINGCAL, 552
  - INPUT\_SENSORCAL, 552
  - INPUT\_SENSOROFF, 552
  - INPUT\_STARTCAL, 552
- InputColorIdxConstants
  - INPUT\_BLANK, 550
  - INPUT\_BLUE, 550
  - INPUT\_GREEN, 550
  - INPUT\_NO\_OF\_COLORS, 550
  - INPUT\_RED, 550
- InputColorValueConstants
  - INPUT\_BLACKCOLOR, 551
  - INPUT\_BLUECOLOR, 551
  - INPUT\_GREENCOLOR, 551
  - INPUT\_REDCOLOR, 551
  - INPUT\_WHITECOLOR, 551
  - INPUT\_YELLOWCOLOR, 551
- InputFieldConstants
  - InputMode, 549
  - InvalidData, 549
  - NormalizedValue, 549
  - RawValue, 549
  - ScaledValue, 549
  - Type, 549
- InputIOMAP
  - InputOffsetADRaw, 554
  - InputOffsetColorADRaw, 554
  - InputOffsetColorCalibration, 554
  - InputOffsetColorCalibrationState, 554
  - InputOffsetColorCalLimits, 554
  - InputOffsetColorSensorBoolean, 555
  - InputOffsetColorSensorRaw, 555
  - InputOffsetColorSensorValue, 555
  - InputOffsetCustomActiveStatus, 555
  - InputOffsetCustomPctFullScale, 555
  - InputOffsetCustomZeroOffset, 555
  - InputOffsetDigiPinsDir, 555
  - InputOffsetDigiPinsIn, 555
  - InputOffsetDigiPinsOut, 555
  - InputOffsetInvalidData, 555
  - InputOffsetSensorBoolean, 555
  - InputOffsetSensorMode, 556
  - InputOffsetSensorRaw, 556
  - InputOffsetSensorType, 556
  - InputOffsetSensorValue, 556
- InputMode
  - InputFieldConstants, 549
  - NBCCCommon.h, 890
- InputModuleConstants
  - INPUT\_CUSTOM9V, 47
  - INPUT\_CUSTOMACTIVE, 47
  - INPUT\_CUSTOMINACTIVE, 47
  - INPUT\_DIGI0, 47
  - INPUT\_DIGI1, 47
  - INPUT\_INVALID\_DATA, 47
- InputModuleFunctions
  - ClearSensor, 208
  - ColorADRaw, 208
  - ColorBoolean, 209
  - ColorCalibration, 209
  - ColorCalibrationState, 209
  - ColorCalLimits, 210
  - ColorSensorRaw, 210
  - ColorSensorValue, 211

- CustomSensorActiveStatus, 211
- CustomSensorPercentFullScale, 211
- CustomSensorZeroOffset, 212
- GetInput, 212
- ReadSensorColorEx, 212
- ReadSensorColorRaw, 213
- ResetSensor, 213
- Sensor, 214
- SensorBoolean, 214
- SensorDigiPinsDirection, 214
- SensorDigiPinsOutputLevel, 215
- SensorDigiPinsStatus, 215
- SensorInvalid, 215
- SensorMode, 216
- SensorNormalized, 216
- SensorRaw, 216
- SensorScaled, 217
- SensorType, 217
- SensorValue, 217
- SensorValueBool, 218
- SensorValueRaw, 218
- SetCustomSensorActiveStatus, 218
- SetCustomSensorPercentFullScale, 219
- SetCustomSensorZeroOffset, 219
- SetInput, 219
- SetSensor, 219
- SetSensorBoolean, 220
- SetSensorColorBlue, 220
- SetSensorColorFull, 220
- SetSensorColorGreen, 221
- SetSensorColorNone, 221
- SetSensorColorRed, 221
- SetSensorDigiPinsDirection, 222
- SetSensorDigiPinsOutputLevel, 222
- SetSensorDigiPinsStatus, 222
- SetSensorLight, 222
- SetSensorLowSpeed, 223
- SetSensorMode, 223
- SetSensorSound, 223
- SetSensorTouch, 224
- SetSensorType, 224
- SysColorSensorRead, 224
- InputModuleID
  - ModuleIDConstants, 179
  - NBCCCommon.h, 890
- InputModuleName
  - ModuleNameConstants, 177
  - NBCCCommon.h, 890
- InputOffsetADRaw
  - InputIOMAP, 554
  - NBCCCommon.h, 890
- InputOffsetColorADRaw
  - InputIOMAP, 554
  - NBCCCommon.h, 890
- InputOffsetColorCalibration
  - InputIOMAP, 554
  - NBCCCommon.h, 890
- InputOffsetColorCalibrationState
  - InputIOMAP, 554
  - NBCCCommon.h, 890
- InputOffsetColorCalLimits
  - InputIOMAP, 554
  - NBCCCommon.h, 890
- InputOffsetColorSensorBoolean
  - InputIOMAP, 555
  - NBCCCommon.h, 890
- InputOffsetColorSensorRaw
  - InputIOMAP, 555
  - NBCCCommon.h, 890
- InputOffsetColorSensorValue
  - InputIOMAP, 555
  - NBCCCommon.h, 890
- InputOffsetCustomActiveStatus
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetCustomPctFullScale
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetCustomZeroOffset
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetDigiPinsDir
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetDigiPinsIn
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetDigiPinsOut
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetInvalidData
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetSensorBoolean
  - InputIOMAP, 555
  - NBCCCommon.h, 891
- InputOffsetSensorMode
  - InputIOMAP, 556
  - NBCCCommon.h, 891
- InputOffsetSensorRaw
  - InputIOMAP, 556
  - NBCCCommon.h, 891
- InputOffsetSensorType
  - InputIOMAP, 556
  - NBCCCommon.h, 891
- InputOffsetSensorValue
  - InputIOMAP, 556
  - NBCCCommon.h, 892



- INT\_MAX
  - NBCCCommon.h, [892](#)
  - NXTLimits, [711](#)
- INT\_MIN
  - NBCCCommon.h, [892](#)
  - NXTLimits, [711](#)
- INTF\_BTOFF
  - CommInterfaceConstants, [630](#)
  - NBCCCommon.h, [892](#)
- INTF\_BTON
  - CommInterfaceConstants, [630](#)
  - NBCCCommon.h, [892](#)
- INTF\_CONNECT
  - CommInterfaceConstants, [630](#)
  - NBCCCommon.h, [892](#)
- INTF\_CONNECTBYNAME
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [892](#)
- INTF\_CONNECTREQ
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [892](#)
- INTF\_DISCONNECT
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [892](#)
- INTF\_DISCONNECTALL
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [892](#)
- INTF\_EXTREAD
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [892](#)
- INTF\_FACTORYRESET
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_OPENSTREAM
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_PINREQ
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_REMOVEDEVICE
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_SEARCH
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_SENDDATA
  - CommInterfaceConstants, [631](#)
  - NBCCCommon.h, [893](#)
- INTF\_SENDFILE
  - CommInterfaceConstants, [632](#)
  - NBCCCommon.h, [893](#)
- INTF\_SETBTNAME
  - CommInterfaceConstants, [632](#)
  - NBCCCommon.h, [893](#)
- INTF\_SETCMDMODE
  - CommInterfaceConstants, [632](#)
  - NBCCCommon.h, [893](#)
- INTF\_STOPSEARCH
  - CommInterfaceConstants, [632](#)
  - NBCCCommon.h, [893](#)
- INTF\_VISIBILITY
  - CommInterfaceConstants, [632](#)
  - NBCCCommon.h, [893](#)
- Invalid
  - ColorSensorReadType, [720](#)
- InvalidData
  - InputFieldConstants, [549](#)
  - NBCCCommon.h, [894](#)
- IOCtrl module, [56](#)
- IOCtrl module constants, [497](#)
- IOCtrl module functions, [302](#)
- IOCtrl module IOMAP offsets, [499](#)
- IOCtrl module types, [301](#)
- IOCTRL\_BOOT
  - IOCtrlPO, [498](#)
  - NBCCCommon.h, [894](#)
- IOCTRL\_POWERDOWN
  - IOCtrlPO, [498](#)
  - NBCCCommon.h, [894](#)
- IOCtrlIOMAP
  - IOCtrlOffsetPowerOn, [499](#)
- IOCtrlModuleFunctions
  - PowerDown, [302](#)
  - RebootInFirmwareMode, [302](#)
  - SleepNow, [302](#)
- IOCtrlModuleID
  - ModuleIDConstants, [179](#)
  - NBCCCommon.h, [894](#)
- IOCtrlModuleName
  - ModuleNameConstants, [177](#)
  - NBCCCommon.h, [894](#)
- IOCtrlOffsetPowerOn
  - IOCtrlIOMAP, [499](#)
  - NBCCCommon.h, [894](#)
- IOCtrlPO
  - IOCTRL\_BOOT, [498](#)
  - IOCTRL\_POWERDOWN, [498](#)
- IOMapRead
  - NBCCCommon.h, [894](#)
  - SysCallConstants, [472](#)
- IOMapReadById
  - NBCCCommon.h, [894](#)
  - SysCallConstants, [472](#)
- IOMapReadByIdType, [780](#)
  - Buffer, [780](#)
  - Count, [780](#)
  - ModuleID, [780](#)
  - Offset, [780](#)



- Result, [781](#)
- IOMapReadType, [782](#)
  - Buffer, [782](#)
  - Count, [782](#)
  - ModuleName, [782](#)
  - Offset, [782](#)
  - Result, [783](#)
- IOMapWrite
  - NBCCCommon.h, [894](#)
  - SysCallConstants, [472](#)
- IOMapWriteByID
  - NBCCCommon.h, [894](#)
  - SysCallConstants, [473](#)
- IOMapWriteByIDType, [784](#)
  - Buffer, [784](#)
  - ModuleID, [784](#)
  - Offset, [784](#)
  - Result, [784](#)
- IOMapWriteType, [786](#)
  - Buffer, [786](#)
  - ModuleName, [786](#)
  - Offset, [786](#)
  - Result, [786](#)
- IR Train channel constants, [684](#)
- IRTrainChannels
  - TRAIN\_CHANNEL\_1, [684](#)
  - TRAIN\_CHANNEL\_2, [684](#)
  - TRAIN\_CHANNEL\_3, [684](#)
  - TRAIN\_CHANNEL\_ALL, [684](#)
- IRTrainFuncs
  - TRAIN\_FUNC\_DECR\_SPEED, [683](#)
  - TRAIN\_FUNC\_INCR\_SPEED, [683](#)
  - TRAIN\_FUNC\_STOP, [683](#)
  - TRAIN\_FUNC\_TOGGLE\_LIGHT, [683](#)
- isalnum
  - ctypeAPI, [460](#)
  - NXCDefs.h, [1157](#)
- isalpha
  - ctypeAPI, [460](#)
  - NXCDefs.h, [1157](#)
- isctrl
  - ctypeAPI, [460](#)
  - NXCDefs.h, [1157](#)
- isdigit
  - ctypeAPI, [460](#)
  - NXCDefs.h, [1158](#)
- isgraph
  - ctypeAPI, [461](#)
  - NXCDefs.h, [1158](#)
- islower
  - ctypeAPI, [461](#)
  - NXCDefs.h, [1158](#)
- isNAN
  - cmathAPI, [420](#)
- NXCDefs.h, [1159](#)
- isprint
  - ctypeAPI, [461](#)
  - NXCDefs.h, [1159](#)
- ispunct
  - ctypeAPI, [462](#)
  - NXCDefs.h, [1159](#)
- isspace
  - ctypeAPI, [462](#)
  - NXCDefs.h, [1160](#)
- isupper
  - ctypeAPI, [462](#)
  - NXCDefs.h, [1160](#)
- isxdigit
  - ctypeAPI, [463](#)
  - NXCDefs.h, [1160](#)
- KeepAlive
  - NBCCCommon.h, [894](#)
  - SysCallConstants, [473](#)
- KeepAliveType, [788](#)
  - Result, [788](#)
- labs
  - cstdlibAPI, [439](#)
  - NXCDefs.h, [1160](#)
- LCD\_LINE1
  - LineConstants, [475](#)
  - NBCCCommon.h, [895](#)
- LCD\_LINE2
  - LineConstants, [475](#)
  - NBCCCommon.h, [895](#)
- LCD\_LINE3
  - LineConstants, [476](#)
  - NBCCCommon.h, [895](#)
- LCD\_LINE4
  - LineConstants, [476](#)
  - NBCCCommon.h, [895](#)
- LCD\_LINE5
  - LineConstants, [476](#)
  - NBCCCommon.h, [896](#)
- LCD\_LINE6
  - LineConstants, [476](#)
  - NBCCCommon.h, [896](#)
- LCD\_LINE7
  - LineConstants, [476](#)
  - NBCCCommon.h, [896](#)
- LCD\_LINE8
  - LineConstants, [477](#)
  - NBCCCommon.h, [896](#)
- ldiv
  - cstdlibAPI, [439](#)
  - NXCDefs.h, [1161](#)
- ldiv\_t, [789](#)

- quot, 789
- rem, 789
- LDR\_APPENDNOTPOSSIBLE
  - LoaderErrors, 502
  - NBCCCommon.h, 896
- LDR\_BTBUSY
  - LoaderErrors, 502
  - NBCCCommon.h, 896
- LDR\_BTCONNECTFAIL
  - LoaderErrors, 503
  - NBCCCommon.h, 896
- LDR\_BTTIMEOUT
  - LoaderErrors, 503
  - NBCCCommon.h, 897
- LDR\_CMD\_BOOTCMD
  - LoaderFunctionConstants, 506
  - NBCCCommon.h, 897
- LDR\_CMD\_BTFACTORYRESET
  - LoaderFunctionConstants, 506
  - NBCCCommon.h, 897
- LDR\_CMD\_BTGETADR
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_CLOSE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_CLOSEMODHANDLE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_CROPDATFILE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_DELETE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_DELETEUSERFLASH
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_DEVICEINFO
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_FINDFIRST
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 897
- LDR\_CMD\_FINDFIRSTMODULE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 898
- LDR\_CMD\_FINDNEXT
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 898
- LDR\_CMD\_FINDNEXTMODULE
  - LoaderFunctionConstants, 507
  - NBCCCommon.h, 898
- LDR\_CMD\_IOMAPREAD
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_IOMAPWRITE
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENAPPENDDATA
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENREAD
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENREADLINEAR
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENWRITE
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENWRITEDATA
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_OPENWRITELINEAR
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 898
- LDR\_CMD\_POLLCMD
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 899
- LDR\_CMD\_POLLCMDLEN
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 899
- LDR\_CMD\_READ
  - LoaderFunctionConstants, 508
  - NBCCCommon.h, 899
- LDR\_CMD\_RENAMEFILE
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_RESIZEDATFILE
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_SEEKFROMCURRENT
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_SEEKFROMEND
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_SEEKFROMSTART
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_SETBRICKNAME
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_VERSIONS
  - LoaderFunctionConstants, 509
  - NBCCCommon.h, 899
- LDR\_CMD\_WRITE

- LoaderFunctionConstants, [509](#)
- NBCCCommon.h, [899](#)
- LDR\_ENDOFFILE
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_EOFEXPECTED
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILEEXISTS
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILEISBUSY
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILEISFULL
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILENOTFOUND
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILETX\_CLOSEERROR
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILETX\_DSTEXISTS
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILETX\_SRCMISSING
  - LoaderErrors, [503](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILETX\_STREAMERROR
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [900](#)
- LDR\_FILETX\_TIMEOUT
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [900](#)
- LDR\_HANDLEALREADYCLOSED
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_ILLEGALFILENAME
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_ILLEGALHANDLE
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_INPROGRESS
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_INVALIDSEEK
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_MODULENOTFOUND
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOLINEARSPACE
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOMOREFILES
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOMOREHANDLES
  - LoaderErrors, [504](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOSPACE
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOTLINEARFILE
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [901](#)
- LDR\_NOWRITEBUFFERS
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [902](#)
- LDR\_OUTOFBOUNDARY
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [902](#)
- LDR\_REQPIN
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [902](#)
- LDR\_SUCCESS
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [902](#)
- LDR\_UNDEFINEDERROR
  - LoaderErrors, [505](#)
  - NBCCCommon.h, [902](#)
- LeftStr
  - cstringAPI, [449](#)
  - NXCDefs.h, [1161](#)
- Length
  - FileFindType, [766](#)
  - FileOpenType, [768](#)
  - FileReadWriteType, [770](#)
  - FileSeekType, [777](#)
  - LoaderExecuteFunctionType, [793](#)
- Line number constants, [475](#)
- LineConstants
  - LCD\_LINE1, [475](#)
  - LCD\_LINE2, [475](#)
  - LCD\_LINE3, [476](#)
  - LCD\_LINE4, [476](#)
  - LCD\_LINE5, [476](#)
  - LCD\_LINE6, [476](#)
  - LCD\_LINE7, [476](#)
  - LCD\_LINE8, [477](#)
- LineOut
  - DisplayModuleFunctions, [258](#)
  - NXCDefs.h, [1161](#)
- ListFiles
  - NBCCCommon.h, [902](#)
  - SysCallConstants, [473](#)

- ListFilesType, 790
  - FileList, 790
  - Pattern, 790
  - Result, 790
- Loader module, 57
- Loader module constants, 500
- Loader module error codes, 502
- Loader module function constants, 506
- Loader module functions, 383
- Loader module IOMAP offsets, 501
- Loader module types, 382
- LoaderErrors
  - LDR\_APPENDNOTPOSSIBLE, 502
  - LDR\_BTBUSY, 502
  - LDR\_BTCONNECTFAIL, 503
  - LDR\_BTTIMEOUT, 503
  - LDR\_ENDOFFILE, 503
  - LDR\_EOFEXPECTED, 503
  - LDR\_FILEEXISTS, 503
  - LDR\_FILEISBUSY, 503
  - LDR\_FILEISFULL, 503
  - LDR\_FILENOTFOUND, 503
  - LDR\_FILETX\_CLOSEERROR, 503
  - LDR\_FILETX\_DSTEXISTS, 503
  - LDR\_FILETX\_SRCMISSING, 503
  - LDR\_FILETX\_STREAMERROR, 504
  - LDR\_FILETX\_TIMEOUT, 504
  - LDR\_HANDLEALREADYCLOSED, 504
  - LDR\_ILLEGALFILENAME, 504
  - LDR\_ILLEGALHANDLE, 504
  - LDR\_INPROGRESS, 504
  - LDR\_INVALIDSEEK, 504
  - LDR\_MODULENOTFOUND, 504
  - LDR\_NOLINEARSPACE, 504
  - LDR\_NOMOREFILES, 504
  - LDR\_NOMOREHANDLES, 504
  - LDR\_NOSPACE, 505
  - LDR\_NOTLINEARFILE, 505
  - LDR\_NOWRITEBUFFERS, 505
  - LDR\_OUTOFBOUNDARY, 505
  - LDR\_REQPIN, 505
  - LDR\_SUCCESS, 505
  - LDR\_UNDEFINEDERROR, 505
- LoaderExecuteFunction
  - NBCCCommon.h, 902
  - SysCallConstants, 473
- LoaderExecuteFunctionType, 792
  - Buffer, 793
  - Cmd, 793
  - Filename, 793
  - Length, 793
  - Result, 793
- LoaderFunctionConstants
  - LDR\_CMD\_BOOTCMD, 506
  - LDR\_CMD\_BTFACTORYRESET, 506
  - LDR\_CMD\_BTGETADR, 507
  - LDR\_CMD\_CLOSE, 507
  - LDR\_CMD\_CLOSEMODHANDLE, 507
  - LDR\_CMD\_CROPDATAFILE, 507
  - LDR\_CMD\_DELETE, 507
  - LDR\_CMD\_DELETEUSERFLASH, 507
  - LDR\_CMD\_DEVICEINFO, 507
  - LDR\_CMD\_FINDFIRST, 507
  - LDR\_CMD\_FINDFIRSTMODULE, 507
  - LDR\_CMD\_FINDNEXT, 507
  - LDR\_CMD\_FINDNEXTMODULE, 507
  - LDR\_CMD\_IOMAPREAD, 508
  - LDR\_CMD\_IOMAPWRITE, 508
  - LDR\_CMD\_OPENAPPENDDATA, 508
  - LDR\_CMD\_OPENREAD, 508
  - LDR\_CMD\_OPENREADLINEAR, 508
  - LDR\_CMD\_OPENWRITE, 508
  - LDR\_CMD\_OPENWRITEDATA, 508
  - LDR\_CMD\_OPENWRITELINEAR, 508
  - LDR\_CMD\_POLLCMD, 508
  - LDR\_CMD\_POLLCMDLEN, 508
  - LDR\_CMD\_READ, 508
  - LDR\_CMD\_RENAMEFILE, 509
  - LDR\_CMD\_RESIZEDATAFILE, 509
  - LDR\_CMD\_SEEKFROMCURRENT, 509
  - LDR\_CMD\_SEEKFROMEND, 509
  - LDR\_CMD\_SEEKFROMSTART, 509
  - LDR\_CMD\_SETBRICKNAME, 509
  - LDR\_CMD\_VERSIONS, 509
  - LDR\_CMD\_WRITE, 509
- LoaderIOMAP
  - LoaderOffsetFreeUserFlash, 501
  - LoaderOffsetPFunc, 501
- LoaderModuleConstants
  - EOF, 500
  - NULL, 500
- LoaderModuleFunctions
  - CloseFile, 385
  - CreateFile, 386
  - CreateFileLinear, 386
  - CreateFileNonLinear, 386
  - DeleteFile, 387
  - FindFirstFile, 387
  - FindNextFile, 388
  - FreeMemory, 388
  - OpenFileAppend, 388
  - OpenFileRead, 389
  - OpenFileReadLinear, 389
  - Read, 390
  - ReadBytes, 390
  - ReadLn, 390
  - ReadLnString, 391
  - RenameFile, 391

- ResizeFile, [391](#)
- ResolveHandle, [392](#)
- SysFileClose, [392](#)
- SysFileDelete, [392](#)
- SysFileFindFirst, [393](#)
- SysFileFindNext, [393](#)
- SysFileOpenAppend, [393](#)
- SysFileOpenRead, [393](#)
- SysFileOpenReadLinear, [394](#)
- SysFileOpenWrite, [394](#)
- SysFileOpenWriteLinear, [394](#)
- SysFileOpenWriteNonLinear, [395](#)
- SysFileRead, [395](#)
- SysFileRename, [395](#)
- SysFileResize, [395](#)
- SysFileResolveHandle, [396](#)
- SysFileSeek, [396](#)
- SysFileWrite, [396](#)
- SysListFiles, [397](#)
- SysLoaderExecuteFunction, [397](#)
- Write, [397](#)
- WriteBytes, [397](#)
- WriteBytesEx, [398](#)
- WriteLn, [398](#)
- WriteLnString, [399](#)
- WriteString, [399](#)
- LoaderModuleID
  - ModuleIDConstants, [180](#)
  - NBCCCommon.h, [902](#)
- LoaderModuleName
  - ModuleNameConstants, [178](#)
  - NBCCCommon.h, [902](#)
- LoaderOffsetFreeUserFlash
  - LoaderIOMAP, [501](#)
  - NBCCCommon.h, [902](#)
- LoaderOffsetPFunc
  - LoaderIOMAP, [501](#)
  - NBCCCommon.h, [903](#)
- Location
  - DrawFontType, [750](#)
  - DrawGraphicArrayType, [752](#)
  - DrawGraphicType, [754](#)
  - DrawPointType, [758](#)
  - DrawRectType, [760](#)
  - DrawTextType, [762](#)
- LocationType, [794](#)
  - X, [794](#)
  - Y, [794](#)
- Log
  - cmathAPI, [409](#)
  - NXCDefs.h, [1051](#)
- log
  - cmathAPI, [420](#)
  - NXCDefs.h, [1162](#)
- Log10
  - cmathAPI, [409](#)
  - NXCDefs.h, [1051](#)
- log10
  - cmathAPI, [421](#)
  - NXCDefs.h, [1162](#)
- LONG\_MAX
  - NBCCCommon.h, [903](#)
  - NXTLimits, [711](#)
- LONG\_MIN
  - NBCCCommon.h, [903](#)
  - NXTLimits, [712](#)
- LongAbort
  - NXCDefs.h, [1163](#)
  - UiModuleFunctions, [375](#)
- Loop
  - SoundPlayFileType, [806](#)
  - SoundPlayToneType, [808](#)
- Low level LowSpeed module functions, [293](#)
- Low Speed module, [60](#)
- Low speed module IOMAP offsets, [583](#)
- LowLevelLowSpeedModuleFunctions
  - GetLSInputBuffer, [294](#)
  - GetLSOutputBuffer, [294](#)
  - LSChannelState, [294](#)
  - LSErrorType, [295](#)
  - LSInputBufferBytesToRx, [295](#)
  - LSInputBufferInPtr, [295](#)
  - LSInputBufferOutPtr, [295](#)
  - LSMode, [296](#)
  - LSNoRestartOnRead, [296](#)
  - LSOutputBufferBytesToRx, [296](#)
  - LSOutputBufferInPtr, [297](#)
  - LSOutputBufferOutPtr, [297](#)
  - LSSpeed, [297](#)
  - LSState, [297](#)
- LowSpeed module constants, [577](#)
- LowSpeed module functions, [279](#)
- LowSpeed module system call functions, [299](#)
- LowSpeed module types, [278](#)
- LOWSPEED\_CH\_NOT\_READY
  - LowSpeedErrorTypeConstants, [582](#)
  - NBCCCommon.h, [903](#)
- LOWSPEED\_COMMUNICATING
  - LowSpeedChannelStateConstants, [579](#)
  - NBCCCommon.h, [903](#)
- LOWSPEED\_DATA\_RECEIVED
  - LowSpeedModeConstants, [581](#)
  - NBCCCommon.h, [903](#)
- LOWSPEED\_DONE
  - LowSpeedChannelStateConstants, [579](#)
  - NBCCCommon.h, [903](#)
- LOWSPEED\_ERROR
  - LowSpeedChannelStateConstants, [579](#)

- NBCCCommon.h, 903
- LOWSPEED\_IDLE
  - LowSpeedChannelStateConstants, 579
  - NBCCCommon.h, 903
- LOWSPEED\_INIT
  - LowSpeedChannelStateConstants, 579
  - NBCCCommon.h, 903
- LOWSPEED\_LOAD\_BUFFER
  - LowSpeedChannelStateConstants, 579
  - NBCCCommon.h, 903
- LOWSPEED\_NO\_ERROR
  - LowSpeedErrorTypeConstants, 582
  - NBCCCommon.h, 904
- LOWSPEED\_RECEIVING
  - LowSpeedModeConstants, 581
  - NBCCCommon.h, 904
- LOWSPEED\_RX\_ERROR
  - LowSpeedErrorTypeConstants, 582
  - NBCCCommon.h, 904
- LOWSPEED\_TRANSMITTING
  - LowSpeedModeConstants, 581
  - NBCCCommon.h, 904
- LOWSPEED\_TX\_ERROR
  - LowSpeedErrorTypeConstants, 582
  - NBCCCommon.h, 904
- LowspeedBytesReady
  - LowSpeedModuleFunctions, 288
  - NXCDefs.h, 1163
- LowSpeedChannelStateConstants
  - LOWSPEED\_COMMUNICATING, 579
  - LOWSPEED\_DONE, 579
  - LOWSPEED\_ERROR, 579
  - LOWSPEED\_IDLE, 579
  - LOWSPEED\_INIT, 579
  - LOWSPEED\_LOAD\_BUFFER, 579
- LowspeedCheckStatus
  - LowSpeedModuleFunctions, 288
  - NXCDefs.h, 1164
- LowSpeedErrorTypeConstants
  - LOWSPEED\_CH\_NOT\_READY, 582
  - LOWSPEED\_NO\_ERROR, 582
  - LOWSPEED\_RX\_ERROR, 582
  - LOWSPEED\_TX\_ERROR, 582
- LowSpeedIOMAP
  - LowSpeedOffsetChannelState, 583
  - LowSpeedOffsetErrorType, 583
  - LowSpeedOffsetInBufBuf, 583
  - LowSpeedOffsetInBufBytesToRx, 583
  - LowSpeedOffsetInBufInPtr, 583
  - LowSpeedOffsetInBufOutPtr, 583
  - LowSpeedOffsetMode, 584
  - LowSpeedOffsetNoRestartOnRead, 584
  - LowSpeedOffsetOutBufBuf, 584
  - LowSpeedOffsetOutBufBytesToRx, 584
  - LowSpeedOffsetOutBufInPtr, 584
  - LowSpeedOffsetOutBufOutPtr, 584
  - LowSpeedOffsetSpeed, 584
  - LowSpeedOffsetState, 584
- LowSpeedModeConstants
  - LOWSPEED\_DATA\_RECEIVED, 581
  - LOWSPEED\_RECEIVING, 581
  - LOWSPEED\_TRANSMITTING, 581
- LowSpeedModuleFunctions
  - I2CBytes, 281
  - I2CBytesReady, 281
  - I2CCheckStatus, 282
  - I2CDeviceId, 282
  - I2CDeviceIdEx, 282
  - I2CDeviceInfo, 283
  - I2CDeviceInfoEx, 283
  - I2CRead, 284
  - I2CSendCommand, 284
  - I2CSendCommandEx, 285
  - I2CStatus, 285
  - I2CVendorId, 286
  - I2CVendorIdEx, 286
  - I2CVersion, 286
  - I2CVersionEx, 287
  - I2CWrite, 287
  - LowspeedBytesReady, 288
  - LowspeedCheckStatus, 288
  - LowspeedRead, 289
  - LowspeedStatus, 289
  - LowspeedWrite, 290
  - ReadI2CRegister, 290
  - ReadSensorUSEx, 291
  - SensorUS, 291
  - WriteI2CRegister, 291
- LowSpeedModuleID
  - ModuleIDConstants, 180
  - NBCCCommon.h, 904
- LowSpeedModuleName
  - ModuleNameConstants, 178
  - NBCCCommon.h, 904
- LowSpeedModuleSystemCallFunctions
  - SysCommLSCheckStatus, 299
  - SysCommLSRead, 299
  - SysCommLSWrite, 299
  - SysCommLSWriteEx, 300
- LowSpeedNoRestartConstants
  - LSREAD\_NO\_RESTART\_1, 585
  - LSREAD\_NO\_RESTART\_2, 585
  - LSREAD\_NO\_RESTART\_3, 585
  - LSREAD\_NO\_RESTART\_4, 585
  - LSREAD\_NO\_RESTART\_MASK, 585
  - LSREAD\_RESTART\_ALL, 585
  - LSREAD\_RESTART\_NONE, 585
- LowSpeedOffsetChannelState



- LowSpeedIOMAP, 583
- NBCCCommon.h, 904
- LowSpeedOffsetErrorType
  - LowSpeedIOMAP, 583
  - NBCCCommon.h, 904
- LowSpeedOffsetInBufBuf
  - LowSpeedIOMAP, 583
  - NBCCCommon.h, 904
- LowSpeedOffsetInBufBytesToRx
  - LowSpeedIOMAP, 583
  - NBCCCommon.h, 904
- LowSpeedOffsetInBufInPtr
  - LowSpeedIOMAP, 583
  - NBCCCommon.h, 905
- LowSpeedOffsetInBufOutPtr
  - LowSpeedIOMAP, 583
  - NBCCCommon.h, 905
- LowSpeedOffsetMode
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetNoRestartOnRead
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetOutBufBuf
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetOutBufBytesToRx
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetOutBufInPtr
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetOutBufOutPtr
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetSpeed
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowSpeedOffsetState
  - LowSpeedIOMAP, 584
  - NBCCCommon.h, 905
- LowspeedRead
  - LowSpeedModuleFunctions, 289
  - NXCDefs.h, 1164
- LowSpeedStateConstants
  - COM\_CHANNEL\_FOUR\_ACTIVE, 578
  - COM\_CHANNEL\_NONE\_ACTIVE, 578
  - COM\_CHANNEL\_ONE\_ACTIVE, 578
  - COM\_CHANNEL\_THREE\_ACTIVE, 578
  - COM\_CHANNEL\_TWO\_ACTIVE, 578
- LowspeedStatus
  - LowSpeedModuleFunctions, 289
  - NXCDefs.h, 1165
- LowspeedWrite
  - LowSpeedModuleFunctions, 290
  - NXCDefs.h, 1165
- LR\_COULD\_NOT\_SAVE
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 905
- LR\_ENTRY\_REMOVED
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 906
- LR\_STORE\_IS\_FULL
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 906
- LR\_SUCCESS
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 906
- LR\_UNKNOWN\_ADDR
  - CommStatusCodesConstants, 633
  - NBCCCommon.h, 906
- LSChannelState
  - LowLevelLowSpeedModuleFunctions, 294
  - NXCDefs.h, 1166
- LSChannelState constants, 579
- LSErrorType
  - LowLevelLowSpeedModuleFunctions, 295
  - NXCDefs.h, 1166
- LSErrorType constants, 582
- LSInputBufferBytesToRx
  - LowLevelLowSpeedModuleFunctions, 295
  - NXCDefs.h, 1166
- LSInputBufferInPtr
  - LowLevelLowSpeedModuleFunctions, 295
  - NXCDefs.h, 1167
- LSInputBufferOutPtr
  - LowLevelLowSpeedModuleFunctions, 295
  - NXCDefs.h, 1167
- LSMode
  - LowLevelLowSpeedModuleFunctions, 296
  - NXCDefs.h, 1167
- LSMode constants, 581
- LSNoRestartOnRead
  - LowLevelLowSpeedModuleFunctions, 296
  - NXCDefs.h, 1168
- LSNoRestartOnRead constants, 585
- LSOutputBufferBytesToRx
  - LowLevelLowSpeedModuleFunctions, 296
  - NXCDefs.h, 1168
- LSOutputBufferInPtr
  - LowLevelLowSpeedModuleFunctions, 297
  - NXCDefs.h, 1168
- LSOutputBufferOutPtr
  - LowLevelLowSpeedModuleFunctions, 297
  - NXCDefs.h, 1169
- LSREAD\_NO\_RESTART\_1
  - LowSpeedNoRestartConstants, 585
  - NBCCCommon.h, 906

- LSREAD\_NO\_RESTART\_2
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSREAD\_NO\_RESTART\_3
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSREAD\_NO\_RESTART\_4
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSREAD\_NO\_RESTART\_MASK
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSREAD\_RESTART\_ALL
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSREAD\_RESTART\_NONE
  - LowSpeedNoRestartConstants, [585](#)
  - NBCCCommon.h, [906](#)
- LSSpeed
  - LowLevelLowSpeedModuleFunctions, [297](#)
  - NXCDefs.h, [1169](#)
- LSState
  - LowLevelLowSpeedModuleFunctions, [297](#)
  - NXCDefs.h, [1169](#)
- LSState constants, [578](#)
- Mailbox constants, [485](#)
- MAILBOX1
  - MailboxConstants, [485](#)
  - NBCCCommon.h, [907](#)
- MAILBOX10
  - MailboxConstants, [485](#)
  - NBCCCommon.h, [907](#)
- MAILBOX2
  - MailboxConstants, [485](#)
  - NBCCCommon.h, [907](#)
- MAILBOX3
  - MailboxConstants, [485](#)
  - NBCCCommon.h, [907](#)
- MAILBOX4
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MAILBOX5
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MAILBOX6
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MAILBOX7
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MAILBOX8
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MAILBOX9
  - MailboxConstants, [486](#)
  - NBCCCommon.h, [907](#)
- MailboxConstants
  - MAILBOX1, [485](#)
  - MAILBOX10, [485](#)
  - MAILBOX2, [485](#)
  - MAILBOX3, [485](#)
  - MAILBOX4, [486](#)
  - MAILBOX5, [486](#)
  - MAILBOX6, [486](#)
  - MAILBOX7, [486](#)
  - MAILBOX8, [486](#)
  - MAILBOX9, [486](#)
- MAX\_BT\_MSG\_SIZE
  - CommMiscConstants, [613](#)
  - NBCCCommon.h, [908](#)
- MaxVal
  - UpdateCalibCacheInfoType, [812](#)
- memcpy
  - cstringAPI, [450](#)
  - NXCDefs.h, [1169](#)
- memcpy
  - cstringAPI, [450](#)
  - NXCDefs.h, [1170](#)
- memmove
  - cstringAPI, [450](#)
  - NXCDefs.h, [1170](#)
- MENUICON\_CENTER
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [908](#)
- MENUICON\_LEFT
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [908](#)
- MENUICON\_RIGHT
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [908](#)
- MENUICONS
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [908](#)
- MENUTEXT
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [908](#)
- Message
  - DatalogWriteType, [742](#)
  - MessageReadType, [795](#)
  - MessageWriteType, [797](#)
- MessageRead
  - NBCCCommon.h, [908](#)
  - SysCallConstants, [473](#)
- MessageReadType, [795](#)
  - Message, [795](#)
  - QueueID, [795](#)
  - Remove, [795](#)



- Result, [795](#)
- MessageWrite
  - NBCCCommon.h, [908](#)
  - SysCallConstants, [473](#)
- MessageWriteType, [797](#)
  - Message, [797](#)
  - QueueID, [797](#)
  - Result, [797](#)
- MidStr
  - cstringAPI, [451](#)
  - NXCDefs.h, [1170](#)
- MIN\_1
  - NBCCCommon.h, [908](#)
  - TimeConstants, [479](#)
- MindSensors API Functions, [101](#)
- MindSensors device constants, [701](#)
- MindSensors DIST-Nx constants, [702](#)
- MindSensors nRLink constants, [709](#)
- MindSensors PSP-Nx button set 1 constants, [706](#)
- MindSensors PSP-Nx button set 2 constants, [707](#)
- MindSensors PSP-Nx constants, [704](#)
- MindSensorsAPI
  - DISTNxDistance, [112](#)
  - DISTNxDistanceEx, [112](#)
  - DISTNxGP2D12, [113](#)
  - DISTNxGP2D120, [113](#)
  - DISTNxGP2D120Ex, [113](#)
  - DISTNxGP2D12Ex, [114](#)
  - DISTNxGP2YA02, [114](#)
  - DISTNxGP2YA02Ex, [114](#)
  - DISTNxGP2YA21, [115](#)
  - DISTNxGP2YA21Ex, [115](#)
  - DISTNxMaxDistance, [115](#)
  - DISTNxMaxDistanceEx, [116](#)
  - DISTNxMinDistance, [116](#)
  - DISTNxMinDistanceEx, [116](#)
  - DISTNxModuleType, [117](#)
  - DISTNxModuleTypeEx, [117](#)
  - DISTNxNumPoints, [117](#)
  - DISTNxNumPointsEx, [118](#)
  - DISTNxVoltage, [118](#)
  - DISTNxVoltageEx, [118](#)
  - MSADPAOff, [119](#)
  - MSADPAOffEx, [119](#)
  - MSADPAOn, [119](#)
  - MSADPAOnEx, [120](#)
  - MSDeenergize, [120](#)
  - MSDeenergizeEx, [120](#)
  - MEnergize, [121](#)
  - MEnergizeEx, [121](#)
  - MSIRTrain, [121](#)
  - MSIRTrainEx, [122](#)
  - MSPFComboDirect, [122](#)
  - MSPFComboDirectEx, [123](#)
  - MSPFComboPWM, [123](#)
  - MSPFComboPWMLEx, [124](#)
  - MSPFRawOutput, [124](#)
  - MSPFRawOutputEx, [125](#)
  - MSPFRepeat, [125](#)
  - MSPFRepeatEx, [125](#)
  - MSPFSingleOutputCST, [126](#)
  - MSPFSingleOutputCSTEx, [126](#)
  - MSPFSingleOutputPWM, [127](#)
  - MSPFSingleOutputPWMLEx, [127](#)
  - MSPFSinglePin, [128](#)
  - MSPFSinglePinEx, [128](#)
  - MSPFTrain, [129](#)
  - MSPFTrainEx, [129](#)
  - MSRCXAbsVar, [130](#)
  - MSRCXAddToDatalog, [130](#)
  - MSRCXAndVar, [130](#)
  - MSRCXBatteryLevel, [131](#)
  - MSRCXBoot, [131](#)
  - MSRCXCalibrateEvent, [131](#)
  - MSRCXClearAllEvents, [131](#)
  - MSRCXClearCounter, [132](#)
  - MSRCXClearMsg, [132](#)
  - MSRCXClearSensor, [132](#)
  - MSRCXClearSound, [132](#)
  - MSRCXClearTimer, [132](#)
  - MSRCXCreateDatalog, [133](#)
  - MSRCXDecCounter, [133](#)
  - MSRCXDeleteSub, [133](#)
  - MSRCXDeleteSubs, [133](#)
  - MSRCXDeleteTask, [133](#)
  - MSRCXDeleteTasks, [134](#)
  - MSRCXDisableOutput, [134](#)
  - MSRCXDivVar, [134](#)
  - MSRCXEnableOutput, [134](#)
  - MSRCXEvent, [135](#)
  - MSRCXFloat, [135](#)
  - MSRCXFwd, [135](#)
  - MSRCXIncCounter, [135](#)
  - MSRCXInvertOutput, [136](#)
  - MSRCXMulVar, [136](#)
  - MSRCXMuteSound, [136](#)
  - MSRCXObvertOutput, [136](#)
  - MSRCXOff, [137](#)
  - MSRCXOn, [137](#)
  - MSRCXOnFor, [137](#)
  - MSRCXOnFwd, [137](#)
  - MSRCXOnRev, [138](#)
  - MSRCXOrVar, [138](#)
  - MSRCXPBTurnOff, [138](#)
  - MSRCXPing, [138](#)
  - MSRCXPlaySound, [139](#)
  - MSRCXPlayTone, [139](#)
  - MSRCXPlayToneVar, [139](#)

- MSRCXPoll, [139](#)
- MSRCXPollMemory, [140](#)
- MSRCXRemote, [140](#)
- MSRCXReset, [140](#)
- MSRCXRev, [140](#)
- MSRCXSelectDisplay, [141](#)
- MSRCXSelectProgram, [141](#)
- MSRCXSendSerial, [141](#)
- MSRCXSet, [141](#)
- MSRCXSetDirection, [142](#)
- MSRCXSetEvent, [142](#)
- MSRCXSetGlobalDirection, [142](#)
- MSRCXSetGlobalOutput, [143](#)
- MSRCXSetMaxPower, [143](#)
- MSRCXSetMessage, [143](#)
- MSRCXSetNRLinkPort, [143](#)
- MSRCXSetNRLinkPortEx, [144](#)
- MSRCXSetOutput, [144](#)
- MSRCXSetPower, [144](#)
- MSRCXSetPriority, [145](#)
- MSRCXSetSensorMode, [145](#)
- MSRCXSetSensorType, [145](#)
- MSRCXSetSleepTime, [145](#)
- MSRCXSetTxPower, [146](#)
- MSRCXSetUserDisplay, [146](#)
- MSRCXSetVar, [146](#)
- MSRCXSetWatch, [146](#)
- MSRCXSgnVar, [147](#)
- MSRCXStartTask, [147](#)
- MSRCXStopAllTasks, [147](#)
- MSRCXStopTask, [147](#)
- MSRCXSubVar, [148](#)
- MSRCXSumVar, [148](#)
- MSRCXToggle, [148](#)
- MSRCXUnlock, [148](#)
- MSRCXUnmuteSound, [149](#)
- MSReadValue, [149](#)
- MSReadValueEx, [149](#)
- MSScoutCalibrateSensor, [150](#)
- MSScoutMuteSound, [150](#)
- MSScoutSelectSounds, [150](#)
- MSScoutSendVLL, [150](#)
- MSScoutSetCounterLimit, [150](#)
- MSScoutSetEventFeedback, [151](#)
- MSScoutSetLight, [151](#)
- MSScoutSetScoutMode, [151](#)
- MSScoutSetScoutRules, [151](#)
- MSScoutSetSensorClickTime, [152](#)
- MSScoutSetSensorHysteresis, [152](#)
- MSScoutSetSensorLowerLimit, [152](#)
- MSScoutSetSensorUpperLimit, [153](#)
- MSScoutSetTimerLimit, [153](#)
- MSScoutUnmuteSound, [153](#)
- NRLink2400, [153](#)
- NRLink2400Ex, [154](#)
- NRLink4800, [154](#)
- NRLink4800Ex, [154](#)
- NRLinkFlush, [155](#)
- NRLinkFlushEx, [155](#)
- NRLinkIRLong, [155](#)
- NRLinkIRLongEx, [156](#)
- NRLinkIRShort, [156](#)
- NRLinkIRShortEx, [156](#)
- NRLinkSetPF, [157](#)
- NRLinkSetPFEx, [157](#)
- NRLinkSetRCX, [157](#)
- NRLinkSetRCXEx, [158](#)
- NRLinkSetTrain, [158](#)
- NRLinkSetTrainEx, [158](#)
- NRLinkStatus, [159](#)
- NRLinkStatusEx, [159](#)
- NRLinkTxRaw, [159](#)
- NRLinkTxRawEx, [160](#)
- PSPNxAnalog, [160](#)
- PSPNxAnalogEx, [160](#)
- PSPNxDigital, [161](#)
- PSPNxDigitalEx, [161](#)
- ReadNRLinkBytes, [161](#)
- ReadNRLinkBytesEx, [162](#)
- ReadSensorMSAccel, [162](#)
- ReadSensorMSAccelEx, [163](#)
- ReadSensorMSPlayStation, [163](#)
- ReadSensorMSPlayStationEx, [163](#)
- ReadSensorMSRTClock, [164](#)
- ReadSensorMSTilt, [164](#)
- ReadSensorMSTiltEx, [165](#)
- RunNRLinkMacro, [165](#)
- RunNRLinkMacroEx, [166](#)
- SensorMSCompass, [166](#)
- SensorMSCompassEx, [166](#)
- SensorMSDROD, [167](#)
- SensorMSPressure, [167](#)
- SensorMSPressureRaw, [167](#)
- SetSensorMSDROD, [167](#)
- SetSensorMSPressure, [168](#)
- WriteNRLinkBytes, [168](#)
- WriteNRLinkBytesEx, [168](#)
- MindSensorsConstants
  - MS\_CMD\_ADPA\_OFF, [701](#)
  - MS\_CMD\_ADPA\_ON, [701](#)
  - MS\_CMD\_DEENERGIZED, [701](#)
  - MS\_CMD\_ENERGIZED, [701](#)
- MinVal
  - UpdateCalibCacheInfoType, [812](#)
- MiscConstants
  - DEGREES\_PER\_RADIAN, [181](#)
  - FALSE, [181](#)
  - NA, [181](#)

- PI, [181](#)
- RADIANS\_PER\_DEGREE, [182](#)
- TRUE, [182](#)
- Miscellaneous Comm module constants, [613](#)
- Miscellaneous NBC/NXC constants, [181](#)
- Mode
  - CommHSControlType, [729](#)
- ModuleID
  - IOMapReadByIDType, [780](#)
  - IOMapWriteByIDType, [784](#)
- ModuleIDConstants
  - ButtonModuleID, [179](#)
  - CommandModuleID, [179](#)
  - CommModuleID, [179](#)
  - DisplayModuleID, [179](#)
  - InputModuleID, [179](#)
  - IOCtrlModuleID, [179](#)
  - LoaderModuleID, [180](#)
  - LowSpeedModuleID, [180](#)
  - OutputModuleID, [180](#)
  - SoundModuleID, [180](#)
  - UIModuleID, [180](#)
- ModuleName
  - IOMapReadType, [782](#)
  - IOMapWriteType, [786](#)
- ModuleNameConstants
  - ButtonModuleName, [177](#)
  - CommandModuleName, [177](#)
  - CommModuleName, [177](#)
  - DisplayModuleName, [177](#)
  - InputModuleName, [177](#)
  - IOCtrlModuleName, [177](#)
  - LoaderModuleName, [178](#)
  - LowSpeedModuleName, [178](#)
  - OutputModuleName, [178](#)
  - SoundModuleName, [178](#)
  - UIModuleName, [178](#)
- MotorActualSpeed
  - NXCDefs.h, [1171](#)
  - OutputModuleFunctions, [230](#)
- MotorBlockTachoCount
  - NXCDefs.h, [1171](#)
  - OutputModuleFunctions, [230](#)
- MotorMode
  - NXCDefs.h, [1171](#)
  - OutputModuleFunctions, [230](#)
- MotorOverload
  - NXCDefs.h, [1171](#)
  - OutputModuleFunctions, [231](#)
- MotorPower
  - NXCDefs.h, [1172](#)
  - OutputModuleFunctions, [231](#)
- MotorPwnFreq
  - NXCDefs.h, [1172](#)
  - OutputModuleFunctions, [231](#)
- MotorRegDValue
  - NXCDefs.h, [1172](#)
  - OutputModuleFunctions, [231](#)
- MotorRegIValue
  - NXCDefs.h, [1172](#)
  - OutputModuleFunctions, [232](#)
- MotorRegPValue
  - NXCDefs.h, [1173](#)
  - OutputModuleFunctions, [232](#)
- MotorRegulation
  - NXCDefs.h, [1173](#)
  - OutputModuleFunctions, [232](#)
- MotorRotationCount
  - NXCDefs.h, [1173](#)
  - OutputModuleFunctions, [232](#)
- MotorRunState
  - NXCDefs.h, [1173](#)
  - OutputModuleFunctions, [233](#)
- MotorTachoCount
  - NXCDefs.h, [1174](#)
  - OutputModuleFunctions, [233](#)
- MotorTachoLimit
  - NXCDefs.h, [1174](#)
  - OutputModuleFunctions, [233](#)
- MotorTurnRatio
  - NXCDefs.h, [1174](#)
  - OutputModuleFunctions, [233](#)
- MS\_1
  - NBCCCommon.h, [908](#)
  - TimeConstants, [479](#)
- MS\_10
  - NBCCCommon.h, [908](#)
  - TimeConstants, [479](#)
- MS\_100
  - NBCCCommon.h, [909](#)
  - TimeConstants, [479](#)
- MS\_150
  - NBCCCommon.h, [909](#)
  - TimeConstants, [479](#)
- MS\_2
  - NBCCCommon.h, [909](#)
  - TimeConstants, [479](#)
- MS\_20
  - NBCCCommon.h, [909](#)
  - TimeConstants, [479](#)
- MS\_200
  - NBCCCommon.h, [909](#)
  - TimeConstants, [479](#)
- MS\_250
  - NBCCCommon.h, [909](#)
  - TimeConstants, [480](#)
- MS\_3
  - NBCCCommon.h, [909](#)

- TimeConstants, [480](#)
- MS\_30
  - NBCCCommon.h, [909](#)
  - TimeConstants, [480](#)
- MS\_300
  - NBCCCommon.h, [909](#)
  - TimeConstants, [480](#)
- MS\_350
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_4
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_40
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_400
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_450
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_5
  - NBCCCommon.h, [910](#)
  - TimeConstants, [480](#)
- MS\_50
  - NBCCCommon.h, [910](#)
  - TimeConstants, [481](#)
- MS\_500
  - NBCCCommon.h, [910](#)
  - TimeConstants, [481](#)
- MS\_6
  - NBCCCommon.h, [910](#)
  - TimeConstants, [481](#)
- MS\_60
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_600
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_7
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_70
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_700
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_8
  - NBCCCommon.h, [911](#)
  - TimeConstants, [481](#)
- MS\_80
  - NBCCCommon.h, [911](#)
- TimeConstants, [482](#)
- MS\_800
  - NBCCCommon.h, [911](#)
  - TimeConstants, [482](#)
- MS\_9
  - NBCCCommon.h, [911](#)
  - TimeConstants, [482](#)
- MS\_90
  - NBCCCommon.h, [911](#)
  - TimeConstants, [482](#)
- MS\_900
  - NBCCCommon.h, [911](#)
  - TimeConstants, [482](#)
- MS\_CMD\_ADPA\_OFF
  - MindSensorsConstants, [701](#)
  - NBCCCommon.h, [912](#)
- MS\_CMD\_ADPA\_ON
  - MindSensorsConstants, [701](#)
  - NBCCCommon.h, [912](#)
- MS\_CMD\_DEENERGIZED
  - MindSensorsConstants, [701](#)
  - NBCCCommon.h, [912](#)
- MS\_CMD\_ENERGIZED
  - MindSensorsConstants, [701](#)
  - NBCCCommon.h, [912](#)
- MSADPAOff
  - MindSensorsAPI, [119](#)
  - NXCDefs.h, [1174](#)
- MSADPAOffEx
  - MindSensorsAPI, [119](#)
  - NXCDefs.h, [1175](#)
- MSADPAOn
  - MindSensorsAPI, [119](#)
  - NXCDefs.h, [1175](#)
- MSADPAOnEx
  - MindSensorsAPI, [120](#)
  - NXCDefs.h, [1175](#)
- MSDeenergize
  - MindSensorsAPI, [120](#)
  - NXCDefs.h, [1176](#)
- MSDeenergizeEx
  - MindSensorsAPI, [120](#)
  - NXCDefs.h, [1176](#)
- MSDistNX
  - DIST\_CMD\_CUSTOM, [702](#)
  - DIST\_CMD\_GP2D12, [702](#)
  - DIST\_CMD\_GP2D120, [702](#)
  - DIST\_CMD\_GP2YA02, [702](#)
  - DIST\_CMD\_GP2YA21, [702](#)
  - DIST\_REG\_DIST, [702](#)
  - DIST\_REG\_DIST1, [703](#)
  - DIST\_REG\_DIST\_MAX, [703](#)
  - DIST\_REG\_DIST\_MIN, [703](#)
  - DIST\_REG\_MODULE\_TYPE, [703](#)

- DIST\_REG\_NUM\_POINTS, 703
- DIST\_REG\_VOLT, 703
- DIST\_REG\_VOLT1, 703
- MSEnerize
  - MindSensorsAPI, 121
  - NXCDefs.h, 1176
- MSEnerizeEx
  - MindSensorsAPI, 121
  - NXCDefs.h, 1177
- MSIRTrain
  - MindSensorsAPI, 121
  - NXCDefs.h, 1177
- MSIRTrainEx
  - MindSensorsAPI, 122
  - NXCDefs.h, 1177
- MSNRLink
  - NRLINK\_CMD\_2400, 709
  - NRLINK\_CMD\_4800, 709
  - NRLINK\_CMD\_FLUSH, 709
  - NRLINK\_CMD\_IR\_LONG, 709
  - NRLINK\_CMD\_IR\_SHORT, 709
  - NRLINK\_CMD\_RUN\_MACRO, 709
  - NRLINK\_CMD\_SET\_PF, 710
  - NRLINK\_CMD\_SET\_RCX, 710
  - NRLINK\_CMD\_SET\_TRAIN, 710
  - NRLINK\_CMD\_TX\_RAW, 710
  - NRLINK\_REG\_BYTES, 710
  - NRLINK\_REG\_DATA, 710
  - NRLINK\_REG\_EEPROM, 710
- MSPFComboDirect
  - MindSensorsAPI, 122
  - NXCDefs.h, 1178
- MSPFComboDirectEx
  - MindSensorsAPI, 123
  - NXCDefs.h, 1178
- MSPFComboPWM
  - MindSensorsAPI, 123
  - NXCDefs.h, 1179
- MSPFComboPWMLEx
  - MindSensorsAPI, 124
  - NXCDefs.h, 1179
- MSPFRawOutput
  - MindSensorsAPI, 124
  - NXCDefs.h, 1180
- MSPFRawOutputEx
  - MindSensorsAPI, 125
  - NXCDefs.h, 1180
- MSPFRepeat
  - MindSensorsAPI, 125
  - NXCDefs.h, 1181
- MSPFRepeatEx
  - MindSensorsAPI, 125
  - NXCDefs.h, 1181
- MSPFSingleOutputCST
  - MindSensorsAPI, 126
  - NXCDefs.h, 1181
- MSPFSingleOutputCSTEx
  - MindSensorsAPI, 126
  - NXCDefs.h, 1182
- MSPFSingleOutputPWM
  - MindSensorsAPI, 127
  - NXCDefs.h, 1182
- MSPFSingleOutputPWMLEx
  - MindSensorsAPI, 127
  - NXCDefs.h, 1183
- MSPFSinglePin
  - MindSensorsAPI, 128
  - NXCDefs.h, 1183
- MSPFSinglePinEx
  - MindSensorsAPI, 128
  - NXCDefs.h, 1184
- MSPFTrain
  - MindSensorsAPI, 129
  - NXCDefs.h, 1184
- MSPFTrainEx
  - MindSensorsAPI, 129
  - NXCDefs.h, 1185
- MSPSPNX
  - PSP\_CMD\_ANALOG, 704
  - PSP\_CMD\_DIGITAL, 704
  - PSP\_REG\_BTNSET1, 704
  - PSP\_REG\_BTNSET2, 704
  - PSP\_REG\_XLEFT, 704
  - PSP\_REG\_XRIGHT, 705
  - PSP\_REG\_YLEFT, 705
  - PSP\_REG\_YRIGHT, 705
- MSPSPNXBtnSet1
  - PSP\_BTNSET1\_DOWN, 706
  - PSP\_BTNSET1\_L3, 706
  - PSP\_BTNSET1\_LEFT, 706
  - PSP\_BTNSET1\_R3, 706
  - PSP\_BTNSET1\_RIGHT, 706
  - PSP\_BTNSET1\_UP, 706
- MSPSPNXBtnSet2
  - PSP\_BTNSET2\_CIRCLE, 707
  - PSP\_BTNSET2\_CROSS, 707
  - PSP\_BTNSET2\_L1, 707
  - PSP\_BTNSET2\_L2, 707
  - PSP\_BTNSET2\_R1, 707
  - PSP\_BTNSET2\_R2, 707
  - PSP\_BTNSET2\_SQUARE, 707
  - PSP\_BTNSET2\_TRIANGLE, 708
- MSRCXAbsVar
  - MindSensorsAPI, 130
  - NXCDefs.h, 1185
- MSRCXAddToDatalog
  - MindSensorsAPI, 130
  - NXCDefs.h, 1186

- MSRCXAndVar
  - MindSensorsAPI, [130](#)
  - NXCDefs.h, [1186](#)
- MSRCXBatteryLevel
  - MindSensorsAPI, [131](#)
  - NXCDefs.h, [1186](#)
- MSRCXBoot
  - MindSensorsAPI, [131](#)
  - NXCDefs.h, [1186](#)
- MSRCXCalibrateEvent
  - MindSensorsAPI, [131](#)
  - NXCDefs.h, [1187](#)
- MSRCXClearAllEvents
  - MindSensorsAPI, [131](#)
  - NXCDefs.h, [1187](#)
- MSRCXClearCounter
  - MindSensorsAPI, [132](#)
  - NXCDefs.h, [1187](#)
- MSRCXClearMsg
  - MindSensorsAPI, [132](#)
  - NXCDefs.h, [1187](#)
- MSRCXClearSensor
  - MindSensorsAPI, [132](#)
  - NXCDefs.h, [1187](#)
- MSRCXClearSound
  - MindSensorsAPI, [132](#)
  - NXCDefs.h, [1188](#)
- MSRCXClearTimer
  - MindSensorsAPI, [132](#)
  - NXCDefs.h, [1188](#)
- MSRCXCreateDatalog
  - MindSensorsAPI, [133](#)
  - NXCDefs.h, [1188](#)
- MSRCXDecCounter
  - MindSensorsAPI, [133](#)
  - NXCDefs.h, [1188](#)
- MSRCXDeleteSub
  - MindSensorsAPI, [133](#)
  - NXCDefs.h, [1188](#)
- MSRCXDeleteSubs
  - MindSensorsAPI, [133](#)
  - NXCDefs.h, [1189](#)
- MSRCXDeleteTask
  - MindSensorsAPI, [133](#)
  - NXCDefs.h, [1189](#)
- MSRCXDeleteTasks
  - MindSensorsAPI, [134](#)
  - NXCDefs.h, [1189](#)
- MSRCXDisableOutput
  - MindSensorsAPI, [134](#)
  - NXCDefs.h, [1189](#)
- MSRCXDivVar
  - MindSensorsAPI, [134](#)
  - NXCDefs.h, [1189](#)
- MSRCXEnableOutput
  - MindSensorsAPI, [134](#)
  - NXCDefs.h, [1190](#)
- MSRCXEvent
  - MindSensorsAPI, [135](#)
  - NXCDefs.h, [1190](#)
- MSRCXFloat
  - MindSensorsAPI, [135](#)
  - NXCDefs.h, [1190](#)
- MSRCXFwd
  - MindSensorsAPI, [135](#)
  - NXCDefs.h, [1190](#)
- MSRCXIncCounter
  - MindSensorsAPI, [135](#)
  - NXCDefs.h, [1191](#)
- MSRCXInvertOutput
  - MindSensorsAPI, [136](#)
  - NXCDefs.h, [1191](#)
- MSRCXMulVar
  - MindSensorsAPI, [136](#)
  - NXCDefs.h, [1191](#)
- MSRCXMuteSound
  - MindSensorsAPI, [136](#)
  - NXCDefs.h, [1191](#)
- MSRCXObvertOutput
  - MindSensorsAPI, [136](#)
  - NXCDefs.h, [1192](#)
- MSRCXOff
  - MindSensorsAPI, [137](#)
  - NXCDefs.h, [1192](#)
- MSRCXOn
  - MindSensorsAPI, [137](#)
  - NXCDefs.h, [1192](#)
- MSRCXOnFor
  - MindSensorsAPI, [137](#)
  - NXCDefs.h, [1192](#)
- MSRCXOnFwd
  - MindSensorsAPI, [137](#)
  - NXCDefs.h, [1193](#)
- MSRCXOnRev
  - MindSensorsAPI, [138](#)
  - NXCDefs.h, [1193](#)
- MSRCXOrVar
  - MindSensorsAPI, [138](#)
  - NXCDefs.h, [1193](#)
- MSRCXPBTurnOff
  - MindSensorsAPI, [138](#)
  - NXCDefs.h, [1193](#)
- MSRCXPing
  - MindSensorsAPI, [138](#)
  - NXCDefs.h, [1193](#)
- MSRCXPlaySound
  - MindSensorsAPI, [139](#)
  - NXCDefs.h, [1194](#)

- MSRCXPlayTone
  - MindSensorsAPI, [139](#)
  - NXCDefs.h, [1194](#)
- MSRCXPlayToneVar
  - MindSensorsAPI, [139](#)
  - NXCDefs.h, [1194](#)
- MSRCXPoll
  - MindSensorsAPI, [139](#)
  - NXCDefs.h, [1194](#)
- MSRCXPollMemory
  - MindSensorsAPI, [140](#)
  - NXCDefs.h, [1195](#)
- MSRCXRemote
  - MindSensorsAPI, [140](#)
  - NXCDefs.h, [1195](#)
- MSRCXReset
  - MindSensorsAPI, [140](#)
  - NXCDefs.h, [1195](#)
- MSRCXRev
  - MindSensorsAPI, [140](#)
  - NXCDefs.h, [1195](#)
- MSRCXSelectDisplay
  - MindSensorsAPI, [141](#)
  - NXCDefs.h, [1196](#)
- MSRCXSelectProgram
  - MindSensorsAPI, [141](#)
  - NXCDefs.h, [1196](#)
- MSRCXSendSerial
  - MindSensorsAPI, [141](#)
  - NXCDefs.h, [1196](#)
- MSRCXSet
  - MindSensorsAPI, [141](#)
  - NXCDefs.h, [1196](#)
- MSRCXSetDirection
  - MindSensorsAPI, [142](#)
  - NXCDefs.h, [1197](#)
- MSRCXSetEvent
  - MindSensorsAPI, [142](#)
  - NXCDefs.h, [1197](#)
- MSRCXSetGlobalDirection
  - MindSensorsAPI, [142](#)
  - NXCDefs.h, [1197](#)
- MSRCXSetGlobalOutput
  - MindSensorsAPI, [143](#)
  - NXCDefs.h, [1198](#)
- MSRCXSetMaxPower
  - MindSensorsAPI, [143](#)
  - NXCDefs.h, [1198](#)
- MSRCXSetMessage
  - MindSensorsAPI, [143](#)
  - NXCDefs.h, [1198](#)
- MSRCXSetNRLinkPort
  - MindSensorsAPI, [143](#)
  - NXCDefs.h, [1198](#)
- MSRCXSetNRLinkPortEx
  - MindSensorsAPI, [144](#)
  - NXCDefs.h, [1199](#)
- MSRCXSetOutput
  - MindSensorsAPI, [144](#)
  - NXCDefs.h, [1199](#)
- MSRCXSetPower
  - MindSensorsAPI, [144](#)
  - NXCDefs.h, [1199](#)
- MSRCXSetPriority
  - MindSensorsAPI, [145](#)
  - NXCDefs.h, [1200](#)
- MSRCXSetSensorMode
  - MindSensorsAPI, [145](#)
  - NXCDefs.h, [1200](#)
- MSRCXSetSensorType
  - MindSensorsAPI, [145](#)
  - NXCDefs.h, [1200](#)
- MSRCXSetSleepTime
  - MindSensorsAPI, [145](#)
  - NXCDefs.h, [1200](#)
- MSRCXSetTxPower
  - MindSensorsAPI, [146](#)
  - NXCDefs.h, [1201](#)
- MSRCXSetUserDisplay
  - MindSensorsAPI, [146](#)
  - NXCDefs.h, [1201](#)
- MSRCXSetVar
  - MindSensorsAPI, [146](#)
  - NXCDefs.h, [1201](#)
- MSRCXSetWatch
  - MindSensorsAPI, [146](#)
  - NXCDefs.h, [1201](#)
- MSRCXSgnVar
  - MindSensorsAPI, [147](#)
  - NXCDefs.h, [1202](#)
- MSRCXStartTask
  - MindSensorsAPI, [147](#)
  - NXCDefs.h, [1202](#)
- MSRCXStopAllTasks
  - MindSensorsAPI, [147](#)
  - NXCDefs.h, [1202](#)
- MSRCXStopTask
  - MindSensorsAPI, [147](#)
  - NXCDefs.h, [1202](#)
- MSRCXSubVar
  - MindSensorsAPI, [148](#)
  - NXCDefs.h, [1203](#)
- MSRCXSumVar
  - MindSensorsAPI, [148](#)
  - NXCDefs.h, [1203](#)
- MSRCXToggle
  - MindSensorsAPI, [148](#)
  - NXCDefs.h, [1203](#)



- MSRCXUnlock
  - MindSensorsAPI, [148](#)
  - NXCDefs.h, [1203](#)
- MSRCXUnmuteSound
  - MindSensorsAPI, [149](#)
  - NXCDefs.h, [1204](#)
- MSReadValue
  - MindSensorsAPI, [149](#)
  - NXCDefs.h, [1204](#)
- MSReadValueEx
  - MindSensorsAPI, [149](#)
  - NXCDefs.h, [1204](#)
- MSScoutCalibrateSensor
  - MindSensorsAPI, [150](#)
  - NXCDefs.h, [1205](#)
- MSScoutMuteSound
  - MindSensorsAPI, [150](#)
  - NXCDefs.h, [1205](#)
- MSScoutSelectSounds
  - MindSensorsAPI, [150](#)
  - NXCDefs.h, [1205](#)
- MSScoutSendVLL
  - MindSensorsAPI, [150](#)
  - NXCDefs.h, [1205](#)
- MSScoutSetCounterLimit
  - MindSensorsAPI, [150](#)
  - NXCDefs.h, [1205](#)
- MSScoutSetEventFeedback
  - MindSensorsAPI, [151](#)
  - NXCDefs.h, [1206](#)
- MSScoutSetLight
  - MindSensorsAPI, [151](#)
  - NXCDefs.h, [1206](#)
- MSScoutSetScoutMode
  - MindSensorsAPI, [151](#)
  - NXCDefs.h, [1206](#)
- MSScoutSetScoutRules
  - MindSensorsAPI, [151](#)
  - NXCDefs.h, [1206](#)
- MSScoutSetSensorClickTime
  - MindSensorsAPI, [152](#)
  - NXCDefs.h, [1207](#)
- MSScoutSetSensorHysteresis
  - MindSensorsAPI, [152](#)
  - NXCDefs.h, [1207](#)
- MSScoutSetSensorLowerLimit
  - MindSensorsAPI, [152](#)
  - NXCDefs.h, [1207](#)
- MSScoutSetSensorUpperLimit
  - MindSensorsAPI, [153](#)
  - NXCDefs.h, [1208](#)
- MSScoutSetTimerLimit
  - MindSensorsAPI, [153](#)
  - NXCDefs.h, [1208](#)
- MSScoutUnmuteSound
  - MindSensorsAPI, [153](#)
  - NXCDefs.h, [1208](#)
- MulDiv32
  - cmathAPI, [409](#)
  - NXCDefs.h, [1052](#)
- muldiv32
  - cmathAPI, [421](#)
  - NXCDefs.h, [1208](#)
- NA
  - MiscConstants, [181](#)
  - NBCCCommon.h, [912](#)
- Name
  - CommBTConnectionType, [722](#)
  - CommExecuteFunctionType, [727](#)
  - ComputeCalibValueType, [739](#)
  - UpdateCalibCacheInfoType, [812](#)
- NBC Input port constants, [543](#)
- NBC sensor mode constants, [547](#)
- NBC sensor type constants, [544](#)
- NBCCCommon.h, [817](#)
  - ActualSpeed, [846](#)
  - BITMAP\_1, [846](#)
  - BITMAP\_2, [846](#)
  - BITMAP\_3, [846](#)
  - BITMAP\_4, [847](#)
  - BITMAPS, [847](#)
  - BlockTachoCount, [847](#)
  - BREAKOUT\_REQ, [847](#)
  - BT\_ARM\_CMD\_MODE, [847](#)
  - BT\_ARM\_DATA\_MODE, [847](#)
  - BT\_ARM\_OFF, [847](#)
  - BT\_BRICK\_PORT\_OPEN, [847](#)
  - BT\_BRICK\_VISIBILITY, [847](#)
  - BT\_CMD\_BYTE, [847](#)
  - BT\_CMD\_READY, [848](#)
  - BT\_CONNECTION\_0\_ENABLE, [848](#)
  - BT\_CONNECTION\_1\_ENABLE, [848](#)
  - BT\_CONNECTION\_2\_ENABLE, [848](#)
  - BT\_CONNECTION\_3\_ENABLE, [848](#)
  - BT\_DEFAULT\_INQUIRY\_MAX, [848](#)
  - BT\_DEFAULT\_INQUIRY\_TIMEOUT\_LO, [848](#)
  - BT\_DEVICE\_AWAY, [848](#)
  - BT\_DEVICE\_EMPTY, [848](#)
  - BT\_DEVICE\_KNOWN, [848](#)
  - BT\_DEVICE\_NAME, [848](#)
  - BT\_DEVICE\_UNKNOWN, [849](#)
  - BT\_DISABLE, [849](#)
  - BT\_ENABLE, [849](#)
  - BTN1, [849](#)
  - BTN2, [849](#)
  - BTN3, [849](#)



- BTN4, 849
- BTNCENTER, 849
- BTNEXIT, 849
- BTNLEFT, 850
- BTNRIGHT, 850
- BTNSTATE\_LONG\_PRESSED\_EV, 850
- BTNSTATE\_LONG\_RELEASED\_EV, 850
- BTNSTATE\_NONE, 850
- BTNSTATE\_PRESSED\_EV, 850
- BTNSTATE\_PRESSED\_STATE, 850
- BTNSTATE\_SHORT\_RELEASED\_EV, 851
- ButtonModuleID, 851
- ButtonModuleName, 851
- ButtonOffsetLongPressCnt, 851
- ButtonOffsetLongRelCnt, 851
- ButtonOffsetPressedCnt, 851
- ButtonOffsetRelCnt, 851
- ButtonOffsetShortRelCnt, 851
- ButtonOffsetState, 851
- CHAR\_BIT, 851
- CHAR\_MAX, 851
- CHAR\_MIN, 852
- CLUMP\_DONE, 852
- CLUMP\_SUSPEND, 852
- ColorSensorRead, 852
- COM\_CHANNEL\_FOUR\_ACTIVE, 852
- COM\_CHANNEL\_NONE\_ACTIVE, 852
- COM\_CHANNEL\_ONE\_ACTIVE, 852
- COM\_CHANNEL\_THREE\_ACTIVE, 852
- COM\_CHANNEL\_TWO\_ACTIVE, 852
- CommandModuleID, 852
- CommandModuleName, 852
- CommandOffsetActivateFlag, 853
- CommandOffsetAwake, 853
- CommandOffsetDeactivateFlag, 853
- CommandOffsetFileName, 853
- CommandOffsetFormatString, 853
- CommandOffsetMemoryPool, 853
- CommandOffsetOffsetDS, 853
- CommandOffsetOffsetDVA, 853
- CommandOffsetPRCHandler, 853
- CommandOffsetProgStatus, 853
- CommandOffsetSyncTick, 854
- CommandOffsetSyncTime, 854
- CommandOffsetTick, 854
- CommBTCheckStatus, 854
- CommBTConnection, 854
- CommBTOnOff, 854
- CommBTRead, 854
- CommBTWrite, 854
- CommExecuteFunction, 854
- CommHSCheckStatus, 854
- CommHSControl, 854
- CommHSRead, 855
- CommHSWrite, 855
- CommLSCheckStatus, 855
- CommLSRead, 855
- CommLSWrite, 855
- CommLSWriteEx, 855
- CommModuleID, 855
- CommModuleName, 855
- CommOffsetBrickDataBdAddr, 855
- CommOffsetBrickDataBluecoreVersion, 855
- CommOffsetBrickDataBtHwStatus, 855
- CommOffsetBrickDataBtStateStatus, 856
- CommOffsetBrickDataName, 856
- CommOffsetBrickDataTimeOutValue, 856
- CommOffsetBtConnectTableBdAddr, 856
- CommOffsetBtConnectTableClassOfDevice, 856
- CommOffsetBtConnectTableHandleNr, 856
- CommOffsetBtConnectTableLinkQuality, 856
- CommOffsetBtConnectTableName, 856
- CommOffsetBtConnectTablePinCode, 856
- CommOffsetBtConnectTableStreamStatus, 856
- CommOffsetBtDeviceCnt, 856
- CommOffsetBtDeviceNameCnt, 857
- CommOffsetBtDeviceTableBdAddr, 857
- CommOffsetBtDeviceTableClassOfDevice, 857
- CommOffsetBtDeviceTableDeviceStatus, 857
- CommOffsetBtDeviceTableName, 857
- CommOffsetBtInBufBuf, 857
- CommOffsetBtInBufInPtr, 857
- CommOffsetBtInBufOutPtr, 857
- CommOffsetBtOutBufBuf, 857
- CommOffsetBtOutBufInPtr, 857
- CommOffsetBtOutBufOutPtr, 857
- CommOffsetHsFlags, 858
- CommOffsetHsInBufBuf, 858
- CommOffsetHsInBufInPtr, 858
- CommOffsetHsInBufOutPtr, 858
- CommOffsetHsMode, 858
- CommOffsetHsOutBufBuf, 858
- CommOffsetHsOutBufInPtr, 858
- CommOffsetHsOutBufOutPtr, 858
- CommOffsetHsSpeed, 858
- CommOffsetHsState, 858
- CommOffsetPFunc, 858
- CommOffsetPFuncTwo, 859
- CommOffsetUsbInBufBuf, 859
- CommOffsetUsbInBufInPtr, 859
- CommOffsetUsbInBufOutPtr, 859
- CommOffsetUsbOutBufBuf, 859
- CommOffsetUsbOutBufInPtr, 859
- CommOffsetUsbOutBufOutPtr, 859
- CommOffsetUsbPollBufBuf, 859

- CommOffsetUsbPollBufInPtr, 859
- CommOffsetUsbPollBufOutPtr, 859
- CommOffsetUsbState, 859
- ComputeCalibValue, 860
- DatalogGetTimes, 860
- DatalogWrite, 860
- DEGREES\_PER\_RADIAN, 860
- DISPLAY\_BUSY, 860
- DISPLAY\_CHAR, 860
- DISPLAY\_CONTRAST\_DEFAULT, 860
- DISPLAY\_CONTRAST\_MAX, 860
- DISPLAY\_ERASE\_ALL, 860
- DISPLAY\_ERASE\_LINE, 861
- DISPLAY\_FILL\_REGION, 861
- DISPLAY\_FRAME, 861
- DISPLAY\_HEIGHT, 861
- DISPLAY\_HORIZONTAL\_LINE, 861
- DISPLAY\_MENUICONS\_X\_DIFF, 861
- DISPLAY\_MENUICONS\_X\_OFFS, 861
- DISPLAY\_MENUICONS\_Y, 861
- DISPLAY\_ON, 861
- DISPLAY\_PIXEL, 861
- DISPLAY\_POPUP, 861
- DISPLAY\_REFRESH, 862
- DISPLAY\_REFRESH\_DISABLED, 862
- DISPLAY\_VERTICAL\_LINE, 862
- DISPLAY\_WIDTH, 862
- DisplayExecuteFunction, 862
- DisplayModuleID, 862
- DisplayModuleName, 862
- DisplayOffsetContrast, 862
- DisplayOffsetDisplay, 862
- DisplayOffsetEraseMask, 862
- DisplayOffsetFlags, 863
- DisplayOffsetNormal, 863
- DisplayOffsetPBitmaps, 863
- DisplayOffsetPFont, 863
- DisplayOffsetPFunc, 863
- DisplayOffsetPMenuIcons, 863
- DisplayOffsetPMenuText, 863
- DisplayOffsetPopup, 863
- DisplayOffsetPScreens, 863
- DisplayOffsetPStatusIcons, 863
- DisplayOffsetPStatusText, 863
- DisplayOffsetPStepIcons, 864
- DisplayOffsetPTextLines, 864
- DisplayOffsetStatusIcons, 864
- DisplayOffsetStepIcons, 864
- DisplayOffsetTextLinesCenterFlags, 864
- DisplayOffsetUpdateMask, 864
- DIST\_CMD\_CUSTOM, 864
- DIST\_CMD\_GP2D12, 864
- DIST\_CMD\_GP2D120, 864
- DIST\_CMD\_GP2YA02, 864
- DIST\_CMD\_GP2YA21, 864
- DIST\_REG\_DIST, 865
- DIST\_REG\_DIST1, 865
- DIST\_REG\_DIST\_MAX, 865
- DIST\_REG\_DIST\_MIN, 865
- DIST\_REG\_MODULE\_TYPE, 865
- DIST\_REG\_NUM\_POINTS, 865
- DIST\_REG\_VOLT, 865
- DIST\_REG\_VOLT1, 865
- DRAW\_OPT\_CLEAR, 865
- DRAW\_OPT\_CLEAR\_EXCEPT\_STATUS\_SCREEN, 865
- DRAW\_OPT\_CLEAR\_PIXELS, 865
- DRAW\_OPT\_CLEAR\_SCREEN\_MODES, 866
- DRAW\_OPT\_CLEAR\_WHOLE\_SCREEN, 866
- DRAW\_OPT\_FILL\_SHAPE, 866
- DRAW\_OPT\_FONT\_DIR\_B2TL, 866
- DRAW\_OPT\_FONT\_DIR\_B2TR, 866
- DRAW\_OPT\_FONT\_DIR\_L2RB, 866
- DRAW\_OPT\_FONT\_DIR\_L2RT, 866
- DRAW\_OPT\_FONT\_DIR\_R2LB, 866
- DRAW\_OPT\_FONT\_DIR\_R2LT, 867
- DRAW\_OPT\_FONT\_DIR\_T2BL, 867
- DRAW\_OPT\_FONT\_DIR\_T2BR, 867
- DRAW\_OPT\_FONT\_DIRECTIONS, 867
- DRAW\_OPT\_FONT\_WRAP, 867
- DRAW\_OPT\_INVERT, 867
- DRAW\_OPT\_LOGICAL\_AND, 867
- DRAW\_OPT\_LOGICAL\_COPY, 867
- DRAW\_OPT\_LOGICAL\_OPERATIONS, 868
- DRAW\_OPT\_LOGICAL\_OR, 868
- DRAW\_OPT\_LOGICAL\_XOR, 868
- DRAW\_OPT\_NORMAL, 868
- DRAW\_OPT\_POLYGON\_POLYLINE, 868
- DrawCircle, 868
- DrawEllipse, 868
- DrawFont, 868
- DrawGraphic, 868
- DrawGraphicArray, 869
- DrawLine, 869
- DrawPoint, 869
- DrawPolygon, 869
- DrawRect, 869
- DrawText, 869
- EOF, 869
- ERR\_ARG, 869
- ERR\_BAD\_POOL\_SIZE, 869
- ERR\_BAD\_PTR, 869
- ERR\_CLUMP\_COUNT, 870
- ERR\_COMM\_BUFFER\_FULL, 870
- ERR\_COMM\_BUS\_ERR, 870

ERR\_COMM\_CHAN\_INVALID, 870  
ERR\_COMM\_CHAN\_NOT\_READY, 870  
ERR\_DEFAULT\_OFFSETS, 870  
ERR\_FILE, 870  
ERR\_INSANE\_OFFSET, 870  
ERR\_INSTR, 870  
ERR\_INVALID\_FIELD, 870  
ERR\_INVALID\_PORT, 870  
ERR\_INVALID\_QUEUE, 871  
ERR\_INVALID\_SIZE, 871  
ERR\_LOADER\_ERR, 871  
ERR\_MEM, 871  
ERR\_MEMMGR\_FAIL, 871  
ERR\_NO\_ACTIVE\_CLUMP, 871  
ERR\_NO\_CODE, 871  
ERR\_NO\_PROG, 871  
ERR\_NON\_FATAL, 871  
ERR\_RC\_BAD\_PACKET, 871  
ERR\_RC\_FAILED, 871  
ERR\_RC\_ILLEGAL\_VAL, 872  
ERR\_RC\_UNKNOWN\_CMD, 872  
ERR\_SPOTCHECK\_FAIL, 872  
ERR\_VER, 872  
FALSE, 872  
FileClose, 872  
FileDelete, 872  
FileFindFirst, 872  
FileFindNext, 872  
FileOpenAppend, 872  
FileOpenRead, 872  
FileOpenReadLinear, 873  
FileOpenWrite, 873  
FileOpenWriteLinear, 873  
FileOpenWriteNonLinear, 873  
FileRead, 873  
FileRename, 873  
FileResize, 873  
FileResolveHandle, 873  
FileSeek, 873  
FileWrite, 873  
FRAME\_SELECT, 873  
FREQUENCY\_MAX, 874  
FREQUENCY\_MIN, 874  
GetStartTick, 874  
GL\_CAMERA\_DEPTH, 874  
GL\_CIRCLE, 874  
GL\_CIRCLE\_SIZE, 874  
GL\_CULL\_BACK, 874  
GL\_CULL\_FRONT, 874  
GL\_CULL\_MODE, 874  
GL\_CULL\_NONE, 874  
GL\_LINE, 875  
GL\_POINT, 875  
GL\_POLYGON, 875  
GL\_ROTATE\_X, 875  
GL\_ROTATE\_Y, 875  
GL\_ROTATE\_Z, 875  
GL\_SCALE\_X, 875  
GL\_SCALE\_Y, 876  
GL\_SCALE\_Z, 876  
GL\_TRANSLATE\_X, 876  
GL\_TRANSLATE\_Y, 876  
GL\_TRANSLATE\_Z, 876  
GL\_ZOOM\_FACTOR, 876  
HS\_BAUD\_115200, 876  
HS\_BAUD\_1200, 876  
HS\_BAUD\_14400, 877  
HS\_BAUD\_19200, 877  
HS\_BAUD\_230400, 877  
HS\_BAUD\_2400, 877  
HS\_BAUD\_28800, 877  
HS\_BAUD\_3600, 877  
HS\_BAUD\_38400, 877  
HS\_BAUD\_460800, 877  
HS\_BAUD\_4800, 877  
HS\_BAUD\_57600, 877  
HS\_BAUD\_7200, 877  
HS\_BAUD\_76800, 878  
HS\_BAUD\_921600, 878  
HS\_BAUD\_9600, 878  
HS\_CMD\_READY, 878  
HS\_CTRL\_EXIT, 878  
HS\_CTRL\_INIT, 878  
HS\_CTRL\_UART, 878  
HS\_DISABLE, 878  
HS\_ENABLE, 878  
HS\_INIT\_RECEIVER, 879  
HS\_INITIALISE, 879  
HS\_MODE\_10\_STOP, 879  
HS\_MODE\_15\_STOP, 879  
HS\_MODE\_20\_STOP, 879  
HS\_MODE\_5\_DATA, 879  
HS\_MODE\_6\_DATA, 879  
HS\_MODE\_7\_DATA, 879  
HS\_MODE\_7E1, 879  
HS\_MODE\_8\_DATA, 879  
HS\_MODE\_8N1, 879  
HS\_MODE\_E\_PARITY, 880  
HS\_MODE\_M\_PARITY, 880  
HS\_MODE\_N\_PARITY, 880  
HS\_MODE\_O\_PARITY, 880  
HS\_MODE\_S\_PARITY, 880  
HS\_SEND\_DATA, 880  
HS\_UPDATE, 880  
HT\_CH1\_A, 880  
HT\_CH1\_B, 880  
HT\_CH2\_A, 880  
HT\_CH2\_B, 881

---

HT\_CH3\_A, 881  
 HT\_CH3\_B, 881  
 HT\_CH4\_A, 881  
 HT\_CH4\_B, 881  
 HT\_CMD\_COLOR2\_50HZ, 881  
 HT\_CMD\_COLOR2\_60HZ, 881  
 HT\_CMD\_COLOR2\_ACTIVE, 881  
 HT\_CMD\_COLOR2\_BLCAL, 881  
 HT\_CMD\_COLOR2\_FAR, 881  
 HT\_CMD\_COLOR2\_LED\_HI, 881  
 HT\_CMD\_COLOR2\_LED\_LOW, 882  
 HT\_CMD\_COLOR2\_NEAR, 882  
 HT\_CMD\_COLOR2\_PASSIVE, 882  
 HT\_CMD\_COLOR2\_RAW, 882  
 HT\_CMD\_COLOR2\_WBCAL, 882  
 HTIR2\_MODE\_1200, 882  
 HTIR2\_MODE\_600, 882  
 HTIR2\_REG\_AC01, 882  
 HTIR2\_REG\_AC02, 882  
 HTIR2\_REG\_AC03, 882  
 HTIR2\_REG\_AC04, 882  
 HTIR2\_REG\_AC05, 883  
 HTIR2\_REG\_ACDIR, 883  
 HTIR2\_REG\_DC01, 883  
 HTIR2\_REG\_DC02, 883  
 HTIR2\_REG\_DC03, 883  
 HTIR2\_REG\_DC04, 883  
 HTIR2\_REG\_DC05, 883  
 HTIR2\_REG\_DCAVG, 883  
 HTIR2\_REG\_DCDIR, 883  
 HTIR2\_REG\_MODE, 883  
 I2C\_REG\_CMD, 883  
 I2C\_REG\_DEVICE\_ID, 884  
 I2C\_REG\_VENDOR\_ID, 884  
 I2C\_REG\_VERSION, 884  
 IN\_1, 884  
 IN\_2, 884  
 IN\_3, 884  
 IN\_4, 884  
 IN\_MODE\_ANGLESTEP, 884  
 IN\_MODE\_BOOLEAN, 885  
 IN\_MODE\_CELSIUS, 885  
 IN\_MODE\_FAHRENHEIT, 885  
 IN\_MODE\_MODEMASK, 885  
 IN\_MODE\_PCTFULLSCALE, 885  
 IN\_MODE\_PERIODCOUNTER, 885  
 IN\_MODE\_RAW, 885  
 IN\_MODE\_SLOPEMASK, 885  
 IN\_MODE\_TRANSITIONCNT, 885  
 IN\_TYPE\_ANGLE, 885  
 IN\_TYPE\_COLORBLUE, 885  
 IN\_TYPE\_COLOREXIT, 886  
 IN\_TYPE\_COLORFULL, 886  
 IN\_TYPE\_COLORGREEN, 886  
 IN\_TYPE\_COLORNONE, 886  
 IN\_TYPE\_COLORRED, 886  
 IN\_TYPE\_CUSTOM, 886  
 IN\_TYPE\_HISPEED, 886  
 IN\_TYPE\_LIGHT\_ACTIVE, 886  
 IN\_TYPE\_LIGHT\_INACTIVE, 886  
 IN\_TYPE\_LOWSPEED, 886  
 IN\_TYPE\_LOWSPEED\_9V, 886  
 IN\_TYPE\_NO\_SENSOR, 887  
 IN\_TYPE\_REFLECTION, 887  
 IN\_TYPE\_SOUND\_DB, 887  
 IN\_TYPE\_SOUND\_DBA, 887  
 IN\_TYPE\_SWITCH, 887  
 IN\_TYPE\_TEMPERATURE, 887  
 INPUT\_BLACKCOLOR, 887  
 INPUT\_BLANK, 887  
 INPUT\_BLUE, 887  
 INPUT\_BLUECOLOR, 887  
 INPUT\_CAL\_POINT\_0, 887  
 INPUT\_CAL\_POINT\_1, 888  
 INPUT\_CAL\_POINT\_2, 888  
 INPUT\_CUSTOM9V, 888  
 INPUT\_CUSTOMACTIVE, 888  
 INPUT\_CUSTOMINACTIVE, 888  
 INPUT\_DIGI0, 888  
 INPUT\_DIGI1, 888  
 INPUT\_GREEN, 888  
 INPUT\_GREENCOLOR, 888  
 INPUT\_INVALID\_DATA, 888  
 INPUT\_NO\_OF\_COLORS, 888  
 INPUT\_NO\_OF\_POINTS, 889  
 INPUT\_RED, 889  
 INPUT\_REDCOLOR, 889  
 INPUT\_RESETCAL, 889  
 INPUT\_RUNNINGCAL, 889  
 INPUT\_SENSORCAL, 889  
 INPUT\_SENSOROFF, 889  
 INPUT\_STARTCAL, 889  
 INPUT\_WHITECOLOR, 889  
 INPUT\_YELLOWCOLOR, 889  
 InputMode, 890  
 InputModuleID, 890  
 InputModuleName, 890  
 InputOffsetADRaw, 890  
 InputOffsetColorADRaw, 890  
 InputOffsetColorCalibration, 890  
 InputOffsetColorCalibrationState, 890  
 InputOffsetColorCalLimits, 890  
 InputOffsetColorSensorBoolean, 890  
 InputOffsetColorSensorRaw, 890  
 InputOffsetColorSensorValue, 890  
 InputOffsetCustomActiveStatus, 891  
 InputOffsetCustomPctFullScale, 891  
 InputOffsetCustomZeroOffset, 891

---

InputOffsetDigiPinsDir, 891  
InputOffsetDigiPinsIn, 891  
InputOffsetDigiPinsOut, 891  
InputOffsetInvalidData, 891  
InputOffsetSensorBoolean, 891  
InputOffsetSensorMode, 891  
InputOffsetSensorRaw, 891  
InputOffsetSensorType, 891  
InputOffsetSensorValue, 892  
INT\_MAX, 892  
INT\_MIN, 892  
INTF\_BTOFF, 892  
INTF\_BTON, 892  
INTF\_CONNECT, 892  
INTF\_CONNECTBYNAME, 892  
INTF\_CONNECTREQ, 892  
INTF\_DISCONNECT, 892  
INTF\_DISCONNECTALL, 892  
INTF\_EXTREAD, 892  
INTF\_FACTORYRESET, 893  
INTF\_OPENSTREAM, 893  
INTF\_PINREQ, 893  
INTF\_REMOVEDEVICE, 893  
INTF\_SEARCH, 893  
INTF\_SENDDATA, 893  
INTF\_SENDFILE, 893  
INTF\_SETBTNAME, 893  
INTF\_SETCMDMODE, 893  
INTF\_STOPSEARCH, 893  
INTF\_VISIBILITY, 893  
InvalidData, 894  
IOCTRL\_BOOT, 894  
IOCTRL\_POWERDOWN, 894  
IOCtrlModuleID, 894  
IOCtrlModuleName, 894  
IOCtrlOffsetPowerOn, 894  
IOMapRead, 894  
IOMapReadByID, 894  
IOMapWrite, 894  
IOMapWriteByID, 894  
KeepAlive, 894  
LCD\_LINE1, 895  
LCD\_LINE2, 895  
LCD\_LINE3, 895  
LCD\_LINE4, 895  
LCD\_LINE5, 896  
LCD\_LINE6, 896  
LCD\_LINE7, 896  
LCD\_LINE8, 896  
LDR\_APPENDNOTPOSSIBLE, 896  
LDR\_BTBUSY, 896  
LDR\_BTCONNECTFAIL, 896  
LDR\_BTTIMEOUT, 897  
LDR\_CMD\_BOOTCMD, 897  
LDR\_CMD\_BTFACTORYRESET, 897  
LDR\_CMD\_BTGETADR, 897  
LDR\_CMD\_CLOSE, 897  
LDR\_CMD\_CLOSEMODHANDLE, 897  
LDR\_CMD\_CROPDATAFILE, 897  
LDR\_CMD\_DELETE, 897  
LDR\_CMD\_DELETEUSERFLASH, 897  
LDR\_CMD\_DEVICEINFO, 897  
LDR\_CMD\_FINDFIRST, 897  
LDR\_CMD\_FINDFIRSTMODULE, 898  
LDR\_CMD\_FINDNEXT, 898  
LDR\_CMD\_FINDNEXTMODULE, 898  
LDR\_CMD\_IOMAPREAD, 898  
LDR\_CMD\_IOMAPWRITE, 898  
LDR\_CMD\_OPENAPPENDDATA, 898  
LDR\_CMD\_OPENREAD, 898  
LDR\_CMD\_OPENREADLINEAR, 898  
LDR\_CMD\_OPENWRITE, 898  
LDR\_CMD\_OPENWRITEDATA, 898  
LDR\_CMD\_OPENWRITELINEAR, 898  
LDR\_CMD\_POLLCMD, 899  
LDR\_CMD\_POLLCMDLEN, 899  
LDR\_CMD\_READ, 899  
LDR\_CMD\_RENAMEFILE, 899  
LDR\_CMD\_RESIZEDATAFILE, 899  
LDR\_CMD\_SEEKFROMCURRENT, 899  
LDR\_CMD\_SEEKFROMEND, 899  
LDR\_CMD\_SEEKFROMSTART, 899  
LDR\_CMD\_SETBRICKNAME, 899  
LDR\_CMD\_VERSIONS, 899  
LDR\_CMD\_WRITE, 899  
LDR\_ENDOFFILE, 900  
LDR\_EOFEXPECTED, 900  
LDR\_FILEEXISTS, 900  
LDR\_FILEISBUSY, 900  
LDR\_FILEISFULL, 900  
LDR\_FILENOTFOUND, 900  
LDR\_FILETX\_CLOSEERROR, 900  
LDR\_FILETX\_DSTEXISTS, 900  
LDR\_FILETX\_SRCMISSING, 900  
LDR\_FILETX\_STREAMERROR, 900  
LDR\_FILETX\_TIMEOUT, 900  
LDR\_HANDLEALREADYCLOSED, 901  
LDR\_ILLEGALFILENAME, 901  
LDR\_ILLEGALHANDLE, 901  
LDR\_INPROGRESS, 901  
LDR\_INVALIDSEEK, 901  
LDR\_MODULENOTFOUND, 901  
LDR\_NOLINEARSPACE, 901  
LDR\_NOMOREFILES, 901  
LDR\_NOMOREHANDLES, 901  
LDR\_NOSPACE, 901  
LDR\_NOTLINEARFILE, 901  
LDR\_NOWRITEBUFFERS, 902

- LDR\_OUTOFBOUNDARY, 902
- LDR\_REQPIN, 902
- LDR\_SUCCESS, 902
- LDR\_UNDEFINEDERROR, 902
- ListFiles, 902
- LoaderExecuteFunction, 902
- LoaderModuleID, 902
- LoaderModuleName, 902
- LoaderOffsetFreeUserFlash, 902
- LoaderOffsetPFunc, 903
- LONG\_MAX, 903
- LONG\_MIN, 903
- LOWSPEED\_CH\_NOT\_READY, 903
- LOWSPEED\_COMMUNICATING, 903
- LOWSPEED\_DATA\_RECEIVED, 903
- LOWSPEED\_DONE, 903
- LOWSPEED\_ERROR, 903
- LOWSPEED\_IDLE, 903
- LOWSPEED\_INIT, 903
- LOWSPEED\_LOAD\_BUFFER, 903
- LOWSPEED\_NO\_ERROR, 904
- LOWSPEED\_RECEIVING, 904
- LOWSPEED\_RX\_ERROR, 904
- LOWSPEED\_TRANSMITTING, 904
- LOWSPEED\_TX\_ERROR, 904
- LowSpeedModuleID, 904
- LowSpeedModuleName, 904
- LowSpeedOffsetChannelState, 904
- LowSpeedOffsetErrorType, 904
- LowSpeedOffsetInBufBuf, 904
- LowSpeedOffsetInBufBytesToRx, 904
- LowSpeedOffsetInBufInPtr, 905
- LowSpeedOffsetInBufOutPtr, 905
- LowSpeedOffsetMode, 905
- LowSpeedOffsetNoRestartOnRead, 905
- LowSpeedOffsetOutBufBuf, 905
- LowSpeedOffsetOutBufBytesToRx, 905
- LowSpeedOffsetOutBufInPtr, 905
- LowSpeedOffsetOutBufOutPtr, 905
- LowSpeedOffsetSpeed, 905
- LowSpeedOffsetState, 905
- LR\_COULD\_NOT\_SAVE, 905
- LR\_ENTRY\_REMOVED, 906
- LR\_STORE\_IS\_FULL, 906
- LR\_SUCCESS, 906
- LR\_UNKNOWN\_ADDR, 906
- LSREAD\_NO\_RESTART\_1, 906
- LSREAD\_NO\_RESTART\_2, 906
- LSREAD\_NO\_RESTART\_3, 906
- LSREAD\_NO\_RESTART\_4, 906
- LSREAD\_NO\_RESTART\_MASK, 906
- LSREAD\_RESTART\_ALL, 906
- LSREAD\_RESTART\_NONE, 906
- MAILBOX1, 907
- MAILBOX10, 907
- MAILBOX2, 907
- MAILBOX3, 907
- MAILBOX4, 907
- MAILBOX5, 907
- MAILBOX6, 907
- MAILBOX7, 907
- MAILBOX8, 907
- MAILBOX9, 907
- MAX\_BT\_MSG\_SIZE, 908
- MENUICON\_CENTER, 908
- MENUICON\_LEFT, 908
- MENUICON\_RIGHT, 908
- MENUICONS, 908
- MENUTEXT, 908
- MessageRead, 908
- MessageWrite, 908
- MIN\_1, 908
- MS\_1, 908
- MS\_10, 908
- MS\_100, 909
- MS\_150, 909
- MS\_2, 909
- MS\_20, 909
- MS\_200, 909
- MS\_250, 909
- MS\_3, 909
- MS\_30, 909
- MS\_300, 909
- MS\_350, 910
- MS\_4, 910
- MS\_40, 910
- MS\_400, 910
- MS\_450, 910
- MS\_5, 910
- MS\_50, 910
- MS\_500, 910
- MS\_6, 910
- MS\_60, 911
- MS\_600, 911
- MS\_7, 911
- MS\_70, 911
- MS\_700, 911
- MS\_8, 911
- MS\_80, 911
- MS\_800, 911
- MS\_9, 911
- MS\_90, 911
- MS\_900, 911
- MS\_CMD\_ADPA\_OFF, 912
- MS\_CMD\_ADPA\_ON, 912
- MS\_CMD\_DEENERGIZED, 912
- MS\_CMD\_ENERGIZED, 912
- NA, 912



NO\_ERR, 912  
NO\_OF\_BTNS, 912  
NormalizedValue, 912  
NRLINK\_CMD\_2400, 913  
NRLINK\_CMD\_4800, 913  
NRLINK\_CMD\_FLUSH, 913  
NRLINK\_CMD\_IR\_LONG, 913  
NRLINK\_CMD\_IR\_SHORT, 913  
NRLINK\_CMD\_RUN\_MACRO, 913  
NRLINK\_CMD\_SET\_PF, 913  
NRLINK\_CMD\_SET\_RCX, 913  
NRLINK\_CMD\_SET\_TRAIN, 913  
NRLINK\_CMD\_TX\_RAW, 913  
NRLINK\_REG\_BYTES, 913  
NRLINK\_REG\_DATA, 914  
NRLINK\_REG\_EEPROM, 914  
NULL, 914  
OPARR\_MAX, 914  
OPARR\_MEAN, 914  
OPARR\_MIN, 914  
OPARR\_SORT, 914  
OPARR\_STD, 914  
OPARR\_SUM, 914  
OPARR\_SUMSQR, 914  
OUT\_A, 914  
OUT\_AB, 915  
OUT\_ABC, 915  
OUT\_AC, 915  
OUT\_B, 915  
OUT\_BC, 915  
OUT\_C, 915  
OUT\_MODE\_BRAKE, 915  
OUT\_MODE\_COAST, 916  
OUT\_MODE\_MOTORON, 916  
OUT\_MODE\_REGMETHOD, 916  
OUT\_MODE\_REGULATED, 916  
OUT\_OPTION\_HOLDATLIMIT, 916  
OUT\_OPTION\_RAMPDOWNTOLIMIT, 916  
OUT\_REGMODE\_IDLE, 916  
OUT\_REGMODE\_SPEED, 916  
OUT\_REGMODE\_SYNC, 916  
OUT\_RUNSTATE\_HOLD, 917  
OUT\_RUNSTATE\_IDLE, 917  
OUT\_RUNSTATE\_RAMPDOWN, 917  
OUT\_RUNSTATE\_RAMPUP, 917  
OUT\_RUNSTATE\_RUNNING, 917  
OutputMode, 917  
OutputModuleID, 917  
OutputModuleName, 917  
OutputOffsetActualSpeed, 917  
OutputOffsetBlockTachoCount, 917  
OutputOffsetFlags, 918  
OutputOffsetMode, 918  
OutputOffsetMotorRPM, 918  
OutputOffsetOptions, 918  
OutputOffsetOverloaded, 918  
OutputOffsetPwnFreq, 918  
OutputOffsetRegDParameter, 918  
OutputOffsetRegIPParameter, 918  
OutputOffsetRegMode, 918  
OutputOffsetRegPPParameter, 918  
OutputOffsetRotationCount, 918  
OutputOffsetRunState, 919  
OutputOffsetSpeed, 919  
OutputOffsetSyncTurnParameter, 919  
OutputOffsetTachoCount, 919  
OutputOffsetTachoLimit, 919  
OutputOptions, 919  
Overload, 919  
PF\_CHANNEL\_1, 919  
PF\_CHANNEL\_2, 920  
PF\_CHANNEL\_3, 920  
PF\_CHANNEL\_4, 920  
PF\_CMD\_BRAKE, 920  
PF\_CMD\_FWD, 920  
PF\_CMD\_REV, 920  
PF\_CMD\_STOP, 920  
PF\_CST\_CLEAR1\_CLEAR2, 920  
PF\_CST\_CLEAR1\_SET2, 920  
PF\_CST\_DECREMENT\_PWM, 921  
PF\_CST\_FULL\_FWD, 921  
PF\_CST\_FULL\_REV, 921  
PF\_CST\_INCREMENT\_PWM, 921  
PF\_CST\_SET1\_CLEAR2, 921  
PF\_CST\_SET1\_SET2, 921  
PF\_CST\_TOGGLE\_DIR, 921  
PF\_FUNC\_CLEAR, 921  
PF\_FUNC\_NOCHANGE, 921  
PF\_FUNC\_SET, 921  
PF\_FUNC\_TOGGLE, 922  
PF\_MODE\_COMBO\_DIRECT, 922  
PF\_MODE\_COMBO\_PWM, 922  
PF\_MODE\_SINGLE\_OUTPUT\_CST, 922  
PF\_MODE\_SINGLE\_OUTPUT\_PWM, 922  
PF\_MODE\_SINGLE\_PIN\_CONT, 922  
PF\_MODE\_SINGLE\_PIN\_TIME, 922  
PF\_MODE\_TRAIN, 922  
PF\_OUT\_A, 922  
PF\_OUT\_B, 923  
PF\_PIN\_C1, 923  
PF\_PIN\_C2, 923  
PF\_PWM\_BRAKE, 923  
PF\_PWM\_FLOAT, 923  
PF\_PWM\_FWD1, 923  
PF\_PWM\_FWD2, 923  
PF\_PWM\_FWD3, 923  
PF\_PWM\_FWD4, 923  
PF\_PWM\_FWD5, 923

PF\_PWM\_FWD6, 924  
 PF\_PWM\_FWD7, 924  
 PF\_PWM\_REV1, 924  
 PF\_PWM\_REV2, 924  
 PF\_PWM\_REV3, 924  
 PF\_PWM\_REV4, 924  
 PF\_PWM\_REV5, 924  
 PF\_PWM\_REV6, 924  
 PF\_PWM\_REV7, 924  
 PI, 925  
 PID\_0, 925  
 PID\_1, 925  
 PID\_2, 925  
 PID\_3, 925  
 PID\_4, 925  
 PID\_5, 925  
 PID\_6, 925  
 PID\_7, 925  
 Power, 925  
 PROG\_ABORT, 926  
 PROG\_ERROR, 926  
 PROG\_IDLE, 926  
 PROG\_OK, 926  
 PROG\_RESET, 926  
 PROG\_RUNNING, 926  
 PSP\_BTNSET1\_DOWN, 926  
 PSP\_BTNSET1\_L3, 926  
 PSP\_BTNSET1\_LEFT, 926  
 PSP\_BTNSET1\_R3, 926  
 PSP\_BTNSET1\_RIGHT, 926  
 PSP\_BTNSET1\_UP, 927  
 PSP\_BTNSET2\_CIRCLE, 927  
 PSP\_BTNSET2\_CROSS, 927  
 PSP\_BTNSET2\_L1, 927  
 PSP\_BTNSET2\_L2, 927  
 PSP\_BTNSET2\_R1, 927  
 PSP\_BTNSET2\_R2, 927  
 PSP\_BTNSET2\_SQUARE, 927  
 PSP\_BTNSET2\_TRIANGLE, 927  
 PSP\_CMD\_ANALOG, 927  
 PSP\_CMD\_DIGITAL, 927  
 PSP\_REG\_BTNSET1, 928  
 PSP\_REG\_BTNSET2, 928  
 PSP\_REG\_XLEFT, 928  
 PSP\_REG\_XRIGHT, 928  
 PSP\_REG\_YLEFT, 928  
 PSP\_REG\_YRIGHT, 928  
 RADIANS\_PER\_DEGREE, 928  
 RAND\_MAX, 928  
 RandomNumber, 928  
 RawValue, 928  
 RCX\_AbsVarOp, 928  
 RCX\_AndVarOp, 929  
 RCX\_AutoOffOp, 929  
 RCX\_BatteryLevelOp, 929  
 RCX\_BatteryLevelSrc, 929  
 RCX\_BootModeOp, 929  
 RCX\_CalibrateEventOp, 929  
 RCX\_ClearAllEventsOp, 929  
 RCX\_ClearCounterOp, 929  
 RCX\_ClearMsgOp, 929  
 RCX\_ClearSensorOp, 929  
 RCX\_ClearSoundOp, 929  
 RCX\_ClearTimerOp, 930  
 RCX\_ClickCounterSrc, 930  
 RCX\_ConstantSrc, 930  
 RCX\_CounterSrc, 930  
 RCX\_DatalogOp, 930  
 RCX\_DatalogRawDirectSrc, 930  
 RCX\_DatalogRawIndirectSrc, 930  
 RCX\_DatalogSrcDirectSrc, 930  
 RCX\_DatalogSrcIndirectSrc, 930  
 RCX\_DatalogValueDirectSrc, 931  
 RCX\_DatalogValueIndirectSrc, 931  
 RCX\_DecCounterOp, 931  
 RCX\_DeleteSubOp, 931  
 RCX\_DeleteSubsOp, 931  
 RCX\_DeleteTaskOp, 931  
 RCX\_DeleteTasksOp, 931  
 RCX\_DirectEventOp, 931  
 RCX\_DisplayOp, 931  
 RCX\_DivVarOp, 931  
 RCX\_DurationSrc, 931  
 RCX\_EventStateSrc, 932  
 RCX\_FirmwareVersionSrc, 932  
 RCX\_GlobalMotorStatusSrc, 932  
 RCX\_GOutputDirOp, 932  
 RCX\_GOutputModeOp, 932  
 RCX\_GOutputPowerOp, 932  
 RCX\_HysteresisSrc, 932  
 RCX\_IncCounterOp, 932  
 RCX\_IndirectVarSrc, 932  
 RCX\_InputBooleanSrc, 932  
 RCX\_InputModeOp, 932  
 RCX\_InputModeSrc, 933  
 RCX\_InputRawSrc, 933  
 RCX\_InputTypeOp, 933  
 RCX\_InputTypeSrc, 933  
 RCX\_InputValueSrc, 933  
 RCX\_IRModeOp, 933  
 RCX\_LightOp, 933  
 RCX\_LowerThresholdSrc, 933  
 RCX\_LSblinkTimeOp, 933  
 RCX\_LSCalibrateOp, 933  
 RCX\_LSHysteresisOp, 933  
 RCX\_LSLowerThreshOp, 934  
 RCX\_LSUpperThreshOp, 934  
 RCX\_MessageOp, 934



- [RCX\\_MessageSrc, 934](#)
- [RCX\\_MulVarOp, 934](#)
- [RCX\\_MuteSoundOp, 934](#)
- [RCX\\_OnOffFloatOp, 934](#)
- [RCX\\_OrVarOp, 934](#)
- [RCX\\_OUT\\_A, 934](#)
- [RCX\\_OUT\\_AB, 935](#)
- [RCX\\_OUT\\_ABC, 935](#)
- [RCX\\_OUT\\_AC, 935](#)
- [RCX\\_OUT\\_B, 935](#)
- [RCX\\_OUT\\_BC, 935](#)
- [RCX\\_OUT\\_C, 935](#)
- [RCX\\_OUT\\_FLOAT, 935](#)
- [RCX\\_OUT\\_FULL, 935](#)
- [RCX\\_OUT\\_FWD, 935](#)
- [RCX\\_OUT\\_HALF, 936](#)
- [RCX\\_OUT\\_LOW, 936](#)
- [RCX\\_OUT\\_OFF, 936](#)
- [RCX\\_OUT\\_ON, 936](#)
- [RCX\\_OUT\\_REV, 936](#)
- [RCX\\_OUT\\_TOGGLE, 936](#)
- [RCX\\_OutputDirOp, 936](#)
- [RCX\\_OutputPowerOp, 936](#)
- [RCX\\_OutputStatusSrc, 936](#)
- [RCX\\_PBTurnOffOp, 937](#)
- [RCX\\_PingOp, 937](#)
- [RCX\\_PlaySoundOp, 937](#)
- [RCX\\_PlayToneOp, 937](#)
- [RCX\\_PlayToneVarOp, 937](#)
- [RCX\\_PollMemoryOp, 937](#)
- [RCX\\_PollOp, 937](#)
- [RCX\\_ProgramSlotSrc, 937](#)
- [RCX\\_RandomSrc, 937](#)
- [RCX\\_RemoteKeysReleased, 937](#)
- [RCX\\_RemoteOp, 937](#)
- [RCX\\_RemoteOutABackward, 938](#)
- [RCX\\_RemoteOutAForward, 938](#)
- [RCX\\_RemoteOutBBackward, 938](#)
- [RCX\\_RemoteOutBForward, 938](#)
- [RCX\\_RemoteOutCBackward, 938](#)
- [RCX\\_RemoteOutCForward, 938](#)
- [RCX\\_RemotePBMessage1, 938](#)
- [RCX\\_RemotePBMessage2, 938](#)
- [RCX\\_RemotePBMessage3, 938](#)
- [RCX\\_RemotePlayASound, 938](#)
- [RCX\\_RemoteSelProgram1, 938](#)
- [RCX\\_RemoteSelProgram2, 939](#)
- [RCX\\_RemoteSelProgram3, 939](#)
- [RCX\\_RemoteSelProgram4, 939](#)
- [RCX\\_RemoteSelProgram5, 939](#)
- [RCX\\_RemoteStopOutOff, 939](#)
- [RCX\\_ScoutCounterLimitSrc, 939](#)
- [RCX\\_ScoutEventFBSrc, 939](#)
- [RCX\\_ScoutLightParamsSrc, 939](#)
- [RCX\\_ScoutOp, 939](#)
- [RCX\\_ScoutRulesOp, 939](#)
- [RCX\\_ScoutRulesSrc, 939](#)
- [RCX\\_ScoutTimerLimitSrc, 940](#)
- [RCX\\_SelectProgramOp, 940](#)
- [RCX\\_SendUARTDataOp, 940](#)
- [RCX\\_SetCounterOp, 940](#)
- [RCX\\_SetDatalogOp, 940](#)
- [RCX\\_SetEventOp, 940](#)
- [RCX\\_SetFeedbackOp, 940](#)
- [RCX\\_SetPriorityOp, 940](#)
- [RCX\\_SetSourceValueOp, 940](#)
- [RCX\\_SetTimerLimitOp, 940](#)
- [RCX\\_SetVarOp, 940](#)
- [RCX\\_SetWatchOp, 941](#)
- [RCX\\_SgnVarOp, 941](#)
- [RCX\\_SoundOp, 941](#)
- [RCX\\_StartTaskOp, 941](#)
- [RCX\\_StopAllTasksOp, 941](#)
- [RCX\\_StopTaskOp, 941](#)
- [RCX\\_SubVarOp, 941](#)
- [RCX\\_SumVarOp, 941](#)
- [RCX\\_TaskEventsSrc, 941](#)
- [RCX\\_TenMSTimerSrc, 941](#)
- [RCX\\_TimerSrc, 941](#)
- [RCX\\_UARTSetupSrc, 942](#)
- [RCX\\_UnlockFirmOp, 942](#)
- [RCX\\_UnlockOp, 942](#)
- [RCX\\_UnmuteSoundOp, 942](#)
- [RCX\\_UploadDatalogOp, 942](#)
- [RCX\\_UpperThresholdSrc, 942](#)
- [RCX\\_VariableSrc, 942](#)
- [RCX\\_ViewSourceValOp, 942](#)
- [RCX\\_VLLOp, 942](#)
- [RCX\\_WatchSrc, 942](#)
- [ReadButton, 943](#)
- [ReadSemData, 943](#)
- [RegDValue, 943](#)
- [RegIValue, 943](#)
- [RegMode, 943](#)
- [RegPValue, 943](#)
- [RESET\\_ALL, 943](#)
- [RESET\\_BLOCK\\_COUNT, 944](#)
- [RESET\\_BLOCKANDTACHO, 944](#)
- [RESET\\_COUNT, 944](#)
- [RESET\\_NONE, 944](#)
- [RESET\\_ROTATION\\_COUNT, 944](#)
- [RICArg, 944](#)
- [RICImgPoint, 944](#)
- [RICImgRect, 945](#)
- [RICMapArg, 945](#)
- [RICMapElement, 945](#)
- [RICMapFunction, 945](#)
- [RICOpCircle, 945](#)

RICOpCopyBits, [946](#)  
RICOpDescription, [946](#)  
RICOpEllipse, [946](#)  
RICOpLine, [947](#)  
RICOpNumBox, [947](#)  
RICOpPixel, [947](#)  
RICOpPolygon, [947](#)  
RICOpRect, [947](#)  
RICOpSprite, [948](#)  
RICOpVarMap, [948](#)  
RICPolygonPoints, [948](#)  
RICSpriteData, [949](#)  
ROTATE\_QUEUE, [949](#)  
RotationCount, [949](#)  
RunState, [949](#)  
SAMPLERATE\_DEFAULT, [949](#)  
SAMPLERATE\_MAX, [949](#)  
SAMPLERATE\_MIN, [949](#)  
ScaledValue, [949](#)  
SCHAR\_MAX, [950](#)  
SCHAR\_MIN, [950](#)  
SCOUT\_FXR\_ALARM, [950](#)  
SCOUT\_FXR\_BUG, [950](#)  
SCOUT\_FXR\_NONE, [950](#)  
SCOUT\_FXR\_RANDOM, [950](#)  
SCOUT\_FXR\_SCIENCE, [950](#)  
SCOUT\_LIGHT\_OFF, [950](#)  
SCOUT\_LIGHT\_ON, [950](#)  
SCOUT\_LR\_AVOID, [950](#)  
SCOUT\_LR\_IGNORE, [951](#)  
SCOUT\_LR\_OFF\_WHEN, [951](#)  
SCOUT\_LR\_SEEK\_DARK, [951](#)  
SCOUT\_LR\_SEEK\_LIGHT, [951](#)  
SCOUT\_LR\_WAIT\_FOR, [951](#)  
SCOUT\_MODE\_POWER, [951](#)  
SCOUT\_MODE\_STANDALONE, [951](#)  
SCOUT\_MR\_CIRCLE\_LEFT, [951](#)  
SCOUT\_MR\_CIRCLE\_RIGHT, [951](#)  
SCOUT\_MR\_FORWARD, [951](#)  
SCOUT\_MR\_LOOP\_A, [952](#)  
SCOUT\_MR\_LOOP\_AB, [952](#)  
SCOUT\_MR\_LOOP\_B, [952](#)  
SCOUT\_MR\_NO\_MOTION, [952](#)  
SCOUT\_MR\_ZIGZAG, [952](#)  
SCOUT\_SNDSET\_ALARM, [952](#)  
SCOUT\_SNDSET\_BASIC, [952](#)  
SCOUT\_SNDSET\_BUG, [952](#)  
SCOUT\_SNDSET\_NONE, [952](#)  
SCOUT\_SNDSET\_RANDOM, [952](#)  
SCOUT\_SNDSET\_SCIENCE, [952](#)  
SCOUT\_SOUND\_1\_BLINK, [953](#)  
SCOUT\_SOUND\_2\_BLINK, [953](#)  
SCOUT\_SOUND\_COUNTER1, [953](#)  
SCOUT\_SOUND\_COUNTER2, [953](#)  
SCOUT\_SOUND\_ENTER\_BRIGHT, [953](#)  
SCOUT\_SOUND\_ENTER\_DARK, [953](#)  
SCOUT\_SOUND\_ENTER\_NORMAL, [953](#)  
SCOUT\_SOUND\_ENTERSA, [953](#)  
SCOUT\_SOUND\_KEYERROR, [953](#)  
SCOUT\_SOUND\_MAIL\_RECEIVED, [953](#)  
SCOUT\_SOUND\_NONE, [953](#)  
SCOUT\_SOUND\_REMOTE, [954](#)  
SCOUT\_SOUND\_SPECIAL1, [954](#)  
SCOUT\_SOUND\_SPECIAL2, [954](#)  
SCOUT\_SOUND\_SPECIAL3, [954](#)  
SCOUT\_SOUND\_TIMER1, [954](#)  
SCOUT\_SOUND\_TIMER2, [954](#)  
SCOUT\_SOUND\_TIMER3, [954](#)  
SCOUT\_SOUND\_TOUCH1\_PRES, [954](#)  
SCOUT\_SOUND\_TOUCH1\_REL, [954](#)  
SCOUT\_SOUND\_TOUCH2\_PRES, [954](#)  
SCOUT\_SOUND\_TOUCH2\_REL, [954](#)  
SCOUT\_TGS\_LONG, [955](#)  
SCOUT\_TGS\_MEDIUM, [955](#)  
SCOUT\_TGS\_SHORT, [955](#)  
SCOUT\_TR\_AVOID, [955](#)  
SCOUT\_TR\_IGNORE, [955](#)  
SCOUT\_TR\_OFF\_WHEN, [955](#)  
SCOUT\_TR\_REVERSE, [955](#)  
SCOUT\_TR\_WAIT\_FOR, [955](#)  
SCREEN\_BACKGROUND, [955](#)  
SCREEN\_LARGE, [955](#)  
SCREEN\_MODE\_CLEAR, [956](#)  
SCREEN\_MODE\_RESTORE, [956](#)  
SCREEN\_SMALL, [956](#)  
SCREENS, [956](#)  
SEC\_1, [956](#)  
SEC\_10, [956](#)  
SEC\_15, [956](#)  
SEC\_2, [957](#)  
SEC\_20, [957](#)  
SEC\_3, [957](#)  
SEC\_30, [957](#)  
SEC\_4, [957](#)  
SEC\_5, [957](#)  
SEC\_6, [957](#)  
SEC\_7, [958](#)  
SEC\_8, [958](#)  
SEC\_9, [958](#)  
SetScreenMode, [958](#)  
SetSleepTimeoutVal, [958](#)  
SHRT\_MAX, [958](#)  
SHRT\_MIN, [958](#)  
SIZE\_OF\_BDADDR, [958](#)  
SIZE\_OF\_BRICK\_NAME, [958](#)  
SIZE\_OF\_BT\_CONNECT\_TABLE, [958](#)  
SIZE\_OF\_BT\_DEVICE\_TABLE, [959](#)  
SIZE\_OF\_BT\_NAME, [959](#)

SIZE\_OF\_BT\_PINCODE, 959  
SIZE\_OF\_BTBUF, 959  
SIZE\_OF\_CLASS\_OF\_DEVICE, 959  
SIZE\_OF\_HSBUF, 959  
SIZE\_OF\_USBBUF, 959  
SIZE\_OF\_USBDATA, 959  
SOUND\_CLICK, 959  
SOUND\_DOUBLE\_BEEP, 959  
SOUND\_DOWN, 959  
SOUND\_FAST\_UP, 960  
SOUND\_FLAGS\_IDLE, 960  
SOUND\_FLAGS\_RUNNING, 960  
SOUND\_FLAGS\_UPDATE, 960  
SOUND\_LOW\_BEEP, 960  
SOUND\_MODE\_LOOP, 960  
SOUND\_MODE\_ONCE, 960  
SOUND\_MODE\_TONE, 960  
SOUND\_STATE\_FILE, 961  
SOUND\_STATE\_IDLE, 961  
SOUND\_STATE\_STOP, 961  
SOUND\_STATE\_TONE, 961  
SOUND\_UP, 961  
SoundGetState, 961  
SoundModuleID, 961  
SoundModuleName, 961  
SoundOffsetDuration, 962  
SoundOffsetFlags, 962  
SoundOffsetFreq, 962  
SoundOffsetMode, 962  
SoundOffsetSampleRate, 962  
SoundOffsetSoundFilename, 962  
SoundOffsetState, 962  
SoundOffsetVolume, 962  
SoundPlayFile, 962  
SoundPlayTone, 962  
SoundSetState, 963  
SPECIALS, 963  
STAT\_COMM\_PENDING, 963  
STAT\_MSG\_EMPTY\_MAILBOX, 963  
STATUSICON\_BATTERY, 963  
STATUSICON\_BLUETOOTH, 963  
STATUSICON\_USB, 963  
STATUSICON\_VM, 963  
STATUSICONS, 963  
STATUSTEXT, 963  
STEPICON\_1, 963  
STEPICON\_2, 964  
STEPICON\_3, 964  
STEPICON\_4, 964  
STEPICON\_5, 964  
STEPICONS, 964  
STEPLINE, 964  
STOP\_REQ, 964  
TachoCount, 964  
TachoLimit, 964  
TEMP\_FQ\_1, 964  
TEMP\_FQ\_2, 964  
TEMP\_FQ\_4, 965  
TEMP\_FQ\_6, 965  
TEMP\_I2C\_ADDRESS, 965  
TEMP\_OS\_ONESHOT, 965  
TEMP\_POL\_HIGH, 965  
TEMP\_POL\_LOW, 965  
TEMP\_REG\_CONFIG, 965  
TEMP\_REG\_TEMP, 965  
TEMP\_REG\_THIGH, 965  
TEMP\_REG\_TLOW, 965  
TEMP\_RES\_10BIT, 965  
TEMP\_RES\_11BIT, 966  
TEMP\_RES\_12BIT, 966  
TEMP\_RES\_9BIT, 966  
TEMP\_SD\_CONTINUOUS, 966  
TEMP\_SD\_SHUTDOWN, 966  
TEMP\_TM\_COMPARATOR, 966  
TEMP\_TM\_INTERRUPT, 966  
TEXTLINE\_1, 966  
TEXTLINE\_2, 966  
TEXTLINE\_3, 966  
TEXTLINE\_4, 967  
TEXTLINE\_5, 967  
TEXTLINE\_6, 967  
TEXTLINE\_7, 967  
TEXTLINE\_8, 967  
TEXTLINES, 967  
TIMES\_UP, 967  
TONE\_A3, 967  
TONE\_A4, 967  
TONE\_A5, 967  
TONE\_A6, 967  
TONE\_A7, 968  
TONE\_AS3, 968  
TONE\_AS4, 968  
TONE\_AS5, 968  
TONE\_AS6, 968  
TONE\_AS7, 968  
TONE\_B3, 968  
TONE\_B4, 968  
TONE\_B5, 968  
TONE\_B6, 968  
TONE\_B7, 968  
TONE\_C4, 969  
TONE\_C5, 969  
TONE\_C6, 969  
TONE\_C7, 969  
TONE\_CS4, 969  
TONE\_CS5, 969  
TONE\_CS6, 969  
TONE\_CS7, 969

TONE\_D4, 969  
 TONE\_D5, 970  
 TONE\_D6, 970  
 TONE\_D7, 970  
 TONE\_DS4, 970  
 TONE\_DS5, 970  
 TONE\_DS6, 970  
 TONE\_DS7, 970  
 TONE\_E4, 970  
 TONE\_E5, 970  
 TONE\_E6, 970  
 TONE\_E7, 971  
 TONE\_F4, 971  
 TONE\_F5, 971  
 TONE\_F6, 971  
 TONE\_F7, 971  
 TONE\_FS4, 971  
 TONE\_FS5, 971  
 TONE\_FS6, 971  
 TONE\_FS7, 971  
 TONE\_G4, 971  
 TONE\_G5, 971  
 TONE\_G6, 972  
 TONE\_G7, 972  
 TONE\_GS4, 972  
 TONE\_GS5, 972  
 TONE\_GS6, 972  
 TONE\_GS7, 972  
 TOPLINE, 972  
 TRAIN\_CHANNEL\_1, 972  
 TRAIN\_CHANNEL\_2, 972  
 TRAIN\_CHANNEL\_3, 972  
 TRAIN\_CHANNEL\_ALL, 973  
 TRAIN\_FUNC\_DECR\_SPEED, 973  
 TRAIN\_FUNC\_INCR\_SPEED, 973  
 TRAIN\_FUNC\_STOP, 973  
 TRAIN\_FUNC\_TOGGLE\_LIGHT, 973  
 TRUE, 973  
 TurnRatio, 973  
 Type, 973  
 UCHAR\_MAX, 974  
 UF\_PENDING\_UPDATES, 974  
 UF\_UPDATE\_MODE, 974  
 UF\_UPDATE\_PID\_VALUES, 974  
 UF\_UPDATE\_RESET\_BLOCK\_COUNT, 974  
 UF\_UPDATE\_RESET\_COUNT, 974  
 UF\_UPDATE\_RESET\_ROTATION\_COUNT, 974  
 UF\_UPDATE\_SPEED, 974  
 UF\_UPDATE\_TACHO\_LIMIT, 974  
 UI\_BT\_CONNECT\_REQUEST, 974  
 UI\_BT\_ERROR\_ATTENTION, 974  
 UI\_BT\_PIN\_REQUEST, 975  
 UI\_BT\_STATE\_CONNECTED, 975  
 UI\_BT\_STATE\_OFF, 975  
 UI\_BT\_STATE\_VISIBLE, 975  
 UI\_BUTTON\_ENTER, 975  
 UI\_BUTTON\_EXIT, 975  
 UI\_BUTTON\_LEFT, 975  
 UI\_BUTTON\_NONE, 975  
 UI\_BUTTON\_RIGHT, 975  
 UI\_FLAGS\_BUSY, 975  
 UI\_FLAGS\_DISABLE\_EXIT, 976  
 UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER, 976  
 UI\_FLAGS\_ENABLE\_STATUS\_UPDATE, 976  
 UI\_FLAGS\_EXECUTE\_LMS\_FILE, 976  
 UI\_FLAGS\_REDRAW\_STATUS, 976  
 UI\_FLAGS\_RESET\_SLEEP\_TIMER, 976  
 UI\_FLAGS\_UPDATE, 976  
 UI\_STATE\_BT\_ERROR, 976  
 UI\_STATE\_CONNECT\_REQUEST, 976  
 UI\_STATE\_DRAW\_MENU, 976  
 UI\_STATE\_ENTER\_PRESSED, 976  
 UI\_STATE\_EXECUTE\_FILE, 977  
 UI\_STATE\_EXECUTING\_FILE, 977  
 UI\_STATE\_EXIT\_PRESSED, 977  
 UI\_STATE\_INIT\_DISPLAY, 977  
 UI\_STATE\_INIT\_INTRO, 977  
 UI\_STATE\_INIT\_LOW\_BATTERY, 977  
 UI\_STATE\_INIT\_MENU, 977  
 UI\_STATE\_INIT\_WAIT, 977  
 UI\_STATE\_LEFT\_PRESSED, 977  
 UI\_STATE\_LOW\_BATTERY, 977  
 UI\_STATE\_NEXT\_MENU, 977  
 UI\_STATE\_RIGHT\_PRESSED, 978  
 UI\_STATE\_TEST\_BUTTONS, 978  
 UI\_VM\_IDLE, 978  
 UI\_VM\_RESET1, 978  
 UI\_VM\_RESET2, 978  
 UI\_VM\_RUN\_FREE, 978  
 UI\_VM\_RUN\_PAUSE, 978  
 UI\_VM\_RUN\_SINGLE, 978  
 UIModuleID, 978  
 UIModuleName, 978  
 UINT\_MAX, 978  
 UIOffsetAbortFlag, 979  
 UIOffsetBatteryState, 979  
 UIOffsetBatteryVoltage, 979  
 UIOffsetBluetoothState, 979  
 UIOffsetButton, 979  
 UIOffsetError, 979  
 UIOffsetFlags, 979  
 UIOffsetForceOff, 979  
 UIOffsetLMSfilename, 979  
 UIOffsetOBPPointer, 979

- UIOffsetPMenu, [979](#)
- UIOffsetRechargeable, [980](#)
- UIOffsetRunState, [980](#)
- UIOffsetSleepTimeout, [980](#)
- UIOffsetSleepTimer, [980](#)
- UIOffsetState, [980](#)
- UIOffsetUsbState, [980](#)
- UIOffsetVolume, [980](#)
- ULONG\_MAX, [980](#)
- UpdateCalibCacheInfo, [980](#)
- UpdateFlags, [980](#)
- US\_CMD\_CONTINUOUS, [980](#)
- US\_CMD\_EVENTCAPTURE, [981](#)
- US\_CMD\_OFF, [981](#)
- US\_CMD\_SINGLESLOT, [981](#)
- US\_CMD\_WARMRESET, [981](#)
- US\_REG\_ACTUAL\_ZERO, [981](#)
- US\_REG\_CM\_INTERVAL, [981](#)
- US\_REG\_FACTORY\_ACTUAL\_ZERO, [981](#)
- US\_REG\_FACTORY\_SCALE\_DIVISOR, [981](#)
- US\_REG\_FACTORY\_SCALE\_FACTOR, [981](#)
- US\_REG\_MEASUREMENT\_UNITS, [981](#)
- US\_REG\_SCALE\_DIVISOR, [981](#)
- US\_REG\_SCALE\_FACTOR, [982](#)
- USB\_CMD\_READY, [982](#)
- USB\_PROTOCOL\_OVERHEAD, [982](#)
- USHRT\_MAX, [982](#)
- WriteSemData, [982](#)
- NBCInputPortConstants
  - IN\_1, [543](#)
  - IN\_2, [543](#)
  - IN\_3, [543](#)
  - IN\_4, [543](#)
- NBCSensorModeConstants
  - IN\_MODE\_ANGLESTEP, [547](#)
  - IN\_MODE\_BOOLEAN, [547](#)
  - IN\_MODE\_CELSIUS, [547](#)
  - IN\_MODE\_FAHRENHEIT, [547](#)
  - IN\_MODE\_MODEMASK, [547](#)
  - IN\_MODE\_PCTFULLSCALE, [547](#)
  - IN\_MODE\_PERIODCOUNTER, [548](#)
  - IN\_MODE\_RAW, [548](#)
  - IN\_MODE\_SLOPEMASK, [548](#)
  - IN\_MODE\_TRANSITIONCNT, [548](#)
- NBCSensorTypeConstants
  - IN\_TYPE\_ANGLE, [544](#)
  - IN\_TYPE\_COLORBLUE, [544](#)
  - IN\_TYPE\_COLOREXIT, [544](#)
  - IN\_TYPE\_COLORFULL, [544](#)
  - IN\_TYPE\_COLORGREEN, [545](#)
  - IN\_TYPE\_COLORNONE, [545](#)
  - IN\_TYPE\_COLORRED, [545](#)
  - IN\_TYPE\_CUSTOM, [545](#)
  - IN\_TYPE\_HISPEED, [545](#)
  - IN\_TYPE\_LIGHT\_ACTIVE, [545](#)
  - IN\_TYPE\_LIGHT\_INACTIVE, [545](#)
  - IN\_TYPE\_LOWSPEED, [545](#)
  - IN\_TYPE\_LOWSPEED\_9V, [545](#)
  - IN\_TYPE\_NO\_SENSOR, [545](#)
  - IN\_TYPE\_REFLECTION, [545](#)
  - IN\_TYPE\_SOUND\_DB, [546](#)
  - IN\_TYPE\_SOUND\_DBA, [546](#)
  - IN\_TYPE\_SWITCH, [546](#)
  - IN\_TYPE\_TEMPERATURE, [546](#)
- NewFilename
  - FileRenameType, [772](#)
- NewSize
  - FileResizeType, [774](#)
- NewVal
  - WriteSemDataType, [814](#)
- NO\_ERR
  - CommandModuleConstants, [53](#)
  - NBCCCommon.h, [912](#)
- NO\_OF\_BTNS
  - ButtonNameConstants, [526](#)
  - NBCCCommon.h, [912](#)
- NoRestartOnRead
  - CommLSWriteExType, [735](#)
- NormalizedArray
  - ColorSensorReadType, [720](#)
- NormalizedValue
  - InputFieldConstants, [549](#)
  - NBCCCommon.h, [912](#)
- NRLink2400
  - MindSensorsAPI, [153](#)
  - NXCDefs.h, [1209](#)
- NRLink2400Ex
  - MindSensorsAPI, [154](#)
  - NXCDefs.h, [1209](#)
- NRLink4800
  - MindSensorsAPI, [154](#)
  - NXCDefs.h, [1209](#)
- NRLink4800Ex
  - MindSensorsAPI, [154](#)
  - NXCDefs.h, [1210](#)
- NRLINK\_CMD\_2400
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_4800
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_FLUSH
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_IR\_LONG
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)

- NRLINK\_CMD\_IR\_SHORT
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_RUN\_MACRO
  - MSNRLink, [709](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_SET\_PF
  - MSNRLink, [710](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_SET\_RCX
  - MSNRLink, [710](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_SET\_TRAIN
  - MSNRLink, [710](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_CMD\_TX\_RAW
  - MSNRLink, [710](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_REG\_BYTES
  - MSNRLink, [710](#)
  - NBCCCommon.h, [913](#)
- NRLINK\_REG\_DATA
  - MSNRLink, [710](#)
  - NBCCCommon.h, [914](#)
- NRLINK\_REG\_EEPROM
  - MSNRLink, [710](#)
  - NBCCCommon.h, [914](#)
- NRLinkFlush
  - MindSensorsAPI, [155](#)
  - NXCDefs.h, [1210](#)
- NRLinkFlushEx
  - MindSensorsAPI, [155](#)
  - NXCDefs.h, [1210](#)
- NRLinkIRLong
  - MindSensorsAPI, [155](#)
  - NXCDefs.h, [1211](#)
- NRLinkIRLongEx
  - MindSensorsAPI, [156](#)
  - NXCDefs.h, [1211](#)
- NRLinkIRShort
  - MindSensorsAPI, [156](#)
  - NXCDefs.h, [1211](#)
- NRLinkIRShortEx
  - MindSensorsAPI, [156](#)
  - NXCDefs.h, [1212](#)
- NRLinkSetPF
  - MindSensorsAPI, [157](#)
  - NXCDefs.h, [1212](#)
- NRLinkSetPFEx
  - MindSensorsAPI, [157](#)
  - NXCDefs.h, [1212](#)
- NRLinkSetRCX
  - MindSensorsAPI, [157](#)
  - NXCDefs.h, [1213](#)
- NRLinkSetRCXEx
  - MindSensorsAPI, [158](#)
  - NXCDefs.h, [1213](#)
- NRLinkSetTrain
  - MindSensorsAPI, [158](#)
  - NXCDefs.h, [1213](#)
- NRLinkSetTrainEx
  - MindSensorsAPI, [158](#)
  - NXCDefs.h, [1214](#)
- NRLinkStatus
  - MindSensorsAPI, [159](#)
  - NXCDefs.h, [1214](#)
- NRLinkStatusEx
  - MindSensorsAPI, [159](#)
  - NXCDefs.h, [1214](#)
- NRLinkTxRaw
  - MindSensorsAPI, [159](#)
  - NXCDefs.h, [1215](#)
- NRLinkTxRawEx
  - MindSensorsAPI, [160](#)
  - NXCDefs.h, [1215](#)
- NULL
  - LoaderModuleConstants, [500](#)
  - NBCCCommon.h, [914](#)
- NumOut
  - DisplayModuleFunctions, [258](#)
  - NXCDefs.h, [1215](#)
- NumToStr
  - cstringAPI, [451](#)
  - NXCDefs.h, [1216](#)
- NXCAPIDocs.h, [983](#)
- NXCDefs.h, [984](#)
  - \_SENSOR\_CFG, [1045](#)
  - abort, [1064](#)
  - AbortFlag, [1064](#)
  - abs, [1065](#)
  - Acos, [1045](#)
  - acos, [1065](#)
  - AcosD, [1046](#)
  - acosd, [1065](#)
  - Acquire, [1066](#)
  - addr, [1066](#)
  - address, [1066](#)
  - ArrayBuild, [1067](#)
  - ArrayInit, [1067](#)
  - ArrayLen, [1067](#)
  - ArrayMax, [1068](#)
  - ArrayMean, [1068](#)
  - ArrayMin, [1069](#)
  - ArrayOp, [1069](#)
  - ArraySort, [1069](#)
  - ArrayStd, [1070](#)
  - ArraySubset, [1070](#)
  - ArraySum, [1070](#)



- ArraySumSqr, [1071](#)
- Asin, [1046](#)
- asin, [1071](#)
- AsinD, [1046](#)
- asind, [1072](#)
- Atan, [1047](#)
- atan, [1072](#)
- Atan2, [1047](#)
- atan2, [1072](#)
- Atan2D, [1047](#)
- atan2d, [1073](#)
- AtanD, [1048](#)
- atand, [1073](#)
- atof, [1074](#)
- atoi, [1074](#)
- atol, [1075](#)
- BatteryLevel, [1075](#)
- BatteryState, [1076](#)
- bcd2dec, [1076](#)
- BluetoothState, [1076](#)
- BluetoothStatus, [1076](#)
- BluetoothWrite, [1077](#)
- BrickDataBluecoreVersion, [1077](#)
- BrickDataBtHardwareStatus, [1077](#)
- BrickDataBtStateStatus, [1078](#)
- BrickDataName, [1078](#)
- BrickDataTimeoutValue, [1078](#)
- BTConnectionClass, [1078](#)
- BTConnectionHandleNum, [1078](#)
- BTConnectionLinkQuality, [1079](#)
- BTConnectionName, [1079](#)
- BTConnectionPinCode, [1079](#)
- BTConnectionStreamStatus, [1080](#)
- BTDeviceClass, [1080](#)
- BTDeviceCount, [1080](#)
- BTDeviceName, [1081](#)
- BTDeviceNameCount, [1081](#)
- BTDeviceStatus, [1081](#)
- BTInputBufferInPtr, [1082](#)
- BTInputBufferOutPtr, [1082](#)
- BTOutputBufferInPtr, [1082](#)
- BTOutputBufferOutPtr, [1082](#)
- ButtonCount, [1083](#)
- ButtonLongPressCount, [1083](#)
- ButtonLongReleaseCount, [1083](#)
- ButtonPressCount, [1083](#)
- ButtonPressed, [1084](#)
- ButtonReleaseCount, [1084](#)
- ButtonShortReleaseCount, [1084](#)
- ButtonState, [1085](#)
- ByteArrayToStr, [1085](#)
- ByteArrayToStrEx, [1085](#)
- Ceil, [1048](#)
- ceil, [1086](#)
- CircleOut, [1086](#)
- ClearLine, [1087](#)
- ClearScreen, [1087](#)
- ClearSensor, [1087](#)
- CloseFile, [1087](#)
- Coast, [1088](#)
- CoastEx, [1088](#)
- ColorADRaw, [1088](#)
- ColorBoolean, [1089](#)
- ColorCalibration, [1089](#)
- ColorCalibrationState, [1089](#)
- ColorCalLimits, [1090](#)
- ColorSensorRaw, [1090](#)
- ColorSensorValue, [1091](#)
- CommandFlags, [1091](#)
- Copy, [1091](#)
- Cos, [1048](#)
- cos, [1092](#)
- CosD, [1049](#)
- cosd, [1092](#)
- Cosh, [1049](#)
- cosh, [1092](#)
- CoshD, [1049](#)
- coshd, [1093](#)
- CreateFile, [1093](#)
- CreateFileLinear, [1093](#)
- CreateFileNonLinear, [1094](#)
- CurrentTick, [1094](#)
- CustomSensorActiveStatus, [1095](#)
- CustomSensorPercentFullScale, [1095](#)
- CustomSensorZeroOffset, [1095](#)
- DeleteFile, [1096](#)
- DisplayContrast, [1096](#)
- DisplayDisplay, [1096](#)
- DisplayEraseMask, [1096](#)
- DisplayFlags, [1097](#)
- DisplayFont, [1097](#)
- DisplayTextLinesCenterFlags, [1097](#)
- DisplayUpdateMask, [1097](#)
- DISTNxDistance, [1098](#)
- DISTNxDistanceEx, [1098](#)
- DISTNxGP2D12, [1098](#)
- DISTNxGP2D120, [1099](#)
- DISTNxGP2D120Ex, [1099](#)
- DISTNxGP2D12Ex, [1099](#)
- DISTNxGP2YA02, [1100](#)
- DISTNxGP2YA02Ex, [1100](#)
- DISTNxGP2YA21, [1100](#)
- DISTNxGP2YA21Ex, [1101](#)
- DISTNxMaxDistance, [1101](#)
- DISTNxMaxDistanceEx, [1101](#)
- DISTNxMinDistance, [1102](#)
- DISTNxMinDistanceEx, [1102](#)
- DISTNxModuleType, [1102](#)

- DISTNxModuleTypeEx, 1103
- DISTNxNumPoints, 1103
- DISTNxNumPointsEx, 1103
- DISTNxVoltage, 1104
- DISTNxVoltageEx, 1104
- div, 1104
- EllipseOut, 1105
- ExitTo, 1105
- Exp, 1050
- exp, 1105
- fclose, 1106
- feof, 1106
- fflush, 1106
- fgetc, 1107
- fgets, 1107
- FindFirstFile, 1107
- FindNextFile, 1108
- FirstTick, 1108
- Flatten, 1108
- FlattenVar, 1109
- Float, 1109
- Floor, 1050
- floor, 1109
- Follows, 1110
- FontNumOut, 1110
- FontTextOut, 1111
- fopen, 1111
- ForceOff, 1112
- FormatNum, 1112
- fprintf, 1112
- fputc, 1113
- fputs, 1113
- Frac, 1050
- frac, 1113
- FreeMemory, 1114
- fseek, 1114
- ftell, 1114
- GetBrickDataAddress, 1115
- GetBTConnectionAddress, 1115
- GetBTDeviceAddress, 1115
- GetBTInputBuffer, 1115
- GetBTOutputBuffer, 1116
- getc, 1051
- getchar, 1116
- GetDisplayNormal, 1116
- GetDisplayPopup, 1117
- GetHSInputBuffer, 1117
- GetHSOutputBuffer, 1117
- GetInput, 1118
- GetLSInputBuffer, 1118
- GetLSOutputBuffer, 1118
- GetOutput, 1119
- GetUSBInputBuffer, 1119
- GetUSBOutputBuffer, 1119
- GetUSBPollBuffer, 1120
- glAddToAngleX, 1120
- glAddToAngleY, 1120
- glAddToAngleZ, 1120
- glAddVertex, 1121
- glBegin, 1121
- glBeginObject, 1121
- glBeginRender, 1121
- glBox, 1121
- glCallObject, 1122
- glCos32768, 1122
- glCube, 1122
- glEnd, 1123
- glEndObject, 1123
- glFinishRender, 1123
- glInit, 1123
- glObjectAction, 1123
- glPyramid, 1123
- glSet, 1124
- glSetAngleX, 1124
- glSetAngleY, 1124
- glSetAngleZ, 1124
- glSin32768, 1125
- GraphicArrayOut, 1125
- GraphicArrayOutEx, 1125
- GraphicOut, 1126
- GraphicOutEx, 1126
- HSFlags, 1127
- HSInputBufferInPtr, 1127
- HSInputBufferOutPtr, 1127
- HSMODE, 1128
- HSOutputBufferInPtr, 1128
- HSOutputBufferOutPtr, 1128
- HSSpeed, 1128
- HSSState, 1129
- HTIRTrain, 1129
- HTPFComboDirect, 1129
- HTPFComboPWM, 1130
- HTPFRawOutput, 1130
- HTPFRepeat, 1131
- HTPFSingleOutputCST, 1131
- HTPFSingleOutputPWM, 1132
- HTPFSinglePin, 1132
- HTPFTrain, 1133
- HTRCXXAddToDatalog, 1133
- HTRCXXBatteryLevel, 1133
- HTRCXXClearAllEvents, 1133
- HTRCXXClearCounter, 1134
- HTRCXXClearMsg, 1134
- HTRCXXClearSensor, 1134
- HTRCXXClearSound, 1134
- HTRCXXClearTimer, 1134
- HTRCXXCreateDatalog, 1135
- HTRCXXDecCounter, 1135



- HTRCXDeleteSub, 1135
- HTRCXDeleteSubs, 1135
- HTRCXDeleteTask, 1135
- HTRCXDeleteTasks, 1136
- HTRCXDisableOutput, 1136
- HTRCXEnableOutput, 1136
- HTRCXEvent, 1136
- HTRCXFloat, 1136
- HTRCXFwd, 1137
- HTRCXIncCounter, 1137
- HTRCXInvertOutput, 1137
- HTRCXMuteSound, 1137
- HTRCXObvertOutput, 1138
- HTRCXOff, 1138
- HTRCXOn, 1138
- HTRCXOnFor, 1138
- HTRCXOnFwd, 1139
- HTRCXOnRev, 1139
- HTRCXPBTurnOff, 1139
- HTRCXPing, 1139
- HTRCXPlaySound, 1139
- HTRCXPlayTone, 1140
- HTRCXPlayToneVar, 1140
- HTRCXPoll, 1140
- HTRCXPollMemory, 1140
- HTRCXRemote, 1141
- HTRCXRev, 1141
- HTRCXSelectDisplay, 1141
- HTRCXSelectProgram, 1141
- HTRCXSendSerial, 1142
- HTRCXSetDirection, 1142
- HTRCXSetEvent, 1142
- HTRCXSetGlobalDirection, 1142
- HTRCXSetGlobalOutput, 1143
- HTRCXSetIRLinkPort, 1143
- HTRCXSetMaxPower, 1143
- HTRCXSetMessage, 1143
- HTRCXSetOutput, 1144
- HTRCXSetPower, 1144
- HTRCXSetPriority, 1144
- HTRCXSetSensorMode, 1144
- HTRCXSetSensorType, 1145
- HTRCXSetSleepTime, 1145
- HTRCXSetTxPower, 1145
- HTRCXSetWatch, 1145
- HTRCXStartTask, 1146
- HTRCXStopAllTasks, 1146
- HTRCXStopTask, 1146
- HTRCXToggle, 1146
- HTRCXUnmuteSound, 1147
- HTScoutCalibrateSensor, 1147
- HTScoutMuteSound, 1147
- HTScoutSelectSounds, 1147
- HTScoutSendVLL, 1147
- HTScoutSetEventFeedback, 1147
- HTScoutSetLight, 1148
- HTScoutSetScoutMode, 1148
- HTScoutSetSensorClickTime, 1148
- HTScoutSetSensorHysteresis, 1148
- HTScoutSetSensorLowerLimit, 1149
- HTScoutSetSensorUpperLimit, 1149
- HTScoutUnmuteSound, 1149
- I2CBytes, 1149
- I2CBytesReady, 1150
- I2CCheckStatus, 1151
- I2CDeviceId, 1151
- I2CDeviceIdEx, 1151
- I2CDeviceInfo, 1152
- I2CDeviceInfoEx, 1152
- I2CRead, 1153
- I2CSendCommand, 1153
- I2CSendCommandEx, 1154
- I2CStatus, 1154
- I2CVendorId, 1155
- I2CVendorIdEx, 1155
- I2CVersion, 1155
- I2CVersionEx, 1156
- I2CWrite, 1156
- isalnum, 1157
- isalpha, 1157
- iscntrl, 1157
- isdigit, 1158
- isgraph, 1158
- islower, 1158
- isNAN, 1159
- isprint, 1159
- ispunct, 1159
- isspace, 1160
- isupper, 1160
- isxdigit, 1160
- labs, 1160
- ldiv, 1161
- LeftStr, 1161
- LineOut, 1161
- Log, 1051
- log, 1162
- Log10, 1051
- log10, 1162
- LongAbort, 1163
- LowSpeedBytesReady, 1163
- LowSpeedCheckStatus, 1164
- LowSpeedRead, 1164
- LowSpeedStatus, 1165
- LowSpeedWrite, 1165
- LSChannelState, 1166
- LSErrorType, 1166
- LSInputBufferBytesToRx, 1166
- LSInputBufferInPtr, 1167

- LSInputBufferOutPtr, [1167](#)
- LSMode, [1167](#)
- LSNoRestartOnRead, [1168](#)
- LSOutputBufferBytesToRx, [1168](#)
- LSOutputBufferInPtr, [1168](#)
- LSOutputBufferOutPtr, [1169](#)
- LSSpeed, [1169](#)
- LSState, [1169](#)
- memcmp, [1169](#)
- memcpy, [1170](#)
- memmove, [1170](#)
- MidStr, [1170](#)
- MotorActualSpeed, [1171](#)
- MotorBlockTachoCount, [1171](#)
- MotorMode, [1171](#)
- MotorOverload, [1171](#)
- MotorPower, [1172](#)
- MotorPwnFreq, [1172](#)
- MotorRegDValue, [1172](#)
- MotorRegIValue, [1172](#)
- MotorRegPValue, [1173](#)
- MotorRegulation, [1173](#)
- MotorRotationCount, [1173](#)
- MotorRunState, [1173](#)
- MotorTachoCount, [1174](#)
- MotorTachoLimit, [1174](#)
- MotorTurnRatio, [1174](#)
- MSADPAOff, [1174](#)
- MSADPAOffEx, [1175](#)
- MSADPAOn, [1175](#)
- MSADPAOnEx, [1175](#)
- MSDeenergize, [1176](#)
- MSDeenergizeEx, [1176](#)
- MEnergize, [1176](#)
- MEnergizeEx, [1177](#)
- MSIRTrain, [1177](#)
- MSIRTrainEx, [1177](#)
- MSPFComboDirect, [1178](#)
- MSPFComboDirectEx, [1178](#)
- MSPFComboPWM, [1179](#)
- MSPFComboPWMEEx, [1179](#)
- MSPFRawOutput, [1180](#)
- MSPFRawOutputEx, [1180](#)
- MSPFRepeat, [1181](#)
- MSPFRepeatEx, [1181](#)
- MSPFSingleOutputCST, [1181](#)
- MSPFSingleOutputCSTEx, [1182](#)
- MSPFSingleOutputPWM, [1182](#)
- MSPFSingleOutputPWMEEx, [1183](#)
- MSPFSinglePin, [1183](#)
- MSPFSinglePinEx, [1184](#)
- MSPFTrain, [1184](#)
- MSPFTrainEx, [1185](#)
- MSRCXAbsVar, [1185](#)
- MSRCXAddToDatalog, [1186](#)
- MSRCXAndVar, [1186](#)
- MSRCXBatteryLevel, [1186](#)
- MSRCXBoot, [1186](#)
- MSRCXCalibrateEvent, [1187](#)
- MSRCXClearAllEvents, [1187](#)
- MSRCXClearCounter, [1187](#)
- MSRCXClearMsg, [1187](#)
- MSRCXClearSensor, [1187](#)
- MSRCXClearSound, [1188](#)
- MSRCXClearTimer, [1188](#)
- MSRCXCreateDatalog, [1188](#)
- MSRCXDecCounter, [1188](#)
- MSRCXDeleteSub, [1188](#)
- MSRCXDeleteSubs, [1189](#)
- MSRCXDeleteTask, [1189](#)
- MSRCXDeleteTasks, [1189](#)
- MSRCXDisableOutput, [1189](#)
- MSRCXDivVar, [1189](#)
- MSRCXEnableOutput, [1190](#)
- MSRCXEvent, [1190](#)
- MSRCXFloat, [1190](#)
- MSRCXFwd, [1190](#)
- MSRCXIncCounter, [1191](#)
- MSRCXInvertOutput, [1191](#)
- MSRCXMulVar, [1191](#)
- MSRCXMuteSound, [1191](#)
- MSRCXObvertOutput, [1192](#)
- MSRCXOff, [1192](#)
- MSRCXOn, [1192](#)
- MSRCXOnFor, [1192](#)
- MSRCXOnFwd, [1193](#)
- MSRCXOnRev, [1193](#)
- MSRCXOrVar, [1193](#)
- MSRCXPBTurnOff, [1193](#)
- MSRCXPing, [1193](#)
- MSRCXPlaySound, [1194](#)
- MSRCXPlayTone, [1194](#)
- MSRCXPlayToneVar, [1194](#)
- MSRCXPoll, [1194](#)
- MSRCXPollMemory, [1195](#)
- MSRCXRemote, [1195](#)
- MSRCXReset, [1195](#)
- MSRCXRev, [1195](#)
- MSRCXSelectDisplay, [1196](#)
- MSRCXSelectProgram, [1196](#)
- MSRCXSendSerial, [1196](#)
- MSRCXSet, [1196](#)
- MSRCXSetDirection, [1197](#)
- MSRCXSetEvent, [1197](#)
- MSRCXSetGlobalDirection, [1197](#)
- MSRCXSetGlobalOutput, [1198](#)
- MSRCXSetMaxPower, [1198](#)
- MSRCXSetMessage, [1198](#)

- MSRCXSetNRLinkPort, 1198
- MSRCXSetNRLinkPortEx, 1199
- MSRCXSetOutput, 1199
- MSRCXSetPower, 1199
- MSRCXSetPriority, 1200
- MSRCXSetSensorMode, 1200
- MSRCXSetSensorType, 1200
- MSRCXSetSleepTime, 1200
- MSRCXSetTxPower, 1201
- MSRCXSetUserDisplay, 1201
- MSRCXSetVar, 1201
- MSRCXSetWatch, 1201
- MSRCXSgnVar, 1202
- MSRCXStartTask, 1202
- MSRCXStopAllTasks, 1202
- MSRCXStopTask, 1202
- MSRCXSubVar, 1203
- MSRCXSumVar, 1203
- MSRCXToggle, 1203
- MSRCXUnlock, 1203
- MSRCXUnmuteSound, 1204
- MSReadValue, 1204
- MSReadValueEx, 1204
- MSScoutCalibrateSensor, 1205
- MSScoutMuteSound, 1205
- MSScoutSelectSounds, 1205
- MSScoutSendVLL, 1205
- MSScoutSetCounterLimit, 1205
- MSScoutSetEventFeedback, 1206
- MSScoutSetLight, 1206
- MSScoutSetScoutMode, 1206
- MSScoutSetScoutRules, 1206
- MSScoutSetSensorClickTime, 1207
- MSScoutSetSensorHysteresis, 1207
- MSScoutSetSensorLowerLimit, 1207
- MSScoutSetSensorUpperLimit, 1208
- MSScoutSetTimerLimit, 1208
- MSScoutUnmuteSound, 1208
- MulDiv32, 1052
- muldiv32, 1208
- NRLink2400, 1209
- NRLink2400Ex, 1209
- NRLink4800, 1209
- NRLink4800Ex, 1210
- NRLinkFlush, 1210
- NRLinkFlushEx, 1210
- NRLinkIRLong, 1211
- NRLinkIRLongEx, 1211
- NRLinkIRShort, 1211
- NRLinkIRShortEx, 1212
- NRLinkSetPF, 1212
- NRLinkSetPFEx, 1212
- NRLinkSetRCX, 1213
- NRLinkSetRCXEx, 1213
- NRLinkSetTrain, 1213
- NRLinkSetTrainEx, 1214
- NRLinkStatus, 1214
- NRLinkStatusEx, 1214
- NRLinkTxRaw, 1215
- NRLinkTxRawEx, 1215
- NumOut, 1215
- NumToStr, 1216
- Off, 1216
- OffEx, 1217
- OnBrickProgramPointer, 1217
- OnFwd, 1217
- OnFwdEx, 1217
- OnFwdReg, 1218
- OnFwdRegEx, 1218
- OnFwdRegExPID, 1218
- OnFwdRegPID, 1219
- OnFwdSync, 1219
- OnFwdSyncEx, 1220
- OnFwdSyncExPID, 1220
- OnFwdSyncPID, 1220
- OnRev, 1221
- OnRevEx, 1221
- OnRevReg, 1221
- OnRevRegEx, 1222
- OnRevRegExPID, 1222
- OnRevRegPID, 1223
- OnRevSync, 1223
- OnRevSyncEx, 1223
- OnRevSyncExPID, 1224
- OnRevSyncPID, 1224
- OpenFileAppend, 1225
- OpenFileRead, 1225
- OpenFileReadLinear, 1226
- PlayFile, 1226
- PlayFileEx, 1226
- PlaySound, 1227
- PlayTone, 1227
- PlayToneEx, 1227
- PlayTones, 1228
- PointOut, 1228
- PolyOut, 1228
- Pow, 1052
- pow, 1229
- PowerDown, 1229
- Precedes, 1229
- printf, 1230
- PSPNxAnalog, 1230
- PSPNxAnalogEx, 1230
- PSPNxDigital, 1231
- PSPNxDigitalEx, 1231
- putc, 1052
- rand, 1231
- Random, 1232

Read, [1232](#)  
ReadButtonEx, [1232](#)  
ReadBytes, [1233](#)  
ReadI2CRegister, [1233](#)  
ReadLn, [1234](#)  
ReadLnString, [1234](#)  
ReadNRLinkBytes, [1234](#)  
ReadNRLinkBytesEx, [1235](#)  
ReadSensorColorEx, [1235](#)  
ReadSensorColorRaw, [1236](#)  
ReadSensorHTAccel, [1236](#)  
ReadSensorHTColor, [1236](#)  
ReadSensorHTColor2Active, [1237](#)  
ReadSensorHTIRReceiver, [1237](#)  
ReadSensorHTIRReceiverEx, [1238](#)  
ReadSensorHTIRSeeker, [1238](#)  
ReadSensorHTIRSeeker2AC, [1239](#)  
ReadSensorHTIRSeeker2DC, [1239](#)  
ReadSensorHTNormalizedColor, [1240](#)  
ReadSensorHTNormalizedColor2Active, [1240](#)  
ReadSensorHTRawColor, [1241](#)  
ReadSensorHTRawColor2, [1241](#)  
ReadSensorHTTouchMultiplexer, [1241](#)  
ReadSensorMSAccel, [1242](#)  
ReadSensorMSAccelEx, [1242](#)  
ReadSensorMSPlayStation, [1243](#)  
ReadSensorMSPlayStationEx, [1243](#)  
ReadSensorMSRTClock, [1244](#)  
ReadSensorMSTilt, [1244](#)  
ReadSensorMSTiltEx, [1245](#)  
ReadSensorUSEx, [1245](#)  
RebootInFirmwareMode, [1245](#)  
ReceiveMessage, [1246](#)  
ReceiveRemoteBool, [1246](#)  
ReceiveRemoteMessageEx, [1246](#)  
ReceiveRemoteNumber, [1247](#)  
ReceiveRemoteString, [1247](#)  
RechargeableBattery, [1247](#)  
RectOut, [1248](#)  
reladdr, [1248](#)  
Release, [1249](#)  
RemoteKeepAlive, [1249](#)  
RemoteMessageRead, [1249](#)  
RemoteMessageWrite, [1249](#)  
RemotePlaySoundFile, [1250](#)  
RemotePlayTone, [1250](#)  
RemoteResetMotorPosition, [1251](#)  
RemoteResetScaledValue, [1251](#)  
RemoteSetInputMode, [1251](#)  
RemoteSetOutputState, [1252](#)  
RemoteStartProgram, [1252](#)  
RemoteStopProgram, [1252](#)  
RemoteStopSound, [1253](#)  
remove, [1253](#)  
rename, [1253](#)  
RenameFile, [1254](#)  
ResetAllTachoCounts, [1254](#)  
ResetBlockTachoCount, [1254](#)  
ResetRotationCount, [1254](#)  
ResetScreen, [1255](#)  
ResetSensor, [1255](#)  
ResetSleepTimer, [1255](#)  
ResetTachoCount, [1255](#)  
ResizeFile, [1256](#)  
ResolveHandle, [1256](#)  
rewind, [1256](#)  
RICSetValue, [1053](#)  
RightStr, [1257](#)  
RotateMotor, [1257](#)  
RotateMotorEx, [1257](#)  
RotateMotorExPID, [1258](#)  
RotateMotorPID, [1258](#)  
RS485Control, [1259](#)  
RS485DataAvailable, [1259](#)  
RS485Exit, [1260](#)  
RS485Init, [1260](#)  
RS485Read, [1260](#)  
RS485SendingData, [1261](#)  
RS485Status, [1261](#)  
RS485Uart, [1261](#)  
RS485Write, [1262](#)  
RunNRLinkMacro, [1262](#)  
RunNRLinkMacroEx, [1262](#)  
S1, [1053](#)  
s16, [1055](#)  
S2, [1055](#)  
S3, [1055](#)  
s32, [1055](#)  
S4, [1055](#)  
s8, [1055](#)  
SEEK\_CUR, [1055](#)  
SEEK\_END, [1055](#)  
SEEK\_SET, [1055](#)  
SendMessage, [1263](#)  
SendRemoteBool, [1263](#)  
SendRemoteNumber, [1263](#)  
SendRemoteString, [1264](#)  
SendResponseBool, [1264](#)  
SendResponseNumber, [1264](#)  
SendResponseString, [1265](#)  
SendRS485Bool, [1265](#)  
SendRS485Number, [1265](#)  
SendRS485String, [1266](#)  
Sensor, [1266](#)  
SENSOR\_1, [1056](#)  
SENSOR\_2, [1056](#)  
SENSOR\_3, [1056](#)  
SENSOR\_4, [1056](#)

- SENSOR\_CELSIUS, [1056](#)
- SENSOR\_COLORBLUE, [1056](#)
- SENSOR\_COLORFULL, [1056](#)
- SENSOR\_COLORGREEN, [1056](#)
- SENSOR\_COLORNONE, [1056](#)
- SENSOR\_COLORRED, [1056](#)
- SENSOR\_EDGE, [1057](#)
- SENSOR\_FAHRENHEIT, [1057](#)
- SENSOR\_LIGHT, [1057](#)
- SENSOR\_LOWSPEED, [1057](#)
- SENSOR\_LOWSPEED\_9V, [1057](#)
- SENSOR\_MODE\_BOOL, [1057](#)
- SENSOR\_MODE\_CELSIUS, [1057](#)
- SENSOR\_MODE\_EDGE, [1057](#)
- SENSOR\_MODE\_FAHRENHEIT, [1057](#)
- SENSOR\_MODE\_PERCENT, [1058](#)
- SENSOR\_MODE\_PULSE, [1058](#)
- SENSOR\_MODE\_RAW, [1058](#)
- SENSOR\_MODE\_ROTATION, [1058](#)
- SENSOR\_NXTLIGHT, [1058](#)
- SENSOR\_PULSE, [1058](#)
- SENSOR\_ROTATION, [1058](#)
- SENSOR\_SOUND, [1058](#)
- SENSOR\_TOUCH, [1058](#)
- SENSOR\_TYPE\_COLORBLUE, [1059](#)
- SENSOR\_TYPE\_COLORFULL, [1059](#)
- SENSOR\_TYPE\_COLORGREEN, [1059](#)
- SENSOR\_TYPE\_COLORNONE, [1059](#)
- SENSOR\_TYPE\_COLORRED, [1059](#)
- SENSOR\_TYPE\_CUSTOM, [1059](#)
- SENSOR\_TYPE\_HIGHSPEED, [1059](#)
- SENSOR\_TYPE\_LIGHT, [1059](#)
- SENSOR\_TYPE\_LIGHT\_ACTIVE, [1059](#)
- SENSOR\_TYPE\_LIGHT\_INACTIVE, [1059](#)
- SENSOR\_TYPE\_LOWSPEED, [1060](#)
- SENSOR\_TYPE\_LOWSPEED\_9V, [1060](#)
- SENSOR\_TYPE\_NONE, [1060](#)
- SENSOR\_TYPE\_ROTATION, [1060](#)
- SENSOR\_TYPE\_SOUND\_DB, [1060](#)
- SENSOR\_TYPE\_SOUND\_DBA, [1060](#)
- SENSOR\_TYPE\_TEMPERATURE, [1060](#)
- SENSOR\_TYPE\_TOUCH, [1060](#)
- SensorBoolean, [1266](#)
- SensorDigiPinsDirection, [1267](#)
- SensorDigiPinsOutputLevel, [1267](#)
- SensorDigiPinsStatus, [1267](#)
- SensorHTColorNum, [1268](#)
- SensorHTCompass, [1268](#)
- SensorHTEOPD, [1268](#)
- SensorHTGyro, [1269](#)
- SensorHTIRSeeker2ACDir, [1269](#)
- SensorHTIRSeeker2Addr, [1269](#)
- SensorHTIRSeeker2DCDir, [1270](#)
- SensorHTIRSeekerDir, [1270](#)
- SensorInvalid, [1270](#)
- SensorMode, [1271](#)
- SensorMSCompass, [1271](#)
- SensorMSCompassEx, [1271](#)
- SensorMSDROD, [1272](#)
- SensorMSPressure, [1272](#)
- SensorMSPressureRaw, [1272](#)
- SensorNormalized, [1273](#)
- SensorRaw, [1273](#)
- SensorScaled, [1273](#)
- SensorType, [1274](#)
- SensorUS, [1274](#)
- SensorValue, [1274](#)
- SensorValueBool, [1275](#)
- SensorValueRaw, [1275](#)
- SetAbortFlag, [1275](#)
- SetBatteryState, [1276](#)
- SetBluetoothState, [1276](#)
- SetBTInputBuffer, [1276](#)
- SetBTInputBufferInPtr, [1276](#)
- SetBTInputBufferOutPtr, [1277](#)
- SetBTOutputBuffer, [1277](#)
- SetBTOutputBufferInPtr, [1277](#)
- SetBTOutputBufferOutPtr, [1277](#)
- SetButtonLongPressCount, [1278](#)
- SetButtonLongReleaseCount, [1278](#)
- SetButtonPressCount, [1278](#)
- SetButtonReleaseCount, [1278](#)
- SetButtonShortReleaseCount, [1279](#)
- SetButtonState, [1279](#)
- SetCommandFlags, [1279](#)
- SetCustomSensorActiveStatus, [1279](#)
- SetCustomSensorPercentFullScale, [1280](#)
- SetCustomSensorZeroOffset, [1280](#)
- SetDisplayContrast, [1280](#)
- SetDisplayDisplay, [1280](#)
- SetDisplayEraseMask, [1281](#)
- SetDisplayFlags, [1281](#)
- SetDisplayFont, [1281](#)
- SetDisplayNormal, [1281](#)
- SetDisplayPopup, [1282](#)
- SetDisplayTextLinesCenterFlags, [1282](#)
- SetDisplayUpdateMask, [1282](#)
- SetHSFlags, [1283](#)
- SetHSInputBuffer, [1283](#)
- SetHSInputBufferInPtr, [1283](#)
- SetHSInputBufferOutPtr, [1283](#)
- SetHSMode, [1284](#)
- SetHSOutputBuffer, [1284](#)
- SetHSOutputBufferInPtr, [1284](#)
- SetHSOutputBufferOutPtr, [1284](#)
- SetHSSpeed, [1285](#)
- SetHSState, [1285](#)
- SetHTColor2Mode, [1285](#)

SetHTIRSeeker2Mode, 1285  
SetInput, 1286  
SetLongAbort, 1286  
SetMotorPwnFreq, 1286  
SetOnBrickProgramPointer, 1287  
SetOutput, 1287  
SetSensor, 1287  
SetSensorBoolean, 1288  
SetSensorColorBlue, 1288  
SetSensorColorFull, 1288  
SetSensorColorGreen, 1288  
SetSensorColorNone, 1289  
SetSensorColorRed, 1289  
SetSensorDigiPinsDirection, 1289  
SetSensorDigiPinsOutputLevel, 1290  
SetSensorDigiPinsStatus, 1290  
SetSensorHTEOPD, 1290  
SetSensorHTGyro, 1290  
SetSensorLight, 1291  
SetSensorLowSpeed, 1291  
SetSensorMode, 1291  
SetSensorMSDROD, 1292  
SetSensorMSPressure, 1292  
SetSensorSound, 1292  
SetSensorTouch, 1292  
SetSensorType, 1293  
SetSleepTime, 1293  
SetSleepTimeout, 1293  
SetSleepTimer, 1294  
SetSoundDuration, 1294  
SetSoundFlags, 1294  
SetSoundFrequency, 1294  
SetSoundMode, 1295  
SetSoundModuleState, 1295  
SetSoundSampleRate, 1295  
SetSoundVolume, 1296  
SetUIButton, 1296  
SetUIState, 1296  
SetUSBInputBuffer, 1296  
SetUSBInputBufferInPtr, 1297  
SetUSBInputBufferOutPtr, 1297  
SetUSBOutputBuffer, 1297  
SetUSBOutputBufferInPtr, 1297  
SetUSBOutputBufferOutPtr, 1298  
SetUSBPollBuffer, 1298  
SetUSBPollBufferInPtr, 1298  
SetUSBPollBufferOutPtr, 1298  
SetUSBState, 1299  
SetVMRunState, 1299  
SetVolume, 1299  
sign, 1299  
Sin, 1060  
sin, 1300  
SinD, 1061  
sind, 1300  
Sinh, 1061  
sinh, 1300  
SinhD, 1061  
sinhd, 1301  
SleepNow, 1301  
SleepTime, 1301  
SleepTimeout, 1302  
SleepTimer, 1302  
SoundDuration, 1302  
SoundFlags, 1302  
SoundFrequency, 1303  
SoundMode, 1303  
SoundSampleRate, 1303  
SoundState, 1304  
SoundVolume, 1304  
sprintf, 1304  
Sqrt, 1062  
sqrt, 1305  
StartTask, 1305  
Stop, 1305  
StopAllTasks, 1305  
StopSound, 1306  
StopTask, 1306  
StrCat, 1307  
strcat, 1306  
strcmp, 1307  
strcpy, 1307  
StrIndex, 1308  
StrLen, 1308  
strlen, 1308  
strncat, 1309  
strncmp, 1309  
strncpy, 1310  
StrReplace, 1310  
StrToByteArray, 1310  
strtod, 1311  
strtol, 1311  
StrToNum, 1312  
strtol, 1312  
SubStr, 1313  
SysCall, 1313  
SysColorSensorRead, 1314  
SysCommBTCheckStatus, 1314  
SysCommBTConnection, 1314  
SysCommBTOnOff, 1314  
SysCommBTWrite, 1315  
SysCommExecuteFunction, 1315  
SysCommHSCheckStatus, 1315  
SysCommHSControl, 1316  
SysCommHSRead, 1316  
SysCommHSWrite, 1316  
SysCommLSCheckStatus, 1317  
SysCommLSRead, 1317



- [SysCommLSWrite, 1317](#)
- [SysCommLSWriteEx, 1317](#)
- [SysComputeCalibValue, 1318](#)
- [SysDatalogGetTimes, 1318](#)
- [SysDatalogWrite, 1318](#)
- [SysDisplayExecuteFunction, 1319](#)
- [SysDrawCircle, 1319](#)
- [SysDrawEllipse, 1319](#)
- [SysDrawFont, 1320](#)
- [SysDrawGraphic, 1320](#)
- [SysDrawGraphicArray, 1320](#)
- [SysDrawLine, 1321](#)
- [SysDrawPoint, 1321](#)
- [SysDrawPolygon, 1321](#)
- [SysDrawRect, 1321](#)
- [SysDrawText, 1322](#)
- [SysFileClose, 1322](#)
- [SysFileDelete, 1322](#)
- [SysFileFindFirst, 1322](#)
- [SysFileFindNext, 1323](#)
- [SysFileOpenAppend, 1323](#)
- [SysFileOpenRead, 1323](#)
- [SysFileOpenReadLinear, 1324](#)
- [SysFileOpenWrite, 1324](#)
- [SysFileOpenWriteLinear, 1324](#)
- [SysFileOpenWriteNonLinear, 1324](#)
- [SysFileRead, 1325](#)
- [SysFileRename, 1325](#)
- [SysFileResize, 1325](#)
- [SysFileResolveHandle, 1326](#)
- [SysFileSeek, 1326](#)
- [SysFileWrite, 1326](#)
- [SysGetStartTick, 1326](#)
- [SysIOMapRead, 1327](#)
- [SysIOMapReadByID, 1327](#)
- [SysIOMapWrite, 1327](#)
- [SysIOMapWriteByID, 1327](#)
- [SysKeepAlive, 1328](#)
- [SysListFiles, 1328](#)
- [SysLoaderExecuteFunction, 1328](#)
- [SysMessageRead, 1329](#)
- [SysMessageWrite, 1329](#)
- [SysRandomNumber, 1329](#)
- [SysReadButton, 1329](#)
- [SysReadSemData, 1330](#)
- [SysSetScreenMode, 1330](#)
- [SysSetSleepTimeout, 1330](#)
- [SysSoundGetState, 1331](#)
- [SysSoundPlayFile, 1331](#)
- [SysSoundPlayTone, 1331](#)
- [SysSoundSetState, 1331](#)
- [SysUpdateCalibCacheInfo, 1332](#)
- [SysWriteSemData, 1332](#)
- [Tan, 1062](#)
- [tan, 1332](#)
- [TanD, 1062](#)
- [tand, 1333](#)
- [Tanh, 1063](#)
- [tanh, 1333](#)
- [TanhD, 1063](#)
- [tanhD, 1333](#)
- [TextOut, 1334](#)
- [tolower, 1334](#)
- [toupper, 1335](#)
- [Trunc, 1063](#)
- [trunc, 1335](#)
- [u16, 1064](#)
- [u32, 1064](#)
- [u8, 1064](#)
- [UIButton, 1335](#)
- [UIState, 1336](#)
- [UnflattenVar, 1336](#)
- [USBInputBufferInPtr, 1336](#)
- [USBInputBufferOutPtr, 1336](#)
- [USBOutputBufferInPtr, 1337](#)
- [USBOutputBufferOutPtr, 1337](#)
- [USBPollBufferInPtr, 1337](#)
- [USBPollBufferOutPtr, 1337](#)
- [USBState, 1338](#)
- [UsbState, 1338](#)
- [VMRunState, 1338](#)
- [Volume, 1338](#)
- [Wait, 1338](#)
- [Write, 1339](#)
- [WriteBytes, 1339](#)
- [WriteBytesEx, 1340](#)
- [WriteI2CRegister, 1340](#)
- [WriteLn, 1340](#)
- [WriteLnString, 1341](#)
- [WriteNRLinkBytes, 1341](#)
- [WriteNRLinkBytesEx, 1342](#)
- [WriteString, 1342](#)
- [Yield, 1342](#)
- [NXT firmware module IDs, 179](#)
- [NXT firmware module names, 177](#)
- [NXT Firmware Modules, 43](#)
- [NXTLimits](#)
  - [CHAR\\_BIT, 711](#)
  - [CHAR\\_MAX, 711](#)
  - [CHAR\\_MIN, 711](#)
  - [INT\\_MAX, 711](#)
  - [INT\\_MIN, 711](#)
  - [LONG\\_MAX, 711](#)
  - [LONG\\_MIN, 712](#)
  - [RAND\\_MAX, 712](#)
  - [SCHAR\\_MAX, 712](#)
  - [SCHAR\\_MIN, 712](#)
  - [SHRT\\_MAX, 712](#)

- SHRT\_MIN, [712](#)
- UCHAR\_MAX, [712](#)
- UINT\_MAX, [712](#)
- ULONG\_MAX, [712](#)
- USHRT\_MAX, [712](#)
- Off
  - NXCDefs.h, [1216](#)
  - OutputModuleFunctions, [234](#)
- OffEx
  - NXCDefs.h, [1217](#)
  - OutputModuleFunctions, [234](#)
- Offset
  - IOMapReadByIDType, [780](#)
  - IOMapReadType, [782](#)
  - IOMapWriteByIDType, [784](#)
  - IOMapWriteType, [786](#)
- OldFilename
  - FileRenameType, [772](#)
- On
  - DisplayExecuteFunctionType, [743](#)
- OnBrickProgramPointer
  - NXCDefs.h, [1217](#)
  - UiModuleFunctions, [375](#)
- OnFwd
  - NXCDefs.h, [1217](#)
  - OutputModuleFunctions, [234](#)
- OnFwdEx
  - NXCDefs.h, [1217](#)
  - OutputModuleFunctions, [235](#)
- OnFwdReg
  - NXCDefs.h, [1218](#)
  - OutputModuleFunctions, [235](#)
- OnFwdRegEx
  - NXCDefs.h, [1218](#)
  - OutputModuleFunctions, [235](#)
- OnFwdRegExPID
  - NXCDefs.h, [1218](#)
  - OutputModuleFunctions, [236](#)
- OnFwdRegPID
  - NXCDefs.h, [1219](#)
  - OutputModuleFunctions, [236](#)
- OnFwdSync
  - NXCDefs.h, [1219](#)
  - OutputModuleFunctions, [236](#)
- OnFwdSyncEx
  - NXCDefs.h, [1220](#)
  - OutputModuleFunctions, [237](#)
- OnFwdSyncExPID
  - NXCDefs.h, [1220](#)
  - OutputModuleFunctions, [237](#)
- OnFwdSyncPID
  - NXCDefs.h, [1220](#)
  - OutputModuleFunctions, [238](#)
- OnRev
  - NXCDefs.h, [1221](#)
  - OutputModuleFunctions, [238](#)
- OnRevEx
  - NXCDefs.h, [1221](#)
  - OutputModuleFunctions, [238](#)
- OnRevReg
  - NXCDefs.h, [1221](#)
  - OutputModuleFunctions, [239](#)
- OnRevRegEx
  - NXCDefs.h, [1222](#)
  - OutputModuleFunctions, [239](#)
- OnRevRegExPID
  - NXCDefs.h, [1222](#)
  - OutputModuleFunctions, [239](#)
- OnRevRegPID
  - NXCDefs.h, [1223](#)
  - OutputModuleFunctions, [240](#)
- OnRevSync
  - NXCDefs.h, [1223](#)
  - OutputModuleFunctions, [240](#)
- OnRevSyncEx
  - NXCDefs.h, [1223](#)
  - OutputModuleFunctions, [241](#)
- OnRevSyncExPID
  - NXCDefs.h, [1224](#)
  - OutputModuleFunctions, [241](#)
- OnRevSyncPID
  - NXCDefs.h, [1224](#)
  - OutputModuleFunctions, [241](#)
- OPARR\_MAX
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_MEAN
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_MIN
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_SORT
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_STD
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_SUM
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OPARR\_SUMSQR
  - ArrayOpConstants, [465](#)
  - NBCCCommon.h, [914](#)
- OpenFileAppend
  - LoaderModuleFunctions, [388](#)
  - NXCDefs.h, [1225](#)



- OpenFileRead
  - LoaderModuleFunctions, 389
  - NXCDefs.h, 1225
- OpenFileReadLinear
  - LoaderModuleFunctions, 389
  - NXCDefs.h, 1226
- Options
  - DrawCircleType, 746
  - DrawEllipseType, 748
  - DrawFontType, 750
  - DrawGraphicArrayType, 752
  - DrawGraphicType, 754
  - DrawLineType, 756
  - DrawPointType, 758
  - DrawPolygonType, 759
  - DrawRectType, 760
  - DrawTextType, 762
- Origin
  - FileSeekType, 777
- OUT\_A
  - NBCCCommon.h, 914
  - OutputPortConstants, 557
- OUT\_AB
  - NBCCCommon.h, 915
  - OutputPortConstants, 557
- OUT\_ABC
  - NBCCCommon.h, 915
  - OutputPortConstants, 557
- OUT\_AC
  - NBCCCommon.h, 915
  - OutputPortConstants, 557
- OUT\_B
  - NBCCCommon.h, 915
  - OutputPortConstants, 558
- OUT\_BC
  - NBCCCommon.h, 915
  - OutputPortConstants, 558
- OUT\_C
  - NBCCCommon.h, 915
  - OutputPortConstants, 558
- OUT\_MODE\_BRAKE
  - NBCCCommon.h, 915
  - OutModeConstants, 565
- OUT\_MODE\_COAST
  - NBCCCommon.h, 916
  - OutModeConstants, 565
- OUT\_MODE\_MOTORON
  - NBCCCommon.h, 916
  - OutModeConstants, 565
- OUT\_MODE\_REGMETHOD
  - NBCCCommon.h, 916
  - OutModeConstants, 565
- OUT\_MODE\_REGULATED
  - NBCCCommon.h, 916
  - OutModeConstants, 565
- OUT\_OPTION\_HOLDATLIMIT
  - NBCCCommon.h, 916
  - OutOptionConstants, 566
- OUT\_OPTION\_RAMPDOWNTOLIMIT
  - NBCCCommon.h, 916
  - OutOptionConstants, 566
- OUT\_REGMODE\_IDLE
  - NBCCCommon.h, 916
  - OutRegModeConstants, 568
- OUT\_REGMODE\_SPEED
  - NBCCCommon.h, 916
  - OutRegModeConstants, 568
- OUT\_REGMODE\_SYNC
  - NBCCCommon.h, 916
  - OutRegModeConstants, 568
- OUT\_RUNSTATE\_HOLD
  - NBCCCommon.h, 917
  - OutRunStateConstants, 567
- OUT\_RUNSTATE\_IDLE
  - NBCCCommon.h, 917
  - OutRunStateConstants, 567
- OUT\_RUNSTATE\_RAMPDOWN
  - NBCCCommon.h, 917
  - OutRunStateConstants, 567
- OUT\_RUNSTATE\_RAMPUP
  - NBCCCommon.h, 917
  - OutRunStateConstants, 567
- OUT\_RUNSTATE\_RUNNING
  - NBCCCommon.h, 917
  - OutRunStateConstants, 567
- OutModeConstants
  - OUT\_MODE\_BRAKE, 565
  - OUT\_MODE\_COAST, 565
  - OUT\_MODE\_MOTORON, 565
  - OUT\_MODE\_REGMETHOD, 565
  - OUT\_MODE\_REGULATED, 565
- OutOptionConstants
  - OUT\_OPTION\_HOLDATLIMIT, 566
  - OUT\_OPTION\_RAMPDOWNTOLIMIT, 566
- Output field constants, 569
- Output module, 49
- Output module constants, 50
- Output module functions, 226
- Output module IOMAP offsets, 574
- Output port constants, 557
- Output port mode constants, 565
- Output port option constants, 566
- Output port regulation mode constants, 568
- Output port run state constants, 567
- Output port update flag constants, 561
- OutputFieldConstants
  - ActualSpeed, 570
  - BlockTachoCount, 570

- OutputMode, 570
- OutputOptions, 570
- Overload, 570
- Power, 571
- RegDValue, 571
- RegIValue, 571
- RegMode, 571
- RegPValue, 571
- RotationCount, 572
- RunState, 572
- TachoCount, 572
- TachoLimit, 572
- TurnRatio, 572
- UpdateFlags, 573
- OutputIOMAP
  - OutputOffsetActualSpeed, 574
  - OutputOffsetBlockTachoCount, 574
  - OutputOffsetFlags, 574
  - OutputOffsetMode, 574
  - OutputOffsetMotorRPM, 574
  - OutputOffsetOptions, 575
  - OutputOffsetOverloaded, 575
  - OutputOffsetPwnFreq, 575
  - OutputOffsetRegDParameter, 575
  - OutputOffsetRegIParameter, 575
  - OutputOffsetRegMode, 575
  - OutputOffsetRegPParameter, 575
  - OutputOffsetRotationCount, 575
  - OutputOffsetRunState, 575
  - OutputOffsetSpeed, 575
  - OutputOffsetSyncTurnParameter, 575
  - OutputOffsetTachoCount, 576
  - OutputOffsetTachoLimit, 576
- OutputMode
  - NBCCCommon.h, 917
  - OutputFieldConstants, 570
- OutputModuleFunctions
  - Coast, 229
  - CoastEx, 229
  - Float, 229
  - GetOutput, 230
  - MotorActualSpeed, 230
  - MotorBlockTachoCount, 230
  - MotorMode, 230
  - MotorOverload, 231
  - MotorPower, 231
  - MotorPwnFreq, 231
  - MotorRegDValue, 231
  - MotorRegIValue, 232
  - MotorRegPValue, 232
  - MotorRegulation, 232
  - MotorRotationCount, 232
  - MotorRunState, 233
  - MotorTachoCount, 233
  - MotorTachoLimit, 233
  - MotorTurnRatio, 233
  - Off, 234
  - OffEx, 234
  - OnFwd, 234
  - OnFwdEx, 235
  - OnFwdReg, 235
  - OnFwdRegEx, 235
  - OnFwdRegExPID, 236
  - OnFwdRegPID, 236
  - OnFwdSync, 236
  - OnFwdSyncEx, 237
  - OnFwdSyncExPID, 237
  - OnFwdSyncPID, 238
  - OnRev, 238
  - OnRevEx, 238
  - OnRevReg, 239
  - OnRevRegEx, 239
  - OnRevRegExPID, 239
  - OnRevRegPID, 240
  - OnRevSync, 240
  - OnRevSyncEx, 241
  - OnRevSyncExPID, 241
  - OnRevSyncPID, 241
  - ResetAllTachoCounts, 242
  - ResetBlockTachoCount, 242
  - ResetRotationCount, 242
  - ResetTachoCount, 243
  - RotateMotor, 243
  - RotateMotorEx, 243
  - RotateMotorExPID, 244
  - RotateMotorPID, 244
  - SetMotorPwnFreq, 245
  - SetOutput, 245
- OutputModuleID
  - ModuleIDConstants, 180
  - NBCCCommon.h, 917
- OutputModuleName
  - ModuleNameConstants, 178
  - NBCCCommon.h, 917
- OutputOffsetActualSpeed
  - NBCCCommon.h, 917
  - OutputIOMAP, 574
- OutputOffsetBlockTachoCount
  - NBCCCommon.h, 917
  - OutputIOMAP, 574
- OutputOffsetFlags
  - NBCCCommon.h, 918
  - OutputIOMAP, 574
- OutputOffsetMode
  - NBCCCommon.h, 918
  - OutputIOMAP, 574
- OutputOffsetMotorRPM
  - NBCCCommon.h, 918

- OutputIOMAP, 574
- OutputOffsetOptions
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetOverloaded
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetPwnFreq
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRegDParameter
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRegIPParameter
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRegMode
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRegPPParameter
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRotationCount
  - NBCCCommon.h, 918
  - OutputIOMAP, 575
- OutputOffsetRunState
  - NBCCCommon.h, 919
  - OutputIOMAP, 575
- OutputOffsetSpeed
  - NBCCCommon.h, 919
  - OutputIOMAP, 575
- OutputOffsetSyncTurnParameter
  - NBCCCommon.h, 919
  - OutputIOMAP, 575
- OutputOffsetTachoCount
  - NBCCCommon.h, 919
  - OutputIOMAP, 576
- OutputOffsetTachoLimit
  - NBCCCommon.h, 919
  - OutputIOMAP, 576
- OutputOptions
  - NBCCCommon.h, 919
  - OutputFieldConstants, 570
- OutputPortConstants
  - OUT\_A, 557
  - OUT\_AB, 557
  - OUT\_ABC, 557
  - OUT\_AC, 557
  - OUT\_B, 558
  - OUT\_BC, 558
  - OUT\_C, 558
- OutRegModeConstants
  - OUT\_REGMODE\_IDLE, 568
  - OUT\_REGMODE\_SPEED, 568
  - OUT\_REGMODE\_SYNC, 568
- OutRunStateConstants
  - OUT\_RUNSTATE\_HOLD, 567
  - OUT\_RUNSTATE\_IDLE, 567
  - OUT\_RUNSTATE\_RAMPDOWN, 567
  - OUT\_RUNSTATE\_RAMPUP, 567
  - OUT\_RUNSTATE\_RUNNING, 567
- OutUFConstants
  - UF\_PENDING\_UPDATES, 561
  - UF\_UPDATE\_MODE, 561
  - UF\_UPDATE\_PID\_VALUES, 561
  - UF\_UPDATE\_RESET\_BLOCK\_COUNT, 561
  - UF\_UPDATE\_RESET\_COUNT, 561
  - UF\_UPDATE\_RESET\_ROTATION\_COUNT, 561
  - UF\_UPDATE\_SPEED, 562
  - UF\_UPDATE\_TACHO\_LIMIT, 562
- Overload
  - NBCCCommon.h, 919
  - OutputFieldConstants, 570
- Param1
  - CommExecuteFunctionType, 727
- Param2
  - CommExecuteFunctionType, 727
- Param3
  - CommExecuteFunctionType, 727
- Pattern
  - ListFilesType, 790
- PF/IR Train function constants, 683
- PF\_CHANNEL\_1
  - NBCCCommon.h, 919
  - PFChannelConstants, 681
- PF\_CHANNEL\_2
  - NBCCCommon.h, 920
  - PFChannelConstants, 681
- PF\_CHANNEL\_3
  - NBCCCommon.h, 920
  - PFChannelConstants, 681
- PF\_CHANNEL\_4
  - NBCCCommon.h, 920
  - PFChannelConstants, 681
- PF\_CMD\_BRAKE
  - NBCCCommon.h, 920
  - PFCmdConstants, 680
- PF\_CMD\_FWD
  - NBCCCommon.h, 920
  - PFCmdConstants, 680
- PF\_CMD\_REV
  - NBCCCommon.h, 920
  - PFCmdConstants, 680
- PF\_CMD\_STOP
  - NBCCCommon.h, 920

- PFCmdConstants, 680
- PF\_CST\_CLEAR1\_CLEAR2
  - NBCCCommon.h, 920
  - PFCSTOptions, 688
- PF\_CST\_CLEAR1\_SET2
  - NBCCCommon.h, 920
  - PFCSTOptions, 688
- PF\_CST\_DECREMENT\_PWM
  - NBCCCommon.h, 921
  - PFCSTOptions, 688
- PF\_CST\_FULL\_FWD
  - NBCCCommon.h, 921
  - PFCSTOptions, 688
- PF\_CST\_FULL\_REV
  - NBCCCommon.h, 921
  - PFCSTOptions, 688
- PF\_CST\_INCREMENT\_PWM
  - NBCCCommon.h, 921
  - PFCSTOptions, 688
- PF\_CST\_SET1\_CLEAR2
  - NBCCCommon.h, 921
  - PFCSTOptions, 688
- PF\_CST\_SET1\_SET2
  - NBCCCommon.h, 921
  - PFCSTOptions, 689
- PF\_CST\_TOGGLE\_DIR
  - NBCCCommon.h, 921
  - PFCSTOptions, 689
- PF\_FUNC\_CLEAR
  - NBCCCommon.h, 921
  - PFPinFuncs, 687
- PF\_FUNC\_NOCHANGE
  - NBCCCommon.h, 921
  - PFPinFuncs, 687
- PF\_FUNC\_SET
  - NBCCCommon.h, 921
  - PFPinFuncs, 687
- PF\_FUNC\_TOGGLE
  - NBCCCommon.h, 922
  - PFPinFuncs, 687
- PF\_MODE\_COMBO\_DIRECT
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_MODE\_COMBO\_PWM
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_MODE\_SINGLE\_OUTPUT\_CST
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_MODE\_SINGLE\_OUTPUT\_PWM
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_MODE\_SINGLE\_PIN\_CONT
  - NBCCCommon.h, 922
- PFModeConstants, 682
- PF\_MODE\_SINGLE\_PIN\_TIME
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_MODE\_TRAIN
  - NBCCCommon.h, 922
  - PFModeConstants, 682
- PF\_OUT\_A
  - NBCCCommon.h, 922
  - PFOutputs, 685
- PF\_OUT\_B
  - NBCCCommon.h, 923
  - PFOutputs, 685
- PF\_PIN\_C1
  - NBCCCommon.h, 923
  - PFPinConstants, 686
- PF\_PIN\_C2
  - NBCCCommon.h, 923
  - PFPinConstants, 686
- PF\_PWM\_BRAKE
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FLOAT
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FWD1
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FWD2
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FWD3
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FWD4
  - NBCCCommon.h, 923
  - PFPWMOptions, 690
- PF\_PWM\_FWD5
  - NBCCCommon.h, 923
  - PFPWMOptions, 691
- PF\_PWM\_FWD6
  - NBCCCommon.h, 924
  - PFPWMOptions, 691
- PF\_PWM\_FWD7
  - NBCCCommon.h, 924
  - PFPWMOptions, 691
- PF\_PWM\_REV1
  - NBCCCommon.h, 924
  - PFPWMOptions, 691
- PF\_PWM\_REV2
  - NBCCCommon.h, 924
  - PFPWMOptions, 691
- PF\_PWM\_REV3
  - NBCCCommon.h, 924

- PFPWMOptions, [691](#)
- PF\_PWM\_REV4
  - NBCCCommon.h, [924](#)
  - PFPWMOptions, [691](#)
- PF\_PWM\_REV5
  - NBCCCommon.h, [924](#)
  - PFPWMOptions, [691](#)
- PF\_PWM\_REV6
  - NBCCCommon.h, [924](#)
  - PFPWMOptions, [691](#)
- PF\_PWM\_REV7
  - NBCCCommon.h, [924](#)
  - PFPWMOptions, [692](#)
- PFChannelConstants
  - PF\_CHANNEL\_1, [681](#)
  - PF\_CHANNEL\_2, [681](#)
  - PF\_CHANNEL\_3, [681](#)
  - PF\_CHANNEL\_4, [681](#)
- PFCmdConstants
  - PF\_CMD\_BRAKE, [680](#)
  - PF\_CMD\_FWD, [680](#)
  - PF\_CMD\_REV, [680](#)
  - PF\_CMD\_STOP, [680](#)
- PFCSTOptions
  - PF\_CST\_CLEAR1\_CLEAR2, [688](#)
  - PF\_CST\_CLEAR1\_SET2, [688](#)
  - PF\_CST\_DECREMENT\_PWM, [688](#)
  - PF\_CST\_FULL\_FWD, [688](#)
  - PF\_CST\_FULL\_REV, [688](#)
  - PF\_CST\_INCREMENT\_PWM, [688](#)
  - PF\_CST\_SET1\_CLEAR2, [688](#)
  - PF\_CST\_SET1\_SET2, [689](#)
  - PF\_CST\_TOGGLE\_DIR, [689](#)
- PFModeConstants
  - PF\_MODE\_COMBO\_DIRECT, [682](#)
  - PF\_MODE\_COMBO\_PWM, [682](#)
  - PF\_MODE\_SINGLE\_OUTPUT\_CST, [682](#)
  - PF\_MODE\_SINGLE\_OUTPUT\_PWM, [682](#)
  - PF\_MODE\_SINGLE\_PIN\_CONT, [682](#)
  - PF\_MODE\_SINGLE\_PIN\_TIME, [682](#)
  - PF\_MODE\_TRAIN, [682](#)
- PFOutputs
  - PF\_OUT\_A, [685](#)
  - PF\_OUT\_B, [685](#)
- PFPinConstants
  - PF\_PIN\_C1, [686](#)
  - PF\_PIN\_C2, [686](#)
- PFPinFuncs
  - PF\_FUNC\_CLEAR, [687](#)
  - PF\_FUNC\_NOCHANGE, [687](#)
  - PF\_FUNC\_SET, [687](#)
  - PF\_FUNC\_TOGGLE, [687](#)
- PFPWMOptions
  - PF\_PWM\_BRAKE, [690](#)
  - PF\_PWM\_FLOAT, [690](#)
  - PF\_PWM\_FWD1, [690](#)
  - PF\_PWM\_FWD2, [690](#)
  - PF\_PWM\_FWD3, [690](#)
  - PF\_PWM\_FWD4, [690](#)
  - PF\_PWM\_FWD5, [691](#)
  - PF\_PWM\_FWD6, [691](#)
  - PF\_PWM\_FWD7, [691](#)
  - PF\_PWM\_REV1, [691](#)
  - PF\_PWM\_REV2, [691](#)
  - PF\_PWM\_REV3, [691](#)
  - PF\_PWM\_REV4, [691](#)
  - PF\_PWM\_REV5, [691](#)
  - PF\_PWM\_REV6, [691](#)
  - PF\_PWM\_REV7, [692](#)
- PI
  - MiscConstants, [181](#)
  - NBCCCommon.h, [925](#)
- PID constants, [559](#)
- PID\_0
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_1
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_2
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_3
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_4
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_5
  - NBCCCommon.h, [925](#)
  - PIDConstants, [559](#)
- PID\_6
  - NBCCCommon.h, [925](#)
  - PIDConstants, [560](#)
- PID\_7
  - NBCCCommon.h, [925](#)
  - PIDConstants, [560](#)
- PIDConstants
  - PID\_0, [559](#)
  - PID\_1, [559](#)
  - PID\_2, [559](#)
  - PID\_3, [559](#)
  - PID\_4, [559](#)
  - PID\_5, [559](#)
  - PID\_6, [560](#)
  - PID\_7, [560](#)
- PlayFile
  - NXCDefs.h, [1226](#)

- SoundModuleFunctions, 270
- PlayFileEx
  - NXCDefs.h, 1226
  - SoundModuleFunctions, 270
- PlaySound
  - NXCDefs.h, 1227
  - SoundModuleFunctions, 271
- PlayTone
  - NXCDefs.h, 1227
  - SoundModuleFunctions, 271
- PlayToneEx
  - NXCDefs.h, 1227
  - SoundModuleFunctions, 271
- PlayTones
  - NXCDefs.h, 1228
  - SoundModuleFunctions, 272
- PointOut
  - DisplayModuleFunctions, 259
  - NXCDefs.h, 1228
- Points
  - DrawPolygonType, 759
- PolyOut
  - DisplayModuleFunctions, 259
  - NXCDefs.h, 1228
- Port
  - ColorSensorReadType, 720
  - CommLSCheckStatusType, 732
  - CommLSReadType, 733
  - CommLSWriteExType, 735
  - CommLSWriteType, 737
- Pow
  - cmathAPI, 410
  - NXCDefs.h, 1052
- pow
  - cmathAPI, 421
  - NXCDefs.h, 1229
- Power
  - NBCCCommon.h, 925
  - OutputFieldConstants, 571
- Power Function channel constants, 681
- Power Function command constants, 680
- Power Function CST options constants, 688
- Power Function mode constants, 682
- Power Function output constants, 685
- Power Function pin constants, 686
- Power Function PWM option constants, 690
- Power Function single pin function constants, 687
- PowerDown
  - IOCtrlModuleFunctions, 302
  - NXCDefs.h, 1229
- PowerOn constants, 498
- PowerState
  - CommBTONOffType, 724
- Precedes
  - CommandModuleFunctions, 307
  - NXCDefs.h, 1229
- Pressed
  - ReadButtonType, 799
- printf
  - cstdioAPI, 432
  - NXCDefs.h, 1230
- PROG\_ABORT
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- PROG\_ERROR
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- PROG\_IDLE
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- PROG\_OK
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- PROG\_RESET
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- PROG\_RUNNING
  - CommandProgStatus, 494
  - NBCCCommon.h, 926
- Program status constants, 494
- PSP\_BTNSET1\_DOWN
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 926
- PSP\_BTNSET1\_L3
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 926
- PSP\_BTNSET1\_LEFT
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 926
- PSP\_BTNSET1\_R3
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 926
- PSP\_BTNSET1\_RIGHT
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 926
- PSP\_BTNSET1\_UP
  - MSPSPNXBtnSet1, 706
  - NBCCCommon.h, 927
- PSP\_BTNSET2\_CIRCLE
  - MSPSPNXBtnSet2, 707
  - NBCCCommon.h, 927
- PSP\_BTNSET2\_CROSS
  - MSPSPNXBtnSet2, 707
  - NBCCCommon.h, 927
- PSP\_BTNSET2\_L1
  - MSPSPNXBtnSet2, 707
  - NBCCCommon.h, 927
- PSP\_BTNSET2\_L2

- MSPSPNXBtnSet2, [707](#)
- NBCCCommon.h, [927](#)
- PSP\_BTNSET2\_R1
  - MSPSPNXBtnSet2, [707](#)
  - NBCCCommon.h, [927](#)
- PSP\_BTNSET2\_R2
  - MSPSPNXBtnSet2, [707](#)
  - NBCCCommon.h, [927](#)
- PSP\_BTNSET2\_SQUARE
  - MSPSPNXBtnSet2, [707](#)
  - NBCCCommon.h, [927](#)
- PSP\_BTNSET2\_TRIANGLE
  - MSPSPNXBtnSet2, [708](#)
  - NBCCCommon.h, [927](#)
- PSP\_CMD\_ANALOG
  - MSPSPNX, [704](#)
  - NBCCCommon.h, [927](#)
- PSP\_CMD\_DIGITAL
  - MSPSPNX, [704](#)
  - NBCCCommon.h, [927](#)
- PSP\_REG\_BTNSET1
  - MSPSPNX, [704](#)
  - NBCCCommon.h, [928](#)
- PSP\_REG\_BTNSET2
  - MSPSPNX, [704](#)
  - NBCCCommon.h, [928](#)
- PSP\_REG\_XLEFT
  - MSPSPNX, [704](#)
  - NBCCCommon.h, [928](#)
- PSP\_REG\_XRIGHT
  - MSPSPNX, [705](#)
  - NBCCCommon.h, [928](#)
- PSP\_REG\_YLEFT
  - MSPSPNX, [705](#)
  - NBCCCommon.h, [928](#)
- PSP\_REG\_YRIGHT
  - MSPSPNX, [705](#)
  - NBCCCommon.h, [928](#)
- PSPNxAnalog
  - MindSensorsAPI, [160](#)
  - NXCDefs.h, [1230](#)
- PSPNxAnalogEx
  - MindSensorsAPI, [160](#)
  - NXCDefs.h, [1230](#)
- PSPNxDigital
  - MindSensorsAPI, [161](#)
  - NXCDefs.h, [1231](#)
- PSPNxDigitalEx
  - MindSensorsAPI, [161](#)
  - NXCDefs.h, [1231](#)
- putc
  - cstdioAPI, [427](#)
  - NXCDefs.h, [1052](#)
- QueueID
  - MessageReadType, [795](#)
  - MessageWriteType, [797](#)
- quot
  - div\_t, [745](#)
  - ldiv\_t, [789](#)
- RADIANS\_PER\_DEGREE
  - MiscConstants, [182](#)
  - NBCCCommon.h, [928](#)
- rand
  - cstdlibAPI, [440](#)
  - NXCDefs.h, [1231](#)
- RAND\_MAX
  - NBCCCommon.h, [928](#)
  - NXTLimits, [712](#)
- Random
  - cstdlibAPI, [440](#)
  - NXCDefs.h, [1232](#)
- RandomNumber
  - NBCCCommon.h, [928](#)
  - SysCallConstants, [473](#)
- RandomNumberType, [798](#)
- Result, [798](#)
- RawArray
  - ColorSensorReadType, [720](#)
- RawVal
  - ComputeCalibValueType, [739](#)
- RawValue
  - InputFieldConstants, [549](#)
  - NBCCCommon.h, [928](#)
- RCX and Scout opcode constants, [670](#)
- RCX and Scout sound constants, [650](#)
- RCX and Scout source constants, [665](#)
- RCX constants, [641](#)
- RCX IR remote constants, [647](#)
- RCX output constants, [642](#)
- RCX output direction constants, [645](#)
- RCX output mode constants, [644](#)
- RCX output power constants, [646](#)
- RCX\_AbsVarOp
  - NBCCCommon.h, [928](#)
  - RCXOpcodeConstants, [671](#)
- RCX\_AndVarOp
  - NBCCCommon.h, [929](#)
  - RCXOpcodeConstants, [671](#)
- RCX\_AutoOffOp
  - NBCCCommon.h, [929](#)
  - RCXOpcodeConstants, [671](#)
- RCX\_BatteryLevelOp
  - NBCCCommon.h, [929](#)
  - RCXOpcodeConstants, [672](#)
- RCX\_BatteryLevelSrc
  - NBCCCommon.h, [929](#)



- RCXSourceConstants, 666
- RCX\_BootModeOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_CalibrateEventOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearAllEventsOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearCounterOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearMsgOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearSensorOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearSoundOp
  - NBCCCommon.h, 929
  - RCXOpcodeConstants, 672
- RCX\_ClearTimerOp
  - NBCCCommon.h, 930
  - RCXOpcodeConstants, 672
- RCX\_ClickCounterSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_ConstantSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_CounterSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_DatalogOp
  - NBCCCommon.h, 930
  - RCXOpcodeConstants, 672
- RCX\_DatalogRawDirectSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_DatalogRawIndirectSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_DatalogSrcDirectSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_DatalogSrcIndirectSrc
  - NBCCCommon.h, 930
  - RCXSourceConstants, 666
- RCX\_DatalogValueDirectSrc
  - NBCCCommon.h, 931
  - RCXSourceConstants, 666
- RCX\_DatalogValueIndirectSrc
  - NBCCCommon.h, 931
- RCXSourceConstants, 666
- RCX\_DecCounterOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 672
- RCX\_DeleteSubOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DeleteSubsOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DeleteTaskOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DeleteTasksOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DirectEventOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DisplayOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DivVarOp
  - NBCCCommon.h, 931
  - RCXOpcodeConstants, 673
- RCX\_DurationSrc
  - NBCCCommon.h, 931
  - RCXSourceConstants, 667
- RCX\_EventStateSrc
  - NBCCCommon.h, 932
  - RCXSourceConstants, 667
- RCX\_FirmwareVersionSrc
  - NBCCCommon.h, 932
  - RCXSourceConstants, 667
- RCX\_GlobalMotorStatusSrc
  - NBCCCommon.h, 932
  - RCXSourceConstants, 667
- RCX\_GOutputDirOp
  - NBCCCommon.h, 932
  - RCXOpcodeConstants, 673
- RCX\_GOutputModeOp
  - NBCCCommon.h, 932
  - RCXOpcodeConstants, 673
- RCX\_GOutputPowerOp
  - NBCCCommon.h, 932
  - RCXOpcodeConstants, 673
- RCX\_HysteresisSrc
  - NBCCCommon.h, 932
  - RCXSourceConstants, 667
- RCX\_IncCounterOp
  - NBCCCommon.h, 932
  - RCXOpcodeConstants, 673
- RCX\_IndirectVarSrc
  - NBCCCommon.h, 932



- RCXSourceConstants, 667
- RCX\_InputBooleanSrc
  - NBCCCommon.h, 932
  - RCXSourceConstants, 667
- RCX\_InputModeOp
  - NBCCCommon.h, 932
  - RCXOpcodeConstants, 674
- RCX\_InputModeSrc
  - NBCCCommon.h, 933
  - RCXSourceConstants, 667
- RCX\_InputRawSrc
  - NBCCCommon.h, 933
  - RCXSourceConstants, 667
- RCX\_InputTypeOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_InputTypeSrc
  - NBCCCommon.h, 933
  - RCXSourceConstants, 667
- RCX\_InputValueSrc
  - NBCCCommon.h, 933
  - RCXSourceConstants, 667
- RCX\_IRModeOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_LightOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_LowerThresholdSrc
  - NBCCCommon.h, 933
  - RCXSourceConstants, 668
- RCX\_LSBLinkTimeOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_LSCalibrateOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_LSHysteresisOp
  - NBCCCommon.h, 933
  - RCXOpcodeConstants, 674
- RCX\_LSLowerThreshOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 674
- RCX\_LSupperThreshOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 674
- RCX\_MessageOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 674
- RCX\_MessageSrc
  - NBCCCommon.h, 934
  - RCXSourceConstants, 668
- RCX\_MulVarOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 674
- RCX\_MuteSoundOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 675
- RCX\_OnOffFloatOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 675
- RCX\_OrVarOp
  - NBCCCommon.h, 934
  - RCXOpcodeConstants, 675
- RCX\_OUT\_A
  - NBCCCommon.h, 934
  - RCXOutputConstants, 642
- RCX\_OUT\_AB
  - NBCCCommon.h, 935
  - RCXOutputConstants, 642
- RCX\_OUT\_ABC
  - NBCCCommon.h, 935
  - RCXOutputConstants, 642
- RCX\_OUT\_AC
  - NBCCCommon.h, 935
  - RCXOutputConstants, 642
- RCX\_OUT\_B
  - NBCCCommon.h, 935
  - RCXOutputConstants, 643
- RCX\_OUT\_BC
  - NBCCCommon.h, 935
  - RCXOutputConstants, 643
- RCX\_OUT\_C
  - NBCCCommon.h, 935
  - RCXOutputConstants, 643
- RCX\_OUT\_FLOAT
  - NBCCCommon.h, 935
  - RCXOutputMode, 644
- RCX\_OUT\_FULL
  - NBCCCommon.h, 935
  - RCXOutputPower, 646
- RCX\_OUT\_FWD
  - NBCCCommon.h, 935
  - RCXOutputDirection, 645
- RCX\_OUT\_HALF
  - NBCCCommon.h, 936
  - RCXOutputPower, 646
- RCX\_OUT\_LOW
  - NBCCCommon.h, 936
  - RCXOutputPower, 646
- RCX\_OUT\_OFF
  - NBCCCommon.h, 936
  - RCXOutputMode, 644
- RCX\_OUT\_ON
  - NBCCCommon.h, 936
  - RCXOutputMode, 644
- RCX\_OUT\_REV
  - NBCCCommon.h, 936

- RCXOutputDirection, 645
- RCX\_OUT\_TOGGLE
  - NBCCCommon.h, 936
  - RCXOutputDirection, 645
- RCX\_OutputDirOp
  - NBCCCommon.h, 936
  - RCXOpcodeConstants, 675
- RCX\_OutputPowerOp
  - NBCCCommon.h, 936
  - RCXOpcodeConstants, 675
- RCX\_OutputStatusSrc
  - NBCCCommon.h, 936
  - RCXSourceConstants, 668
- RCX\_PBTurnOffOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PingOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PlaySoundOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PlayToneOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PlayToneVarOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PollMemoryOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 675
- RCX\_PollOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 676
- RCX\_ProgramSlotSrc
  - NBCCCommon.h, 937
  - RCXSourceConstants, 668
- RCX\_RandomSrc
  - NBCCCommon.h, 937
  - RCXSourceConstants, 668
- RCX\_RemoteKeysReleased
  - NBCCCommon.h, 937
  - RCXRemoteConstants, 647
- RCX\_RemoteOp
  - NBCCCommon.h, 937
  - RCXOpcodeConstants, 676
- RCX\_RemoteOutABackward
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 647
- RCX\_RemoteOutAForward
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 647
- RCX\_RemoteOutBBackward
  - NBCCCommon.h, 938
- RCXRemoteConstants, 647
- RCX\_RemoteOutBForward
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 647
- RCX\_RemoteOutCBackward
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemoteOutCForward
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemotePBMessage1
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemotePBMessage2
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemotePBMessage3
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemotePlayASound
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemoteSelProgram1
  - NBCCCommon.h, 938
  - RCXRemoteConstants, 648
- RCX\_RemoteSelProgram2
  - NBCCCommon.h, 939
  - RCXRemoteConstants, 648
- RCX\_RemoteSelProgram3
  - NBCCCommon.h, 939
  - RCXRemoteConstants, 648
- RCX\_RemoteSelProgram4
  - NBCCCommon.h, 939
  - RCXRemoteConstants, 648
- RCX\_RemoteSelProgram5
  - NBCCCommon.h, 939
  - RCXRemoteConstants, 648
- RCX\_RemoteStopOutOff
  - NBCCCommon.h, 939
  - RCXRemoteConstants, 649
- RCX\_ScoutCounterLimitSrc
  - NBCCCommon.h, 939
  - RCXSourceConstants, 668
- RCX\_ScoutEventFBSrc
  - NBCCCommon.h, 939
  - RCXSourceConstants, 668
- RCX\_ScoutLightParamsSrc
  - NBCCCommon.h, 939
  - RCXSourceConstants, 668
- RCX\_ScoutOp
  - NBCCCommon.h, 939
  - RCXOpcodeConstants, 676
- RCX\_ScoutRulesOp
  - NBCCCommon.h, 939

- RCXOpcodeConstants, 676
- RCX\_ScoutRulesSrc
  - NBCCCommon.h, 939
  - RCXSourceConstants, 668
- RCX\_ScoutTimerLimitSrc
  - NBCCCommon.h, 940
  - RCXSourceConstants, 668
- RCX\_SelectProgramOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SendUARTDataOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetCounterOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetDatalogOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetEventOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetFeedbackOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetPriorityOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 676
- RCX\_SetSourceValueOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 677
- RCX\_SetTimerLimitOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 677
- RCX\_SetVarOp
  - NBCCCommon.h, 940
  - RCXOpcodeConstants, 677
- RCX\_SetWatchOp
  - NBCCCommon.h, 941
  - RCXOpcodeConstants, 677
- RCX\_SgnVarOp
  - NBCCCommon.h, 941
  - RCXOpcodeConstants, 677
- RCX\_SoundOp
  - NBCCCommon.h, 941
  - RCXOpcodeConstants, 677
- RCX\_StartTaskOp
  - NBCCCommon.h, 941
  - RCXOpcodeConstants, 677
- RCX\_StopAllTasksOp
  - NBCCCommon.h, 941
  - RCXOpcodeConstants, 677
- RCX\_StopTaskOp
  - NBCCCommon.h, 941
- RCXOpcodeConstants, 677
  - RCX\_SubVarOp
    - NBCCCommon.h, 941
    - RCXOpcodeConstants, 677
  - RCX\_SumVarOp
    - NBCCCommon.h, 941
    - RCXOpcodeConstants, 677
  - RCX\_TaskEventsSrc
    - NBCCCommon.h, 941
    - RCXSourceConstants, 669
  - RCX\_TenMSTimerSrc
    - NBCCCommon.h, 941
    - RCXSourceConstants, 669
  - RCX\_TimerSrc
    - NBCCCommon.h, 941
    - RCXSourceConstants, 669
  - RCX\_UARTSetupSrc
    - NBCCCommon.h, 942
    - RCXSourceConstants, 669
  - RCX\_UnlockFirmOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_UnlockOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_UnmuteSoundOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_UploadDatalogOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_UpperThresholdSrc
    - NBCCCommon.h, 942
    - RCXSourceConstants, 669
  - RCX\_VariableSrc
    - NBCCCommon.h, 942
    - RCXSourceConstants, 669
  - RCX\_ViewSourceValOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_VLLOp
    - NBCCCommon.h, 942
    - RCXOpcodeConstants, 678
  - RCX\_WatchSrc
    - NBCCCommon.h, 942
    - RCXSourceConstants, 669
- RCXOpcodeConstants
  - RCX\_AbsVarOp, 671
  - RCX\_AndVarOp, 671
  - RCX\_AutoOffOp, 671
  - RCX\_BatteryLevelOp, 672
  - RCX\_BootModeOp, 672
  - RCX\_CalibrateEventOp, 672
  - RCX\_ClearAllEventsOp, 672

- RCX\_ClearCounterOp, [672](#)
- RCX\_ClearMsgOp, [672](#)
- RCX\_ClearSensorOp, [672](#)
- RCX\_ClearSoundOp, [672](#)
- RCX\_ClearTimerOp, [672](#)
- RCX\_DatalogOp, [672](#)
- RCX\_DecCounterOp, [672](#)
- RCX\_DeleteSubOp, [673](#)
- RCX\_DeleteSubsOp, [673](#)
- RCX\_DeleteTaskOp, [673](#)
- RCX\_DeleteTasksOp, [673](#)
- RCX\_DirectEventOp, [673](#)
- RCX\_DisplayOp, [673](#)
- RCX\_DivVarOp, [673](#)
- RCX\_GOutputDirOp, [673](#)
- RCX\_GOutputModeOp, [673](#)
- RCX\_GOutputPowerOp, [673](#)
- RCX\_IncCounterOp, [673](#)
- RCX\_InputModeOp, [674](#)
- RCX\_InputTypeOp, [674](#)
- RCX\_IRModeOp, [674](#)
- RCX\_LightOp, [674](#)
- RCX\_LSblinkTimeOp, [674](#)
- RCX\_LSCalibrateOp, [674](#)
- RCX\_LSHysteresisOp, [674](#)
- RCX\_LSLowerThreshOp, [674](#)
- RCX\_LSupperThreshOp, [674](#)
- RCX\_MessageOp, [674](#)
- RCX\_MulVarOp, [674](#)
- RCX\_MuteSoundOp, [675](#)
- RCX\_OnOffFloatOp, [675](#)
- RCX\_OrVarOp, [675](#)
- RCX\_OutputDirOp, [675](#)
- RCX\_OutputPowerOp, [675](#)
- RCX\_PBTurnOffOp, [675](#)
- RCX\_PingOp, [675](#)
- RCX\_PlaySoundOp, [675](#)
- RCX\_PlayToneOp, [675](#)
- RCX\_PlayToneVarOp, [675](#)
- RCX\_PollMemoryOp, [675](#)
- RCX\_PollOp, [676](#)
- RCX\_RemoteOp, [676](#)
- RCX\_ScoutOp, [676](#)
- RCX\_ScoutRulesOp, [676](#)
- RCX\_SelectProgramOp, [676](#)
- RCX\_SendUARTDataOp, [676](#)
- RCX\_SetCounterOp, [676](#)
- RCX\_SetDatalogOp, [676](#)
- RCX\_SetEventOp, [676](#)
- RCX\_SetFeedbackOp, [676](#)
- RCX\_SetPriorityOp, [676](#)
- RCX\_SetSourceValueOp, [677](#)
- RCX\_SetTimerLimitOp, [677](#)
- RCX\_SetVarOp, [677](#)
- RCX\_SetWatchOp, [677](#)
- RCX\_SgnVarOp, [677](#)
- RCX\_SoundOp, [677](#)
- RCX\_StartTaskOp, [677](#)
- RCX\_StopAllTasksOp, [677](#)
- RCX\_StopTaskOp, [677](#)
- RCX\_SubVarOp, [677](#)
- RCX\_SumVarOp, [677](#)
- RCX\_UnlockFirmOp, [678](#)
- RCX\_UnlockOp, [678](#)
- RCX\_UnmuteSoundOp, [678](#)
- RCX\_UploadDatalogOp, [678](#)
- RCX\_ViewSourceValOp, [678](#)
- RCX\_VLLOp, [678](#)
- RCXOutputConstants
  - RCX\_OUT\_A, [642](#)
  - RCX\_OUT\_AB, [642](#)
  - RCX\_OUT\_ABC, [642](#)
  - RCX\_OUT\_AC, [642](#)
  - RCX\_OUT\_B, [643](#)
  - RCX\_OUT\_BC, [643](#)
  - RCX\_OUT\_C, [643](#)
- RCXOutputDirection
  - RCX\_OUT\_FWD, [645](#)
  - RCX\_OUT\_REV, [645](#)
  - RCX\_OUT\_TOGGLE, [645](#)
- RCXOutputMode
  - RCX\_OUT\_FLOAT, [644](#)
  - RCX\_OUT\_OFF, [644](#)
  - RCX\_OUT\_ON, [644](#)
- RCXOutputPower
  - RCX\_OUT\_FULL, [646](#)
  - RCX\_OUT\_HALF, [646](#)
  - RCX\_OUT\_LOW, [646](#)
- RCXRemoteConstants
  - RCX\_RemoteKeysReleased, [647](#)
  - RCX\_RemoteOutABackward, [647](#)
  - RCX\_RemoteOutAForward, [647](#)
  - RCX\_RemoteOutBBackward, [647](#)
  - RCX\_RemoteOutBForward, [647](#)
  - RCX\_RemoteOutCBackward, [648](#)
  - RCX\_RemoteOutCForward, [648](#)
  - RCX\_RemotePBMessage1, [648](#)
  - RCX\_RemotePBMessage2, [648](#)
  - RCX\_RemotePBMessage3, [648](#)
  - RCX\_RemotePlayASound, [648](#)
  - RCX\_RemoteSelProgram1, [648](#)
  - RCX\_RemoteSelProgram2, [648](#)
  - RCX\_RemoteSelProgram3, [648](#)
  - RCX\_RemoteSelProgram4, [648](#)
  - RCX\_RemoteSelProgram5, [648](#)
  - RCX\_RemoteStopOutOff, [649](#)
- RCXSoundConstants
  - SOUND\_CLICK, [650](#)

- SOUND\_DOUBLE\_BEEP, 650
- SOUND\_DOWN, 650
- SOUND\_FAST\_UP, 650
- SOUND\_LOW\_BEEP, 650
- SOUND\_UP, 651
- RCXSourceConstants
  - RCX\_BatteryLevelSrc, 666
  - RCX\_ClickCounterSrc, 666
  - RCX\_ConstantSrc, 666
  - RCX\_CounterSrc, 666
  - RCX\_DatalogRawDirectSrc, 666
  - RCX\_DatalogRawIndirectSrc, 666
  - RCX\_DatalogSrcDirectSrc, 666
  - RCX\_DatalogSrcIndirectSrc, 666
  - RCX\_DatalogValueDirectSrc, 666
  - RCX\_DatalogValueIndirectSrc, 666
  - RCX\_DurationSrc, 667
  - RCX\_EventStateSrc, 667
  - RCX\_FirmwareVersionSrc, 667
  - RCX\_GlobalMotorStatusSrc, 667
  - RCX\_HysteresisSrc, 667
  - RCX\_IndirectVarSrc, 667
  - RCX\_InputBooleanSrc, 667
  - RCX\_InputModeSrc, 667
  - RCX\_InputRawSrc, 667
  - RCX\_InputTypeSrc, 667
  - RCX\_InputValueSrc, 667
  - RCX\_LowerThresholdSrc, 668
  - RCX\_MessageSrc, 668
  - RCX\_OutputStatusSrc, 668
  - RCX\_ProgramSlotSrc, 668
  - RCX\_RandomSrc, 668
  - RCX\_ScoutCounterLimitSrc, 668
  - RCX\_ScoutEventFBSrc, 668
  - RCX\_ScoutLightParamsSrc, 668
  - RCX\_ScoutRulesSrc, 668
  - RCX\_ScoutTimerLimitSrc, 668
  - RCX\_TaskEventsSrc, 669
  - RCX\_TenMSTimerSrc, 669
  - RCX\_TimerSrc, 669
  - RCX\_UARTSetupSrc, 669
  - RCX\_UpperThresholdSrc, 669
  - RCX\_VariableSrc, 669
  - RCX\_WatchSrc, 669
- Read
  - LoaderModuleFunctions, 390
  - NXCDefs.h, 1232
- ReadButton
  - NBCCCommon.h, 943
  - SysCallConstants, 473
- ReadButtonEx
  - ButtonModuleFunctions, 368
  - NXCDefs.h, 1232
- ReadButtonType, 799
  - Count, 799
  - Index, 799
  - Pressed, 799
  - Reset, 799
  - Result, 800
- ReadBytes
  - LoaderModuleFunctions, 390
  - NXCDefs.h, 1233
- ReadI2CRegister
  - LowSpeedModuleFunctions, 290
  - NXCDefs.h, 1233
- ReadLn
  - LoaderModuleFunctions, 390
  - NXCDefs.h, 1234
- ReadLnString
  - LoaderModuleFunctions, 391
  - NXCDefs.h, 1234
- ReadNRLinkBytes
  - MindSensorsAPI, 161
  - NXCDefs.h, 1234
- ReadNRLinkBytesEx
  - MindSensorsAPI, 162
  - NXCDefs.h, 1235
- ReadSemData
  - NBCCCommon.h, 943
  - SysCallConstants, 473
- ReadSemDataType, 801
  - Request, 801
  - SemData, 801
- ReadSensorColorEx
  - InputModuleFunctions, 212
  - NXCDefs.h, 1235
- ReadSensorColorRaw
  - InputModuleFunctions, 213
  - NXCDefs.h, 1236
- ReadSensorHTAccel
  - HiTechnicAPI, 90
  - NXCDefs.h, 1236
- ReadSensorHTColor
  - HiTechnicAPI, 91
  - NXCDefs.h, 1236
- ReadSensorHTColor2Active
  - HiTechnicAPI, 91
  - NXCDefs.h, 1237
- ReadSensorHTIRReceiver
  - HiTechnicAPI, 91
  - NXCDefs.h, 1237
- ReadSensorHTIRReceiverEx
  - HiTechnicAPI, 92
  - NXCDefs.h, 1238
- ReadSensorHTIRSeeker
  - HiTechnicAPI, 92
  - NXCDefs.h, 1238
- ReadSensorHTIRSeeker2AC

- HiTechnicAPI, 93
- NXCDefs.h, 1239
- ReadSensorHTIRSeeker2DC
  - HiTechnicAPI, 93
  - NXCDefs.h, 1239
- ReadSensorHTNormalizedColor
  - HiTechnicAPI, 94
  - NXCDefs.h, 1240
- ReadSensorHTNormalizedColor2Active
  - HiTechnicAPI, 94
  - NXCDefs.h, 1240
- ReadSensorHTRawColor
  - HiTechnicAPI, 95
  - NXCDefs.h, 1241
- ReadSensorHTRawColor2
  - HiTechnicAPI, 95
  - NXCDefs.h, 1241
- ReadSensorHTTouchMultiplexer
  - HiTechnicAPI, 95
  - NXCDefs.h, 1241
- ReadSensorMSAccel
  - MindSensorsAPI, 162
  - NXCDefs.h, 1242
- ReadSensorMSAccelEx
  - MindSensorsAPI, 163
  - NXCDefs.h, 1242
- ReadSensorMSPlayStation
  - MindSensorsAPI, 163
  - NXCDefs.h, 1243
- ReadSensorMSPlayStationEx
  - MindSensorsAPI, 163
  - NXCDefs.h, 1243
- ReadSensorMSRTClock
  - MindSensorsAPI, 164
  - NXCDefs.h, 1244
- ReadSensorMSTilt
  - MindSensorsAPI, 164
  - NXCDefs.h, 1244
- ReadSensorMSTiltEx
  - MindSensorsAPI, 165
  - NXCDefs.h, 1245
- ReadSensorUSEx
  - LowSpeedModuleFunctions, 291
  - NXCDefs.h, 1245
- RebootInFirmwareMode
  - IOCtrlModuleFunctions, 302
  - NXCDefs.h, 1245
- ReceiveMessage
  - CommModuleFunctions, 339
  - NXCDefs.h, 1246
- ReceiveRemoteBool
  - CommModuleFunctions, 339
  - NXCDefs.h, 1246
- ReceiveRemoteMessageEx
  - CommModuleFunctions, 339
  - NXCDefs.h, 1246
- ReceiveRemoteNumber
  - CommModuleFunctions, 340
  - NXCDefs.h, 1247
- ReceiveRemoteString
  - CommModuleFunctions, 340
  - NXCDefs.h, 1247
- RechargeableBattery
  - NXCDefs.h, 1247
  - UiModuleFunctions, 376
- RectOut
  - DisplayModuleFunctions, 260
  - NXCDefs.h, 1248
- RegDValue
  - NBCCCommon.h, 943
  - OutputFieldConstants, 571
- RegIValue
  - NBCCCommon.h, 943
  - OutputFieldConstants, 571
- RegMode
  - NBCCCommon.h, 943
  - OutputFieldConstants, 571
- RegPValue
  - NBCCCommon.h, 943
  - OutputFieldConstants, 571
- reladdr
  - cstringAPI, 451
  - NXCDefs.h, 1248
- Release
  - CommandModuleFunctions, 307
  - NXCDefs.h, 1249
- rem
  - div\_t, 745
  - ldiv\_t, 789
- Remote control (direct commands) errors, 493
- RemoteKeepAlive
  - CommModuleFunctions, 341
  - NXCDefs.h, 1249
- RemoteMessageRead
  - CommModuleFunctions, 341
  - NXCDefs.h, 1249
- RemoteMessageWrite
  - CommModuleFunctions, 341
  - NXCDefs.h, 1249
- RemotePlaySoundFile
  - CommModuleFunctions, 342
  - NXCDefs.h, 1250
- RemotePlayTone
  - CommModuleFunctions, 342
  - NXCDefs.h, 1250
- RemoteResetMotorPosition
  - CommModuleFunctions, 342
  - NXCDefs.h, 1251



- RemoteResetScaledValue
  - CommModuleFunctions, [343](#)
  - NXCDefs.h, [1251](#)
- RemoteSetInputMode
  - CommModuleFunctions, [343](#)
  - NXCDefs.h, [1251](#)
- RemoteSetOutputState
  - CommModuleFunctions, [343](#)
  - NXCDefs.h, [1252](#)
- RemoteStartProgram
  - CommModuleFunctions, [344](#)
  - NXCDefs.h, [1252](#)
- RemoteStopProgram
  - CommModuleFunctions, [344](#)
  - NXCDefs.h, [1252](#)
- RemoteStopSound
  - CommModuleFunctions, [345](#)
  - NXCDefs.h, [1253](#)
- Remove
  - MessageReadType, [795](#)
- remove
  - cstdioAPI, [432](#)
  - NXCDefs.h, [1253](#)
- rename
  - cstdioAPI, [433](#)
  - NXCDefs.h, [1253](#)
- RenameFile
  - LoaderModuleFunctions, [391](#)
  - NXCDefs.h, [1254](#)
- Request
  - ReadSemDataType, [801](#)
  - WriteSemDataType, [814](#)
- Reset
  - ReadButtonType, [799](#)
- RESET\_ALL
  - NBCCCommon.h, [943](#)
  - TachoResetConstants, [563](#)
- RESET\_BLOCK\_COUNT
  - NBCCCommon.h, [944](#)
  - TachoResetConstants, [563](#)
- RESET\_BLOCKANDTACHO
  - NBCCCommon.h, [944](#)
  - TachoResetConstants, [563](#)
- RESET\_COUNT
  - NBCCCommon.h, [944](#)
  - TachoResetConstants, [563](#)
- RESET\_NONE
  - NBCCCommon.h, [944](#)
  - TachoResetConstants, [563](#)
- RESET\_ROTATION\_COUNT
  - NBCCCommon.h, [944](#)
  - TachoResetConstants, [563](#)
- ResetAllTachoCounts
  - NXCDefs.h, [1254](#)
- OutputModuleFunctions, [242](#)
- ResetBlockTachoCount
  - NXCDefs.h, [1254](#)
- OutputModuleFunctions, [242](#)
- ResetRotationCount
  - NXCDefs.h, [1254](#)
- OutputModuleFunctions, [242](#)
- ResetScreen
  - DisplayModuleFunctions, [260](#)
  - NXCDefs.h, [1255](#)
- ResetSensor
  - InputModuleFunctions, [213](#)
  - NXCDefs.h, [1255](#)
- ResetSleepTimer
  - CommandModuleFunctions, [307](#)
  - NXCDefs.h, [1255](#)
- ResetTachoCount
  - NXCDefs.h, [1255](#)
- OutputModuleFunctions, [243](#)
- ResizeFile
  - LoaderModuleFunctions, [391](#)
  - NXCDefs.h, [1256](#)
- ResolveHandle
  - LoaderModuleFunctions, [392](#)
  - NXCDefs.h, [1256](#)
- Result
  - ColorSensorReadType, [720](#)
  - CommBTCheckStatusType, [721](#)
  - CommBTConnectionType, [722](#)
  - CommBTOnOffType, [724](#)
  - CommBTWriteType, [725](#)
  - CommExecuteFunctionType, [727](#)
  - CommHSControlType, [729](#)
  - CommLSCheckStatusType, [732](#)
  - CommLSReadType, [733](#)
  - CommLSWriteExType, [735](#)
  - CommLSWriteType, [737](#)
  - ComputeCalibValueType, [739](#)
  - DatalogWriteType, [742](#)
  - DrawCircleType, [746](#)
  - DrawEllipseType, [748](#)
  - DrawFontType, [750](#)
  - DrawGraphicArrayType, [753](#)
  - DrawGraphicType, [755](#)
  - DrawLineType, [756](#)
  - DrawPointType, [758](#)
  - DrawPolygonType, [759](#)
  - DrawRectType, [760](#)
  - DrawTextType, [762](#)
  - FileCloseType, [764](#)
  - FileDeleteType, [765](#)
  - FileFindType, [766](#)
  - FileOpenType, [769](#)
  - FileReadWriteType, [770](#)

- FileRenameType, [772](#)
- FileResizeType, [774](#)
- FileResolveHandleType, [775](#)
- FileSeekType, [777](#)
- GetStartTickType, [779](#)
- IOMapReadByIDType, [781](#)
- IOMapReadType, [783](#)
- IOMapWriteByIDType, [784](#)
- IOMapWriteType, [786](#)
- KeepAliveType, [788](#)
- ListFilesType, [790](#)
- LoaderExecuteFunctionType, [793](#)
- MessageReadType, [795](#)
- MessageWriteType, [797](#)
- RandomNumberType, [798](#)
- ReadButtonType, [800](#)
- SetScreenModeType, [802](#)
- SetSleepTimeoutType, [803](#)
- SoundPlayFileType, [806](#)
- SoundPlayToneType, [808](#)
- SoundSetStateType, [810](#)
- UpdateCalibCacheInfoType, [813](#)
- ReturnLen
  - CommLSWriteExType, [736](#)
  - CommLSWriteType, [737](#)
- RetVal
  - CommExecuteFunctionType, [727](#)
- rewind
  - cstdioAPI, [433](#)
  - NXCDefs.h, [1256](#)
- RIC Macro Wrappers, [170](#)
- RICArg
  - NBCCCommon.h, [944](#)
  - RICMacros, [171](#)
- RICImgPoint
  - NBCCCommon.h, [944](#)
  - RICMacros, [172](#)
- RICImgRect
  - NBCCCommon.h, [945](#)
  - RICMacros, [172](#)
- RICMacros
  - RICArg, [171](#)
  - RICImgPoint, [172](#)
  - RICImgRect, [172](#)
  - RICMapArg, [172](#)
  - RICMapElement, [172](#)
  - RICMapFunction, [172](#)
  - RICOpCircle, [173](#)
  - RICOpCopyBits, [173](#)
  - RICOpDescription, [173](#)
  - RICOpEllipse, [173](#)
  - RICOpLine, [174](#)
  - RICOpNumBox, [174](#)
  - RICOpPixel, [174](#)
- RICOpPolygon, [174](#)
- RICOpRect, [175](#)
- RICOpSprite, [175](#)
- RICOpVarMap, [175](#)
- RICPolygonPoints, [176](#)
- RICSetValue, [176](#)
- RICSpriteData, [176](#)
- RICMapArg
  - NBCCCommon.h, [945](#)
  - RICMacros, [172](#)
- RICMapElement
  - NBCCCommon.h, [945](#)
  - RICMacros, [172](#)
- RICMapFunction
  - NBCCCommon.h, [945](#)
  - RICMacros, [172](#)
- RICOpCircle
  - NBCCCommon.h, [945](#)
  - RICMacros, [173](#)
- RICOpCopyBits
  - NBCCCommon.h, [946](#)
  - RICMacros, [173](#)
- RICOpDescription
  - NBCCCommon.h, [946](#)
  - RICMacros, [173](#)
- RICOpEllipse
  - NBCCCommon.h, [946](#)
  - RICMacros, [173](#)
- RICOpLine
  - NBCCCommon.h, [947](#)
  - RICMacros, [174](#)
- RICOpNumBox
  - NBCCCommon.h, [947](#)
  - RICMacros, [174](#)
- RICOpPixel
  - NBCCCommon.h, [947](#)
  - RICMacros, [174](#)
- RICOpPolygon
  - NBCCCommon.h, [947](#)
  - RICMacros, [174](#)
- RICOpRect
  - NBCCCommon.h, [947](#)
  - RICMacros, [175](#)
- RICOpSprite
  - NBCCCommon.h, [948](#)
  - RICMacros, [175](#)
- RICOpVarMap
  - NBCCCommon.h, [948](#)
  - RICMacros, [175](#)
- RICPolygonPoints
  - NBCCCommon.h, [948](#)
  - RICMacros, [176](#)
- RICSetValue
  - NXCDefs.h, [1053](#)



- RICMacros, [176](#)
- RICSpriteData
  - NBCCCommon.h, [949](#)
  - RICMacros, [176](#)
- RightStr
  - cstringAPI, [452](#)
  - NXCDefs.h, [1257](#)
- ROTATE\_QUEUE
  - CommandVMState, [487](#)
  - NBCCCommon.h, [949](#)
- RotateMotor
  - NXCDefs.h, [1257](#)
  - OutputModuleFunctions, [243](#)
- RotateMotorEx
  - NXCDefs.h, [1257](#)
  - OutputModuleFunctions, [243](#)
- RotateMotorExPID
  - NXCDefs.h, [1258](#)
  - OutputModuleFunctions, [244](#)
- RotateMotorPID
  - NXCDefs.h, [1258](#)
  - OutputModuleFunctions, [244](#)
- RotationCount
  - NBCCCommon.h, [949](#)
  - OutputFieldConstants, [572](#)
- RS485Control
  - CommModuleFunctions, [345](#)
  - NXCDefs.h, [1259](#)
- RS485DataAvailable
  - CommModuleFunctions, [345](#)
  - NXCDefs.h, [1259](#)
- RS485Exit
  - CommModuleFunctions, [346](#)
  - NXCDefs.h, [1260](#)
- RS485Init
  - CommModuleFunctions, [346](#)
  - NXCDefs.h, [1260](#)
- RS485Read
  - CommModuleFunctions, [346](#)
  - NXCDefs.h, [1260](#)
- RS485SendingData
  - CommModuleFunctions, [347](#)
  - NXCDefs.h, [1261](#)
- RS485Status
  - CommModuleFunctions, [347](#)
  - NXCDefs.h, [1261](#)
- RS485Uart
  - CommModuleFunctions, [347](#)
  - NXCDefs.h, [1261](#)
- RS485Write
  - CommModuleFunctions, [348](#)
  - NXCDefs.h, [1262](#)
- RunNRLinkMacro
  - MindSensorsAPI, [165](#)
  - NXCDefs.h, [1262](#)
- RunNRLinkMacroEx
  - MindSensorsAPI, [166](#)
  - NXCDefs.h, [1262](#)
- RunState
  - NBCCCommon.h, [949](#)
  - OutputFieldConstants, [572](#)
- S1
  - InPorts, [193](#)
  - NXCDefs.h, [1053](#)
- s16
  - NXCDefs.h, [1055](#)
  - TypeAliases, [192](#)
- S2
  - InPorts, [194](#)
  - NXCDefs.h, [1055](#)
- S3
  - InPorts, [194](#)
  - NXCDefs.h, [1055](#)
- s32
  - NXCDefs.h, [1055](#)
  - TypeAliases, [192](#)
- S4
  - InPorts, [195](#)
  - NXCDefs.h, [1055](#)
- s8
  - NXCDefs.h, [1055](#)
  - TypeAliases, [192](#)
- SAMPLERATE\_DEFAULT
  - NBCCCommon.h, [949](#)
  - SoundMisc, [516](#)
- SAMPLERATE\_MAX
  - NBCCCommon.h, [949](#)
  - SoundMisc, [516](#)
- SAMPLERATE\_MIN
  - NBCCCommon.h, [949](#)
  - SoundMisc, [516](#)
- ScaledArray
  - ColorSensorReadType, [720](#)
- ScaledValue
  - InputFieldConstants, [549](#)
  - NBCCCommon.h, [949](#)
- SCHAR\_MAX
  - NBCCCommon.h, [950](#)
  - NXTLimits, [712](#)
- SCHAR\_MIN
  - NBCCCommon.h, [950](#)
  - NXTLimits, [712](#)
- Scout constants, [652](#)
- Scout light constants, [653](#)
- Scout light rule constants, [662](#)
- Scout mode constants, [658](#)
- Scout motion rule constants, [659](#)

- Scout sound constants, 654
- Scout sound set constants, 657
- Scout special effect constants, 664
- Scout touch rule constants, 661
- Scout transmit rule constants, 663
- SCOUT\_FXR\_ALARM
  - NBCCCommon.h, 950
  - ScoutSpecialEffectConstants, 664
- SCOUT\_FXR\_BUG
  - NBCCCommon.h, 950
  - ScoutSpecialEffectConstants, 664
- SCOUT\_FXR\_NONE
  - NBCCCommon.h, 950
  - ScoutSpecialEffectConstants, 664
- SCOUT\_FXR\_RANDOM
  - NBCCCommon.h, 950
  - ScoutSpecialEffectConstants, 664
- SCOUT\_FXR\_SCIENCE
  - NBCCCommon.h, 950
  - ScoutSpecialEffectConstants, 664
- SCOUT\_LIGHT\_OFF
  - NBCCCommon.h, 950
  - ScoutLightConstants, 653
- SCOUT\_LIGHT\_ON
  - NBCCCommon.h, 950
  - ScoutLightConstants, 653
- SCOUT\_LR\_AVOID
  - NBCCCommon.h, 950
  - ScoutLightRuleConstants, 662
- SCOUT\_LR\_IGNORE
  - NBCCCommon.h, 951
  - ScoutLightRuleConstants, 662
- SCOUT\_LR\_OFF\_WHEN
  - NBCCCommon.h, 951
  - ScoutLightRuleConstants, 662
- SCOUT\_LR\_SEEK\_DARK
  - NBCCCommon.h, 951
  - ScoutLightRuleConstants, 662
- SCOUT\_LR\_SEEK\_LIGHT
  - NBCCCommon.h, 951
  - ScoutLightRuleConstants, 662
- SCOUT\_LR\_WAIT\_FOR
  - NBCCCommon.h, 951
  - ScoutLightRuleConstants, 662
- SCOUT\_MODE\_POWER
  - NBCCCommon.h, 951
  - ScoutModeConstants, 658
- SCOUT\_MODE\_STANDALONE
  - NBCCCommon.h, 951
  - ScoutModeConstants, 658
- SCOUT\_MR\_CIRCLE\_LEFT
  - NBCCCommon.h, 951
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_CIRCLE\_RIGHT
  - NBCCCommon.h, 951
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_FORWARD
  - NBCCCommon.h, 951
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_LOOP\_A
  - NBCCCommon.h, 952
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_LOOP\_AB
  - NBCCCommon.h, 952
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_LOOP\_B
  - NBCCCommon.h, 952
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_NO\_MOTION
  - NBCCCommon.h, 952
  - ScoutMotionRuleConstants, 659
- SCOUT\_MR\_ZIGZAG
  - NBCCCommon.h, 952
  - ScoutMotionRuleConstants, 660
- SCOUT\_SNDSET\_ALARM
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SNDSET\_BASIC
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SNDSET\_BUG
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SNDSET\_NONE
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SNDSET\_RANDOM
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SNDSET\_SCIENCE
  - NBCCCommon.h, 952
  - ScoutSndSetConstants, 657
- SCOUT\_SOUND\_1\_BLINK
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 654
- SCOUT\_SOUND\_2\_BLINK
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 654
- SCOUT\_SOUND\_COUNTER1
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 654
- SCOUT\_SOUND\_COUNTER2
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 654
- SCOUT\_SOUND\_ENTER\_BRIGHT
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_ENTER\_DARK

- NBCCCommon.h, 953
- ScoutSoundConstants, 655
- SCOUT\_SOUND\_ENTER\_NORMAL
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_ENTERSA
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_KEYERROR
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_MAIL\_RECEIVED
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_NONE
  - NBCCCommon.h, 953
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_REMOTE
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_SPECIAL1
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_SPECIAL2
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_SPECIAL3
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 655
- SCOUT\_SOUND\_TIMER1
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TIMER2
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TIMER3
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TOUCH1\_PRE
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TOUCH1\_REL
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TOUCH2\_PRE
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_SOUND\_TOUCH2\_REL
  - NBCCCommon.h, 954
  - ScoutSoundConstants, 656
- SCOUT\_TGS\_LONG
  - NBCCCommon.h, 955
  - ScoutTransmitRuleConstants, 663
- SCOUT\_TGS\_MEDIUM
  - NBCCCommon.h, 955
  - ScoutTransmitRuleConstants, 663
- SCOUT\_TGS\_SHORT
  - NBCCCommon.h, 955
  - ScoutTransmitRuleConstants, 663
- SCOUT\_TR\_AVOID
  - NBCCCommon.h, 955
  - ScoutTouchRuleConstants, 661
- SCOUT\_TR\_IGNORE
  - NBCCCommon.h, 955
  - ScoutTouchRuleConstants, 661
- SCOUT\_TR\_OFF\_WHEN
  - NBCCCommon.h, 955
  - ScoutTouchRuleConstants, 661
- SCOUT\_TR\_REVERSE
  - NBCCCommon.h, 955
  - ScoutTouchRuleConstants, 661
- SCOUT\_TR\_WAIT\_FOR
  - NBCCCommon.h, 955
  - ScoutTouchRuleConstants, 661
- ScoutLightConstants
  - SCOUT\_LIGHT\_OFF, 653
  - SCOUT\_LIGHT\_ON, 653
- ScoutLightRuleConstants
  - SCOUT\_LR\_AVOID, 662
  - SCOUT\_LR\_IGNORE, 662
  - SCOUT\_LR\_OFF\_WHEN, 662
  - SCOUT\_LR\_SEEK\_DARK, 662
  - SCOUT\_LR\_SEEK\_LIGHT, 662
  - SCOUT\_LR\_WAIT\_FOR, 662
- ScoutModeConstants
  - SCOUT\_MODE\_POWER, 658
  - SCOUT\_MODE\_STANDALONE, 658
- ScoutMotionRuleConstants
  - SCOUT\_MR\_CIRCLE\_LEFT, 659
  - SCOUT\_MR\_CIRCLE\_RIGHT, 659
  - SCOUT\_MR\_FORWARD, 659
  - SCOUT\_MR\_LOOP\_A, 659
  - SCOUT\_MR\_LOOP\_AB, 659
  - SCOUT\_MR\_LOOP\_B, 659
  - SCOUT\_MR\_NO\_MOTION, 659
  - SCOUT\_MR\_ZIGZAG, 660
- ScoutSndSetConstants
  - SCOUT\_SNDSET\_ALARM, 657
  - SCOUT\_SNDSET\_BASIC, 657
  - SCOUT\_SNDSET\_BUG, 657
  - SCOUT\_SNDSET\_NONE, 657
  - SCOUT\_SNDSET\_RANDOM, 657
  - SCOUT\_SNDSET\_SCIENCE, 657
- ScoutSoundConstants
  - SCOUT\_SOUND\_1\_BLINK, 654
  - SCOUT\_SOUND\_2\_BLINK, 654
  - SCOUT\_SOUND\_COUNTER1, 654
  - SCOUT\_SOUND\_COUNTER2, 654

- SCOUT\_SOUND\_ENTER\_BRIGHT, [655](#)
- SCOUT\_SOUND\_ENTER\_DARK, [655](#)
- SCOUT\_SOUND\_ENTER\_NORMAL, [655](#)
- SCOUT\_SOUND\_ENTERSA, [655](#)
- SCOUT\_SOUND\_KEYERROR, [655](#)
- SCOUT\_SOUND\_MAIL\_RECEIVED, [655](#)
- SCOUT\_SOUND\_NONE, [655](#)
- SCOUT\_SOUND\_REMOTE, [655](#)
- SCOUT\_SOUND\_SPECIAL1, [655](#)
- SCOUT\_SOUND\_SPECIAL2, [655](#)
- SCOUT\_SOUND\_SPECIAL3, [655](#)
- SCOUT\_SOUND\_TIMER1, [656](#)
- SCOUT\_SOUND\_TIMER2, [656](#)
- SCOUT\_SOUND\_TIMER3, [656](#)
- SCOUT\_SOUND\_TOUCH1\_PRES, [656](#)
- SCOUT\_SOUND\_TOUCH1\_REL, [656](#)
- SCOUT\_SOUND\_TOUCH2\_PRES, [656](#)
- SCOUT\_SOUND\_TOUCH2\_REL, [656](#)
- ScoutSpecialEffectConstants
  - SCOUT\_FXR\_ALARM, [664](#)
  - SCOUT\_FXR\_BUG, [664](#)
  - SCOUT\_FXR\_NONE, [664](#)
  - SCOUT\_FXR\_RANDOM, [664](#)
  - SCOUT\_FXR\_SCIENCE, [664](#)
- ScoutTouchRuleConstants
  - SCOUT\_TR\_AVOID, [661](#)
  - SCOUT\_TR\_IGNORE, [661](#)
  - SCOUT\_TR\_OFF\_WHEN, [661](#)
  - SCOUT\_TR\_REVERSE, [661](#)
  - SCOUT\_TR\_WAIT\_FOR, [661](#)
- ScoutTransmitRuleConstants
  - SCOUT\_TGS\_LONG, [663](#)
  - SCOUT\_TGS\_MEDIUM, [663](#)
  - SCOUT\_TGS\_SHORT, [663](#)
- SCREEN\_BACKGROUND
  - DisplayModuleConstants, [595](#)
  - NBCCCommon.h, [955](#)
- SCREEN\_LARGE
  - DisplayModuleConstants, [596](#)
  - NBCCCommon.h, [955](#)
- SCREEN\_MODE\_CLEAR
  - DisplayModuleConstants, [596](#)
  - NBCCCommon.h, [956](#)
- SCREEN\_MODE\_RESTORE
  - DisplayModuleConstants, [596](#)
  - NBCCCommon.h, [956](#)
- SCREEN\_SMALL
  - DisplayModuleConstants, [596](#)
  - NBCCCommon.h, [956](#)
- ScreenMode
  - SetScreenModeType, [802](#)
- SCREENS
  - DisplayModuleConstants, [596](#)
  - NBCCCommon.h, [956](#)
- SEC\_1
  - NBCCCommon.h, [956](#)
  - TimeConstants, [482](#)
- SEC\_10
  - NBCCCommon.h, [956](#)
  - TimeConstants, [482](#)
- SEC\_15
  - NBCCCommon.h, [956](#)
  - TimeConstants, [482](#)
- SEC\_2
  - NBCCCommon.h, [957](#)
  - TimeConstants, [482](#)
- SEC\_20
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_3
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_30
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_4
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_5
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_6
  - NBCCCommon.h, [957](#)
  - TimeConstants, [483](#)
- SEC\_7
  - NBCCCommon.h, [958](#)
  - TimeConstants, [483](#)
- SEC\_8
  - NBCCCommon.h, [958](#)
  - TimeConstants, [484](#)
- SEC\_9
  - NBCCCommon.h, [958](#)
  - TimeConstants, [484](#)
- SEEK\_CUR
  - fseekConstants, [435](#)
  - NXCDefs.h, [1055](#)
- SEEK\_END
  - fseekConstants, [435](#)
  - NXCDefs.h, [1055](#)
- SEEK\_SET
  - fseekConstants, [435](#)
  - NXCDefs.h, [1055](#)
- SemData
  - ReadSemDataType, [801](#)
  - WriteSemDataType, [814](#)
- SendingData
  - CommHSCheckStatusType, [728](#)
- SendMessage

- CommModuleFunctions, [348](#)
- NXCDefs.h, [1263](#)
- SendRemoteBool
  - CommModuleFunctions, [349](#)
  - NXCDefs.h, [1263](#)
- SendRemoteNumber
  - CommModuleFunctions, [349](#)
  - NXCDefs.h, [1263](#)
- SendRemoteString
  - CommModuleFunctions, [349](#)
  - NXCDefs.h, [1264](#)
- SendResponseBool
  - CommModuleFunctions, [350](#)
  - NXCDefs.h, [1264](#)
- SendResponseNumber
  - CommModuleFunctions, [350](#)
  - NXCDefs.h, [1264](#)
- SendResponseString
  - CommModuleFunctions, [350](#)
  - NXCDefs.h, [1265](#)
- SendRS485Bool
  - CommModuleFunctions, [351](#)
  - NXCDefs.h, [1265](#)
- SendRS485Number
  - CommModuleFunctions, [351](#)
  - NXCDefs.h, [1265](#)
- SendRS485String
  - CommModuleFunctions, [351](#)
  - NXCDefs.h, [1266](#)
- Sensor
  - InputModuleFunctions, [214](#)
  - NXCDefs.h, [1266](#)
- Sensor mode constants, [199](#)
- Sensor type constants, [196](#)
- Sensor types and modes, [48](#)
- SENSOR\_1
  - BasicSensorValues, [225](#)
  - NXCDefs.h, [1056](#)
- SENSOR\_2
  - BasicSensorValues, [225](#)
  - NXCDefs.h, [1056](#)
- SENSOR\_3
  - BasicSensorValues, [225](#)
  - NXCDefs.h, [1056](#)
- SENSOR\_4
  - BasicSensorValues, [225](#)
  - NXCDefs.h, [1056](#)
- SENSOR\_CELSIUS
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_COLORBLUE
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_COLORFULL
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_COLORGREEN
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_COLORNONE
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_COLORRED
  - NXCDefs.h, [1056](#)
  - SensorTypeModes, [202](#)
- SENSOR\_EDGE
  - NXCDefs.h, [1057](#)
  - SensorTypeModes, [202](#)
- SENSOR\_FAHRENHEIT
  - NXCDefs.h, [1057](#)
  - SensorTypeModes, [202](#)
- SENSOR\_LIGHT
  - NXCDefs.h, [1057](#)
  - SensorTypeModes, [202](#)
- SENSOR\_LOWSPEED
  - NXCDefs.h, [1057](#)
  - SensorTypeModes, [203](#)
- SENSOR\_LOWSPEED\_9V
  - NXCDefs.h, [1057](#)
  - SensorTypeModes, [203](#)
- SENSOR\_MODE\_BOOL
  - NXCDefs.h, [1057](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_CELSIUS
  - NXCDefs.h, [1057](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_EDGE
  - NXCDefs.h, [1057](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_FAHRENHEIT
  - NXCDefs.h, [1057](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_PERCENT
  - NXCDefs.h, [1058](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_PULSE
  - NXCDefs.h, [1058](#)
  - SensorModes, [199](#)
- SENSOR\_MODE\_RAW
  - NXCDefs.h, [1058](#)
  - SensorModes, [200](#)
- SENSOR\_MODE\_ROTATION
  - NXCDefs.h, [1058](#)
  - SensorModes, [200](#)
- SENSOR\_NXTLIGHT
  - NXCDefs.h, [1058](#)
  - SensorTypeModes, [203](#)
- SENSOR\_PULSE

- NXCDefs.h, [1058](#)
- SensorTypeModes, [203](#)
- SENSOR\_ROTATION
  - NXCDefs.h, [1058](#)
  - SensorTypeModes, [203](#)
- SENSOR\_SOUND
  - NXCDefs.h, [1058](#)
  - SensorTypeModes, [203](#)
- SENSOR\_TOUCH
  - NXCDefs.h, [1058](#)
  - SensorTypeModes, [203](#)
- SENSOR\_TYPE\_COLORBLUE
  - NXCDefs.h, [1059](#)
  - SensorTypes, [196](#)
- SENSOR\_TYPE\_COLORFULL
  - NXCDefs.h, [1059](#)
  - SensorTypes, [196](#)
- SENSOR\_TYPE\_COLORGREEN
  - NXCDefs.h, [1059](#)
  - SensorTypes, [196](#)
- SENSOR\_TYPE\_COLORNONE
  - NXCDefs.h, [1059](#)
  - SensorTypes, [196](#)
- SENSOR\_TYPE\_COLORRED
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_CUSTOM
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_HIGHSPEED
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_LIGHT
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_LIGHT\_ACTIVE
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_LIGHT\_INACTIVE
  - NXCDefs.h, [1059](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_LOWSPEED
  - NXCDefs.h, [1060](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_LOWSPEED\_9V
  - NXCDefs.h, [1060](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_NONE
  - NXCDefs.h, [1060](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_ROTATION
  - NXCDefs.h, [1060](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_SOUND\_DB
  - NXCDefs.h, [1060](#)
  - SensorTypes, [197](#)
- SENSOR\_TYPE\_SOUND\_DBA
  - NXCDefs.h, [1060](#)
  - SensorTypes, [198](#)
- SENSOR\_TYPE\_TEMPERATURE
  - NXCDefs.h, [1060](#)
  - SensorTypes, [198](#)
- SENSOR\_TYPE\_TOUCH
  - NXCDefs.h, [1060](#)
  - SensorTypes, [198](#)
- SensorBoolean
  - InputModuleFunctions, [214](#)
  - NXCDefs.h, [1266](#)
- SensorDigiPinsDirection
  - InputModuleFunctions, [214](#)
  - NXCDefs.h, [1267](#)
- SensorDigiPinsOutputLevel
  - InputModuleFunctions, [215](#)
  - NXCDefs.h, [1267](#)
- SensorDigiPinsStatus
  - InputModuleFunctions, [215](#)
  - NXCDefs.h, [1267](#)
- SensorHTColorNum
  - HiTechnicAPI, [96](#)
  - NXCDefs.h, [1268](#)
- SensorHTCompass
  - HiTechnicAPI, [96](#)
  - NXCDefs.h, [1268](#)
- SensorHTEOPD
  - HiTechnicAPI, [96](#)
  - NXCDefs.h, [1268](#)
- SensorHTGyro
  - HiTechnicAPI, [97](#)
  - NXCDefs.h, [1269](#)
- SensorHTIRSeeker2ACDir
  - HiTechnicAPI, [97](#)
  - NXCDefs.h, [1269](#)
- SensorHTIRSeeker2Addr
  - HiTechnicAPI, [97](#)
  - NXCDefs.h, [1269](#)
- SensorHTIRSeeker2DCDir
  - HiTechnicAPI, [98](#)
  - NXCDefs.h, [1270](#)
- SensorHTIRSeekerDir
  - HiTechnicAPI, [98](#)
  - NXCDefs.h, [1270](#)
- SensorInvalid
  - InputModuleFunctions, [215](#)
  - NXCDefs.h, [1270](#)
- SensorMode
  - InputModuleFunctions, [216](#)
  - NXCDefs.h, [1271](#)
- SensorModes



- SENSOR\_MODE\_BOOL, 199
- SENSOR\_MODE\_CELSIUS, 199
- SENSOR\_MODE\_EDGE, 199
- SENSOR\_MODE\_FAHRENHEIT, 199
- SENSOR\_MODE\_PERCENT, 199
- SENSOR\_MODE\_PULSE, 199
- SENSOR\_MODE\_RAW, 200
- SENSOR\_MODE\_ROTATION, 200
- SensorMSCompass
  - MindSensorsAPI, 166
  - NXCDefs.h, 1271
- SensorMSCompassEx
  - MindSensorsAPI, 166
  - NXCDefs.h, 1271
- SensorMSDROD
  - MindSensorsAPI, 167
  - NXCDefs.h, 1272
- SensorMSPressure
  - MindSensorsAPI, 167
  - NXCDefs.h, 1272
- SensorMSPressureRaw
  - MindSensorsAPI, 167
  - NXCDefs.h, 1272
- SensorNormalized
  - InputModuleFunctions, 216
  - NXCDefs.h, 1273
- SensorRaw
  - InputModuleFunctions, 216
  - NXCDefs.h, 1273
- SensorScaled
  - InputModuleFunctions, 217
  - NXCDefs.h, 1273
- SensorType
  - InputModuleFunctions, 217
  - NXCDefs.h, 1274
- SensorTypeModes
  - \_SENSOR\_CFG, 202
  - SENSOR\_CELSIUS, 202
  - SENSOR\_COLORBLUE, 202
  - SENSOR\_COLORFULL, 202
  - SENSOR\_COLORGREEN, 202
  - SENSOR\_COLORNONE, 202
  - SENSOR\_COLORRED, 202
  - SENSOR\_EDGE, 202
  - SENSOR\_FAHRENHEIT, 202
  - SENSOR\_LIGHT, 202
  - SENSOR\_LOWSPEED, 203
  - SENSOR\_LOWSPEED\_9V, 203
  - SENSOR\_NXTLIGHT, 203
  - SENSOR\_PULSE, 203
  - SENSOR\_ROTATION, 203
  - SENSOR\_SOUND, 203
  - SENSOR\_TOUCH, 203
- SensorTypes
  - SENSOR\_TYPE\_COLORBLUE, 196
  - SENSOR\_TYPE\_COLORFULL, 196
  - SENSOR\_TYPE\_COLORGREEN, 196
  - SENSOR\_TYPE\_COLORNONE, 196
  - SENSOR\_TYPE\_COLORRED, 197
  - SENSOR\_TYPE\_CUSTOM, 197
  - SENSOR\_TYPE\_HIGHSPEED, 197
  - SENSOR\_TYPE\_LIGHT, 197
  - SENSOR\_TYPE\_LIGHT\_ACTIVE, 197
  - SENSOR\_TYPE\_LIGHT\_INACTIVE, 197
  - SENSOR\_TYPE\_LOWSPEED, 197
  - SENSOR\_TYPE\_LOWSPEED\_9V, 197
  - SENSOR\_TYPE\_NONE, 197
  - SENSOR\_TYPE\_ROTATION, 197
  - SENSOR\_TYPE\_SOUND\_DB, 197
  - SENSOR\_TYPE\_SOUND\_DBA, 198
  - SENSOR\_TYPE\_TEMPERATURE, 198
  - SENSOR\_TYPE\_TOUCH, 198
- SensorUS
  - LowSpeedModuleFunctions, 291
  - NXCDefs.h, 1274
- SensorValue
  - InputModuleFunctions, 217
  - NXCDefs.h, 1274
- SensorValueBool
  - InputModuleFunctions, 218
  - NXCDefs.h, 1275
- SensorValueRaw
  - InputModuleFunctions, 218
  - NXCDefs.h, 1275
- SetAbortFlag
  - NXCDefs.h, 1275
  - UiModuleFunctions, 376
- SetBatteryState
  - NXCDefs.h, 1276
  - UiModuleFunctions, 376
- SetBluetoothState
  - NXCDefs.h, 1276
  - UiModuleFunctions, 376
- SetBTInputBuffer
  - CommModuleFunctions, 352
  - NXCDefs.h, 1276
- SetBTInputBufferInPtr
  - CommModuleFunctions, 352
  - NXCDefs.h, 1276
- SetBTInputBufferOutPtr
  - CommModuleFunctions, 352
  - NXCDefs.h, 1277
- SetBTOutputBuffer
  - CommModuleFunctions, 353
  - NXCDefs.h, 1277
- SetBTOutputBufferInPtr
  - CommModuleFunctions, 353
  - NXCDefs.h, 1277

- SetBTOutputBufferOutPtr
  - CommModuleFunctions, [353](#)
  - NXCDefs.h, [1277](#)
- SetButtonLongPressCount
  - ButtonModuleFunctions, [369](#)
  - NXCDefs.h, [1278](#)
- SetButtonLongReleaseCount
  - ButtonModuleFunctions, [369](#)
  - NXCDefs.h, [1278](#)
- SetButtonPressCount
  - ButtonModuleFunctions, [369](#)
  - NXCDefs.h, [1278](#)
- SetButtonReleaseCount
  - ButtonModuleFunctions, [369](#)
  - NXCDefs.h, [1278](#)
- SetButtonShortReleaseCount
  - ButtonModuleFunctions, [370](#)
  - NXCDefs.h, [1279](#)
- SetButtonState
  - ButtonModuleFunctions, [370](#)
  - NXCDefs.h, [1279](#)
- SetCommandFlags
  - NXCDefs.h, [1279](#)
  - UiModuleFunctions, [377](#)
- SetCustomSensorActiveStatus
  - InputModuleFunctions, [218](#)
  - NXCDefs.h, [1279](#)
- SetCustomSensorPercentFullScale
  - InputModuleFunctions, [219](#)
  - NXCDefs.h, [1280](#)
- SetCustomSensorZeroOffset
  - InputModuleFunctions, [219](#)
  - NXCDefs.h, [1280](#)
- SetDisplayContrast
  - DisplayModuleFunctions, [261](#)
  - NXCDefs.h, [1280](#)
- SetDisplayDisplay
  - DisplayModuleFunctions, [261](#)
  - NXCDefs.h, [1280](#)
- SetDisplayEraseMask
  - DisplayModuleFunctions, [261](#)
  - NXCDefs.h, [1281](#)
- SetDisplayFlags
  - DisplayModuleFunctions, [261](#)
  - NXCDefs.h, [1281](#)
- SetDisplayFont
  - DisplayModuleFunctions, [262](#)
  - NXCDefs.h, [1281](#)
- SetDisplayNormal
  - DisplayModuleFunctions, [262](#)
  - NXCDefs.h, [1281](#)
- SetDisplayPopup
  - DisplayModuleFunctions, [262](#)
  - NXCDefs.h, [1282](#)
- SetDisplayTextLinesCenterFlags
  - DisplayModuleFunctions, [263](#)
  - NXCDefs.h, [1282](#)
- SetDisplayUpdateMask
  - DisplayModuleFunctions, [263](#)
  - NXCDefs.h, [1282](#)
- SetHSFlags
  - CommModuleFunctions, [353](#)
  - NXCDefs.h, [1283](#)
- SetHSInputBuffer
  - CommModuleFunctions, [353](#)
  - NXCDefs.h, [1283](#)
- SetHSInputBufferInPtr
  - CommModuleFunctions, [354](#)
  - NXCDefs.h, [1283](#)
- SetHSInputBufferOutPtr
  - CommModuleFunctions, [354](#)
  - NXCDefs.h, [1283](#)
- SetHSMMode
  - CommModuleFunctions, [354](#)
  - NXCDefs.h, [1284](#)
- SetHSOutputBuffer
  - CommModuleFunctions, [354](#)
  - NXCDefs.h, [1284](#)
- SetHSOutputBufferInPtr
  - CommModuleFunctions, [355](#)
  - NXCDefs.h, [1284](#)
- SetHSOutputBufferOutPtr
  - CommModuleFunctions, [355](#)
  - NXCDefs.h, [1284](#)
- SetHSSpeed
  - CommModuleFunctions, [355](#)
  - NXCDefs.h, [1285](#)
- SetHSSState
  - CommModuleFunctions, [355](#)
  - NXCDefs.h, [1285](#)
- SetHTColor2Mode
  - HiTechnicAPI, [98](#)
  - NXCDefs.h, [1285](#)
- SetHTIRSeeker2Mode
  - HiTechnicAPI, [99](#)
  - NXCDefs.h, [1285](#)
- SetInput
  - InputModuleFunctions, [219](#)
  - NXCDefs.h, [1286](#)
- SetLongAbort
  - NXCDefs.h, [1286](#)
  - UiModuleFunctions, [377](#)
- SetMotorPwnFreq
  - NXCDefs.h, [1286](#)
  - OutputModuleFunctions, [245](#)
- SetOnBrickProgramPointer
  - NXCDefs.h, [1287](#)
  - UiModuleFunctions, [377](#)



- SetOutput
  - NXCDefs.h, [1287](#)
  - OutputModuleFunctions, [245](#)
- SetScreenMode
  - NBCCCommon.h, [958](#)
  - SysCallConstants, [473](#)
- SetScreenModeType, [802](#)
  - Result, [802](#)
  - ScreenMode, [802](#)
- SetSensor
  - InputModuleFunctions, [219](#)
  - NXCDefs.h, [1287](#)
- SetSensorBoolean
  - InputModuleFunctions, [220](#)
  - NXCDefs.h, [1288](#)
- SetSensorColorBlue
  - InputModuleFunctions, [220](#)
  - NXCDefs.h, [1288](#)
- SetSensorColorFull
  - InputModuleFunctions, [220](#)
  - NXCDefs.h, [1288](#)
- SetSensorColorGreen
  - InputModuleFunctions, [221](#)
  - NXCDefs.h, [1288](#)
- SetSensorColorNone
  - InputModuleFunctions, [221](#)
  - NXCDefs.h, [1289](#)
- SetSensorColorRed
  - InputModuleFunctions, [221](#)
  - NXCDefs.h, [1289](#)
- SetSensorDigiPinsDirection
  - InputModuleFunctions, [222](#)
  - NXCDefs.h, [1289](#)
- SetSensorDigiPinsOutputLevel
  - InputModuleFunctions, [222](#)
  - NXCDefs.h, [1290](#)
- SetSensorDigiPinsStatus
  - InputModuleFunctions, [222](#)
  - NXCDefs.h, [1290](#)
- SetSensorHTEOPD
  - HiTechnicAPI, [99](#)
  - NXCDefs.h, [1290](#)
- SetSensorHTGyro
  - HiTechnicAPI, [99](#)
  - NXCDefs.h, [1290](#)
- SetSensorLight
  - InputModuleFunctions, [222](#)
  - NXCDefs.h, [1291](#)
- SetSensorLowspeed
  - InputModuleFunctions, [223](#)
  - NXCDefs.h, [1291](#)
- SetSensorMode
  - InputModuleFunctions, [223](#)
  - NXCDefs.h, [1291](#)
- SetSensorMSDROD
  - MindSensorsAPI, [167](#)
  - NXCDefs.h, [1292](#)
- SetSensorMSPressure
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1292](#)
- SetSensorSound
  - InputModuleFunctions, [223](#)
  - NXCDefs.h, [1292](#)
- SetSensorTouch
  - InputModuleFunctions, [224](#)
  - NXCDefs.h, [1292](#)
- SetSensorType
  - InputModuleFunctions, [224](#)
  - NXCDefs.h, [1293](#)
- SetSleepTime
  - NXCDefs.h, [1293](#)
  - UiModuleFunctions, [377](#)
- SetSleepTimeout
  - NXCDefs.h, [1293](#)
  - UiModuleFunctions, [378](#)
- SetSleepTimeoutType, [803](#)
  - Result, [803](#)
  - TheSleepTimeoutMS, [803](#)
- SetSleepTimeoutVal
  - NBCCCommon.h, [958](#)
  - SysCallConstants, [473](#)
- SetSleepTimer
  - NXCDefs.h, [1294](#)
  - UiModuleFunctions, [378](#)
- SetSoundDuration
  - NXCDefs.h, [1294](#)
  - SoundModuleFunctions, [272](#)
- SetSoundFlags
  - NXCDefs.h, [1294](#)
  - SoundModuleFunctions, [272](#)
- SetSoundFrequency
  - NXCDefs.h, [1294](#)
  - SoundModuleFunctions, [273](#)
- SetSoundMode
  - NXCDefs.h, [1295](#)
  - SoundModuleFunctions, [273](#)
- SetSoundModuleState
  - NXCDefs.h, [1295](#)
  - SoundModuleFunctions, [273](#)
- SetSoundSampleRate
  - NXCDefs.h, [1295](#)
  - SoundModuleFunctions, [274](#)
- SetSoundVolume
  - NXCDefs.h, [1296](#)
  - SoundModuleFunctions, [274](#)
- SetUIButton
  - NXCDefs.h, [1296](#)
  - UiModuleFunctions, [378](#)

- SetUIState
  - NXCDefs.h, [1296](#)
  - UiModuleFunctions, [378](#)
- SetUSBInputBuffer
  - CommModuleFunctions, [356](#)
  - NXCDefs.h, [1296](#)
- SetUSBInputBufferInPtr
  - CommModuleFunctions, [356](#)
  - NXCDefs.h, [1297](#)
- SetUSBInputBufferOutPtr
  - CommModuleFunctions, [356](#)
  - NXCDefs.h, [1297](#)
- SetUSBOutputBuffer
  - CommModuleFunctions, [356](#)
  - NXCDefs.h, [1297](#)
- SetUSBOutputBufferInPtr
  - CommModuleFunctions, [357](#)
  - NXCDefs.h, [1297](#)
- SetUSBOutputBufferOutPtr
  - CommModuleFunctions, [357](#)
  - NXCDefs.h, [1298](#)
- SetUSBPollBuffer
  - CommModuleFunctions, [357](#)
  - NXCDefs.h, [1298](#)
- SetUSBPollBufferInPtr
  - CommModuleFunctions, [357](#)
  - NXCDefs.h, [1298](#)
- SetUSBPollBufferOutPtr
  - CommModuleFunctions, [358](#)
  - NXCDefs.h, [1298](#)
- SetUSBState
  - CommModuleFunctions, [358](#)
  - NXCDefs.h, [1299](#)
- SetVMRunState
  - NXCDefs.h, [1299](#)
  - UiModuleFunctions, [379](#)
- SetVolume
  - NXCDefs.h, [1299](#)
  - UiModuleFunctions, [379](#)
- SHRT\_MAX
  - NBCCCommon.h, [958](#)
  - NXTLimits, [712](#)
- SHRT\_MIN
  - NBCCCommon.h, [958](#)
  - NXTLimits, [712](#)
- sign
  - cmathAPI, [422](#)
  - NXCDefs.h, [1299](#)
- Sin
  - cmathAPI, [410](#)
  - NXCDefs.h, [1060](#)
- sin
  - cmathAPI, [422](#)
  - NXCDefs.h, [1300](#)
- SinD
  - cmathAPI, [410](#)
  - NXCDefs.h, [1061](#)
- sind
  - cmathAPI, [422](#)
  - NXCDefs.h, [1300](#)
- Sinh
  - cmathAPI, [411](#)
  - NXCDefs.h, [1061](#)
- sinh
  - cmathAPI, [423](#)
  - NXCDefs.h, [1300](#)
- SinhD
  - cmathAPI, [411](#)
  - NXCDefs.h, [1061](#)
- sinhd
  - cmathAPI, [423](#)
  - NXCDefs.h, [1301](#)
- Size
  - DrawCircleType, [746](#)
  - DrawRectType, [760](#)
- SIZE\_OF\_BDADDR
  - CommMiscConstants, [613](#)
  - NBCCCommon.h, [958](#)
- SIZE\_OF\_BRICK\_NAME
  - CommMiscConstants, [613](#)
  - NBCCCommon.h, [958](#)
- SIZE\_OF\_BT\_CONNECT\_TABLE
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [958](#)
- SIZE\_OF\_BT\_DEVICE\_TABLE
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_BT\_NAME
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_BT\_PINCODE
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_BTBUF
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_CLASS\_OF\_DEVICE
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_HSBUF
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_USBBUF
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)
- SIZE\_OF\_USBDATA
  - CommMiscConstants, [614](#)
  - NBCCCommon.h, [959](#)

- SizeType, 804
  - Height, 804
  - Width, 804
- SizeX
  - DrawEllipseType, 748
- SizeY
  - DrawEllipseType, 749
- SleepNow
  - IOCtrlModuleFunctions, 302
  - NXCDefs.h, 1301
- SleepTime
  - NXCDefs.h, 1301
  - UiModuleFunctions, 379
- SleepTimeout
  - NXCDefs.h, 1302
  - UiModuleFunctions, 379
- SleepTimer
  - NXCDefs.h, 1302
  - UiModuleFunctions, 380
- Sound module, 58
- Sound module constants, 510
- Sound module functions, 269
- Sound module IOMAP offsets, 514
- Sound module miscellaneous constants, 516
- Sound module types, 268
- SOUND\_CLICK
  - NBCCCommon.h, 959
  - RCXSoundConstants, 650
- SOUND\_DOUBLE\_BEEP
  - NBCCCommon.h, 959
  - RCXSoundConstants, 650
- SOUND\_DOWN
  - NBCCCommon.h, 959
  - RCXSoundConstants, 650
- SOUND\_FAST\_UP
  - NBCCCommon.h, 960
  - RCXSoundConstants, 650
- SOUND\_FLAGS\_IDLE
  - NBCCCommon.h, 960
  - SoundFlagsConstants, 511
- SOUND\_FLAGS\_RUNNING
  - NBCCCommon.h, 960
  - SoundFlagsConstants, 511
- SOUND\_FLAGS\_UPDATE
  - NBCCCommon.h, 960
  - SoundFlagsConstants, 511
- SOUND\_LOW\_BEEP
  - NBCCCommon.h, 960
  - RCXSoundConstants, 650
- SOUND\_MODE\_LOOP
  - NBCCCommon.h, 960
  - SoundModeConstants, 513
- SOUND\_MODE\_ONCE
  - NBCCCommon.h, 960
  - SoundModeConstants, 513
- SOUND\_MODE\_TONE
  - NBCCCommon.h, 960
  - SoundModeConstants, 513
- SOUND\_STATE\_FILE
  - NBCCCommon.h, 961
  - SoundStateConstants, 512
- SOUND\_STATE\_IDLE
  - NBCCCommon.h, 961
  - SoundStateConstants, 512
- SOUND\_STATE\_STOP
  - NBCCCommon.h, 961
  - SoundStateConstants, 512
- SOUND\_STATE\_TONE
  - NBCCCommon.h, 961
  - SoundStateConstants, 512
- SOUND\_UP
  - NBCCCommon.h, 961
  - RCXSoundConstants, 651
- SoundDuration
  - NXCDefs.h, 1302
  - SoundModuleFunctions, 274
- SoundFlags
  - NXCDefs.h, 1302
  - SoundModuleFunctions, 274
- SoundFlags constants, 511
- SoundFlagsConstants
  - SOUND\_FLAGS\_IDLE, 511
  - SOUND\_FLAGS\_RUNNING, 511
  - SOUND\_FLAGS\_UPDATE, 511
- SoundFrequency
  - NXCDefs.h, 1303
  - SoundModuleFunctions, 275
- SoundGetState
  - NBCCCommon.h, 961
  - SysCallConstants, 474
- SoundGetStateType, 805
  - Flags, 805
  - State, 805
- SoundIOMAP
  - SoundOffsetDuration, 514
  - SoundOffsetFlags, 514
  - SoundOffsetFreq, 514
  - SoundOffsetMode, 514
  - SoundOffsetSampleRate, 514
  - SoundOffsetSoundFilename, 514
  - SoundOffsetState, 515
  - SoundOffsetVolume, 515
- SoundLevel
  - SoundPlayFileType, 806
  - SoundPlayToneType, 809
- SoundMisc
  - FREQUENCY\_MAX, 516
  - FREQUENCY\_MIN, 516

- SAMPLERATE\_DEFAULT, 516
- SAMPLERATE\_MAX, 516
- SAMPLERATE\_MIN, 516
- SoundMode
  - NXCDefs.h, 1303
  - SoundModuleFunctions, 275
- SoundMode constants, 513
- SoundModeConstants
  - SOUND\_MODE\_LOOP, 513
  - SOUND\_MODE\_ONCE, 513
  - SOUND\_MODE\_TONE, 513
- SoundModuleFunctions
  - PlayFile, 270
  - PlayFileEx, 270
  - PlaySound, 271
  - PlayTone, 271
  - PlayToneEx, 271
  - PlayTones, 272
  - SetSoundDuration, 272
  - SetSoundFlags, 272
  - SetSoundFrequency, 273
  - SetSoundMode, 273
  - SetSoundModuleState, 273
  - SetSoundSampleRate, 274
  - SetSoundVolume, 274
  - SoundDuration, 274
  - SoundFlags, 274
  - SoundFrequency, 275
  - SoundMode, 275
  - SoundSampleRate, 275
  - SoundState, 276
  - SoundVolume, 276
  - StopSound, 276
  - SysSoundGetState, 276
  - SysSoundPlayFile, 277
  - SysSoundPlayTone, 277
  - SysSoundSetState, 277
- SoundModuleID
  - ModuleIDConstants, 180
  - NBCCCommon.h, 961
- SoundModuleName
  - ModuleNameConstants, 178
  - NBCCCommon.h, 961
- SoundOffsetDuration
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetFlags
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetFreq
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetMode
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetSampleRate
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetSoundFilename
  - NBCCCommon.h, 962
  - SoundIOMAP, 514
- SoundOffsetState
  - NBCCCommon.h, 962
  - SoundIOMAP, 515
- SoundOffsetVolume
  - NBCCCommon.h, 962
  - SoundIOMAP, 515
- SoundPlayFile
  - NBCCCommon.h, 962
  - SysCallConstants, 474
- SoundPlayFileType, 806
  - Filename, 806
  - Loop, 806
  - Result, 806
  - SoundLevel, 806
- SoundPlayTone
  - NBCCCommon.h, 962
  - SysCallConstants, 474
- SoundPlayToneType, 808
  - Duration, 808
  - Frequency, 808
  - Loop, 808
  - Result, 808
  - SoundLevel, 809
- SoundSampleRate
  - NXCDefs.h, 1303
  - SoundModuleFunctions, 275
- SoundSetState
  - NBCCCommon.h, 963
  - SysCallConstants, 474
- SoundSetStateType, 810
  - Flags, 810
  - Result, 810
  - State, 810
- SoundState
  - NXCDefs.h, 1304
  - SoundModuleFunctions, 276
- SoundState constants, 512
- SoundStateConstants
  - SOUND\_STATE\_FILE, 512
  - SOUND\_STATE\_IDLE, 512
  - SOUND\_STATE\_STOP, 512
  - SOUND\_STATE\_TONE, 512
- SoundVolume
  - NXCDefs.h, 1304
  - SoundModuleFunctions, 276
- SPECIALS
  - DisplayModuleConstants, 596

- NBCCCommon.h, 963
- sprintf
  - cstdioAPI, 433
  - NXCDefs.h, 1304
- Sqrt
  - cmathAPI, 411
  - NXCDefs.h, 1062
- sqrt
  - cmathAPI, 423
  - NXCDefs.h, 1305
- Standard I2C constants, 587
- Standard-C API functions, 184
- StartLoc
  - DrawLineType, 756
- StartTask
  - CommandModuleFunctions, 308
  - NXCDefs.h, 1305
- STAT\_COMM\_PENDING
  - CommandModuleConstants, 53
  - NBCCCommon.h, 963
- STAT\_MSG\_EMPTY\_MAILBOX
  - CommandModuleConstants, 53
  - NBCCCommon.h, 963
- State
  - SoundGetStateType, 805
  - SoundSetStateType, 810
- Status
  - CommHSReadWriteType, 731
  - DisplayExecuteFunctionType, 744
- STATUSICON\_BATTERY
  - DisplayModuleConstants, 596
  - NBCCCommon.h, 963
- STATUSICON\_BLUETOOTH
  - DisplayModuleConstants, 596
  - NBCCCommon.h, 963
- STATUSICON\_USB
  - DisplayModuleConstants, 596
  - NBCCCommon.h, 963
- STATUSICON\_VM
  - DisplayModuleConstants, 596
  - NBCCCommon.h, 963
- STATUSICONS
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 963
- STATUSTEXT
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 963
- STEPICON\_1
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 963
- STEPICON\_2
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- STEPICON\_3
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- STEPICON\_4
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- STEPICON\_5
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- STEPICONS
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- STEPLINE
  - DisplayModuleConstants, 597
  - NBCCCommon.h, 964
- Stop
  - CommandModuleFunctions, 308
  - NXCDefs.h, 1305
- STOP\_REQ
  - CommandVMState, 487
  - NBCCCommon.h, 964
- StopAllTasks
  - CommandModuleFunctions, 308
  - NXCDefs.h, 1305
- StopSound
  - NXCDefs.h, 1306
  - SoundModuleFunctions, 276
- StopTask
  - CommandModuleFunctions, 308
  - NXCDefs.h, 1306
- StrCat
  - cstringAPI, 453
  - NXCDefs.h, 1307
- strcat
  - cstringAPI, 452
  - NXCDefs.h, 1306
- strcmp
  - cstringAPI, 453
  - NXCDefs.h, 1307
- strcpy
  - cstringAPI, 453
  - NXCDefs.h, 1307
- StrIndex
  - cstringAPI, 454
  - NXCDefs.h, 1308
- StrLen
  - cstringAPI, 454
  - NXCDefs.h, 1308
- strlen
  - cstringAPI, 454
  - NXCDefs.h, 1308
- strncat
  - cstringAPI, 455
  - NXCDefs.h, 1309
- strncmp

- cstringAPI, [455](#)
  - NXCDefs.h, [1309](#)
- strncpy
  - cstringAPI, [455](#)
  - NXCDefs.h, [1310](#)
- StrReplace
  - cstringAPI, [456](#)
  - NXCDefs.h, [1310](#)
- StrToByteArray
  - cstringAPI, [456](#)
  - NXCDefs.h, [1310](#)
- strtod
  - cstdlibAPI, [440](#)
  - NXCDefs.h, [1311](#)
- strtol
  - cstdlibAPI, [441](#)
  - NXCDefs.h, [1311](#)
- StrToNum
  - cstringAPI, [456](#)
  - NXCDefs.h, [1312](#)
- strtoul
  - cstdlibAPI, [442](#)
  - NXCDefs.h, [1312](#)
- SubStr
  - cstringAPI, [457](#)
  - NXCDefs.h, [1313](#)
- SyncTick
  - DatalogGetTimesType, [741](#)
- SyncTime
  - DatalogGetTimesType, [741](#)
- SysCall
  - CommandModuleFunctions, [309](#)
  - NXCDefs.h, [1313](#)
- SysCallConstants
  - ColorSensorRead, [468](#)
  - CommBTCheckStatus, [468](#)
  - CommBTConnection, [468](#)
  - CommBTOnOff, [468](#)
  - CommBTRead, [468](#)
  - CommBTWrite, [469](#)
  - CommExecuteFunction, [469](#)
  - CommHSCheckStatus, [469](#)
  - CommHSControl, [469](#)
  - CommHSRead, [469](#)
  - CommHSWrite, [469](#)
  - CommLSCheckStatus, [469](#)
  - CommLSRead, [469](#)
  - CommLSWrite, [469](#)
  - CommLSWriteEx, [469](#)
  - ComputeCalibValue, [469](#)
  - DatalogGetTimes, [470](#)
  - DatalogWrite, [470](#)
  - DisplayExecuteFunction, [470](#)
  - DrawCircle, [470](#)
  - DrawEllipse, [470](#)
  - DrawFont, [470](#)
  - DrawGraphic, [470](#)
  - DrawGraphicArray, [470](#)
  - DrawLine, [470](#)
  - DrawPoint, [470](#)
  - DrawPolygon, [470](#)
  - DrawRect, [471](#)
  - DrawText, [471](#)
  - FileClose, [471](#)
  - FileDelete, [471](#)
  - FileFindFirst, [471](#)
  - FileFindNext, [471](#)
  - FileOpenAppend, [471](#)
  - FileOpenRead, [471](#)
  - FileOpenReadLinear, [471](#)
  - FileOpenWrite, [471](#)
  - FileOpenWriteLinear, [471](#)
  - FileOpenWriteNonLinear, [472](#)
  - FileRead, [472](#)
  - FileRename, [472](#)
  - FileResize, [472](#)
  - FileResolveHandle, [472](#)
  - FileSeek, [472](#)
  - FileWrite, [472](#)
  - GetStartTick, [472](#)
  - IOMapRead, [472](#)
  - IOMapReadByID, [472](#)
  - IOMapWrite, [472](#)
  - IOMapWriteByID, [473](#)
  - KeepAlive, [473](#)
  - ListFiles, [473](#)
  - LoaderExecuteFunction, [473](#)
  - MessageRead, [473](#)
  - MessageWrite, [473](#)
  - RandomNumber, [473](#)
  - ReadButton, [473](#)
  - ReadSemData, [473](#)
  - SetScreenMode, [473](#)
  - SetSleepTimeoutVal, [473](#)
  - SoundGetState, [474](#)
  - SoundPlayFile, [474](#)
  - SoundPlayTone, [474](#)
  - SoundSetState, [474](#)
  - UpdateCalibCacheInfo, [474](#)
  - WriteSemData, [474](#)
- SysColorSensorRead
  - InputModuleFunctions, [224](#)
  - NXCDefs.h, [1314](#)
- SysCommBTCheckStatus
  - CommModuleFunctions, [358](#)
  - NXCDefs.h, [1314](#)
- SysCommBTConnection
  - CommModuleFunctions, [358](#)

- NXCDefs.h, 1314
- SysCommBTOff
  - CommModuleFunctions, 359
  - NXCDefs.h, 1314
- SysCommBTWrite
  - CommModuleFunctions, 359
  - NXCDefs.h, 1315
- SysCommExecuteFunction
  - CommModuleFunctions, 359
  - NXCDefs.h, 1315
- SysCommHSCheckStatus
  - CommModuleFunctions, 360
  - NXCDefs.h, 1315
- SysCommHSControl
  - CommModuleFunctions, 360
  - NXCDefs.h, 1316
- SysCommHSRead
  - CommModuleFunctions, 360
  - NXCDefs.h, 1316
- SysCommHSWrite
  - CommModuleFunctions, 361
  - NXCDefs.h, 1316
- SysCommLSCheckStatus
  - LowSpeedModuleSystemCallFunctions, 299
  - NXCDefs.h, 1317
- SysCommLSRead
  - LowSpeedModuleSystemCallFunctions, 299
  - NXCDefs.h, 1317
- SysCommLSWrite
  - LowSpeedModuleSystemCallFunctions, 299
  - NXCDefs.h, 1317
- SysCommLSWriteEx
  - LowSpeedModuleSystemCallFunctions, 300
  - NXCDefs.h, 1317
- SysComputeCalibValue
  - CommandModuleFunctions, 309
  - NXCDefs.h, 1318
- SysDatalogGetTimes
  - CommandModuleFunctions, 309
  - NXCDefs.h, 1318
- SysDatalogWrite
  - CommandModuleFunctions, 310
  - NXCDefs.h, 1318
- SysDisplayExecuteFunction
  - DisplayModuleFunctions, 263
  - NXCDefs.h, 1319
- SysDrawCircle
  - DisplayModuleFunctions, 263
  - NXCDefs.h, 1319
- SysDrawEllipse
  - DisplayModuleFunctions, 264
  - NXCDefs.h, 1319
- SysDrawFont
  - DisplayModuleFunctions, 264
  - NXCDefs.h, 1320
- SysDrawGraphic
  - DisplayModuleFunctions, 264
  - NXCDefs.h, 1320
- SysDrawGraphicArray
  - DisplayModuleFunctions, 265
  - NXCDefs.h, 1320
- SysDrawLine
  - DisplayModuleFunctions, 265
  - NXCDefs.h, 1321
- SysDrawPoint
  - DisplayModuleFunctions, 265
  - NXCDefs.h, 1321
- SysDrawPolygon
  - DisplayModuleFunctions, 265
  - NXCDefs.h, 1321
- SysDrawRect
  - DisplayModuleFunctions, 266
  - NXCDefs.h, 1321
- SysDrawText
  - DisplayModuleFunctions, 266
  - NXCDefs.h, 1322
- SysFileClose
  - LoaderModuleFunctions, 392
  - NXCDefs.h, 1322
- SysFileDelete
  - LoaderModuleFunctions, 392
  - NXCDefs.h, 1322
- SysFileFindFirst
  - LoaderModuleFunctions, 393
  - NXCDefs.h, 1322
- SysFileFindNext
  - LoaderModuleFunctions, 393
  - NXCDefs.h, 1323
- SysFileOpenAppend
  - LoaderModuleFunctions, 393
  - NXCDefs.h, 1323
- SysFileOpenRead
  - LoaderModuleFunctions, 393
  - NXCDefs.h, 1323
- SysFileOpenReadLinear
  - LoaderModuleFunctions, 394
  - NXCDefs.h, 1324
- SysFileOpenWrite
  - LoaderModuleFunctions, 394
  - NXCDefs.h, 1324
- SysFileOpenWriteLinear
  - LoaderModuleFunctions, 394
  - NXCDefs.h, 1324
- SysFileOpenWriteNonLinear
  - LoaderModuleFunctions, 395
  - NXCDefs.h, 1324
- SysFileRead
  - LoaderModuleFunctions, 395



- NXCDefs.h, [1325](#)
- SysFileRename
  - LoaderModuleFunctions, [395](#)
  - NXCDefs.h, [1325](#)
- SysFileResize
  - LoaderModuleFunctions, [395](#)
  - NXCDefs.h, [1325](#)
- SysFileResolveHandle
  - LoaderModuleFunctions, [396](#)
  - NXCDefs.h, [1326](#)
- SysFileSeek
  - LoaderModuleFunctions, [396](#)
  - NXCDefs.h, [1326](#)
- SysFileWrite
  - LoaderModuleFunctions, [396](#)
  - NXCDefs.h, [1326](#)
- SysGetStartTick
  - CommandModuleFunctions, [310](#)
  - NXCDefs.h, [1326](#)
- SysIOMapRead
  - CommandModuleFunctions, [310](#)
  - NXCDefs.h, [1327](#)
- SysIOMapReadByID
  - CommandModuleFunctions, [311](#)
  - NXCDefs.h, [1327](#)
- SysIOMapWrite
  - CommandModuleFunctions, [311](#)
  - NXCDefs.h, [1327](#)
- SysIOMapWriteByID
  - CommandModuleFunctions, [311](#)
  - NXCDefs.h, [1327](#)
- SysKeepAlive
  - CommandModuleFunctions, [312](#)
  - NXCDefs.h, [1328](#)
- SysListFiles
  - LoaderModuleFunctions, [397](#)
  - NXCDefs.h, [1328](#)
- SysLoaderExecuteFunction
  - LoaderModuleFunctions, [397](#)
  - NXCDefs.h, [1328](#)
- SysMessageRead
  - CommModuleFunctions, [361](#)
  - NXCDefs.h, [1329](#)
- SysMessageWrite
  - CommModuleFunctions, [361](#)
  - NXCDefs.h, [1329](#)
- SysRandomNumber
  - cstdlibAPI, [442](#)
  - NXCDefs.h, [1329](#)
- SysReadButton
  - ButtonModuleFunctions, [370](#)
  - NXCDefs.h, [1329](#)
- SysReadSemData
  - CommandModuleFunctions, [312](#)
- NXCDefs.h, [1330](#)
- SysSetScreenMode
  - DisplayModuleFunctions, [266](#)
  - NXCDefs.h, [1330](#)
- SysSetSleepTimeout
  - NXCDefs.h, [1330](#)
  - UiModuleFunctions, [380](#)
- SysSoundGetState
  - NXCDefs.h, [1331](#)
  - SoundModuleFunctions, [276](#)
- SysSoundPlayFile
  - NXCDefs.h, [1331](#)
  - SoundModuleFunctions, [277](#)
- SysSoundPlayTone
  - NXCDefs.h, [1331](#)
  - SoundModuleFunctions, [277](#)
- SysSoundSetState
  - NXCDefs.h, [1331](#)
  - SoundModuleFunctions, [277](#)
- System Call function constants, [467](#)
- SysUpdateCalibCacheInfo
  - CommandModuleFunctions, [312](#)
  - NXCDefs.h, [1332](#)
- SysWriteSemData
  - CommandModuleFunctions, [312](#)
  - NXCDefs.h, [1332](#)
- TachoCount
  - NBCCCommon.h, [964](#)
  - OutputFieldConstants, [572](#)
- TachoLimit
  - NBCCCommon.h, [964](#)
  - OutputFieldConstants, [572](#)
- Tachometer counter reset flags, [563](#)
- TachoResetConstants
  - RESET\_ALL, [563](#)
  - RESET\_BLOCK\_COUNT, [563](#)
  - RESET\_BLOCKANDTACHO, [563](#)
  - RESET\_COUNT, [563](#)
  - RESET\_NONE, [563](#)
  - RESET\_ROTATION\_COUNT, [563](#)
- Tan
  - cmathAPI, [412](#)
  - NXCDefs.h, [1062](#)
- tan
  - cmathAPI, [424](#)
  - NXCDefs.h, [1332](#)
- TanD
  - cmathAPI, [412](#)
  - NXCDefs.h, [1062](#)
- tand
  - cmathAPI, [424](#)
  - NXCDefs.h, [1333](#)
- Tanh



- cmathAPI, [412](#)
- NXCDefs.h, [1063](#)
- tanh
  - cmathAPI, [424](#)
  - NXCDefs.h, [1333](#)
- TanhD
  - cmathAPI, [413](#)
  - NXCDefs.h, [1063](#)
- tanhD
  - cmathAPI, [425](#)
  - NXCDefs.h, [1333](#)
- TEMP\_FQ\_1
  - NBCCCommon.h, [964](#)
  - TempI2CConstants, [590](#)
- TEMP\_FQ\_2
  - NBCCCommon.h, [964](#)
  - TempI2CConstants, [590](#)
- TEMP\_FQ\_4
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [590](#)
- TEMP\_FQ\_6
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [590](#)
- TEMP\_I2C\_ADDRESS
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [590](#)
- TEMP\_OS\_ONESHOT
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_POL\_HIGH
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_POL\_LOW
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_REG\_CONFIG
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_REG\_TEMP
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_REG\_THIGH
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_REG\_TLOW
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_RES\_10BIT
  - NBCCCommon.h, [965](#)
  - TempI2CConstants, [591](#)
- TEMP\_RES\_11BIT
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [591](#)
- TEMP\_RES\_12BIT
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [591](#)
- TEMP\_RES\_9BIT
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [591](#)
- TEMP\_SD\_CONTINUOUS
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [592](#)
- TEMP\_SD\_SHUTDOWN
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [592](#)
- TEMP\_TM\_COMPARATOR
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [592](#)
- TEMP\_TM\_INTERRUPT
  - NBCCCommon.h, [966](#)
  - TempI2CConstants, [592](#)
- Temperature sensor constants, [590](#)
- TempI2CConstants
  - TEMP\_FQ\_1, [590](#)
  - TEMP\_FQ\_2, [590](#)
  - TEMP\_FQ\_4, [590](#)
  - TEMP\_FQ\_6, [590](#)
  - TEMP\_I2C\_ADDRESS, [590](#)
  - TEMP\_OS\_ONESHOT, [591](#)
  - TEMP\_POL\_HIGH, [591](#)
  - TEMP\_POL\_LOW, [591](#)
  - TEMP\_REG\_CONFIG, [591](#)
  - TEMP\_REG\_TEMP, [591](#)
  - TEMP\_REG\_THIGH, [591](#)
  - TEMP\_REG\_TLOW, [591](#)
  - TEMP\_RES\_10BIT, [591](#)
  - TEMP\_RES\_11BIT, [591](#)
  - TEMP\_RES\_12BIT, [591](#)
  - TEMP\_RES\_9BIT, [591](#)
  - TEMP\_SD\_CONTINUOUS, [592](#)
  - TEMP\_SD\_SHUTDOWN, [592](#)
  - TEMP\_TM\_COMPARATOR, [592](#)
  - TEMP\_TM\_INTERRUPT, [592](#)
- Text
  - DrawFontType, [751](#)
  - DrawTextType, [762](#)
- Text line constants, [607](#)
- TEXTLINE\_1
  - DisplayTextLineConstants, [607](#)
  - NBCCCommon.h, [966](#)
- TEXTLINE\_2
  - DisplayTextLineConstants, [607](#)
  - NBCCCommon.h, [966](#)
- TEXTLINE\_3
  - DisplayTextLineConstants, [607](#)
  - NBCCCommon.h, [966](#)
- TEXTLINE\_4
  - DisplayTextLineConstants, [607](#)

- NBCCCommon.h, 967
- TEXTLINE\_5
  - DisplayTextLineConstants, 607
  - NBCCCommon.h, 967
- TEXTLINE\_6
  - DisplayTextLineConstants, 608
  - NBCCCommon.h, 967
- TEXTLINE\_7
  - DisplayTextLineConstants, 608
  - NBCCCommon.h, 967
- TEXTLINE\_8
  - DisplayTextLineConstants, 608
  - NBCCCommon.h, 967
- TEXTLINES
  - DisplayTextLineConstants, 608
  - NBCCCommon.h, 967
- TextOut
  - DisplayModuleFunctions, 266
  - NXCDefs.h, 1334
- TheSleepTimeoutMS
  - SetSleepTimeoutType, 803
- Third-party NXT devices, 183
- Time constants, 478
- TimeConstants
  - MIN\_1, 479
  - MS\_1, 479
  - MS\_10, 479
  - MS\_100, 479
  - MS\_150, 479
  - MS\_2, 479
  - MS\_20, 479
  - MS\_200, 479
  - MS\_250, 480
  - MS\_3, 480
  - MS\_30, 480
  - MS\_300, 480
  - MS\_350, 480
  - MS\_4, 480
  - MS\_40, 480
  - MS\_400, 480
  - MS\_450, 480
  - MS\_5, 480
  - MS\_50, 481
  - MS\_500, 481
  - MS\_6, 481
  - MS\_60, 481
  - MS\_600, 481
  - MS\_7, 481
  - MS\_70, 481
  - MS\_700, 481
  - MS\_8, 481
  - MS\_80, 482
  - MS\_800, 482
  - MS\_9, 482
  - MS\_90, 482
  - MS\_900, 482
  - SEC\_1, 482
  - SEC\_10, 482
  - SEC\_15, 482
  - SEC\_2, 482
  - SEC\_20, 483
  - SEC\_3, 483
  - SEC\_30, 483
  - SEC\_4, 483
  - SEC\_5, 483
  - SEC\_6, 483
  - SEC\_7, 483
  - SEC\_8, 484
  - SEC\_9, 484
- TIMES\_UP
  - CommandVMState, 487
  - NBCCCommon.h, 967
- tolower
  - ctypeAPI, 463
  - NXCDefs.h, 1334
- Tone, 811
  - Duration, 811
  - Frequency, 811
- Tone constants, 517
- TONE\_A3
  - NBCCCommon.h, 967
  - ToneConstants, 518
- TONE\_A4
  - NBCCCommon.h, 967
  - ToneConstants, 518
- TONE\_A5
  - NBCCCommon.h, 967
  - ToneConstants, 518
- TONE\_A6
  - NBCCCommon.h, 967
  - ToneConstants, 518
- TONE\_A7
  - NBCCCommon.h, 968
  - ToneConstants, 518
- TONE\_AS3
  - NBCCCommon.h, 968
  - ToneConstants, 518
- TONE\_AS4
  - NBCCCommon.h, 968
  - ToneConstants, 518
- TONE\_AS5
  - NBCCCommon.h, 968
  - ToneConstants, 519
- TONE\_AS6
  - NBCCCommon.h, 968
  - ToneConstants, 519
- TONE\_AS7
  - NBCCCommon.h, 968

- ToneConstants, [519](#)
- TONE\_B3
  - NBCCCommon.h, [968](#)
  - ToneConstants, [519](#)
- TONE\_B4
  - NBCCCommon.h, [968](#)
  - ToneConstants, [519](#)
- TONE\_B5
  - NBCCCommon.h, [968](#)
  - ToneConstants, [519](#)
- TONE\_B6
  - NBCCCommon.h, [968](#)
  - ToneConstants, [519](#)
- TONE\_B7
  - NBCCCommon.h, [968](#)
  - ToneConstants, [519](#)
- TONE\_C4
  - NBCCCommon.h, [969](#)
  - ToneConstants, [519](#)
- TONE\_C5
  - NBCCCommon.h, [969](#)
  - ToneConstants, [519](#)
- TONE\_C6
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_C7
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_CS4
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_CS5
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_CS6
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_CS7
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_D4
  - NBCCCommon.h, [969](#)
  - ToneConstants, [520](#)
- TONE\_D5
  - NBCCCommon.h, [970](#)
  - ToneConstants, [520](#)
- TONE\_D6
  - NBCCCommon.h, [970](#)
  - ToneConstants, [520](#)
- TONE\_D7
  - NBCCCommon.h, [970](#)
  - ToneConstants, [520](#)
- TONE\_DS4
  - NBCCCommon.h, [970](#)
- ToneConstants, [521](#)
- TONE\_DS5
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_DS6
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_DS7
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_E4
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_E5
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_E6
  - NBCCCommon.h, [970](#)
  - ToneConstants, [521](#)
- TONE\_E7
  - NBCCCommon.h, [971](#)
  - ToneConstants, [521](#)
- TONE\_F4
  - NBCCCommon.h, [971](#)
  - ToneConstants, [521](#)
- TONE\_F5
  - NBCCCommon.h, [971](#)
  - ToneConstants, [521](#)
- TONE\_F6
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_F7
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_FS4
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_FS5
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_FS6
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_FS7
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_G4
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_G5
  - NBCCCommon.h, [971](#)
  - ToneConstants, [522](#)
- TONE\_G6
  - NBCCCommon.h, [972](#)

- ToneConstants, [522](#)
- TONE\_G7
  - NBCCCommon.h, [972](#)
  - ToneConstants, [522](#)
- TONE\_GS4
  - NBCCCommon.h, [972](#)
  - ToneConstants, [523](#)
- TONE\_GS5
  - NBCCCommon.h, [972](#)
  - ToneConstants, [523](#)
- TONE\_GS6
  - NBCCCommon.h, [972](#)
  - ToneConstants, [523](#)
- TONE\_GS7
  - NBCCCommon.h, [972](#)
  - ToneConstants, [523](#)
- ToneConstants
  - TONE\_A3, [518](#)
  - TONE\_A4, [518](#)
  - TONE\_A5, [518](#)
  - TONE\_A6, [518](#)
  - TONE\_A7, [518](#)
  - TONE\_AS3, [518](#)
  - TONE\_AS4, [518](#)
  - TONE\_AS5, [519](#)
  - TONE\_AS6, [519](#)
  - TONE\_AS7, [519](#)
  - TONE\_B3, [519](#)
  - TONE\_B4, [519](#)
  - TONE\_B5, [519](#)
  - TONE\_B6, [519](#)
  - TONE\_B7, [519](#)
  - TONE\_C4, [519](#)
  - TONE\_C5, [519](#)
  - TONE\_C6, [520](#)
  - TONE\_C7, [520](#)
  - TONE\_CS4, [520](#)
  - TONE\_CS5, [520](#)
  - TONE\_CS6, [520](#)
  - TONE\_CS7, [520](#)
  - TONE\_D4, [520](#)
  - TONE\_D5, [520](#)
  - TONE\_D6, [520](#)
  - TONE\_D7, [520](#)
  - TONE\_DS4, [521](#)
  - TONE\_DS5, [521](#)
  - TONE\_DS6, [521](#)
  - TONE\_DS7, [521](#)
  - TONE\_E4, [521](#)
  - TONE\_E5, [521](#)
  - TONE\_E6, [521](#)
  - TONE\_E7, [521](#)
  - TONE\_F4, [521](#)
  - TONE\_F5, [521](#)
  - TONE\_F6, [522](#)
  - TONE\_F7, [522](#)
  - TONE\_FS4, [522](#)
  - TONE\_FS5, [522](#)
  - TONE\_FS6, [522](#)
  - TONE\_FS7, [522](#)
  - TONE\_G4, [522](#)
  - TONE\_G5, [522](#)
  - TONE\_G6, [522](#)
  - TONE\_G7, [522](#)
  - TONE\_GS4, [523](#)
  - TONE\_GS5, [523](#)
  - TONE\_GS6, [523](#)
  - TONE\_GS7, [523](#)
- TOPLINE
  - DisplayModuleConstants, [597](#)
  - NBCCCommon.h, [972](#)
- toupper
  - ctypeAPI, [463](#)
  - NXCDefs.h, [1335](#)
- TRAIN\_CHANNEL\_1
  - IRTrainChannels, [684](#)
  - NBCCCommon.h, [972](#)
- TRAIN\_CHANNEL\_2
  - IRTrainChannels, [684](#)
  - NBCCCommon.h, [972](#)
- TRAIN\_CHANNEL\_3
  - IRTrainChannels, [684](#)
  - NBCCCommon.h, [972](#)
- TRAIN\_CHANNEL\_ALL
  - IRTrainChannels, [684](#)
  - NBCCCommon.h, [973](#)
- TRAIN\_FUNC\_DECR\_SPEED
  - IRTrainFuncs, [683](#)
  - NBCCCommon.h, [973](#)
- TRAIN\_FUNC\_INCR\_SPEED
  - IRTrainFuncs, [683](#)
  - NBCCCommon.h, [973](#)
- TRAIN\_FUNC\_STOP
  - IRTrainFuncs, [683](#)
  - NBCCCommon.h, [973](#)
- TRAIN\_FUNC\_TOGGLE\_LIGHT
  - IRTrainFuncs, [683](#)
  - NBCCCommon.h, [973](#)
- TRUE
  - MiscConstants, [182](#)
  - NBCCCommon.h, [973](#)
- Trunc
  - cmathAPI, [413](#)
  - NXCDefs.h, [1063](#)
- trunc
  - cmathAPI, [425](#)
  - NXCDefs.h, [1335](#)
- TurnRatio

- NBCCCommon.h, [973](#)
- OutputFieldConstants, [572](#)
- Type
  - InputFieldConstants, [549](#)
  - NBCCCommon.h, [973](#)
- Type aliases, [192](#)
- TypeAliases
  - s16, [192](#)
  - s32, [192](#)
  - s8, [192](#)
  - u16, [192](#)
  - u32, [192](#)
  - u8, [192](#)
- u16
  - NXCDefs.h, [1064](#)
  - TypeAliases, [192](#)
- u32
  - NXCDefs.h, [1064](#)
  - TypeAliases, [192](#)
- u8
  - NXCDefs.h, [1064](#)
  - TypeAliases, [192](#)
- UCHAR\_MAX
  - NBCCCommon.h, [974](#)
  - NXTLimits, [712](#)
- UF\_PENDING\_UPDATES
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_MODE
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_PID\_VALUES
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_RESET\_BLOCK\_COUNT
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_RESET\_COUNT
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_RESET\_ROTATION\_COUNT
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [561](#)
- UF\_UPDATE\_SPEED
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [562](#)
- UF\_UPDATE\_TACHO\_LIMIT
  - NBCCCommon.h, [974](#)
  - OutUFConstants, [562](#)
- Ui module, [59](#)
- Ui module constants, [530](#)
- Ui module functions, [372](#)
- Ui module IOMAP offsets, [540](#)
- Ui module types, [371](#)
- UI\_BT\_CONNECT\_REQUEST
  - NBCCCommon.h, [974](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BT\_ERROR\_ATTENTION
  - NBCCCommon.h, [974](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BT\_PIN\_REQUEST
  - NBCCCommon.h, [975](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BT\_STATE\_CONNECTED
  - NBCCCommon.h, [975](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BT\_STATE\_OFF
  - NBCCCommon.h, [975](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BT\_STATE\_VISIBLE
  - NBCCCommon.h, [975](#)
  - UiBluetoothStateConstants, [537](#)
- UI\_BUTTON\_ENTER
  - NBCCCommon.h, [975](#)
  - UiButtonConstants, [536](#)
- UI\_BUTTON\_EXIT
  - NBCCCommon.h, [975](#)
  - UiButtonConstants, [536](#)
- UI\_BUTTON\_LEFT
  - NBCCCommon.h, [975](#)
  - UiButtonConstants, [536](#)
- UI\_BUTTON\_NONE
  - NBCCCommon.h, [975](#)
  - UiButtonConstants, [536](#)
- UI\_BUTTON\_RIGHT
  - NBCCCommon.h, [975](#)
  - UiButtonConstants, [536](#)
- UI\_FLAGS\_BUSY
  - NBCCCommon.h, [975](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_DISABLE\_EXIT
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ENTER
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_ENABLE\_STATUS\_UPDATE
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_EXECUTE\_LMS\_FILE
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_REDRAW\_STATUS
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [531](#)
- UI\_FLAGS\_RESET\_SLEEP\_TIMER
  - NBCCCommon.h, [976](#)

- UiFlagsConstants, [532](#)
- UI\_FLAGS\_UPDATE
  - NBCCCommon.h, [976](#)
  - UiFlagsConstants, [532](#)
- UI\_STATE\_BT\_ERROR
  - NBCCCommon.h, [976](#)
  - UiStateConstants, [533](#)
- UI\_STATE\_CONNECT\_REQUEST
  - NBCCCommon.h, [976](#)
  - UiStateConstants, [533](#)
- UI\_STATE\_DRAW\_MENU
  - NBCCCommon.h, [976](#)
  - UiStateConstants, [533](#)
- UI\_STATE\_ENTER\_PRESSED
  - NBCCCommon.h, [976](#)
  - UiStateConstants, [533](#)
- UI\_STATE\_EXECUTE\_FILE
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [533](#)
- UI\_STATE\_EXECUTING\_FILE
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_EXIT\_PRESSED
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_INIT\_DISPLAY
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_INIT\_INTRO
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_INIT\_LOW\_BATTERY
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_INIT\_MENU
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_INIT\_WAIT
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_LEFT\_PRESSED
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_LOW\_BATTERY
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_NEXT\_MENU
  - NBCCCommon.h, [977](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_RIGHT\_PRESSED
  - NBCCCommon.h, [978](#)
  - UiStateConstants, [534](#)
- UI\_STATE\_TEST\_BUTTONS
  - NBCCCommon.h, [978](#)
- UiStateConstants, [535](#)
- UI\_VM\_IDLE
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UI\_VM\_RESET1
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UI\_VM\_RESET2
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UI\_VM\_RUN\_FREE
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UI\_VM\_RUN\_PAUSE
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UI\_VM\_RUN\_SINGLE
  - NBCCCommon.h, [978](#)
  - UiVMRunStateConstants, [539](#)
- UiBluetoothStateConstants
  - UI\_BT\_CONNECT\_REQUEST, [537](#)
  - UI\_BT\_ERROR\_ATTENTION, [537](#)
  - UI\_BT\_PIN\_REQUEST, [537](#)
  - UI\_BT\_STATE\_CONNECTED, [537](#)
  - UI\_BT\_STATE\_OFF, [537](#)
  - UI\_BT\_STATE\_VISIBLE, [537](#)
- UIButton
  - NXCDefs.h, [1335](#)
  - UiModuleFunctions, [380](#)
- UIButton constants, [536](#)
- UIButtonConstants
  - UI\_BUTTON\_ENTER, [536](#)
  - UI\_BUTTON\_EXIT, [536](#)
  - UI\_BUTTON\_LEFT, [536](#)
  - UI\_BUTTON\_NONE, [536](#)
  - UI\_BUTTON\_RIGHT, [536](#)
- UiFlagsConstants
  - UI\_FLAGS\_BUSY, [531](#)
  - UI\_FLAGS\_DISABLE\_EXIT, [531](#)
  - UI\_FLAGS\_DISABLE\_LEFT\_RIGHT\_ -  
ENTER, [531](#)
  - UI\_FLAGS\_ENABLE\_STATUS\_UPDATE, [531](#)
  - UI\_FLAGS\_EXECUTE\_LMS\_FILE, [531](#)
  - UI\_FLAGS\_REDRAW\_STATUS, [531](#)
  - UI\_FLAGS\_RESET\_SLEEP\_TIMER, [532](#)
  - UI\_FLAGS\_UPDATE, [532](#)
- UiIOMAP
  - UIOffsetAbortFlag, [540](#)
  - UIOffsetBatteryState, [540](#)
  - UIOffsetBatteryVoltage, [540](#)
  - UIOffsetBluetoothState, [540](#)
  - UIOffsetButton, [540](#)
  - UIOffsetError, [541](#)

- UIOffsetFlags, 541
- UIOffsetForceOff, 541
- UIOffsetLMSfilename, 541
- UIOffsetOBPPointer, 541
- UIOffsetPMenu, 541
- UIOffsetRechargeable, 541
- UIOffsetRunState, 541
- UIOffsetSleepTimeout, 541
- UIOffsetSleepTimer, 541
- UIOffsetState, 541
- UIOffsetUsbState, 542
- UIOffsetVolume, 542
- UiModuleFunctions
  - AbortFlag, 374
  - BatteryLevel, 374
  - BatteryState, 374
  - BluetoothState, 374
  - CommandFlags, 375
  - ForceOff, 375
  - LongAbort, 375
  - OnBrickProgramPointer, 375
  - RechargeableBattery, 376
  - SetAbortFlag, 376
  - SetBatteryState, 376
  - SetBluetoothState, 376
  - SetCommandFlags, 377
  - SetLongAbort, 377
  - SetOnBrickProgramPointer, 377
  - SetSleepTime, 377
  - SetSleepTimeout, 378
  - SetSleepTimer, 378
  - SetUIButton, 378
  - SetUIState, 378
  - SetVMRunState, 379
  - SetVolume, 379
  - SleepTime, 379
  - SleepTimeout, 379
  - SleepTimer, 380
  - SysSetSleepTimeout, 380
  - UIButton, 380
  - UIState, 380
  - UsbState, 381
  - VMRunState, 381
  - Volume, 381
- UiModuleID
  - ModuleIDConstants, 180
  - NBCCCommon.h, 978
- UiModuleName
  - ModuleNameConstants, 178
  - NBCCCommon.h, 978
- UINT\_MAX
  - NBCCCommon.h, 978
  - NXTLimits, 712
- UIOffsetAbortFlag
  - NBCCCommon.h, 979
  - UiIOMAP, 540
- UIOffsetBatteryState
  - NBCCCommon.h, 979
  - UiIOMAP, 540
- UIOffsetBatteryVoltage
  - NBCCCommon.h, 979
  - UiIOMAP, 540
- UIOffsetBluetoothState
  - NBCCCommon.h, 979
  - UiIOMAP, 540
- UIOffsetButton
  - NBCCCommon.h, 979
  - UiIOMAP, 540
- UIOffsetError
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetFlags
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetForceOff
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetLMSfilename
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetOBPPointer
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetPMenu
  - NBCCCommon.h, 979
  - UiIOMAP, 541
- UIOffsetRechargeable
  - NBCCCommon.h, 980
  - UiIOMAP, 541
- UIOffsetRunState
  - NBCCCommon.h, 980
  - UiIOMAP, 541
- UIOffsetSleepTimeout
  - NBCCCommon.h, 980
  - UiIOMAP, 541
- UIOffsetSleepTimer
  - NBCCCommon.h, 980
  - UiIOMAP, 541
- UIOffsetState
  - NBCCCommon.h, 980
  - UiIOMAP, 541
- UIOffsetUsbState
  - NBCCCommon.h, 980
  - UiIOMAP, 542
- UIOffsetVolume
  - NBCCCommon.h, 980
  - UiIOMAP, 542
- UIState



- NXCDefs.h, 1336
- UiModuleFunctions, 380
- UIState constants, 533
- UiStateConstants
  - UI\_STATE\_BT\_ERROR, 533
  - UI\_STATE\_CONNECT\_REQUEST, 533
  - UI\_STATE\_DRAW\_MENU, 533
  - UI\_STATE\_ENTER\_PRESSED, 533
  - UI\_STATE\_EXECUTE\_FILE, 533
  - UI\_STATE\_EXECUTING\_FILE, 534
  - UI\_STATE\_EXIT\_PRESSED, 534
  - UI\_STATE\_INIT\_DISPLAY, 534
  - UI\_STATE\_INIT\_INTRO, 534
  - UI\_STATE\_INIT\_LOW\_BATTERY, 534
  - UI\_STATE\_INIT\_MENU, 534
  - UI\_STATE\_INIT\_WAIT, 534
  - UI\_STATE\_LEFT\_PRESSED, 534
  - UI\_STATE\_LOW\_BATTERY, 534
  - UI\_STATE\_NEXT\_MENU, 534
  - UI\_STATE\_RIGHT\_PRESSED, 534
  - UI\_STATE\_TEST\_BUTTONS, 535
- UiVMRunStateConstants
  - UI\_VM\_IDLE, 539
  - UI\_VM\_RESET1, 539
  - UI\_VM\_RESET2, 539
  - UI\_VM\_RUN\_FREE, 539
  - UI\_VM\_RUN\_PAUSE, 539
  - UI\_VM\_RUN\_SINGLE, 539
- ULONG\_MAX
  - NBCCCommon.h, 980
  - NXTLimits, 712
- Ultrasonic sensor constants, 588
- UnflattenVar
  - cstringAPI, 457
  - NXCDefs.h, 1336
- UpdateCalibCacheInfo
  - NBCCCommon.h, 980
  - SysCallConstants, 474
- UpdateCalibCacheInfoType, 812
  - MaxVal, 812
  - MinVal, 812
  - Name, 812
  - Result, 813
- UpdateFlags
  - NBCCCommon.h, 980
  - OutputFieldConstants, 573
- US\_CMD\_CONTINUOUS
  - NBCCCommon.h, 980
  - USI2CConstants, 588
- US\_CMD\_EVENTCAPTURE
  - NBCCCommon.h, 981
  - USI2CConstants, 588
- US\_CMD\_OFF
  - NBCCCommon.h, 981
  - USI2CConstants, 588
- US\_CMD\_SINGLESHOT
  - NBCCCommon.h, 981
  - USI2CConstants, 588
- US\_CMD\_WARMRESET
  - NBCCCommon.h, 981
  - USI2CConstants, 588
- US\_REG\_ACTUAL\_ZERO
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_CM\_INTERVAL
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_FACTORY\_ACTUAL\_ZERO
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_FACTORY\_SCALE\_DIVISOR
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_FACTORY\_SCALE\_FACTOR
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_MEASUREMENT\_UNITS
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_SCALE\_DIVISOR
  - NBCCCommon.h, 981
  - USI2CConstants, 589
- US\_REG\_SCALE\_FACTOR
  - NBCCCommon.h, 982
  - USI2CConstants, 589
- USB\_CMD\_READY
  - CommStatusCodesConstants, 634
  - NBCCCommon.h, 982
- USB\_PROTOCOL\_OVERHEAD
  - CommMiscConstants, 614
  - NBCCCommon.h, 982
- USBInputBufferInPtr
  - CommModuleFunctions, 361
  - NXCDefs.h, 1336
- USBInputBufferOutPtr
  - CommModuleFunctions, 362
  - NXCDefs.h, 1336
- USBOutputBufferInPtr
  - CommModuleFunctions, 362
  - NXCDefs.h, 1337
- USBOutputBufferOutPtr
  - CommModuleFunctions, 362
  - NXCDefs.h, 1337
- USBPollBufferInPtr
  - CommModuleFunctions, 362
  - NXCDefs.h, 1337
- USBPollBufferOutPtr
  - CommModuleFunctions, 363



- NXCDefs.h, [1337](#)
- USBState
  - CommModuleFunctions, [363](#)
  - NXCDefs.h, [1338](#)
- UsbState
  - NXCDefs.h, [1338](#)
  - UiModuleFunctions, [381](#)
- USHRT\_MAX
  - NBCCCommon.h, [982](#)
  - NXTLimits, [712](#)
- USI2CConstants
  - US\_CMD\_CONTINUOUS, [588](#)
  - US\_CMD\_EVENTCAPTURE, [588](#)
  - US\_CMD\_OFF, [588](#)
  - US\_CMD\_SINGLESOT, [588](#)
  - US\_CMD\_WARMRESET, [588](#)
  - US\_REG\_ACTUAL\_ZERO, [589](#)
  - US\_REG\_CM\_INTERVAL, [589](#)
  - US\_REG\_FACTORY\_ACTUAL\_ZERO, [589](#)
  - US\_REG\_FACTORY\_SCALE\_DIVISOR, [589](#)
  - US\_REG\_FACTORY\_SCALE\_FACTOR, [589](#)
  - US\_REG\_MEASUREMENT\_UNITS, [589](#)
  - US\_REG\_SCALE\_DIVISOR, [589](#)
  - US\_REG\_SCALE\_FACTOR, [589](#)
- Variables
  - DrawGraphicArrayType, [753](#)
  - DrawGraphicType, [755](#)
- VM run state constants, [539](#)
- VM state constants, [487](#)
- VMRunState
  - NXCDefs.h, [1338](#)
  - UiModuleFunctions, [381](#)
- Volume
  - NXCDefs.h, [1338](#)
  - UiModuleFunctions, [381](#)
- Wait
  - CommandModuleFunctions, [313](#)
  - NXCDefs.h, [1338](#)
- Width
  - SizeType, [804](#)
- Write
  - LoaderModuleFunctions, [397](#)
  - NXCDefs.h, [1339](#)
- WriteBytes
  - LoaderModuleFunctions, [397](#)
  - NXCDefs.h, [1339](#)
- WriteBytesEx
  - LoaderModuleFunctions, [398](#)
  - NXCDefs.h, [1340](#)
- WriteHandle
  - FileResolveHandleType, [775](#)
- WriteI2CRegister
  - LowSpeedModuleFunctions, [291](#)
  - NXCDefs.h, [1340](#)
- WriteLn
  - LoaderModuleFunctions, [398](#)
  - NXCDefs.h, [1340](#)
- WriteLnString
  - LoaderModuleFunctions, [399](#)
  - NXCDefs.h, [1341](#)
- WriteNRLinkBytes
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1341](#)
- WriteNRLinkBytesEx
  - MindSensorsAPI, [168](#)
  - NXCDefs.h, [1342](#)
- WriteSemData
  - NBCCCommon.h, [982](#)
  - SysCallConstants, [474](#)
- WriteSemDataType, [814](#)
  - ClearBits, [814](#)
  - NewVal, [814](#)
  - Request, [814](#)
  - SemData, [814](#)
- WriteString
  - LoaderModuleFunctions, [399](#)
  - NXCDefs.h, [1342](#)
- X
  - LocationType, [794](#)
- X1
  - DisplayExecuteFunctionType, [744](#)
- X2
  - DisplayExecuteFunctionType, [744](#)
- Y
  - LocationType, [794](#)
- Y1
  - DisplayExecuteFunctionType, [744](#)
- Y2
  - DisplayExecuteFunctionType, [744](#)
- Yield
  - CommandModuleFunctions, [313](#)
  - NXCDefs.h, [1342](#)