

NXC

Version 1.2.1 r4

Generated by Doxygen 1.6.2

Tue Oct 5 21:42:44 2010

Contents

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

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

6.11	IOCtrl module	53
6.11.1	Detailed Description	54
6.12	Loader module	54
6.12.1	Detailed Description	54
6.13	Sound module	55
6.13.1	Detailed Description	55
6.14	Ui module	56
6.14.1	Detailed Description	56
6.15	Low Speed module	56
6.15.1	Detailed Description	56
6.16	Display module	57
6.16.1	Detailed Description	58
6.17	HiTechnic API Functions	58
6.17.1	Detailed Description	67
6.17.2	Function Documentation	67
6.18	MindSensors API Functions	107
6.18.1	Detailed Description	119
6.18.2	Function Documentation	119
6.19	RIC Macro Wrappers	179
6.19.1	Detailed Description	181
6.19.2	Define Documentation	181
6.20	NXT firmware module names	187
6.20.1	Detailed Description	188
6.20.2	Define Documentation	188
6.21	NXT firmware module IDs	189
6.21.1	Detailed Description	190
6.21.2	Define Documentation	190
6.22	Miscellaneous NBC/NXC constants	191
6.22.1	Detailed Description	192
6.22.2	Define Documentation	192
6.23	Third-party NXT devices	193

6.23.1	Detailed Description	193
6.24	Standard-C API functions	194
6.24.1	Detailed Description	194
6.25	A simple 3D graphics library	194
6.25.1	Detailed Description	196
6.25.2	Function Documentation	196
6.26	Type aliases	203
6.26.1	Detailed Description	203
6.26.2	Define Documentation	203
6.27	Input port constants	204
6.27.1	Detailed Description	204
6.27.2	Define Documentation	204
6.28	Sensor type constants	207
6.28.1	Detailed Description	207
6.28.2	Define Documentation	208
6.29	Sensor mode constants	210
6.29.1	Detailed Description	210
6.29.2	Define Documentation	210
6.30	Combined sensor type and mode constants	211
6.30.1	Detailed Description	212
6.30.2	Define Documentation	213
6.31	Input module types	215
6.31.1	Detailed Description	215
6.32	Input module functions	215
6.32.1	Detailed Description	219
6.32.2	Function Documentation	219
6.33	Basic analog sensor value names	240
6.33.1	Detailed Description	241
6.33.2	Define Documentation	241
6.34	Output module functions	241
6.34.1	Detailed Description	245

6.34.2 Function Documentation	246
6.35 Display module types	269
6.35.1 Detailed Description	270
6.36 Display module functions	270
6.36.1 Detailed Description	274
6.36.2 Function Documentation	274
6.37 Sound module types	296
6.37.1 Detailed Description	296
6.38 Sound module functions	297
6.38.1 Detailed Description	299
6.38.2 Function Documentation	299
6.39 LowSpeed module types	308
6.39.1 Detailed Description	309
6.40 LowSpeed module functions	309
6.40.1 Detailed Description	311
6.40.2 Function Documentation	311
6.41 Low level LowSpeed module functions	324
6.41.1 Detailed Description	325
6.41.2 Function Documentation	325
6.42 LowSpeed module system call functions	331
6.42.1 Detailed Description	331
6.42.2 Function Documentation	331
6.43 Command module types	333
6.43.1 Detailed Description	334
6.44 Command module functions	334
6.44.1 Detailed Description	339
6.44.2 Function Documentation	339
6.45 Array API functions	364
6.45.1 Detailed Description	365
6.45.2 Function Documentation	366
6.46 IOCtrl module types	372

6.47	IOCtrl module functions	372
6.47.1	Detailed Description	372
6.47.2	Function Documentation	372
6.48	Comm module types	373
6.48.1	Detailed Description	374
6.49	Comm module functions	374
6.49.1	Detailed Description	383
6.49.2	Function Documentation	383
6.50	Button module types	430
6.50.1	Detailed Description	430
6.51	Button module functions	430
6.51.1	Detailed Description	431
6.51.2	Function Documentation	432
6.52	Ui module types	438
6.52.1	Detailed Description	438
6.53	Ui module functions	438
6.53.1	Detailed Description	440
6.53.2	Function Documentation	440
6.54	Loader module types	451
6.54.1	Detailed Description	452
6.55	Loader module functions	452
6.55.1	Detailed Description	455
6.55.2	Function Documentation	455
6.56	cmath API	474
6.56.1	Detailed Description	479
6.56.2	Define Documentation	479
6.56.3	Function Documentation	491
6.57	cstdio API	507
6.57.1	Detailed Description	509
6.57.2	Define Documentation	509
6.57.3	Function Documentation	510

6.58	fseek origin constants	517
6.58.1	Detailed Description	517
6.58.2	Define Documentation	517
6.59	cstdlib API	518
6.59.1	Detailed Description	519
6.59.2	Function Documentation	519
6.60	cstdlib API types	527
6.60.1	Detailed Description	527
6.61	cstring API	527
6.61.1	Detailed Description	530
6.61.2	Function Documentation	530
6.62	ctype API	545
6.62.1	Detailed Description	546
6.62.2	Function Documentation	546
6.63	Array operation constants	551
6.63.1	Detailed Description	551
6.63.2	Define Documentation	552
6.64	System Call function constants	552
6.64.1	Detailed Description	554
6.64.2	Define Documentation	554
6.65	Line number constants	561
6.65.1	Detailed Description	562
6.65.2	Define Documentation	562
6.66	Time constants	564
6.66.1	Detailed Description	566
6.66.2	Define Documentation	566
6.67	Mailbox constants	572
6.67.1	Detailed Description	573
6.67.2	Define Documentation	573
6.68	VM state constants	574
6.68.1	Detailed Description	574

6.68.2 Define Documentation	575
6.69 Fatal errors	575
6.69.1 Detailed Description	576
6.69.2 Define Documentation	576
6.70 General errors	578
6.70.1 Detailed Description	578
6.70.2 Define Documentation	578
6.71 Communications specific errors	579
6.71.1 Detailed Description	579
6.71.2 Define Documentation	579
6.72 Remote control (direct commands) errors	579
6.72.1 Detailed Description	580
6.72.2 Define Documentation	580
6.73 Program status constants	580
6.73.1 Detailed Description	581
6.73.2 Define Documentation	581
6.74 Command module IOMAP offsets	581
6.74.1 Detailed Description	582
6.74.2 Define Documentation	582
6.75 IOCtrl module constants	584
6.75.1 Detailed Description	584
6.76 PowerOn constants	584
6.76.1 Detailed Description	584
6.76.2 Define Documentation	584
6.77 IOCtrl module IOMAP offsets	585
6.77.1 Detailed Description	585
6.77.2 Define Documentation	585
6.78 Loader module constants	585
6.78.1 Detailed Description	586
6.78.2 Define Documentation	586
6.79 Loader module IOMAP offsets	586

6.79.1	Detailed Description	586
6.79.2	Define Documentation	586
6.80	Loader module error codes	587
6.80.1	Detailed Description	588
6.80.2	Define Documentation	588
6.81	Loader module function constants	591
6.81.1	Detailed Description	592
6.81.2	Define Documentation	592
6.82	Sound module constants	596
6.82.1	Detailed Description	596
6.83	SoundFlags constants	596
6.83.1	Detailed Description	597
6.83.2	Define Documentation	597
6.84	SoundState constants	597
6.84.1	Detailed Description	597
6.84.2	Define Documentation	598
6.85	SoundMode constants	598
6.85.1	Detailed Description	598
6.85.2	Define Documentation	599
6.86	Sound module IOMAP offsets	599
6.86.1	Detailed Description	599
6.86.2	Define Documentation	599
6.87	Sound module miscellaneous constants	600
6.87.1	Detailed Description	601
6.87.2	Define Documentation	601
6.88	Tone constants	601
6.88.1	Detailed Description	603
6.88.2	Define Documentation	603
6.89	Button module constants	609
6.89.1	Detailed Description	609
6.90	Button name constants	609

6.90.1	Detailed Description	610
6.90.2	Define Documentation	610
6.91	ButtonState constants	612
6.91.1	Detailed Description	612
6.91.2	Define Documentation	612
6.92	Button module IOMAP offsets	613
6.92.1	Detailed Description	613
6.92.2	Define Documentation	613
6.93	Ui module constants	614
6.93.1	Detailed Description	615
6.94	CommandFlags constants	615
6.94.1	Detailed Description	615
6.94.2	Define Documentation	615
6.95	UIState constants	616
6.95.1	Detailed Description	617
6.95.2	Define Documentation	617
6.96	UIButton constants	619
6.96.1	Detailed Description	619
6.96.2	Define Documentation	619
6.97	BluetoothState constants	620
6.97.1	Detailed Description	620
6.97.2	Define Documentation	620
6.98	VM run state constants	621
6.98.1	Detailed Description	622
6.98.2	Define Documentation	622
6.99	Ui module IOMAP offsets	622
6.99.1	Detailed Description	623
6.99.2	Define Documentation	623
6.100	NBC Input port constants	625
6.100.1	Detailed Description	625
6.100.2	Define Documentation	625

6.101	NBC sensor type constants	626
6.101.1	Detailed Description	627
6.101.2	Define Documentation	627
6.102	NBC sensor mode constants	629
6.102.1	Detailed Description	629
6.102.2	Define Documentation	629
6.103	Input field constants	630
6.103.1	Detailed Description	631
6.103.2	Define Documentation	631
6.104	Color sensor array indices	632
6.104.1	Detailed Description	632
6.104.2	Define Documentation	632
6.105	Color values	633
6.105.1	Detailed Description	633
6.105.2	Define Documentation	633
6.106	Color calibration state constants	634
6.106.1	Detailed Description	634
6.106.2	Define Documentation	634
6.107	Color calibration constants	635
6.107.1	Detailed Description	635
6.107.2	Define Documentation	635
6.108	Input module IOMAP offsets	636
6.108.1	Detailed Description	637
6.108.2	Define Documentation	637
6.109	Output port constants	639
6.109.1	Detailed Description	639
6.109.2	Define Documentation	639
6.110	PID constants	640
6.110.1	Detailed Description	641
6.110.2	Define Documentation	641
6.111	Output port update flag constants	642

6.111.1 Detailed Description	642
6.111.2 Define Documentation	643
6.112 Tachometer counter reset flags	643
6.112.1 Detailed Description	644
6.112.2 Define Documentation	644
6.113 Output port mode constants	645
6.113.1 Detailed Description	645
6.113.2 Define Documentation	645
6.114 Output port option constants	646
6.114.1 Detailed Description	646
6.114.2 Define Documentation	646
6.115 Output port run state constants	647
6.115.1 Detailed Description	647
6.115.2 Define Documentation	647
6.116 Output port regulation mode constants	648
6.116.1 Detailed Description	648
6.116.2 Define Documentation	648
6.117 Output field constants	649
6.117.1 Detailed Description	650
6.117.2 Define Documentation	650
6.118 Output module IOMAP offsets	655
6.118.1 Detailed Description	655
6.118.2 Define Documentation	655
6.119 LowSpeed module constants	657
6.119.1 Detailed Description	658
6.120 LSState constants	658
6.120.1 Detailed Description	659
6.120.2 Define Documentation	659
6.121 LSChannelState constants	659
6.121.1 Detailed Description	660
6.121.2 Define Documentation	660

6.122LSMode constants	661
6.122.1 Detailed Description	661
6.122.2 Define Documentation	661
6.123LSErrorType constants	661
6.123.1 Detailed Description	662
6.123.2 Define Documentation	662
6.124Low speed module IOMAP offsets	662
6.124.1 Detailed Description	663
6.124.2 Define Documentation	663
6.125LSNoRestartOnRead constants	665
6.125.1 Detailed Description	665
6.125.2 Define Documentation	665
6.126Standard I2C constants	666
6.126.1 Detailed Description	666
6.126.2 Define Documentation	666
6.127LEGO I2C address constants	667
6.127.1 Detailed Description	668
6.127.2 Define Documentation	668
6.128Ultrasonic sensor constants	668
6.128.1 Detailed Description	669
6.128.2 Define Documentation	669
6.129LEGO temperature sensor constants	670
6.129.1 Detailed Description	671
6.129.2 Define Documentation	671
6.130E-Meter sensor constants	673
6.130.1 Detailed Description	673
6.130.2 Define Documentation	673
6.131Display module constants	674
6.131.1 Detailed Description	676
6.131.2 Define Documentation	676
6.132DisplayExecuteFunction constants	680

6.132.1 Detailed Description	680
6.132.2 Define Documentation	680
6.133 Drawing option constants	681
6.133.1 Detailed Description	682
6.133.2 Define Documentation	683
6.134 Font drawing option constants	685
6.134.1 Detailed Description	685
6.134.2 Define Documentation	685
6.135 Display flags	687
6.135.1 Detailed Description	687
6.135.2 Define Documentation	687
6.136 Display contrast constants	688
6.136.1 Detailed Description	688
6.136.2 Define Documentation	688
6.137 Text line constants	689
6.137.1 Detailed Description	689
6.137.2 Define Documentation	689
6.138 Display module IOMAP offsets	690
6.138.1 Detailed Description	691
6.138.2 Define Documentation	691
6.139 Comm module constants	693
6.139.1 Detailed Description	694
6.140 Miscellaneous Comm module constants	694
6.140.1 Detailed Description	695
6.140.2 Define Documentation	695
6.141 Bluetooth State constants	697
6.141.1 Detailed Description	697
6.141.2 Define Documentation	697
6.142 Bluetooth state status constants	697
6.142.1 Detailed Description	698
6.142.2 Define Documentation	698

6.143Bluetooth hardware status constants	698
6.143.1 Detailed Description	699
6.143.2 Define Documentation	699
6.144Hi-speed port constants	699
6.144.1 Detailed Description	700
6.145Hi-speed port flags constants	700
6.145.1 Detailed Description	700
6.145.2 Define Documentation	700
6.146Hi-speed port state constants	700
6.146.1 Detailed Description	701
6.146.2 Define Documentation	701
6.147Hi-speed port SysCommHSControl constants	701
6.147.1 Detailed Description	702
6.147.2 Define Documentation	702
6.148Hi-speed port baud rate constants	702
6.148.1 Detailed Description	703
6.148.2 Define Documentation	703
6.149Hi-speed port data bits constants	705
6.149.1 Detailed Description	705
6.149.2 Define Documentation	705
6.150Hi-speed port stop bits constants	706
6.150.1 Detailed Description	706
6.150.2 Define Documentation	706
6.151Hi-speed port parity constants	706
6.151.1 Detailed Description	707
6.151.2 Define Documentation	707
6.152Hi-speed port combined UART constants	707
6.152.1 Detailed Description	707
6.152.2 Define Documentation	708
6.153Device status constants	708
6.153.1 Detailed Description	708

6.153.2 Define Documentation	708
6.154 Comm module interface function constants	709
6.154.1 Detailed Description	710
6.154.2 Define Documentation	710
6.155 Comm module status code constants	712
6.155.1 Detailed Description	712
6.155.2 Define Documentation	712
6.156 Comm module IOMAP offsets	713
6.156.1 Detailed Description	714
6.156.2 Define Documentation	714
6.157 RCX constants	719
6.157.1 Detailed Description	720
6.158 RCX output constants	720
6.158.1 Detailed Description	720
6.158.2 Define Documentation	721
6.159 RCX output mode constants	722
6.159.1 Detailed Description	722
6.159.2 Define Documentation	722
6.160 RCX output direction constants	722
6.160.1 Detailed Description	723
6.160.2 Define Documentation	723
6.161 RCX output power constants	723
6.161.1 Detailed Description	724
6.161.2 Define Documentation	724
6.162 RCX IR remote constants	724
6.162.1 Detailed Description	725
6.162.2 Define Documentation	725
6.163 RCX and Scout sound constants	727
6.163.1 Detailed Description	727
6.163.2 Define Documentation	727
6.164 Scout constants	728

6.164.1 Detailed Description	729
6.165 Scout light constants	729
6.165.1 Detailed Description	729
6.165.2 Define Documentation	729
6.166 Scout sound constants	730
6.166.1 Detailed Description	730
6.166.2 Define Documentation	731
6.167 Scout sound set constants	733
6.167.1 Detailed Description	733
6.167.2 Define Documentation	733
6.168 Scout mode constants	734
6.168.1 Detailed Description	734
6.168.2 Define Documentation	734
6.169 Scout motion rule constants	735
6.169.1 Detailed Description	735
6.169.2 Define Documentation	735
6.170 Scout touch rule constants	736
6.170.1 Detailed Description	736
6.170.2 Define Documentation	737
6.171 Scout light rule constants	737
6.171.1 Detailed Description	738
6.171.2 Define Documentation	738
6.172 Scout transmit rule constants	738
6.172.1 Detailed Description	739
6.172.2 Define Documentation	739
6.173 Scout special effect constants	739
6.173.1 Detailed Description	740
6.173.2 Define Documentation	740
6.174 RCX and Scout source constants	740
6.174.1 Detailed Description	741
6.174.2 Define Documentation	742

6.175RCX and Scout opcode constants	746
6.175.1 Detailed Description	748
6.175.2 Define Documentation	748
6.176HiTechnic/mindsensors Power Function/IR Train constants	756
6.176.1 Detailed Description	757
6.177Power Function command constants	757
6.177.1 Detailed Description	757
6.177.2 Define Documentation	757
6.178Power Function channel constants	758
6.178.1 Detailed Description	758
6.178.2 Define Documentation	758
6.179Power Function mode constants	759
6.179.1 Detailed Description	759
6.179.2 Define Documentation	759
6.180PF/IR Train function constants	760
6.180.1 Detailed Description	760
6.180.2 Define Documentation	760
6.181IR Train channel constants	761
6.181.1 Detailed Description	761
6.181.2 Define Documentation	761
6.182Power Function output constants	762
6.182.1 Detailed Description	762
6.182.2 Define Documentation	762
6.183Power Function pin constants	763
6.183.1 Detailed Description	763
6.183.2 Define Documentation	763
6.184Power Function single pin function constants	763
6.184.1 Detailed Description	763
6.184.2 Define Documentation	764
6.185Power Function CST options constants	764
6.185.1 Detailed Description	764

6.185.2 Define Documentation	765
6.186 Power Function PWM option constants	766
6.186.1 Detailed Description	766
6.186.2 Define Documentation	766
6.187 HiTechnic device constants	768
6.187.1 Detailed Description	769
6.187.2 Define Documentation	769
6.188 HiTechnic IRSeeker2 constants	770
6.188.1 Detailed Description	770
6.188.2 Define Documentation	770
6.189 HiTechnic IRReceiver constants	772
6.189.1 Detailed Description	773
6.189.2 Define Documentation	773
6.190 HiTechnic Color2 constants	774
6.190.1 Detailed Description	774
6.190.2 Define Documentation	774
6.191 MindSensors device constants	775
6.191.1 Detailed Description	776
6.191.2 Define Documentation	777
6.192 MindSensors DIST-Nx constants	779
6.192.1 Detailed Description	780
6.192.2 Define Documentation	780
6.193 MindSensors PSP-Nx constants	781
6.193.1 Detailed Description	782
6.193.2 Define Documentation	782
6.194 MindSensors PSP-Nx button set 1 constants	783
6.194.1 Detailed Description	783
6.194.2 Define Documentation	783
6.195 MindSensors PSP-Nx button set 2 constants	784
6.195.1 Detailed Description	784
6.195.2 Define Documentation	784

6.196	MindSensors nRLink constants	785
6.196.1	Detailed Description	786
6.196.2	Define Documentation	786
6.197	MindSensors ACCL-Nx constants	787
6.197.1	Detailed Description	788
6.197.2	Define Documentation	788
6.198	MindSensors ACCL-Nx sensitivity level constants	790
6.198.1	Detailed Description	790
6.198.2	Define Documentation	791
6.199	Data type limits	791
6.199.1	Detailed Description	792
6.199.2	Define Documentation	792
6.200	Graphics library begin modes	793
6.200.1	Detailed Description	794
6.200.2	Define Documentation	794
6.201	Graphics library actions	794
6.201.1	Detailed Description	795
6.201.2	Define Documentation	795
6.202	Graphics library settings	797
6.202.1	Detailed Description	797
6.202.2	Define Documentation	797
6.203	Graphics library cull mode	797
6.203.1	Detailed Description	798
6.203.2	Define Documentation	798
7	Data Structure Documentation	798
7.1	ColorSensorReadType Struct Reference	798
7.1.1	Detailed Description	799
7.1.2	Field Documentation	799
7.2	CommBTCheckStatusType Struct Reference	800
7.2.1	Detailed Description	800

7.2.2	Field Documentation	801
7.3	CommBTConnectionType Struct Reference	801
7.3.1	Detailed Description	802
7.3.2	Field Documentation	802
7.4	CommBTOnOffType Struct Reference	803
7.4.1	Detailed Description	803
7.4.2	Field Documentation	803
7.5	CommBTWriteType Struct Reference	804
7.5.1	Detailed Description	804
7.5.2	Field Documentation	804
7.6	CommExecuteFunctionType Struct Reference	805
7.6.1	Detailed Description	805
7.6.2	Field Documentation	806
7.7	CommHSCheckStatusType Struct Reference	807
7.7.1	Detailed Description	808
7.7.2	Field Documentation	808
7.8	CommHSControlType Struct Reference	808
7.8.1	Detailed Description	809
7.8.2	Field Documentation	809
7.9	CommHSReadWriteType Struct Reference	810
7.9.1	Detailed Description	810
7.9.2	Field Documentation	810
7.10	CommLSCheckStatusType Struct Reference	811
7.10.1	Detailed Description	811
7.10.2	Field Documentation	811
7.11	CommLSReadType Struct Reference	812
7.11.1	Detailed Description	812
7.11.2	Field Documentation	813
7.12	CommLSWriteExType Struct Reference	814
7.12.1	Detailed Description	814
7.12.2	Field Documentation	814

7.13 CommLSWriteType Struct Reference	815
7.13.1 Detailed Description	816
7.13.2 Field Documentation	816
7.14 ComputeCalibValueType Struct Reference	817
7.14.1 Detailed Description	817
7.14.2 Field Documentation	817
7.15 DatalogGetTimesType Struct Reference	818
7.15.1 Detailed Description	818
7.15.2 Field Documentation	819
7.16 DatalogWriteType Struct Reference	819
7.16.1 Detailed Description	820
7.16.2 Field Documentation	820
7.17 DisplayExecuteFunctionType Struct Reference	820
7.17.1 Detailed Description	821
7.17.2 Field Documentation	821
7.18 div_t Struct Reference	823
7.18.1 Detailed Description	823
7.18.2 Field Documentation	823
7.19 DrawCircleType Struct Reference	824
7.19.1 Detailed Description	824
7.19.2 Field Documentation	824
7.20 DrawEllipseType Struct Reference	825
7.20.1 Detailed Description	825
7.20.2 Field Documentation	826
7.21 DrawFontType Struct Reference	827
7.21.1 Detailed Description	827
7.21.2 Field Documentation	827
7.22 DrawGraphicArrayType Struct Reference	828
7.22.1 Detailed Description	829
7.22.2 Field Documentation	829
7.23 DrawGraphicType Struct Reference	830

7.23.1 Detailed Description	830
7.23.2 Field Documentation	830
7.24 DrawLineType Struct Reference	831
7.24.1 Detailed Description	832
7.24.2 Field Documentation	832
7.25 DrawPointType Struct Reference	833
7.25.1 Detailed Description	833
7.25.2 Field Documentation	833
7.26 DrawPolygonType Struct Reference	834
7.26.1 Detailed Description	834
7.26.2 Field Documentation	835
7.27 DrawRectType Struct Reference	835
7.27.1 Detailed Description	836
7.27.2 Field Documentation	836
7.28 DrawTextType Struct Reference	837
7.28.1 Detailed Description	837
7.28.2 Field Documentation	837
7.29 FileCloseType Struct Reference	838
7.29.1 Detailed Description	838
7.29.2 Field Documentation	839
7.30 FileDeleteType Struct Reference	839
7.30.1 Detailed Description	839
7.30.2 Field Documentation	840
7.31 FileFindType Struct Reference	840
7.31.1 Detailed Description	840
7.31.2 Field Documentation	841
7.32 FileOpenType Struct Reference	841
7.32.1 Detailed Description	842
7.32.2 Field Documentation	842
7.33 FileReadWriteType Struct Reference	843
7.33.1 Detailed Description	843

7.33.2 Field Documentation	843
7.34 FileRenameType Struct Reference	844
7.34.1 Detailed Description	845
7.34.2 Field Documentation	845
7.35 FileResizeType Struct Reference	846
7.35.1 Detailed Description	846
7.35.2 Field Documentation	846
7.36 FileResolveHandleType Struct Reference	847
7.36.1 Detailed Description	847
7.36.2 Field Documentation	847
7.37 FileSeekType Struct Reference	848
7.37.1 Detailed Description	849
7.37.2 Field Documentation	849
7.38 GetStartTickType Struct Reference	850
7.38.1 Detailed Description	850
7.38.2 Field Documentation	850
7.39 IOMapReadByIDType Struct Reference	851
7.39.1 Detailed Description	851
7.39.2 Field Documentation	851
7.40 IOMapReadType Struct Reference	852
7.40.1 Detailed Description	853
7.40.2 Field Documentation	853
7.41 IOMapWriteByIDType Struct Reference	854
7.41.1 Detailed Description	854
7.41.2 Field Documentation	854
7.42 IOMapWriteType Struct Reference	855
7.42.1 Detailed Description	856
7.42.2 Field Documentation	856
7.43 KeepAliveType Struct Reference	857
7.43.1 Detailed Description	857
7.43.2 Field Documentation	857

7.44	Idiv_t Struct Reference	857
7.44.1	Detailed Description	858
7.44.2	Field Documentation	858
7.45	ListFileType Struct Reference	858
7.45.1	Detailed Description	859
7.45.2	Field Documentation	859
7.46	LoaderExecuteFunctionType Struct Reference	860
7.46.1	Detailed Description	860
7.46.2	Field Documentation	862
7.47	LocationType Struct Reference	862
7.47.1	Detailed Description	863
7.47.2	Field Documentation	863
7.48	MessageReadType Struct Reference	864
7.48.1	Detailed Description	864
7.48.2	Field Documentation	864
7.49	MessageWriteType Struct Reference	865
7.49.1	Detailed Description	865
7.49.2	Field Documentation	866
7.50	RandomNumberType Struct Reference	866
7.50.1	Detailed Description	866
7.50.2	Field Documentation	867
7.51	ReadButtonType Struct Reference	867
7.51.1	Detailed Description	867
7.51.2	Field Documentation	868
7.52	ReadSemDataType Struct Reference	868
7.52.1	Detailed Description	869
7.52.2	Field Documentation	869
7.53	SetScreenModeType Struct Reference	869
7.53.1	Detailed Description	870
7.53.2	Field Documentation	870
7.54	SetSleepTimeoutType Struct Reference	870

7.54.1	Detailed Description	871
7.54.2	Field Documentation	871
7.55	SizeType Struct Reference	871
7.55.1	Detailed Description	872
7.55.2	Field Documentation	872
7.56	SoundGetStateType Struct Reference	872
7.56.1	Detailed Description	873
7.56.2	Field Documentation	873
7.57	SoundPlayFileType Struct Reference	873
7.57.1	Detailed Description	874
7.57.2	Field Documentation	874
7.58	SoundPlayToneType Struct Reference	875
7.58.1	Detailed Description	875
7.58.2	Field Documentation	875
7.59	SoundSetStateType Struct Reference	876
7.59.1	Detailed Description	877
7.59.2	Field Documentation	877
7.60	Tone Struct Reference	878
7.60.1	Detailed Description	878
7.60.2	Field Documentation	878
7.61	UpdateCalibCacheInfoType Struct Reference	878
7.61.1	Detailed Description	879
7.61.2	Field Documentation	879
7.62	WriteSemDataType Struct Reference	880
7.62.1	Detailed Description	880
7.62.2	Field Documentation	881
8	File Documentation	881
8.1	NBCCommon.h File Reference	881
8.1.1	Detailed Description	917
8.1.2	Define Documentation	917

8.2	NXCAPIDocs.h File Reference	1085
8.2.1	Detailed Description	1085
8.3	NXCDefs.h File Reference	1085
8.3.1	Detailed Description	1160
8.3.2	Define Documentation	1160
8.3.3	Function Documentation	1183
9	Example Documentation	1549
9.1	alternating_tasks.nxc	1549
9.2	ex_abort.nxc	1550
9.3	ex_AbortFlag.nxc	1550
9.4	ex_abs.nxc	1550
9.5	ex_ACCLNxCalibrateX.nxc	1550
9.6	ex_ACCLNxCalibrateXEnd.nxc	1551
9.7	ex_ACCLNxCalibrateY.nxc	1551
9.8	ex_ACCLNxCalibrateYEnd.nxc	1551
9.9	ex_ACCLNxCalibrateZ.nxc	1551
9.10	ex_ACCLNxCalibrateZEnd.nxc	1551
9.11	ex_ACCLNxResetCalibration.nxc	1551
9.12	ex_ACCLNxSensitivity.nxc	1552
9.13	ex_ACCLNxXOffset.nxc	1552
9.14	ex_ACCLNxXRange.nxc	1552
9.15	ex_ACCLNxYOffset.nxc	1552
9.16	ex_ACCLNxYRange.nxc	1552
9.17	ex_ACCLNxZOffset.nxc	1552
9.18	ex_ACCLNxZRange.nxc	1552
9.19	ex_acos.nxc	1553
9.20	ex_acosd.nxc	1553
9.21	ex_Acquire.nxc	1554
9.22	ex_addressof.nxc	1554
9.23	ex_addressofex.nxc	1555

9.24 ex_arraybuild.nxc	1556
9.25 ex_arrayinit.nxc	1557
9.26 ex_arraylen.nxc	1557
9.27 ex_ArrayMax.nxc	1557
9.28 ex_ArrayMean.nxc	1557
9.29 ex_ArrayMin.nxc	1557
9.30 ex_ArrayOp.nxc	1558
9.31 ex_ArraySort.nxc	1558
9.32 ex_ArrayStd.nxc	1559
9.33 ex_arraysubset.nxc	1559
9.34 ex_ArraySum.nxc	1559
9.35 ex_ArraySumSqr.nxc	1559
9.36 ex_asin.nxc	1560
9.37 ex_asind.nxc	1560
9.38 ex_atan.nxc	1561
9.39 ex_atan2.nxc	1561
9.40 ex_atan2d.nxc	1562
9.41 ex_atand.nxc	1562
9.42 ex_atof.nxc	1563
9.43 ex_atoi.nxc	1563
9.44 ex_atol.nxc	1563
9.45 ex_BatteryState.nxc	1564
9.46 ex_bcd2dec.nxc	1564
9.47 ex_BluetoothState.nxc	1564
9.48 ex_bluetoothstatus.nxc	1564
9.49 ex_bluetoothwrite.nxc	1564
9.50 ex_BrickDataBluecoreVersion.nxc	1564
9.51 ex_BrickDataBtHardwareStatus.nxc	1565
9.52 ex_BrickDataBtStateStatus.nxc	1565
9.53 ex_BrickDataName.nxc	1565
9.54 ex_BrickDataTimeoutValue.nxc	1565

9.55 ex_BTConnectionClass.nxc	1565
9.56 ex_BTConnectionHandleNum.nxc	1565
9.57 ex_BTConnectionLinkQuality.nxc	1566
9.58 ex_BTConnectionName.nxc	1566
9.59 ex_BTConnectionPinCode.nxc	1566
9.60 ex_BTConnectionStreamStatus.nxc	1566
9.61 ex_BTDeviceClass.nxc	1566
9.62 ex_BTDeviceCount.nxc	1566
9.63 ex_BTDeviceName.nxc	1567
9.64 ex_BTDeviceNameCount.nxc	1567
9.65 ex_BTDeviceStatus.nxc	1567
9.66 ex_BTInputBufferInPtr.nxc	1567
9.67 ex_BTInputBufferOutPtr.nxc	1567
9.68 ex_BTOutputBufferInPtr.nxc	1567
9.69 ex_BTOutputBufferOutPtr.nxc	1568
9.70 ex_ButtonCount.nxc	1568
9.71 ex_ButtonLongPressCount.nxc	1568
9.72 ex_ButtonLongReleaseCount.nxc	1568
9.73 ex_ButtonPressCount.nxc	1568
9.74 ex_ButtonPressed.nxc	1568
9.75 ex_ButtonReleaseCount.nxc	1569
9.76 ex_ButtonShortReleaseCount.nxc	1569
9.77 ex_ButtonState.nxc	1569
9.78 ex_ByteArrayToStr.nxc	1569
9.79 ex_ByteArrayToStrEx.nxc	1569
9.80 ex_ceil.nxc	1570
9.81 ex_CircleOut.nxc	1570
9.82 ex_ClearLine.nxc	1570
9.83 ex_ClearScreen.nxc	1570
9.84 ex_ClearSensor.nxc	1571
9.85 ex_CloseFile.nxc	1571

9.86 ex_coast.nxc	1571
9.87 ex_coastex.nxc	1571
9.88 ex_ColorADRaw.nxc	1571
9.89 ex_ColorBoolean.nxc	1571
9.90 ex_ColorCalibration.nxc	1571
9.91 ex_ColorCalibrationState.nxc	1572
9.92 ex_ColorCalLimits.nxc	1572
9.93 ex_ColorSensorRaw.nxc	1572
9.94 ex_ColorSensorValue.nxc	1572
9.95 ex_CommandFlags.nxc	1572
9.96 ex_contrast.nxc	1572
9.97 ex_Copy.nxc	1573
9.98 ex_cosh.nxc	1573
9.99 ex_CreateFile.nxc	1573
9.100ex_CreateFileLinear.nxc	1573
9.101ex_CreateFileNonLinear.nxc	1574
9.102ex_cstdio.nxc	1574
9.103ex_cstring.nxc	1574
9.104ex_ctype.nxc	1575
9.105ex_CurrentTick.nxc	1575
9.106ex_CustomSensorActiveStatus.nxc	1576
9.107ex_CustomSensorPercentFullScale.nxc	1576
9.108ex_CustomSensorZeroOffset.nxc	1576
9.109ex_delete_data_file.nxc	1576
9.110ex_DeleteFile.nxc	1577
9.111ex_dispfnout.nxc	1577
9.112ex_dispftout.nxc	1577
9.113ex_dispfunc.nxc	1578
9.114ex_dispgaout.nxc	1578
9.115ex_dispgaoutex.nxc	1579
9.116ex_dispgout.nxc	1584

9.117ex_dispoutex.nxc	1585
9.118ex_DisplayDisplay.nxc	1586
9.119ex_DisplayEraseMask.nxc	1586
9.120ex_DisplayFlags.nxc	1586
9.121ex_DisplayFont.nxc	1586
9.122ex_DisplayTextLinesCenterFlags.nxc	1587
9.123ex_DisplayUpdateMask.nxc	1588
9.124ex_dispmisc.nxc	1588
9.125ex_DISTNxDistance.nxc	1588
9.126ex_DISTNxGP2D12.nxc	1588
9.127ex_DISTNxGP2D120.nxc	1589
9.128ex_DISTNxGP2YA02.nxc	1589
9.129ex_DISTNxGP2YA21.nxc	1589
9.130ex_DISTNxMaxDistance.nxc	1589
9.131ex_DISTNxMinDistance.nxc	1589
9.132ex_DISTNxModuleType.nxc	1589
9.133ex_DISTNxNumPoints.nxc	1590
9.134ex_DISTNxVoltage.nxc	1590
9.135ex_div.nxc	1590
9.136ex_EllipseOut.nxc	1590
9.137ex_exp.nxc	1590
9.138ex_fclose.nxc	1591
9.139ex_feof.nxc	1591
9.140ex fflush.nxc	1591
9.141ex_fgetc.nxc	1591
9.142ex_fgets.nxc	1591
9.143ex_file_system.nxc	1591
9.144ex_FindFirstFile.nxc	1595
9.145ex_FindNextFile.nxc	1595
9.146ex_FirstTick.nxc	1595
9.147ex_Flatten.nxc	1595

9.148ex_FlattenVar.nxc	1595
9.149ex_float.nxc	1596
9.150ex_floor.nxc	1596
9.151ex_Follows.nxc	1596
9.152ex_fopen.nxc	1596
9.153ex_ForceOff.nxc	1597
9.154ex_FormatNum.nxc	1597
9.155ex_fprintf.nxc	1597
9.156ex_fputc.nxc	1597
9.157ex_fputs.nxc	1597
9.158ex_frac.nxc	1597
9.159ex_FreeMemory.nxc	1598
9.160ex_fseek.nxc	1598
9.161ex_ftell.nxc	1598
9.162ex_GetBrickDataAddress.nxc	1598
9.163ex_GetBTConnectionAddress.nxc	1598
9.164ex_GetBTDeviceAddress.nxc	1599
9.165ex_GetBTInputBuffer.nxc	1599
9.166ex_GetBTOutputBuffer.nxc	1599
9.167ex_getc.nxc	1599
9.168ex_getchar.nxc	1599
9.169ex_GetDisplayNormal.nxc	1600
9.170ex_GetDisplayPopup.nxc	1600
9.171ex_GetHSInputBuffer.nxc	1600
9.172ex_GetHSOutputBuffer.nxc	1600
9.173ex_GetInput.nxc	1600
9.174ex_GetLSInputBuffer.nxc	1600
9.175ex_GetLSOutputBuffer.nxc	1601
9.176ex_getoutput.nxc	1601
9.177ex_GetUSBInputBuffer.nxc	1601
9.178ex_GetUSBOOutputBuffer.nxc	1601

9.179ex_GetUSBPollBuffer.nxc	1601
9.180ex_GraphicOut.nxc	1601
9.181ex_GraphicOutEx.nxc	1602
9.182ex_HSFlags.nxc	1602
9.183ex_HSInputBufferInPtr.nxc	1602
9.184ex_HSInputBufferOutPtr.nxc	1602
9.185ex_HSMode.nxc	1602
9.186ex_HSOutputBufferInPtr.nxc	1602
9.187ex_HSOutputBufferOutPtr.nxc	1603
9.188ex_HSSpeed.nxc	1603
9.189ex_HSState.nxc	1603
9.190ex_HTIRTrain.nxc	1603
9.191ex_HTPFComboDirect.nxc	1603
9.192ex_HTPFComboPWM.nxc	1603
9.193ex_HTPFRawOutput.nxc	1604
9.194ex_HTPFRepeat.nxc	1604
9.195ex_HTPFSingleOutputCST.nxc	1604
9.196ex_HTPFSingleOutputPWM.nxc	1604
9.197ex_HTPFSinglePin.nxc	1604
9.198ex_HTPFTrain.nxc	1604
9.199ex_HTRCXAddToDatalog.nxc	1605
9.200ex_HTRCXBatteryLevel.nxc	1605
9.201ex_HTRCXClearAllEvents.nxc	1605
9.202ex_HTRCXClearCounter.nxc	1605
9.203ex_HTRCXClearMsg.nxc	1605
9.204ex_HTRCXClearSensor.nxc	1605
9.205ex_HTRCXClearSound.nxc	1606
9.206ex_HTRCXClearTimer.nxc	1606
9.207ex_HTRCXCreateDatalog.nxc	1606
9.208ex_HTRCXDecCounter.nxc	1606
9.209ex_HTRCXDeleteSub.nxc	1606

9.210ex_HTRCXDeleteSubs.nxc	1606
9.211ex_HTRCXDeleteTask.nxc	1607
9.212ex_HTRCXDeleteTasks.nxc	1607
9.213ex_HTRCXDisableOutput.nxc	1607
9.214ex_HTRCXEnableOutput.nxc	1607
9.215ex_HTRCXEvent.nxc	1607
9.216ex_HTRCXFloat.nxc	1607
9.217ex_HTRCXFwd.nxc	1608
9.218ex_HTRCXIncCounter.nxc	1608
9.219ex_HTRCXInvertOutput.nxc	1608
9.220ex_HTRCXMuteSound.nxc	1608
9.221ex_HTRCXObvertOutput.nxc	1608
9.222ex_HTRCXOff.nxc	1608
9.223ex_HTRCXOn.nxc	1609
9.224ex_HTRCXOnFor.nxc	1609
9.225ex_HTRCXOnFwd.nxc	1609
9.226ex_HTRCXOnRev.nxc	1609
9.227ex_HTRCXPBTurnOff.nxc	1609
9.228ex_HTRCXPing.nxc	1609
9.229ex_HTRCXPlaySound.nxc	1610
9.230ex_HTRCXPlayTone.nxc	1610
9.231ex_HTRCXPlayToneVar.nxc	1610
9.232ex_HTRCXPoll.nxc	1610
9.233ex_HTRCXPollMemory.nxc	1610
9.234ex_HTRCXRemote.nxc	1610
9.235ex_HTRCXRev.nxc	1611
9.236ex_HTRCXSelectDisplay.nxc	1611
9.237ex_HTRCXSelectProgram.nxc	1611
9.238ex_HTRCXSendSerial.nxc	1611
9.239ex_HTRCXSetDirection.nxc	1611
9.240ex_HTRCXSetEvent.nxc	1611

9.241ex_HTRCXSetGlobalDirection.nxc	1612
9.242ex_HTRCXSetGlobalOutput.nxc	1612
9.243ex_HTRCXSetIRLinkPort.nxc	1612
9.244ex_HTRCXSetMaxPower.nxc	1612
9.245ex_HTRCXSetMessage.nxc	1612
9.246ex_HTRCXSetOutput.nxc	1612
9.247ex_HTRCXSetPower.nxc	1613
9.248ex_HTRCXSetPriority.nxc	1613
9.249ex_HTRCXSetSensorMode.nxc	1613
9.250ex_HTRCXSetSensorType.nxc	1613
9.251ex_HTRCXSetSleepTime.nxc	1613
9.252ex_HTRCXSetTxPower.nxc	1613
9.253ex_HTRCXSetWatch.nxc	1614
9.254ex_HTRCXStartTask.nxc	1614
9.255ex_HTRCXStopAllTasks.nxc	1614
9.256ex_HTRCXStopTask.nxc	1614
9.257ex_HTRCXToggle.nxc	1614
9.258ex_HTRCXUnmuteSound.nxc	1614
9.259ex_HTSOundCalibrateSensor.nxc	1615
9.260ex_HTSOundMuteSound.nxc	1615
9.261ex_HTSOundSelectSounds.nxc	1615
9.262ex_HTSOundSendVLL.nxc	1615
9.263ex_HTSOundSetEventFeedback.nxc	1615
9.264ex_HTSOundSetLight.nxc	1615
9.265ex_HTSOundSetScoutMode.nxc	1616
9.266ex_HTSOundSetSensorClickTime.nxc	1616
9.267ex_HTSOundSetSensorHysteresis.nxc	1616
9.268ex_HTSOundSetSensorLowerLimit.nxc	1616
9.269ex_HTSOundSetSensorUpperLimit.nxc	1616
9.270ex_HTSOundUnmuteSound.nxc	1616
9.271ex_I2CBytes.nxc	1617

9.272ex_I2CBytesReady.nxc	1617
9.273ex_I2CCheckStatus.nxc	1617
9.274ex_I2CDeviceId.nxc	1617
9.275ex_I2CDeviceInfo.nxc	1617
9.276ex_I2CRead.nxc	1618
9.277ex_I2CSendCommand.nxc	1618
9.278ex_I2CStatus.nxc	1618
9.279ex_I2CVendorId.nxc	1618
9.280ex_I2CVersion.nxc	1618
9.281ex_i2cwrite.nxc	1619
9.282ex_isalnum.nxc	1619
9.283ex_isalpha.nxc	1619
9.284ex_iscntrl.nxc	1619
9.285ex_isdigit.nxc	1619
9.286ex_isgraph.nxc	1619
9.287ex_islower.nxc	1620
9.288ex_isNAN.nxc	1620
9.289ex_isprint.nxc	1620
9.290ex_ispunct.nxc	1620
9.291ex_isspace.nxc	1620
9.292ex_isupper.nxc	1621
9.293ex_isxdigit.nxc	1621
9.294ex_labs.nxc	1621
9.295ex_ldiv.nxc	1621
9.296ex_leftstr.nxc	1622
9.297ex_LineOut.nxc	1622
9.298ex_log.nxc	1622
9.299ex_log10.nxc	1622
9.300ex_LongAbort.nxc	1622
9.301ex_LowspeedBytesReady.nxc	1623
9.302ex_LowspeedCheckStatus.nxc	1623

9.303ex_LowspeedRead.nxc	1623
9.304ex_lowspeedstatus.nxc	1623
9.305ex_LowspeedWrite.nxc	1623
9.306ex_LSChannelState.nxc	1623
9.307ex_LSErrorType.nxc	1624
9.308ex_LSInputBufferBytesToRx.nxc	1624
9.309ex_LSInputBufferInPtr.nxc	1624
9.310ex_LSInputBufferOutPtr.nxc	1624
9.311ex_LSMODE.nxc	1624
9.312ex_LSNorestartOnRead.nxc	1624
9.313ex_LSOOutputBufferBytesToRx.nxc	1625
9.314ex_LSOOutputBufferInPtr.nxc	1625
9.315ex_LSOOutputBufferOutPtr.nxc	1625
9.316ex_LSSpeed.nxc	1625
9.317ex_LSSState.nxc	1625
9.318ex_memcmp.nxc	1625
9.319ex_memcpy.nxc	1626
9.320ex_memmove.nxc	1626
9.321ex_midstr.nxc	1626
9.322ex_motoractualspeed.nxc	1626
9.323ex_motorblocktachocount.nxc	1626
9.324ex_motormode.nxc	1627
9.325ex_motoroverload.nxc	1627
9.326ex_motorpower.nxc	1627
9.327ex_motorpwnfreq.nxc	1627
9.328ex_motorregdvalue.nxc	1627
9.329ex_motorregivalue.nxc	1627
9.330ex_motorregpvalue.nxc	1628
9.331ex_motorregulation.nxc	1628
9.332ex_motorrotationcount.nxc	1628
9.333ex_motorrunstate.nxc	1628

9.334ex_motortachocount.nxc	1628
9.335ex_motortacholimit.nxc	1628
9.336ex_motorturnratio.nxc	1628
9.337ex_MSADPAOff.nxc	1629
9.338ex_MSADPAOn.nxc	1629
9.339ex_MSDenergize.nxc	1629
9.340ex_MSEnergize.nxc	1629
9.341ex_MSIRTrain.nxc	1629
9.342ex_MSPFComboDirect.nxc	1629
9.343ex_MSPFComboPWM.nxc	1630
9.344ex_MSPFRawOutput.nxc	1630
9.345ex_MSPFRepeat.nxc	1630
9.346ex_MSPFSingleOutputCST.nxc	1630
9.347ex_MSPFSingleOutputPWM.nxc	1630
9.348ex_MSPFSinglePin.nxc	1630
9.349ex_MSPFTrain.nxc	1631
9.350ex_MSRCXAbsVar.nxc	1631
9.351ex_MSRCXAddToDatalog.nxc	1631
9.352ex_MSRCXAndVar.nxc	1631
9.353ex_MSRCXBatteryLevel.nxc	1631
9.354ex_MSRCXBoot.nxc	1631
9.355ex_MSRCXCalibrateEvent.nxc	1632
9.356ex_MSRCXClearAllEvents.nxc	1632
9.357ex_MSRCXClearCounter.nxc	1632
9.358ex_MSRCXClearMsg.nxc	1632
9.359ex_MSRCXClearSensor.nxc	1632
9.360ex_MSRCXClearSound.nxc	1632
9.361ex_MSRCXClearTimer.nxc	1633
9.362ex_MSRCXCreateDatalog.nxc	1633
9.363ex_MSRCXDecCounter.nxc	1633
9.364ex_MSRCXDeleteSub.nxc	1633

9.365ex_MSRCXDeleteSubs.nxc	1633
9.366ex_MSRCXDeleteTask.nxc	1633
9.367ex_MSRCXDeleteTasks.nxc	1634
9.368ex_MSRCXDisableOutput.nxc	1634
9.369ex_MSRCXDivVar.nxc	1634
9.370ex_MSRCXEnableOutput.nxc	1634
9.371ex_MSRCXEvent.nxc	1634
9.372ex_MSRCXFloat.nxc	1634
9.373ex_MSRCXFwd.nxc	1635
9.374ex_MSRCXIIncCounter.nxc	1635
9.375ex_MSRCXIInvertOutput.nxc	1635
9.376ex_MSRCXMulVar.nxc	1635
9.377ex_MSRCXMuteSound.nxc	1635
9.378ex_MSRCXObvertOutput.nxc	1635
9.379ex_MSRCXOff.nxc	1636
9.380ex_MSRCXOn.nxc	1636
9.381ex_MSRCXOnFor.nxc	1636
9.382ex_MSRCXOnFwd.nxc	1636
9.383ex_MSRCXOnRev.nxc	1636
9.384ex_MSRCXOrVar.nxc	1636
9.385ex_MSRCXPBTurnOff.nxc	1637
9.386ex_MSRCXPing.nxc	1637
9.387ex_MSRCXPlaySound.nxc	1637
9.388ex_MSRCXPlayTone.nxc	1637
9.389ex_MSRCXPlayToneVar.nxc	1637
9.390ex_MSRCXPoll.nxc	1637
9.391ex_MSRCXPollMemory.nxc	1638
9.392ex_MSRCXRemote.nxc	1638
9.393ex_MSRCXReset.nxc	1638
9.394ex_MSRCXRev.nxc	1638
9.395ex_MSRCXSelectDisplay.nxc	1638

9.396ex_MSRCXSelectProgram.nxc	1638
9.397ex_MSRCXSendSerial.nxc	1639
9.398ex_MSRCXSet.nxc	1639
9.399ex_MSRCXSetDirection.nxc	1639
9.400ex_MSRCXSetEvent.nxc	1639
9.401ex_MSRCXSetGlobalDirection.nxc	1639
9.402ex_MSRCXSetGlobalOutput.nxc	1639
9.403ex_MSRCXSetMaxPower.nxc	1640
9.404ex_MSRCXSetMessage.nxc	1640
9.405ex_MSRCXSetNRLLinkPort.nxc	1640
9.406ex_MSRCXSetOutput.nxc	1640
9.407ex_MSRCXSetPower.nxc	1640
9.408ex_MSRCXSetPriority.nxc	1640
9.409ex_MSRCXSetSensorMode.nxc	1641
9.410ex_MSRCXSetSensorType.nxc	1641
9.411ex_MSRCXSetSleepTime.nxc	1641
9.412ex_MSRCXSetTxPower.nxc	1641
9.413ex_MSRCXSetUserDisplay.nxc	1641
9.414ex_MSRCXSetVar.nxc	1641
9.415ex_MSRCXSetWatch.nxc	1642
9.416ex_MSRCXSgnVar.nxc	1642
9.417ex_MSRCXStartTask.nxc	1642
9.418ex_MSRCXStopAllTasks.nxc	1642
9.419ex_MSRCXStopTask.nxc	1642
9.420ex_MSRCXSubVar.nxc	1642
9.421ex_MSRCXSumVar.nxc	1643
9.422ex_MSRCXToggle.nxc	1643
9.423ex_MSRCXUnlock.nxc	1643
9.424ex_MSRCXUnmuteSound.nxc	1643
9.425ex_MSReadValue.nxc	1643
9.426ex_MSScoutCalibrateSensor.nxc	1643

9.427ex_MSScoutMuteSound.nxc	1644
9.428ex_MSScoutSelectSounds.nxc	1644
9.429ex_MSScoutSendVLL.nxc	1644
9.430ex_MSScoutSetCounterLimit.nxc	1644
9.431ex_MSScoutSetEventFeedback.nxc	1644
9.432ex_MSScoutSetLight.nxc	1644
9.433ex_MSScoutSetScoutMode.nxc	1645
9.434ex_MSScoutSetScoutRules.nxc	1645
9.435ex_MSScoutSetSensorClickTime.nxc	1645
9.436ex_MSScoutSetSensorHysteresis.nxc	1645
9.437ex_MSScoutSetSensorLowerLimit.nxc	1645
9.438ex_MSScoutSetSensorUpperLimit.nxc	1645
9.439ex_MSScoutSetTimerLimit.nxc	1646
9.440ex_MSScoutUnmuteSound.nxc	1646
9.441ex_muldiv32.nxc	1646
9.442ex_NRLink2400.nxc	1646
9.443ex_NRLink4800.nxc	1646
9.444ex_NRLinkFlush.nxc	1646
9.445ex_NRLinkIRLong.nxc	1647
9.446ex_NRLinkIRShort.nxc	1647
9.447ex_NRLinkSetPF.nxc	1647
9.448ex_NRLinkSetRCX.nxc	1647
9.449ex_NRLinkSetTrain.nxc	1647
9.450ex_NRLinkStatus.nxc	1647
9.451ex_NRLinkTxRaw.nxc	1647
9.452ex_NumOut.nxc	1648
9.453ex_NumToStr.nxc	1648
9.454ex_off.nxc	1648
9.455ex_offex.nxc	1648
9.456ex_OnBrickProgramPointer.nxc	1648
9.457ex_onfwd.nxc	1648

9.458ex_onfwdex.nxc	1649
9.459ex_onfwdreg.nxc	1649
9.460ex_onfwdregex.nxc	1649
9.461ex_onfwdregexpid.nxc	1649
9.462ex_onfwdregpid.nxc	1649
9.463ex_onfwdsync.nxc	1649
9.464ex_onfwdsyncexc.nxc	1649
9.465ex_onfwdsyncexpid.nxc	1650
9.466ex_onfwdsyncpid.nxc	1650
9.467ex_onrev.nxc	1650
9.468ex_onrevex.nxc	1650
9.469ex_onrevreg.nxc	1650
9.470ex_onrevregex.nxc	1650
9.471ex_onrevregexpid.nxc	1650
9.472ex_onrevregpid.nxc	1651
9.473ex_onrevsync.nxc	1651
9.474ex_onrevsyncexc.nxc	1651
9.475ex_onrevsyncexpid.nxc	1651
9.476ex_onrevsyncpid.nxc	1651
9.477ex_OpenFileAppend.nxc	1651
9.478ex_OpenFileRead.nxc	1651
9.479ex_OpenFileReadLinear.nxc	1652
9.480ex_PlayFile.nxc	1652
9.481ex_PlayFileEx.nxc	1652
9.482ex_playsound.nxc	1652
9.483ex_PlayTone.nxc	1652
9.484ex_PlayToneEx.nxc	1653
9.485ex_playtones.nxc	1653
9.486ex_PointOut.nxc	1653
9.487ex_PolyOut.nxc	1653
9.488ex_pow.nxc	1654

9.489ex_powerdown.nxc	1654
9.490ex_Precedes.nxc	1654
9.491ex_printf.nxc	1654
9.492ex_PSPNxAnalog.nxc	1654
9.493ex_PSPNxDigital.nxc	1655
9.494ex_putc.nxc	1655
9.495ex_rand.nxc	1655
9.496ex_Random.nxc	1655
9.497ex_Read.nxc	1655
9.498ex_ReadButtonEx.nxc	1655
9.499ex_ReadBytes.nxc	1656
9.500ex_readi2cregister.nxc	1656
9.501ex_ReadLn.nxc	1656
9.502ex_ReadNRLinkBytes.nxc	1656
9.503ex_ReadSensorColorEx.nxc	1656
9.504ex_ReadSensorColorRaw.nxc	1656
9.505ex_ReadSensorHTAccel.nxc	1657
9.506ex_ReadSensorHTColor.nxc	1657
9.507ex_ReadSensorHTColor2Active.nxc	1657
9.508ex_ReadSensorHTIRReceiver.nxc	1657
9.509ex_ReadSensorHTIRReceiverEx.nxc	1657
9.510ex_ReadSensorHTIRSeeker.nxc	1657
9.511ex_ReadSensorHTIRSeeker2AC.nxc	1658
9.512ex_ReadSensorHTIRSeeker2DC.nxc	1658
9.513ex_ReadSensorHTNormalizedColor.nxc	1658
9.514ex_ReadSensorHTNormalizedColor2Active.nxc	1658
9.515ex_ReadSensorHTRawColor.nxc	1658
9.516ex_ReadSensorHTRawColor2.nxc	1658
9.517ex_ReadSensorHTTouchMultiplexer.nxc	1659
9.518ex_ReadSensorMSAccel.nxc	1659
9.519ex_ReadSensorMSPlayStation.nxc	1659

9.520ex_ReadSensorMSRTClock.nxc	1659
9.521ex_ReadSensorMSTilt.nxc	1660
9.522ex_readsensoreusex.nxc	1660
9.523ex_rebootinfirmwaremode.nxc	1660
9.524ex_receivemessage.nxc	1660
9.525ex_ReceiveRemoteBool.nxc	1660
9.526ex_ReceiveRemoteMessageEx.nxc	1660
9.527ex_ReceiveRemoteNumber.nxc	1661
9.528ex_ReceiveRemoteString.nxc	1661
9.529ex_RechargeableBattery.nxc	1661
9.530ex_RectOut.nxc	1661
9.531ex_reladdressof.nxc	1661
9.532ex_Release.nxc	1662
9.533ex_RemoteKeepAlive.nxc	1662
9.534ex_RemoteMessageRead.nxc	1662
9.535ex_RemoteMessageWrite.nxc	1662
9.536ex_RemotePlaySoundFile.nxc	1663
9.537ex_RemotePlayTone.nxc	1663
9.538ex_RemoteResetMotorPosition.nxc	1663
9.539ex_RemoteResetScaledValue.nxc	1663
9.540ex_RemoteSetInputMode.nxc	1663
9.541ex_RemoteSetOutputState.nxc	1663
9.542ex_RemoteStartProgram.nxc	1664
9.543ex_RemoteStopProgram.nxc	1664
9.544ex_RemoteStopSound.nxc	1664
9.545ex_remove.nxc	1664
9.546ex_rename.nxc	1664
9.547ex_RenameFile.nxc	1665
9.548ex_resetalltachocounts.nxc	1665
9.549ex_resetblocktachocount.nxc	1665
9.550ex_resetrotationcount.nxc	1665

9.551ex_ResetScreen.nxc	1665
9.552ex_ResetSensor.nxc	1665
9.553ex_ResetSleepTimer.nxc	1666
9.554ex_resettachocount.nxc	1666
9.555ex_ResizeFile.nxc	1666
9.556ex_ResolveHandle.nxc	1666
9.557ex_rewind.nxc	1666
9.558ex_rightstr.nxc	1666
9.559ex_rotatemotor.nxc	1667
9.560ex_rotatemotorex.nxc	1667
9.561ex_rotatemotorexpid.nxc	1667
9.562ex_rotatemotorpid.nxc	1667
9.563ex_RS485Control.nxc	1667
9.564ex_RS485DataAvailable.nxc	1667
9.565ex_RS485Exit.nxc	1667
9.566ex_RS485Init.nxc	1668
9.567ex_RS485Read.nxc	1668
9.568ex_RS485SendingData.nxc	1668
9.569ex_RS485Status.nxc	1668
9.570ex_RS485Uart.nxc	1668
9.571ex_RS485Write.nxc	1668
9.572ex_RunNRLLinkMacro.nxc	1669
9.573ex_sendmessage.nxc	1669
9.574ex_sendremotebool.nxc	1669
9.575ex_SendRemoteNumber.nxc	1669
9.576ex_SendRemoteString.nxc	1669
9.577ex_SendResponseBool.nxc	1669
9.578ex_SendResponseNumber.nxc	1670
9.579ex_SendResponseString.nxc	1670
9.580ex_SendRS485Bool.nxc	1670
9.581ex_SendRS485Number.nxc	1670

9.582ex_SendRS485String.nxc	1670
9.583ex_Sensor.nxc	1670
9.584ex_SensorBoolean.nxc	1671
9.585ex_SensorDigiPinsDirection.nxc	1671
9.586ex_SensorDigiPinsOutputLevel.nxc	1671
9.587ex_SensorDigiPinsStatus.nxc	1671
9.588ex_SensorHTColorNum.nxc	1671
9.589ex_SensorHTCompass.nxc	1671
9.590ex_SensorHTEOPD.nxc	1672
9.591ex_SensorHTGyro.nxc	1672
9.592ex_SensorHTIRSeeker2ACDir.nxc	1672
9.593ex_SensorHTIRSeeker2Addr.nxc	1672
9.594ex_SensorHTIRSeeker2DCDir.nxc	1672
9.595ex_SensorHTIRSeekerDir.nxc	1672
9.596ex_SensorInvalid.nxc	1673
9.597ex_SensorMode.nxc	1673
9.598ex_SensorMSCompass.nxc	1673
9.599ex_SensorMSDROD.nxc	1673
9.600ex_SensorMSPressure.nxc	1673
9.601ex_SensorMSPressureRaw.nxc	1673
9.602ex_SensorNormalized.nxc	1674
9.603ex_SensorRaw.nxc	1674
9.604ex_SensorScaled.nxc	1674
9.605ex_SensorType.nxc	1674
9.606ex_sensorus.nxc	1674
9.607ex_SensorValue.nxc	1674
9.608ex_SensorValueBool.nxc	1675
9.609ex_SensorValueRaw.nxc	1675
9.610ex_SetAbortFlag.nxc	1675
9.611ex_SetACCLNxSensitivity.nxc	1675
9.612ex_SetBatteryState.nxc	1675

9.613ex_SetBluetoothState.nxc	1675
9.614ex_SetBTInputBuffer.nxc	1676
9.615ex_SetBTInputBufferInPtr.nxc	1676
9.616ex_SetBTInputBufferOutPtr.nxc	1676
9.617ex_SetBTOutputBuffer.nxc	1676
9.618ex_SetBTOutputBufferInPtr.nxc	1676
9.619ex_SetBTOutputBufferOutPtr.nxc	1676
9.620ex_SetButtonLongPressCount.nxc	1677
9.621ex_SetButtonLongReleaseCount.nxc	1677
9.622ex_SetButtonPressCount.nxc	1677
9.623ex_SetButtonReleaseCount.nxc	1677
9.624ex_SetButtonShortReleaseCount.nxc	1677
9.625ex_SetButtonState.nxc	1677
9.626ex_SetCommandFlags.nxc	1678
9.627ex_SetCustomSensorActiveStatus.nxc	1678
9.628ex_SetCustomSensorPercentFullScale.nxc	1678
9.629ex_SetCustomSensorZeroOffset.nxc	1678
9.630ex_SetDisplayContrast.nxc	1678
9.631ex_SetDisplayDisplay.nxc	1678
9.632ex_SetDisplayEraseMask.nxc	1679
9.633ex_SetDisplayFlags.nxc	1679
9.634ex_SetDisplayFont.nxc	1679
9.635ex_SetDisplayNormal.nxc	1680
9.636ex_SetDisplayPopup.nxc	1680
9.637ex_SetDisplayTextLinesCenterFlags.nxc	1680
9.638ex_SetDisplayUpdateMask.nxc	1681
9.639ex_SetHSFlags.nxc	1681
9.640ex_SetHSInputBuffer.nxc	1681
9.641ex_SetHSInputBufferInPtr.nxc	1681
9.642ex_SetHSInputBufferOutPtr.nxc	1681
9.643ex_SetHSMode.nxc	1681

9.644ex_SetHSOutputBuffer.nxc	1682
9.645ex_SetHSOutputBufferInPtr.nxc	1682
9.646ex_SetHSOutputBufferOutPtr.nxc	1682
9.647ex_SetHSSpeed.nxc	1682
9.648ex_SetHSState.nxc	1682
9.649ex_SetHTColor2Mode.nxc	1682
9.650ex_SetHTIRSeeker2Mode.nxc	1683
9.651ex_SetInput.nxc	1683
9.652ex_SetLongAbort.nxc	1683
9.653ex_setmotorpwnfreq.nxc	1683
9.654ex_SetOnBrickProgramPointer.nxc	1683
9.655ex_setoutput.nxc	1683
9.656ex_SetSensor.nxc	1684
9.657ex_SetSensorBoolean.nxc	1684
9.658ex_SetSensorColorBlue.nxc	1684
9.659ex_SetSensorColorFull.nxc	1684
9.660ex_SetSensorColorGreen.nxc	1684
9.661ex_SetSensorColorNone.nxc	1684
9.662ex_SetSensorColorRed.nxc	1684
9.663ex_SetSensorDigiPinsDirection.nxc	1685
9.664ex_SetSensorDigiPinsOutputLevel.nxc	1685
9.665ex_SetSensorDigiPinsStatus.nxc	1685
9.666ex_SetSensorHTEOPD.nxc	1685
9.667ex_SetSensorHTGyro.nxc	1685
9.668ex_SetSensorLight.nxc	1685
9.669ex_SetSensorLowspeed.nxc	1686
9.670ex_SetSensorMode.nxc	1686
9.671ex_SetSensorMSDROD.nxc	1686
9.672ex_SetSensorMSPressure.nxc	1686
9.673ex_SetSensorSound.nxc	1686
9.674ex_SetSensorTouch.nxc	1686

9.675ex_SetSensorType.nxc	1687
9.676ex_SetSleepTime.nxc	1687
9.677ex_SetSleepTimeout.nxc	1687
9.678ex_SetSleepTimer.nxc	1687
9.679ex_SetSoundDuration.nxc	1687
9.680ex_SetSoundFlags.nxc	1687
9.681ex_SetSoundFrequency.nxc	1688
9.682ex_SetSoundMode.nxc	1688
9.683ex_SetSoundModuleState.nxc	1688
9.684ex_SetSoundSampleRate.nxc	1688
9.685ex_SetSoundVolume.nxc	1688
9.686ex_SetUIButton.nxc	1688
9.687ex_SetUIState.nxc	1689
9.688ex_SetUSBInputBuffer.nxc	1689
9.689ex_SetUSBInputBufferInPtr.nxc	1689
9.690ex_SetUSBInputBufferOutPtr.nxc	1689
9.691ex_SetUSBOutputBuffer.nxc	1689
9.692ex_SetUSBOutputBufferInPtr.nxc	1689
9.693ex_SetUSBOutputBufferOutPtr.nxc	1690
9.694ex_SetUSBPollBuffer.nxc	1690
9.695ex_SetUSBPollBufferInPtr.nxc	1690
9.696ex_SetUSBPollBufferOutPtr.nxc	1690
9.697ex_SetUSBState.nxc	1690
9.698ex_SetVMRunState.nxc	1690
9.699ex_SetVolume.nxc	1691
9.700ex_sign.nxc	1691
9.701ex_sin_cos.nxc	1691
9.702ex_sind_cosd.nxc	1692
9.703ex_sinh.nxc	1692
9.704ex_sleepnow.nxc	1692
9.705ex_SleepTime.nxc	1693

9.706ex_SleepTimeout.nxc	1693
9.707ex_SleepTimer.nxc	1693
9.708ex_SoundDuration.nxc	1693
9.709ex_SoundFlags.nxc	1693
9.710ex_SoundFrequency.nxc	1693
9.711ex_SoundMode.nxc	1694
9.712ex_SoundSampleRate.nxc	1694
9.713ex_SoundState.nxc	1694
9.714ex_SoundVolume.nxc	1694
9.715ex_sprintf.nxc	1694
9.716ex_sqrt.nxc	1694
9.717ex_starttask.nxc	1695
9.718ex_stop.nxc	1695
9.719ex_stopalltasks.nxc	1695
9.720ex_StopSound.nxc	1695
9.721ex_stoptask.nxc	1695
9.722ex_strcat.nxc	1695
9.723ex_StrCatOld.nxc	1696
9.724ex_strcmp.nxc	1696
9.725ex strcpy.nxc	1696
9.726ex_StrIndex.nxc	1696
9.727ex_string.nxc	1696
9.728ex_strlen.nxc	1697
9.729ex_StrLenOld.nxc	1697
9.730ex_strncat.nxc	1698
9.731ex_strncmp.nxc	1698
9.732ex_strncpy.nxc	1698
9.733ex_StrReplace.nxc	1698
9.734ex_StrToByteArray.nxc	1699
9.735ex strtod.nxc	1699
9.736ex strtol.nxc	1699

CONTENTS**li**

9.737ex_StrToNum.nxc	1699
9.738ex_strtoul.nxc	1699
9.739ex_SubStr.nxc	1700
9.740ex_syscall.nxc	1700
9.741ex_SysColorSensorRead.nxc	1700
9.742ex_syscommbtcheckstatus.nxc	1701
9.743ex_SysCommBTConnection.nxc	1701
9.744ex_SysCommBTOnOff.nxc	1701
9.745ex_syscommbtwrite.nxc	1702
9.746ex_syscommexecufunction.nxc	1702
9.747ex_SysCommHSCheckStatus.nxc	1702
9.748ex_SysCommHSControl.nxc	1703
9.749ex_SysCommHSRead.nxc	1703
9.750ex_SysCommHSWrite.nxc	1703
9.751ex_syscommlscheckstatus.nxc	1704
9.752ex_syscommlsread.nxc	1704
9.753ex_syscommlswrite.nxc	1704
9.754ex_syscommlswriteex.nxc	1704
9.755ex_SysComputeCalibValue.nxc	1705
9.756ex_SysDatalogGetTimes.nxc	1705
9.757ex_SysDatalogWrite.nxc	1705
9.758ex_sysdisplayexecufunction.nxc	1706
9.759ex_sysdrawcircle.nxc	1706
9.760ex_sysdrawellipse.nxc	1706
9.761ex_sysdrawfont.nxc	1707
9.762ex_sysdrawgraphic.nxc	1707
9.763ex_sysdrawgraphicarray.nxc	1707
9.764ex_sysdrawline.nxc	1708
9.765ex_sysdrawpoint.nxc	1709
9.766ex_sysdrawpolygon.nxc	1709
9.767ex_sysdrawrect.nxc	1709

9.768ex_sysdrawtext.nxc	1710
9.769ex_sysfileclose.nxc	1710
9.770ex_sysfiledelete.nxc	1710
9.771ex_sysfilefindfirst.nxc	1711
9.772ex_sysfilefindnext.nxc	1711
9.773ex_sysfileopenappend.nxc	1711
9.774ex_sysfileopenread.nxc	1711
9.775ex_sysfileopenreadlinear.nxc	1712
9.776ex_sysfileopenwrite.nxc	1712
9.777ex_sysfileopenwritelinear.nxc	1712
9.778ex_sysfileopenwritenonlinear.nxc	1713
9.779ex_sysfileread.nxc	1713
9.780ex_sysfilerename.nxc	1713
9.781ex_sysfileresize.nxc	1714
9.782ex_sysfileresolvehandle.nxc	1714
9.783ex_sysfileseek.nxc	1715
9.784ex_sysfilewrite.nxc	1715
9.785ex_sysgetstarttick.nxc	1715
9.786ex_sysiomapread.nxc	1716
9.787ex_sysiomapreadbyid.nxc	1716
9.788ex_sysiomapwrite.nxc	1716
9.789ex_sysiomapwritebyid.nxc	1716
9.790ex_syskeepalive.nxc	1717
9.791ex_syslistfiles.nxc	1717
9.792ex_sysloaderexecutefunction.nxc	1717
9.793ex_sysmessageread.nxc	1718
9.794ex_sysmessagewrite.nxc	1718
9.795ex_sysrandomnumber.nxc	1718
9.796ex_sysreadbutton.nxc	1718
9.797ex_SysReadSemData.nxc	1719
9.798ex_systssetscreenmode.nxc	1719

9.799ex_SysSetSleepTimeout.nxc	1719
9.800ex_syssoundgetstate.nxc	1719
9.801ex_syssoundplayfile.nxc	1720
9.802ex_syssoundplaytone.nxc	1720
9.803ex_syssoundsetstate.nxc	1720
9.804ex_SysUpdateCalibCacheInfo.nxc	1721
9.805ex_SysWriteSemData.nxc	1721
9.806ex_tan.nxc	1721
9.807ex_tand.nxc	1722
9.808ex_tanh.nxc	1723
9.809ex_TextOut.nxc	1723
9.810ex_tolower.nxc	1723
9.811ex_toupper.nxc	1723
9.812ex_trunc.nxc	1723
9.813ex_UIButton.nxc	1723
9.814ex_UIState.nxc	1724
9.815ex_UiUsbState.nxc	1724
9.816ex_UnflattenVar.nxc	1724
9.817ex_USBInputBufferInPtr.nxc	1724
9.818ex_USBInputBufferOutPtr.nxc	1725
9.819ex_USBOutputBufferInPtr.nxc	1725
9.820ex_USBOutputBufferOutPtr.nxc	1725
9.821ex_USBPollBufferInPtr.nxc	1725
9.822ex_USBPollBufferOutPtr.nxc	1725
9.823ex_USBState.nxc	1725
9.824ex_VMRunState.nxc	1726
9.825ex_Volume.nxc	1726
9.826ex_wait.nxc	1726
9.827ex_Write.nxc	1726
9.828ex_WriteBytes.nxc	1726
9.829ex_WriteBytesEx.nxc	1726

9.830ex_writei2cregister.nxc	1727
9.831ex_WriteLn.nxc	1727
9.832ex_WriteLnString.nxc	1727
9.833ex_WriteNRLinkBytes.nxc	1727
9.834ex_WriteString.nxc	1727
9.835ex_yield.nxc	1727
9.836glBoxDemo.nxc	1728
9.837glCircleDemo.nxc	1729
9.838glRotateDemo.nxc	1729
9.839glScaleDemo.nxc	1730
9.840glTranslateDemo.nxc	1731
9.841util_battery_1.nxc	1732
9.842util_battery_2.nxc	1733
9.843util_rpm.nxc	1733

1 NXC Programmer's Guide

June 8, 2010

by John Hansen

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

2 Introduction

NXC stands for Not eXactly C.

It is a simple language for programming the LEGO MINDSTORMS NXT product. The NXT has a bytecode interpreter (provided by LEGO), which can be used to execute programs. The NXC compiler translates a source program into NXT bytecodes, which can then be executed on the target itself. Although the preprocessor and control

structures of NXC are very similar to C, NXC is not a general-purpose programming language - there are many restrictions that stem from limitations of the NXT bytecode interpreter.

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

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

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

3 The NXC Language

This section describes the NXC language.

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

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

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

3.1 Lexical Rules

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

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

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

3.1.1 Comments

Two forms of comments are supported in NXC.

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

```
/* this is a comment */

/* this is a two
   line comment */

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

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

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

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

3.1.2 Whitespace

Whitespace consists of all spaces, tabs, and newlines.

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

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

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

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

3.1.3 Numerical Constants

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

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

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

3.1.4 String Constants

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

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

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

3.1.5 Character Constants

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

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

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

3.1.6 Identifiers and Keywords

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

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

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

- [The asm statement](#)

- [bool](#)
- [The break statement](#)
- [byte](#)
- [The case label](#)
- [char](#)
- [const](#)
- [The continue statement](#)
- [The default label](#)
- [The do statement](#)
- [The if-else statement](#)
- [enum](#)
- [The false condition](#)
- [float](#)
- [The for statement](#)
- [The goto statement](#)
- [The if statement](#)
- [The inline keyword](#)
- [int](#)
- [long](#)
- [mutex](#)
- [The priority statement](#)
- [The repeat statement](#)
- [The return statement](#)
- [The safecall keyword](#)
- [short](#)
- [The start statement](#)
- [The stop statement](#)
- [string](#)

- Structures
- The sub keyword
- The switch statement
- Tasks
- The true condition
- `typedef`
- `unsigned`
- The until statement
- The void keyword
- The while statement

3.1.6.1 const

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

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

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

3.1.6.2 enum

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

The syntax is show below.

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

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

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

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

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

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

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

Days tomorrow;

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

3.1.6.3 **typedef**

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

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

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

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

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

3.2 Program Structure

An NXC program is composed of code blocks and variables.

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

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

3.2.1 Code Order

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

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

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

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

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

The rules for lexical ordering are:

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

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

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

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

3.2.2 Tasks

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

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

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

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

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

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

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

```
task music() {
    while (true) {
        PlayTone(TONE_A4, MS_500);
        Wait(MS_600);
    }
}

task movement() {
    while (true) {
        OnFwd(OUT_A, Random(100));
        Wait(Random(SEC_1));
    }
}

task controller() {
    Wait(SEC_10);
    stop music;
    Wait(SEC_5);
    StopAllTasks();
}

task main() {
    Precedes(music, movement, controller);
}
```

3.2.3 Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
void foo(const int x)
{
    PlayTone(x, MS_500);
    x = 1; // error - cannot modify argument
    Wait(SEC_1);
}

task main()
{
    int x = TONE_A4;
    foo(TONE_A5); // ok
    foo(4*TONE_A3); // expression is still constant
    foo(x); // x is not a constant but is okay
}
```

The third type, `arg_type &`, passes arguments by reference rather than by value. This allows the called function to modify the value and have those changes be available in the calling function after the called function returns. However, only variables may be used when calling a function using `arg_type &` arguments:

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

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

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

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

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

```
void foo(int bar, const int baz)
{
```

```
// do something here...
}

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

3.2.3.1 The safecall keyword

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

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

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

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

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

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

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

3.2.3.2 The inline keyword

You can optionally mark NXC functions as inline functions.

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

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

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

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

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

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

3.2.3.3 The void keyword

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

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

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

- [The sub keyword](#)

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

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

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

3.2.4 Variables

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

- [bool](#)
- [byte](#)
- [char](#)
- [int](#)
- [short](#)
- [long](#)
- [unsigned](#)
- [float](#)
- [mutex](#)
- [string](#)
- [Structures](#)
- [Arrays](#)

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

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

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

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

```
int x; // x is global

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

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

3.2.4.1 bool

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

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

```
bool flag=true;
```

3.2.4.2 byte

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

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

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

3.2.4.3 char

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

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

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

3.2.4.4 int

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

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

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

3.2.4.5 short

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

This type can store values from [SHRT_MIN](#) to [SHRT_MAX](#). This is an alias for the int type.

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

3.2.4.6 long

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

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

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

3.2.4.7 unsigned

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

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

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

3.2.4.8 float

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

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

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

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

3.2.4.9 mutex

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

For this reason there is never a reason to declare a mutex variable inside a task or a function. It is designed for global variables that all tasks or functions can [Acquire](#) or [Release](#) in order to obtain exclusive access to a resource that other tasks or functions are also trying to use.

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

```
    Precedes(t1, t2);  
}
```

3.2.4.10 string

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

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

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

3.2.5 Structures

NXC supports user-defined aggregate types known as structs.

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

```
struct car  
{  
    string car_type;  
    int manu_year;  
};  
  
struct person  
{  
    string name;  
    int age;  
    car vehicle;  
};  
  
person myPerson;
```

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

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

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

3.2.6 Arrays

NXC also support arrays.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Statements

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

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

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

3.3.1 Variable Declaration

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

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

```
arg_type variables;
```

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

```
name [=expression]
```

Arrays of variables may also be declared:

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

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

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

3.3.2 Assignment

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

```
variable assign_operator expression;
```

There are 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

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

There are several types described below.

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

3.3.3.1 The compound statement

The simplest control structure is a compound statement.

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

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

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

3.3.3.2 The if statement

The if statement evaluates a condition.

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

```
if (condition) consequence
```

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

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

3.3.3.3 The if-else statement

The if-else statement evaluates a condition.

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

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

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

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

3.3.3.4 The while statement

The while statement is used to construct a conditional loop.

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

```
while (condition) body
```

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

```
while(x < 10)
```

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

3.3.3.5 The do statement

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

The syntax for this control structure is shown below.

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

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

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

3.3.3.6 The for statement

Another kind of loop is the for loop.

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

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

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

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

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

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

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

3.3.3.7 The repeat statement

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

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

```
repeat (expression) body
```

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

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

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

3.3.3.8 The switch statement

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

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

```
switch (expression) body
```

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

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

A typical switch statement might look like this:

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

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

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

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

```
case constant_expression :
```

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

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

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

```
default :
```

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

3.3.3.9 The goto statement

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

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

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

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

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

3.3.3.10 The until statement

NXC also defines an until macro for compatibility with NQC.

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

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

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

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

3.3.4 The asm statement

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

The syntax of the statement is shown below.

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

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

```
asm {
//      jmp __lbl00D5
dseg segment
    s10000  slong
    s10005  slong
    bGTTrue byte
dseg ends
    mov     s10000, 0x0
    mov     s10005, s10000
    mov     s10000, 0x1
    cmp     GT, bGTTrue, s10005, s10000
    set    bGTTrue, FALSE
```

```
    brdst      EQ, __lbl00D5, bGTrue
__lbl00D5:
}
```

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

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

Table 4. ASM Keywords

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

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

3.3.5 Other NXC Statements

NXC supports a few other statement types.

The other NXC statements are described below.

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

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

```
x++;
```

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

3.3.5.1 The function call statement

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

```
name(arguments);
```

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

3.3.5.2 The start statement

You can start a task with the start statement.

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

```
start task_name;
```

3.3.5.3 The stop statement

You can stop a task with the stop statement.

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

```
stop task_name;
```

3.3.5.4 The priority statement

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

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

```
priority task_name, new_priority;
```

3.3.5.5 The break statement

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

It only exits out of the innermost loop

```
break;
```

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

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

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

3.3.5.6 The continue statement

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

```
continue;
```

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

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

3.3.5.7 The return statement

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

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

```
return [expression];
```

3.4 Expressions

Values are the most primitive type of expressions.

More complicated expressions are formed from values using various operators.

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

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

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

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

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

Table 5. Expression Operators

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

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

- Conditions

3.4.1 Conditions

Comparing two expressions forms a condition.

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

The table below summarizes the different types of conditions.

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

Table 6. Conditions

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

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

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

3.4.1.1 The true condition

The keyword true has a value of one.

It represents a condition that is always true.

3.4.1.2 The false condition

The keyword `false` has a value of zero.

It represents a condition that is always false.

3.5 The Preprocessor

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

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

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

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

3.5.1 #include

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

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

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

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

3.5.2 #define

The #define command is used for macro substitution.

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

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

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

3.5.3 ## (Concatenation)

The ## directive works similar to the C preprocessor.

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

```
#define ELEMENT_OUT(n) \  
    NumOut(0, LCD_LINE##n, b##n)

bool b1 = false;  
bool b2 = true;

task main()  
{  
    ELEMENT_OUT(1);  
    ELEMENT_OUT(2);  
    Wait(SEC_2);  
}
```

This is the same as writing

```
bool b1 = false;  
bool b2 = true;

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

3.5.4 Conditional Compilation

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

The following preprocessor directives may be used:

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

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 .rxe 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.ric"
```

4 Todo List

Global <globalScope>::StopSound() ?.

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

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

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

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

Global CommHSControlType::Result values?

Global ComputeCalibValueType::Name ?.

Global ComputeCalibValueType::RawVal ?.

Global ComputeCalibValueType::Result ?.

Global UpdateCalibCacheInfoType::Name ?.

Global UpdateCalibCacheInfoType::Result ?.

5 Deprecated List

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 Module Documentation

6.1 NXT Firmware Modules

Documentation common to all NXT firmware modules.

Modules

- [Input module](#)

Constants and functions related to the Input module.

- [Output module](#)

Constants and functions related to the Output module.

- [Display module](#)

Constants and functions related to the Display module.

- [Sound module](#)

Constants and functions related to the Sound module.

- [Low Speed module](#)

Constants and functions related to the Low Speed module.

- [Command module](#)

Constants and functions related to the Command module.

- [IOCtrl module](#)

Constants and functions related to the IOCtrl module.

- [Comm module](#)

Constants and functions related to the Comm module.

- [Button module](#)

Constants and functions related to the Button module.

- [Ui module](#)

Constants and functions related to the Ui module.

- [Loader module](#)

Constants and functions related to the Loader module.

- [NXT firmware module names](#)

Constant string names for all the NXT firmware modules.

- [NXT firmware module IDs](#)

Constant numeric IDs for all the NXT firmware modules.

6.1.1 Detailed Description

Documentation common to all NXT firmware modules.

6.2 Input module

Constants and functions related to the Input module.

Modules

- [Input module types](#)

Types used by various input module functions.

- [Input module functions](#)

Functions for accessing and modifying input module features.

- [Input module constants](#)

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

6.2.1 Detailed Description

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

There are four sensors, which internally are numbered 0, 1, 2, and 3. This is potentially confusing since they are externally labeled on the NXT as sensors 1, 2, 3, and 4. To help mitigate this confusion, the sensor port names [S1](#), [S2](#), [S3](#), and [S4](#) have been defined. See [Input port constants](#). These sensor names may be used in any function that requires a sensor port as an argument. Alternatively, the NBC port name constants [IN_1](#), [IN_2](#), [IN_3](#), and [IN_4](#) may also be used when a sensor port is required, although this is not recommended. See [NBC Input port constants](#). Sensor value names [SENSOR_1](#), [SENSOR_2](#), [SENSOR_3](#), and [SENSOR_4](#) have also been defined. These names may also be used whenever a program wishes to read the current value of the analog sensor:

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

6.3 Input module constants

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

Modules

- [Input port constants](#)

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

- [NBC Input port constants](#)

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

- [Input field constants](#)

Constants for use with `SetInput()` and `GetInput()`.

- [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

6.3.1 Detailed Description

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

6.3.2 Define Documentation

6.3.2.1 `#define INPUT_CUSTOM9V 0x01`

Custom sensor 9V

6.3.2.2 `#define INPUT_CUSTOMACTIVE 0x02`

Custom sensor active

6.3.2.3 `#define INPUT_CUSTOMINACTIVE 0x00`

Custom sensor inactive

6.3.2.4 `#define INPUT_DIGI0 1`

Digital pin 0

6.3.2.5 `#define INPUT_DIGI1 2`

Digital pin 1

6.3.2.6 `#define INPUT_INVALID_DATA 0x01`

Invalid data flag

6.4 Sensor types and modes

Constants that are used for defining sensor types and modes.

Modules

- **Sensor type constants**

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

- **Sensor mode constants**

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

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

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

- [NBC sensor type constants](#)

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

- [NBC sensor mode constants](#)

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

6.4.1 Detailed Description

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

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

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

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

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

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

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

6.5 Output module

Constants and functions related to the Output module.

Modules

- [Output module functions](#)

Functions for accessing and modifying output module features.

- [Output module constants](#)

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

6.5.1 Detailed Description

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

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

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

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

6.6 Output module constants

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

Modules

- [Output port constants](#)

Output port constants are used when calling motor control API functions.

- [PID constants](#)

PID constants are for adjusting the Proportional, Integral, and Derivative motor controller parameters.

- [Output port update flag constants](#)

Use these constants to specify which motor values need to be updated.

- [Tachometer counter reset flags](#)

Use these constants to specify which of the three tachometer counters should be reset.

- [Output port mode constants](#)

Use these constants to configure the desired mode for the specified motor(s): coast, motoron, brake, or regulated.

- [Output port option constants](#)

Use these constants to configure the desired options for the specified motor(s): hold at limit and ramp down to limit.

- [Output 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.

6.6.1 Detailed Description

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

6.7 Command module

Constants and functions related to the Command module.

Modules

- [Command module types](#)

Types used by various Command module functions.

- [Command module functions](#)

Functions for accessing and modifying Command module features.

- [Command module constants](#)

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

6.7.1 Detailed Description

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

6.8 Command module constants

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

Modules

- [Array operation constants](#)

Constants for use with the NXC ArrayOp function and the NBC arrop statement.

- [System Call function constants](#)

Constants for use in the SysCall() function or NBC syscall statement.

- [Time constants](#)

Constants for use with the Wait() function.

- [VM state constants](#)

Constants defining possible VM states.

- Fatal errors

Constants defining various fatal error conditions.

- General errors

Constants defining general error conditions.

- Communications specific errors

Constants defining communication error conditions.

- Remote control (direct commands) errors

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

- Program status constants

Constants defining various states of the command module virtual machine.

- Command module IOMAP offsets

Constant offsets into the Command module IOMAP structure.

Defines

- #define STAT_MSG_EMPTY_MAILBOX 64
- #define STAT_COMM_PENDING 32
- #define NO_ERR 0

6.8.1 Detailed Description

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

6.8.2 Define Documentation

6.8.2.1 #define NO_ERR 0

Successful execution of the specified command

Examples:

```
ex_SysColorSensorRead.nxc,      ex_SysCommBTConnection.nxc,      ex-
SysCommBTOnOff.nxc, ex_SysCommHSRead.nxc, ex_SysCommHSWrite.nxc,
ex_syscommlswriteex.nxc,       ex_SysComputeCalibValue.nxc,      ex-
SysDatalogWrite.nxc,   ex_sysfileopenappend.nxc,   ex_sysfileopenread.nxc,
```

`ex_sysfileopenreadlinear.nxc`, `ex_sysfileopenwrite.nxc`, `ex_sysfileopenwritelinear.nxc`, `ex_sysfileopenwrittenonlinear.nxc`, `ex_sysfileread.nxc`, `ex_sysfileresize.nxc`, `ex_sysfileseek.nxc`, `ex_sysfilewrite.nxc`, `ex_systiomapread.nxc`, `ex_systiomapreadbyid.nxc`, `ex_syslistfiles.nxc`, and `ex_sysmessageread.nxc`.

6.8.2.2 #define STAT_COMM_PENDING 32

Pending setup operation in progress

6.8.2.3 #define STAT_MSG_EMPTY_MAILBOX 64

Specified mailbox contains no new messages

6.9 Comm module

Constants and functions related to the Comm module.

Modules

- [Comm module types](#)

Types used by various Comm module functions.

- [Comm module functions](#)

Functions for accessing and modifying Comm module features.

- [Comm module constants](#)

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

6.9.1 Detailed Description

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

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

Communication via Bluetooth uses a master/slave connection system. One device must be designated as the master device before you run a program using Bluetooth. If the

NXT is the master device then you can configure up to three slave devices using connection 1, 2, and 3 on the NXT brick. If your NXT is a slave device then connection 0 on the brick must be reserved for the master device.

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

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

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

6.10 Button module

Constants and functions related to the Button module.

Modules

- [Button module types](#)

Types used by various Button module functions.

- [Button module functions](#)

Functions for accessing and modifying Button module features.

- [Button module constants](#)

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

6.10.1 Detailed Description

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

6.11 IOCtrl module

Constants and functions related to the IOCtrl module.

Modules

- [IOCtrl module types](#)

Types used by various IOCctrl module functions.

- [IOCctrl module functions](#)

Functions for accessing and modifying IOCctrl module features.

- [IOCctrl module constants](#)

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

6.11.1 Detailed Description

Constants and functions related to the IOCctrl 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.

6.12 Loader module

Constants and functions related to the Loader module.

Modules

- [Loader module types](#)

Types used by various Loader module functions.

- [Loader module functions](#)

Functions for accessing and modifying Loader module features.

- [Loader module constants](#)

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

6.12.1 Detailed Description

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

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

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

6.13 Sound module

Constants and functions related to the Sound module.

Modules

- [Sound module types](#)

Types used by various sound module functions.

- [Sound module functions](#)

Functions for accessing and modifying sound module features.

- [Sound module constants](#)

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

6.13.1 Detailed Description

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

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

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

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

The NXC API defines frequency and duration constants which may be used in calls to [PlayTone](#) or [PlayToneEx](#). Frequency constants start with [TONE_A3](#) (the 'A' pitch in octave 3) and go to [TONE_B7](#) (the 'B' pitch in octave 7). Duration constants start with [MS_1](#) (1 millisecond) and go up to [MIN_1](#) (60000 milliseconds) with several constants in between. See [NBCCommon.h](#) for the complete list.

6.14 Ui module

Constants and functions related to the Ui module.

Modules

- [Ui module types](#)

Types used by various Ui module functions.

- [Ui module functions](#)

Functions for accessing and modifying Ui module features.

- [Ui module constants](#)

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

6.14.1 Detailed Description

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

6.15 Low Speed module

Constants and functions related to the Low Speed module.

Modules

- [LowSpeed module types](#)

Types used by various low speed module functions.

- [LowSpeed module functions](#)

Functions for accessing and modifying low speed module features.

- [LowSpeed module constants](#)

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

6.15.1 Detailed Description

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

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

You must set the input port's Type property to `SENSOR_TYPE_LOWSPEED` or `SENSOR_TYPE_LOWSPEED_9V` on a given port before using an I2C device on that port. Use `SENSOR_TYPE_LOWSPEED_9V` if your device requires 9V power from the NXT brick. Remember that you also need to set the input port's `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.

6.16 Display module

Constants and functions related to the Display module.

Modules

- [Display module types](#)

Types used by various display module functions.

- [Display module functions](#)

Functions for accessing and modifying display module features.

- [Display module constants](#)

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

6.16.1 Detailed Description

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

The LCD screen has its origin (0, 0) at the bottom left-hand corner of the screen with the positive Y-axis extending upward and the positive X-axis extending toward the right. The NXC API provides constants for use in the [NumOut](#) and [TextOut](#) functions which make it possible to specify LCD line numbers between 1 and 8 with line 1 being at the top of the screen and line 8 being at the bottom of the screen. These constants ([LCD_LINE1](#), [LCD_LINE2](#), [LCD_LINE3](#), [LCD_LINE4](#), [LCD_LINE5](#), [LCD_LINE6](#), [LCD_LINE7](#), [LCD_LINE8](#)) should be used as the Y coordinate in NumOut and TextOut calls. Values of Y other than these constants will be adjusted so that text and numbers are on one of 8 fixed line positions.

6.17 HiTechnic API Functions

Functions for accessing and modifying HiTechnic devices.

Modules

- [HiTechnic device constants](#)

Constants that are for use with HiTechnic devices.

Functions

- int **SensorHTGyro** (const byte &port, int offset=0)
Read HiTechnic Gyro sensor.
- int **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 reg)
Read HiTechnic IRSeeker2 register.
- int **SensorHTIRSeeker2DCDir** (const byte &port)
Read HiTechnic IRSeeker2 DC direction.
- int **SensorHTIRSeeker2ACDir** (const byte &port)
Read HiTechnic IRSeeker2 AC direction.
- char **SetHTColor2Mode** (const byte &port, byte mode)
Set HiTechnic Color2 mode.
- char **SetHTIRSeeker2Mode** (const byte &port, const byte mode)
Set HiTechnic IRSeeker2 mode.
- bool **ReadSensorHTAccel** (const byte port, int &x, int &y, int &z)
Read HiTechnic acceleration values.
- bool **ReadSensorHTColor** (const byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue)

Read HiTechnic Color values.

- bool `ReadSensorHTIRSeeker` (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)

Read HiTechnic IRSeeker values.

- bool `ReadSensorHTNormalizedColor` (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)

Read HiTechnic Color normalized values.

- bool `ReadSensorHTRawColor` (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue)

Read HiTechnic Color raw values.

- bool `ReadSensorHTColor2Active` (byte port, byte &ColorNum, byte &Red, byte &Green, byte &Blue, byte &White)

Read HiTechnic Color2 active values.

- bool `ReadSensorHTNormalizedColor2Active` (const byte port, byte &ColorIdx, byte &Red, byte &Green, byte &Blue)

Read HiTechnic Color2 normalized active values.

- bool `ReadSensorHTRawColor2` (const byte port, unsigned int &Red, unsigned int &Green, unsigned int &Blue, unsigned int &White)

Read HiTechnic Color2 raw values.

- bool `ReadSensorHTIRReceiver` (const byte port, char &pfdatal[])

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 **HTPFCComboDirect** (const byte port, const byte channel, const byte outa, const byte outb)
HTPFCComboDirect function.
- char **HTPFCComboPWM** (const byte port, const byte channel, const byte outa, const byte outb)
HTPFCComboPWM function.
- char **HTPFRawOutput** (const byte port, const byte nibble0, const byte nibble1, const byte nibble2)
HTPFRawOutput function.
- char **HTPFRRepeat** (const byte port, const byte count, const unsigned int delay)
HTPFRRepeat 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 **HTPFTTrain** (const byte port, const byte channel, const byte func)
HTPFTTrain 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 **HTRCXBFloat** (const byte outputs)
HTRCXBFloat 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 pwrvrl)

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 pwrvrl)

HTRCXSetPower function.

- void **HTRCXSetPriority** (const byte p)

HTRCXSetPriority function.

- void **HTRCXSetSensorMode** (const byte port, const byte mode)

HTRCXSetSensorMode function.

- void **HTRCXSetSensorType** (const byte port, const byte type)

HTRCXSetSensorType function.

- void **HTRCXSetSleepTime** (const byte t)

HTRCXSetSleepTime function.

- void **HTRCXSetTxPower** (const byte pwr)

HTRCXSetTxPower function.

- void **HTRCXSetWatch** (const byte hours, const byte minutes)

HTRCXSetWatch function.

- void **HTRCXStartTask** (const byte t)

HTRCXStartTask function.

- void **HTRCXStopAllTasks** (void)

HTRCXStopAllTasks function.

- void **HTRCXStopTask** (const byte t)

HTRCXStopTask function.

- void **HTRCXToggle** (const byte outputs)
HTRCXToggle function.
- void **HTRCXUnmuteSound** (void)
HTRCXUnmuteSound function.
- void **HTScoutCalibrateSensor** (void)
HTScoutCalibrateSensor function.
- void **HTScoutMuteSound** (void)
HTScoutMuteSound function.
- void **HTScoutSelectSounds** (const byte grp)
HTScoutSelectSounds function.
- void **HTScoutSendVLL** (const byte src, const unsigned int value)
HTScoutSendVLL function.
- void **HTScoutSetEventFeedback** (const byte src, const unsigned int value)
HTScoutSetEventFeedback function.
- void **HTScoutSetLight** (const byte x)
HTScoutSetLight function.
- void **HTScoutSetScoutMode** (const byte mode)
HTScoutSetScoutMode function.
- void **HTScoutSetSensorClickTime** (const byte src, const unsigned int value)
HTScoutSetSensorClickTime function.
- void **HTScoutSetSensorHysteresis** (const byte src, const unsigned int value)
HTScoutSetSensorHysteresis function.
- void **HTScoutSetSensorLowerLimit** (const byte src, const unsigned int value)
HTScoutSetSensorLowerLimit function.
- void **HTScoutSetSensorUpperLimit** (const byte src, const unsigned int value)
HTScoutSetSensorUpperLimit function.
- void **HTScoutUnmuteSound** (void)
HTScoutUnmuteSound function.

6.17.1 Detailed Description

Functions for accessing and modifying HiTechnic devices.

6.17.2 Function Documentation

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

HTIRTrain function. Control an IR Train receiver set to the specified channel using the HiTechnic iRLink device. Valid func values are `TRAIN_FUNC_STOP`, `TRAIN_FUNC_INCR_SPEED`, `TRAIN_FUNC_DECR_SPEED`, and `TRAIN_FUNC_TOGGLE_LIGHT`. Valid channel values are `TRAIN_CHANNEL_1` through `TRAIN_CHANNEL_3` and `TRAIN_CHANNEL_ALL`. The port must be configured as a 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](#).

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

HTPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the HiTechnic iRLink device. Commands for outa and outb are `PF_CMD_STOP`, `PF_CMD_REV`, `PF_CMD_FWD`, and `PF_CMD_BRAKE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

Returns:

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

Examples:

[ex_HTPFComboDirect.nxc](#).

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

HTPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Valid output values are [PF_PWM_FLOAT](#), [PF_PWM_FWD1](#), [PF_PWM_FWD2](#), [PF_PWM_FWD3](#), [PF_PWM_FWD4](#), [PF_PWM_FWD5](#), [PF_PWM_FWD6](#), [PF_PWM_FWD7](#), [PF_PWM_BRAKE](#), [PF_PWM_REV7](#), [PF_PWM_REV6](#), [PF_PWM_REV5](#), [PF_PWM_REV4](#), [PF_PWM_REV3](#), [PF_PWM_REV2](#), and [PF_PWM_REV1](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).
channel The Power Function channel. See [Power Function channel constants](#).
outa The Power Function PWM command for output A. See [Power Function PWM option constants](#).
outb The Power Function PWM command for output B. See [Power Function PWM option constants](#).

Returns:

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

Examples:

[ex_HTPFComboPWM.nxc](#).

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

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

Parameters:

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

nibble0 The first raw data nibble.

nibble1 The second raw data nibble.

nibble2 The third raw data nibble.

Returns:

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

Examples:

[ex_HTPFRawOutput.nxc](#).

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

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

Parameters:

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

count The number of times to repeat the command.

delay The number of milliseconds to delay between each repetition.

Returns:

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

Examples:

[ex_HTPFRepeat.nxc](#).

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

HTPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Valid functions are [PF_CST_CLEAR1_CLEAR2](#), [PF_CST_SET1_CLEAR2](#), [PF_CST_CLEAR1_SET2](#), [PF_CST_SET1_SET2](#), [PF_CST_INCREMENT_PWM](#), [PF_CST_DECREMENT_PWM](#), [PF_CST_FULL_FWD](#), [PF_CST_FULL_REV](#), and [PF_CST_TOGGLE_DIR](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

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

func The Power Function CST function. See [Power Function CST options constants](#).

Returns:

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

Examples:

[ex_HTPFSingleOutputCST.nxc](#).

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

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

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

HTPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the HiTechnic iRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Select the desired pin using [PF_PIN_C1](#) or [PF_PIN_C2](#). Valid functions are [PF_FUNC_NOCHANGE](#), [PF_FUNC_CLEAR](#), [PF_FUNC_SET](#), and [PF_FUNC_TOGGLE](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). Specify whether the mode by passing true (continuous) or false (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](#).

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

HTPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the HiTechnic iRLink device as if it were an IR Train receiver. Valid function values are `TRAIN_FUNC_STOP`, `TRAIN_FUNC_INCR_SPEED`, `TRAIN_FUNC_DECR_SPEED`, and `TRAIN_FUNC_TOGGLE_LIGHT`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

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

Returns:

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

Examples:

[ex_HTPFTrain.nxc](#).

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

HTREXAddToDatalog 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_HTREXAddToDatalog.nxc](#).

6.17.2.11 int HTREXBatteryLevel (void) [inline]

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

Returns:

The RCX battery level.

Examples:

[ex_HTRCXBatteryLevel.nxc](#).

6.17.2.12 void HTRCXClearAllEvents (void) [inline]

HTRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

Examples:

[ex_HTRCXClearAllEvents.nxc](#).

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

HTRCXClearCounter function. Send the ClearCounter command to an RCX.

Parameters:

counter The counter to clear.

Examples:

[ex_HTRCXClearCounter.nxc](#).

6.17.2.14 void HTRCXClearMsg (void) [inline]

HTRCXClearMsg function. Send the ClearMsg command to an RCX.

Examples:

[ex_HTRCXClearMsg.nxc](#).

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

HTRECXClearSensor function. Send the ClearSensor command to an RCX.

Parameters:

port The RCX port number.

Examples:

[ex_HTRCXClearSensor.nxc](#).

6.17.2.16 void HTRECXClearSound (void) [inline]

HTRECXClearSound function. Send the ClearSound command to an RCX.

Examples:

[ex_HTRCXClearSound.nxc](#).

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

HTRECXClearTimer function. Send the ClearTimer command to an RCX.

Parameters:

timer The timer to clear.

Examples:

[ex_HTRCXClearTimer.nxc](#).

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

HTRECXCreateDatalog function. Send the CreateDatalog command to an RCX.

Parameters:

size The new datalog size.

Examples:

[ex_HTRCXCreateDatalog.nxc](#).

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

HTRCXDecCounter function. Send the DecCounter command to an RCX.

Parameters:

counter The counter to decrement.

Examples:

[ex_HTRCXDecCounter.nxc](#).

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

HTRCXDeleteSub function. Send the DeleteSub command to an RCX.

Parameters:

s The subroutine number to delete.

Examples:

[ex_HTRCXDeleteSub.nxc](#).

6.17.2.21 void HTRCXDeleteSubs (void) [inline]

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

Examples:

[ex_HTRCXDeleteSubs.nxc](#).

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

HTRCXDeleteTask function. Send the DeleteTask command to an RCX.

Parameters:

t The task number to delete.

Examples:

[ex_HTRCXDeleteTask.nxc](#).

6.17.2.23 void HTRCXDeleteTasks (void) [inline]

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

Examples:

[ex_HTRCXDeleteTasks.nxc](#).

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

HTRCXDisableOutput function. Send the DisableOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXDisableOutput.nxc](#).

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

HTRCXEnableOutput function. Send the EnableOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXEnableOutput.nxc](#).

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

HTRCXEvent function. Send the Event command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXEvent.nxc](#).

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

HTRCXBFloat 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_HTRCXBFloat.nxc](#).

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

HTRCXBwd 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_HTRCXBwd.nxc](#).

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

HTRECXIncCounter function. Send the IncCounter command to an RCX.

Parameters:

counter The counter to increment.

Examples:

[ex_HTRCXIncCounter.nxc](#).

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

HTRECXInvertOutput function. Send the InvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXInvertOutput.nxc](#).

6.17.2.31 void HTRECXMuteSound (void) [inline]

HTRECXMuteSound function. Send the MuteSound command to an RCX.

Examples:

[ex_HTRCXMuteSound.nxc](#).

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

HTRECXObvertOutput function. Send the ObvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXObvertOutput.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOff.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOn.nxc](#).

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

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

Parameters:

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

ms The number of milliseconds to leave the outputs on

Examples:

[ex_HTRCXOnFor.nxc](#).

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

HTRECXOnFwd 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_HTRECXOnFwd.nxc](#).

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

HTRECXOnRev 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_HTRECXOnRev.nxc](#).

6.17.2.38 void HTRECXPBTurnOff (void) [inline]

HTRECXPBTurnOff function. Send the PBTurnOff command to an RCX.

Examples:

[ex_HTRECXPBTurnOff.nxc](#).

6.17.2.39 void HTRECXPing (void) [inline]

HTRECXPing function. Send the Ping command to an RCX.

Examples:

[ex_HTRCXPing.nxc](#).

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

HTRCXPlaySound function. Send the PlaySound command to an RCX.

Parameters:

snd The sound number to play.

Examples:

[ex_HTRCXPlaySound.nxc](#).

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

HTRCXPlayTone function. Send the PlayTone command to an RCX.

Parameters:

freq The frequency of the tone to play.

duration The duration of the tone to play.

Examples:

[ex_HTRCXPlayTone.nxc](#).

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

HTRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

Parameters:

varnum The variable containing the tone frequency to play.

duration The duration of the tone to play.

Examples:

[ex_HTRCXPlayToneVar.nxc](#).

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

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

Parameters:

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

value The RCX value.

Returns:

The value read from the specified port and value.

Examples:

[ex_HTRCXPoll.nxc](#).

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

HTRCXPollMemory function. Send the PollMemory command to an RCX.

Parameters:

address The RCX memory address.

Returns:

The value read from the specified address.

Examples:

[ex_HTRCXPollMemory.nxc](#).

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

HTRECXRemote function. Send the Remote command to an RCX.

Parameters:

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

Examples:

[ex_HTRECXRemote.nxc](#).

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

HTRECXRev 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_HTRECXRev.nxc](#).

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

HTRECXSelectDisplay 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_HTRECXSelectDisplay.nxc](#).

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

HTRCXSelectProgram function. Send the SelectProgram command to an RCX.

Parameters:

prog The program number to select.

Examples:

[ex_HTRCXSelectProgram.nxc](#).

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

HTRCXSendSerial function. Send the SendSerial command to an RCX.

Parameters:

first The first byte address.

count The number of bytes to send.

Examples:

[ex_HTRCXSendSerial.nxc](#).

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

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

Parameters:

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

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

Examples:

[ex_HTRCXSetDirection.nxc](#).

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

HTRCXSetEvent function. Send the SetEvent command to an RCX.

Parameters:

evt The event number to set.

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

type The event type.

Examples:

[ex_HTRCXSetEvent.nxc](#).

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

HTRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

Parameters:

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

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

Examples:

[ex_HTRCXSetGlobalDirection.nxc](#).

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

HTRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

Parameters:

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

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

Examples:

[ex_HTRCXSetGlobalOutput.nxc](#).

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

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

Parameters:

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

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

HTRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

Parameters:

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

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

pwrval The RCX value.

Examples:

[ex_HTRCXSetMaxPower.nxc](#).

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

HTRCXSetMessage function. Send the SetMessage command to an RCX.

Parameters:

msg The numeric message to send.

Examples:

[ex_HTRCXSetMessage.nxc](#).

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

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

Parameters:

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

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

Examples:

[ex_HTRCXSetOutput.nxc](#).

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

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

Parameters:

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

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

pwrval The RCX value.

Examples:

[ex_HTRCXSetPower.nxc](#).

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

HTRECXSetPriority function. Send the SetPriority command to an RCX.

Parameters:

p The new task priority.

Examples:

[ex_HTRCXSetPriority.nxc](#).

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

HTRECXSetSensorMode function. Send the SetSensorMode command to an RCX.

Parameters:

port The RCX sensor port.
mode The RCX sensor mode.

Examples:

[ex_HTRCXSetSensorMode.nxc](#).

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

HTRECXSetSensorType function. Send the SetSensorType command to an RCX.

Parameters:

port The RCX sensor port.
type The RCX sensor type.

Examples:

[ex_HTRCXSetSensorType.nxc](#).

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

HTRECXSetSleepTime function. Send the SetSleepTime command to an RCX.

Parameters:

t The new sleep time value.

Examples:

[ex_HTRCXSetSleepTime.nxc](#).

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

HTRCXSetTxPower function. Send the SetTxPower command to an RCX.

Parameters:

pwr The IR transmit power level.

Examples:

[ex_HTRCXSetTxPower.nxc](#).

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

HTRCXSetWatch function. Send the SetWatch command to an RCX.

Parameters:

hours The new watch time hours value.

minutes The new watch time minutes value.

Examples:

[ex_HTRCXSetWatch.nxc](#).

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

HTRCXStartTask function. Send the StartTask command to an RCX.

Parameters:

t The task number to start.

Examples:

[ex_HTRCXStartTask.nxc](#).

6.17.2.66 void HTRECXStopAllTasks (void) [inline]

HTRECXStopAllTasks function. Send the StopAllTasks command to an RCX.

Examples:

[ex_HTRCXStopAllTasks.nxc](#).

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

HTRECXStopTask function. Send the StopTask command to an RCX.

Parameters:

t The task number to stop.

Examples:

[ex_HTRCXStopTask.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXToggle.nxc](#).

6.17.2.69 void HTRECXUnmuteSound (void) [inline]

HTRECXUnmuteSound function. Send the UnmuteSound command to an RCX.

Examples:

[ex_HTRCXUnmuteSound.nxc](#).

6.17.2.70 void HTScoutCalibrateSensor (void) [inline]

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

Examples:

[ex_HTScoutCalibrateSensor.nxc](#).

6.17.2.71 void HTScoutMuteSound (void) [inline]

HTScoutMuteSound function. Send the MuteSound command to a Scout.

Examples:

[ex_HTScoutMuteSound.nxc](#).

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

HTScoutSelectSounds function. Send the SelectSounds command to a Scout.

Parameters:

grp The Scout sound group to select.

Examples:

[ex_HTScoutSelectSounds.nxc](#).

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

HTScoutSendVLL function. Send the SendVLL command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSendVLL.nxc](#).

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

HTScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSetEventFeedback.nxc](#).

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

HTScoutSetLight function. Send the SetLight command to a Scout.

Parameters:

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

Examples:

[ex_HTScoutSetLight.nxc](#).

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

HTScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

Parameters:

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

Examples:

[ex_HTScoutSetScoutMode.nxc](#).

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

HTScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSetSensorClickTime.nxc](#).

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

HTScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSetSensorHysteresis.nxc](#).

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

HTScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSetSensorLowerLimit.nxc](#).

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

HTScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_HTScoutSetSensorUpperLimit.nxc](#).

6.17.2.81 void HTScoutUnmuteSound (void) [inline]

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

Examples:

[ex_HTScoutUnmuteSound.nxc](#).

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

Read HiTechnic acceleration values. Read X, Y, and Z axis acceleration values from the HiTechnic Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a 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_ReadSensorHTAccel.nxc](#).

6.17.2.83 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](#).

6.17.2.84 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](#).

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

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

Parameters:

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

pfdata Eight bytes of power function remote IR data.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRReceiver.nxc](#).

6.17.2.86 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 IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

offset The power function data offset. See [HiTechnic IRReceiver constants](#).

pfchar A single byte of power function remote IR data.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRReceiverEx.nxc](#).

**6.17.2.87 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](#).

6.17.2.88 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](#).

6.17.2.89 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](#).

6.17.2.90 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](#).

6.17.2.91 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](#).

6.17.2.92 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](#).

**6.17.2.93 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](#).

**6.17.2.94 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](#).

6.17.2.95 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](#).

6.17.2.96 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](#).

6.17.2.97 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](#).

6.17.2.98 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](#).

6.17.2.99 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](#).

**6.17.2.100 int SensorHTIRSeeker2Addr (const byte & *port*, const byte *reg*)
[inline]**

Read HiTechnic IRSeeker2 register. Read a register value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

Returns:

The IRSeeker2 register value.

Examples:

[ex_SensorHTIRSeeker2Addr.nxc](#).

6.17.2.101 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](#).

6.17.2.102 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](#).

6.17.2.103 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](#).

**6.17.2.104 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](#).

**6.17.2.105 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](#).

6.17.2.106 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](#).

6.18 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, const byte i2caddr)
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, const byte i2caddr, int &x, int &y, int &z)
Read mindsensors acceleration values.

- bool **ReadSensorMSPlayStation** (const byte port, const byte i2caddr, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)
Read mindsensors playstation controller values.
- bool **ReadSensorMSRTClock** (const byte port, byte &sec, byte &min, byte &hrs, byte &dow, byte &date, byte &month, byte &year)
Read mindsensors RTClock values.
- bool **ReadSensorMSTilt** (const byte port, const byte i2caddr, byte &x, byte &y, byte &z)
Read mindsensors tilt values.
- int **MSReadValue** (const byte port, const byte i2caddr, const byte reg, const byte numbytes)
Read a mindsensors device value.
- char **MSEnergize** (const byte port, const byte i2caddr)
Turn on power to device.
- char **MSDeenergize** (const byte port, const byte i2caddr)
Turn off power to device.
- char **MSADPAOn** (const byte port, const byte i2caddr)
Turn on mindsensors ADPA mode.
- char **MSADPAOff** (const byte port, const byte i2caddr)
Turn off mindsensors ADPA mode.
- char **DISTNxGP2D12** (const byte port, const byte i2caddr)
Configure DISTNx as GP2D12.
- char **DISTNxGP2D120** (const byte port, const byte i2caddr)
Configure DISTNx as GP2D120.
- char **DISTNxGP2YA02** (const byte port, const byte i2caddr)
Configure DISTNx as GP2YA02.
- char **DISTNxGP2YA21** (const byte port, const byte i2caddr)
Configure DISTNx as GP2YA21.
- int **DISTNxDistance** (const byte port, const byte i2caddr)
Read DISTNx distance value.

- int **DISTNxMaxDistance** (const byte port, const byte i2caddr)
Read DISTNx maximum distance value.
- int **DISTNxMinDistance** (const byte port, const byte i2caddr)
Read DISTNx minimum distance value.
- byte **DISTNxModuleType** (const byte port, const byte i2caddr)
Read DISTNx module type value.
- byte **DISTNxNumPoints** (const byte port, const byte i2caddr)
Read DISTNx num points value.
- int **DISTNxVoltage** (const byte port, const byte i2caddr)
Read DISTNx voltage value.
- char **ACCLNxCalibrateX** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx X-axis.
- char **ACCLNxCalibrateXEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx X-axis.
- char **ACCLNxCalibrateY** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx Y-axis.
- char **ACCLNxCalibrateYEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx Y-axis.
- char **ACCLNxCalibrateZ** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx Z-axis.
- char **ACCLNxCalibrateZEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx Z-axis.
- char **ACCLNxResetCalibration** (const byte port, const byte i2caddr)
Reset ACCL-Nx calibration.
- char **SetACCLNxSensitivity** (const byte port, const byte i2caddr, byte slevel)
Set ACCL-Nx sensitivity.
- byte **ACCLNxSensitivity** (const byte port, const byte i2caddr)
Read ACCL-Nx sensitivity value.

- int **ACCLNxXOffset** (const byte port, const byte i2caddr)
Read ACCL-Nx X offset value.
- int **ACCLNxXRange** (const byte port, const byte i2caddr)
Read ACCL-Nx X range value.
- int **ACCLNxYOffset** (const byte port, const byte i2caddr)
Read ACCL-Nx Y offset value.
- int **ACCLNxYRange** (const byte port, const byte i2caddr)
Read ACCL-Nx Y range value.
- int **ACCLNxZOffset** (const byte port, const byte i2caddr)
Read ACCL-Nx Z offset value.
- int **ACCLNxZRange** (const byte port, const byte i2caddr)
Read ACCL-Nx Z range value.
- char **PSPNxDigital** (const byte &port, const byte &i2caddr)
Configure PSPNx in digital mode.
- char **PSPNxAnalog** (const byte &port, const byte &i2caddr)
Configure PSPNx in analog mode.
- char **NRLLink2400** (const byte port, const byte i2caddr)
Configure NRLLink in 2400 baud mode.
- char **NRLLink4800** (const byte port, const byte i2caddr)
Configure NRLLink in 4800 baud mode.
- char **NRLLinkFlush** (const byte port, const byte i2caddr)
Flush NRLLink buffers.
- char **NRLLinkIRLong** (const byte port, const byte i2caddr)
Configure NRLLink in IR long mode.
- char **NRLLinkIRShort** (const byte port, const byte i2caddr)
Configure NRLLink in IR short mode.
- char **NRLLinkSetPF** (const byte port, const byte i2caddr)
Configure NRLLink in power function mode.

- char **NRLLinkSetRCX** (const byte port, const byte i2caddr)
Configure NRLLink in RCX mode.
- char **NRLLinkSetTrain** (const byte port, const byte i2caddr)
Configure NRLLink in IR train mode.
- char **NRLLinkTxRaw** (const byte port, const byte i2caddr)
Configure NRLLink in raw IR transmit mode.
- byte **NRLLinkStatus** (const byte port, const byte i2caddr)
Read NRLLink status.
- char **RunNRLLinkMacro** (const byte port, const byte i2caddr, const byte macro)
Run NRLLink macro.
- char **WriteNRLLinkBytes** (const byte port, const byte i2caddr, const byte data[])
Write data to NRLLink.
- bool **ReadNRLLinkBytes** (const byte port, const byte i2caddr, byte &data[])
Read data from NRLLink.
- char **MSIRTrain** (const byte port, const byte i2caddr, const byte channel, const byte func)
MSIRTrain function.
- char **MSPFComboDirect** (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)
MSPFComboDirect function.
- char **MSPFComboPWM** (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)
MSPFComboPWM function.
- char **MSPFRawOutput** (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2)
MSPFRawOutput function.
- char **MSPFRepeat** (const byte port, const byte i2caddr, const byte count, const unsigned int delay)
MSPFRepeat function.
- char **MSPFSingleOutputCST** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)

MSPFSingleOutputCST function.

- char **MSPFSingleOutputPWM** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)

MSPFSingleOutputPWM function.

- char **MSPFSinglePin** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont)

MSPFSinglePin function.

- char **MSPFTrain** (const byte port, const byte i2caddr, const byte channel, const byte func)

MSPFTrain function.

- void **MSRCXSetNRLLinkPort** (const byte port, const byte i2caddr)

MSRCXSetIRLinkPort function.

- int **MSRCXBatteryLevel** (void)

MSRCXBatteryLevel function.

- int **MSRCXPoll** (const byte src, const byte value)

MSRCXPoll function.

- int **MSRCXPollMemory** (const unsigned int address)

MSRCXPollMemory function.

- void **MSRCXAbsVar** (const byte varnum, const byte 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.

6.18.1 Detailed Description

Functions for accessing and modifying MindSensors devices.

6.18.2 Function Documentation

6.18.2.1 `char ACCLNxCalibrateX (const byte port, const byte i2caddr)` [`inline`]

Calibrate ACCL-Nx X-axis. Calibrate the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateX.nxc](#).

6.18.2.2 `char ACCLNxCalibrateXEnd (const byte port, const byte i2caddr)` [`inline`]

Stop calibrating ACCL-Nx X-axis. Stop calibrating the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateXEnd.nxc](#).

**6.18.2.3 char ACCLNxCalibrateY (const byte *port*, const byte *i2caddr*)
[inline]**

Calibrate ACCL-Nx Y-axis. Calibrate the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateY.nxc](#).

**6.18.2.4 char ACCLNxCalibrateYEnd (const byte *port*, const byte *i2caddr*)
[inline]**

Stop calibrating ACCL-Nx Y-axis. Stop calibrating the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateYEnd.nxc](#).

**6.18.2.5 char ACCLNxCalibrateZ (const byte *port*, const byte *i2caddr*)
[inline]**

Calibrate ACCL-Nx Z-axis. Calibrate the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateZ.nxc](#).

**6.18.2.6 char ACCLNxCalibrateZEnd (const byte *port*, const byte *i2caddr*)
[inline]**

Stop calibrating ACCL-Nx Z-axis. Stop calibrating the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateZEnd.nxc](#).

**6.18.2.7 char ACCLNxResetCalibration (const byte *port*, const byte *i2caddr*)
[inline]**

Reset ACCL-Nx calibration. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxResetCalibration.nxc](#).

**6.18.2.8 byte ACCLNxSensitivity (const byte *port*, const byte *i2caddr*)
[inline]**

Read ACCL-Nx sensitivity value. Read the mindsensors ACCL-Nx sensitivity value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The sensitivity value.

Examples:

[ex_ACCLNxSensitivity.nxc](#).

6.18.2.9 int ACCLNxXOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx X offset value. Read the mindsensors ACCL-Nx sensor's X offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The X offset value.

Examples:

[ex_ACCLNxXOffset.nxc](#).

**6.18.2.10 int ACCLNxXRange (const byte *port*, const byte *i2caddr*)
[inline]**

Read ACCL-Nx X range value. Read the mindsensors ACCL-Nx sensor's X range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The X range value.

Examples:

[ex_ACCLNxXRange.nxc](#).

6.18.2.11 int ACCLNxYOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Y offset value. Read the mindsensors ACCL-Nx sensor's Y offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Y offset value.

Examples:

[ex_ACCLNxYOffset.nxc](#).

**6.18.2.12 int ACCLNxYRange (const byte *port*, const byte *i2caddr*)
[inline]**

Read ACCL-Nx Y range value. Read the mindsensors ACCL-Nx sensor's Y range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Y range value.

Examples:

[ex_ACCLNxYRange.nxc](#).

6.18.2.13 int ACCLNxZOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Z offset value. Read the mindsensors ACCL-Nx sensor's Z offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Z offset value.

Examples:

[ex_ACCLNxZOffset.nxc](#).

6.18.2.14 int ACCLNxZRange (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Z range value. Read the mindsensors ACCL-Nx sensor's Z range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Z range value.

Examples:

[ex_ACCLNxZRange.nxc](#).

6.18.2.15 int DISTNxDistance (const byte *port*, const byte *i2caddr*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The distance value.

Examples:

[ex_DISTNxDistance.nxc](#).

6.18.2.16 char DISTNxGP2D12 (const byte *port*, const byte *i2caddr*) [inline]

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2D12.nxc](#).

**6.18.2.17 char DISTNxGP2D120 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2D120.nxc](#).

**6.18.2.18 char DISTNxGP2YA02 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2YA02. Configure the mindsensors DISTNx sensor as GP2YA02. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2YA02.nxc](#).

**6.18.2.19 char DISTNxGP2YA21 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2YA21. Configure the mindsensors DISTNx sensor as GP2YA21. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2YA21.nxc](#).

**6.18.2.20 int DISTNxMaxDistance (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The maximum distance value.

Examples:

[ex_DISTNxMaxDistance.nxc](#).

**6.18.2.21 int DISTNxMinDistance (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The distance value.

Examples:

[ex_DISTNxMinDistance.nxc](#).

**6.18.2.22 byte DISTNxModuleType (const byte *port*, const byte *i2caddr*)
[inline]**

Read DISTNx module type value. Read the mindsensors DISTNx sensor's module type value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The module type value.

Examples:

[ex_DISTNxModuleType.nxc](#).

**6.18.2.23 byte DISTNxNumPoints (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The num points value.

Examples:

[ex_DISTNxNumPoints.nxc](#).

6.18.2.24 int DISTNxVoltage (const byte *port*, const byte *i2caddr*) [inline]

Read DISTNx voltage value. Read the mindsensors DISTNx sensor's voltage value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The voltage value.

Examples:

[ex_DISTNxVoltage.nxc](#).

6.18.2.25 char MSADPAOff (const byte *port*, const byte *i2caddr*) [inline]

Turn off mindsensors ADPA mode. Turn ADPA mode off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSADPAOff.nxc](#).

6.18.2.26 char MSADPAOn (const byte *port*, const byte *i2caddr*) [inline]

Turn on mindsensors ADPA mode. Turn ADPA mode on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSADPAOn.nxc](#).

6.18.2.27 char MSDeenergize (const byte *port*, const byte *i2caddr*) [inline]

Turn off power to device. Turn power off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSDeenergize.nxc](#).

6.18.2.28 char MSEnergize (const byte *port*, const byte *i2caddr*) [inline]

Turn on power to device. Turn the power on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSEnergize.nxc](#).

6.18.2.29 char MSIRTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [inline]

MSIRTrain function. Control an IR Train receiver set to the specified channel using the mindsensors NRLink device. Valid function values are [TRAIN_FUNC_STOP](#), [TRAIN_FUNC_INCR_SPEED](#), [TRAIN_FUNC_DECR_SPEED](#), and [TRAIN_FUNC_TOGGLE_LIGHT](#). Valid channels are [TRAIN_CHANNEL_1](#) through [TRAIN_CHANNEL_3](#) and [TRAIN_CHANNEL_ALL](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The IR Train channel. See [IR Train channel constants](#).

func The IR Train function. See [PF/IR Train function constants](#)

Returns:

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

Examples:

[ex_MSIRTrain.nxc](#).

6.18.2.30 char MSPFComboDirect (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

MSPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the mindsensors NRLink device. Commands for outa and outb are PF_CMD_STOP, PF_CMD_REV, PF_CMD_FWD, and [PF_CMD_BRAKE](#). Valid channels are PF_CHANNEL_1 through PF_CHANNEL_4. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

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

Returns:

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

Examples:

[ex_MSPFComboDirect.nxc](#).

6.18.2.31 char MSPFComboPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

MSPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Valid output values are [PF_PWM_FLOAT](#), [PF_PWM_FWD1](#), [PF_PWM_FWD2](#), [PF_PWM_FWD3](#), [PF_PWM_FWD4](#), [PF_PWM_FWD5](#), [PF_PWM_FWD6](#), [PF_PWM_FWD7](#), [PF_PWM_BRAKE](#), [PF_PWM_REV7](#), [PF_PWM_REV6](#), [PF_PWM_REV5](#), [PF_PWM_REV4](#), [PF_PWM_REV3](#), [PF_PWM_REV2](#), and [PF_PWM_REV1](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

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

Returns:

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

Examples:

[ex_MSPFComboPWM.nxc](#).

6.18.2.32 char MSPFRawOutput (const byte *port*, const byte *i2caddr*, const byte *nibble0*, const byte *nibble1*, const byte *nibble2*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

nibble0 The first raw data nibble.

nibble1 The second raw data nibble.

nibble2 The third raw data nibble.

Returns:

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

Examples:

[ex_MSPFRawOutput.nxc](#).

6.18.2.33 char MSPFRepeat (const byte *port*, const byte *i2caddr*, const byte *count*, const unsigned int *delay*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

count The number of times to repeat the command.

delay The number of milliseconds to delay between each repetition.

Returns:

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

Examples:

[ex_MSPFRepeat.nxc](#).

6.18.2.34 char MSPFSingleOutputCST (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]

MSPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Valid functions are [PF_CST_CLEAR1_CLEAR2](#), [PF_CST_SET1_CLEAR2](#), [PF_CST_CLEAR1_SET2](#), [PF_CST_SET1_SET2](#), [PF_CST_INCREMENT_PWM](#), [PF_CST_DECREMENT_PWM](#), [PF_CST_FULL_FWD](#), [PF_CST_FULL_REV](#), and [PF_CST_TOGGLE_DIR](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

func The Power Function CST function. See [Power Function CST options constants](#).

Returns:

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

Examples:

[ex_MSPFSingleOutputCST.nxc](#).

6.18.2.35 char MSPFSingleOutputPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]

MSPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Valid functions are [PF_PWM_FLOAT](#), [PF_PWM_FWD1](#), [PF_PWM_FWD2](#), [PF_PWM_FWD3](#), [PF_PWM_FWD4](#), [PF_PWM_FWD5](#), [PF_PWM_FWD6](#), [PF_PWM_FWD7](#), [PF_PWM_BRAKE](#), [PF_PWM_REV7](#), [PF_PWM_REV6](#), [PF_PWM_REV5](#), [PF_PWM_REV4](#), [PF_PWM_REV3](#), [PF_PWM_REV2](#), and [PF_PWM_REV1](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

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

Returns:

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

Examples:

[ex_MSPFSingleOutputPWM.nxc](#).

6.18.2.36 char MSPFSinglePin (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *pin*, const byte *func*, bool *cont*) [inline]

MSPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Select the desired pin using [PF_PIN_C1](#) or [PF_PIN_C2](#). Valid functions are [PF_FUNC_NOCHANGE](#), [PF_FUNC_CLEAR](#), [PF_FUNC_SET](#), and [PF_FUNC_TOGGLE](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). Specify whether the mode by passing true (continuous) or false (timeout) as the final parameter. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

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

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

cont Control whether the mode is continuous or timeout.

Returns:

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

Examples:

[ex_MSPFSinglePin.nxc](#).

6.18.2.37 char MSPFTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [inline]

MSPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device as if it were an IR Train receiver. Valid function values are [TRAIN_FUNC_STOP](#), [TRAIN_FUNC_INCR_SPEED](#), [TRAIN_FUNC_DECR_SPEED](#), and [TRAIN_FUNC_TOGGLE_LIGHT](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

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

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

Returns:

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

Examples:

[ex_MSPFTrain.nxc](#).

6.18.2.38 void MSRCXAbsVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXAbsVar function. Send the AbsVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXAbsVar.nxc](#).

6.18.2.39 void MSRCXAddToDatalog (const byte *src*, const unsigned int *value*) [inline]

MSRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

Parameters:

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

value The RCX value.

Examples:

[ex_MSRCXAddToDatalog.nxc](#).

6.18.2.40 void MSRCXAndVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXAndVar function. Send the AndVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXAndVar.nxc](#).

6.18.2.41 int MSRCXBatteryLevel (void) [inline]

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

Returns:

The RCX battery level.

Examples:

[ex_MSRCXBatteryLevel.nxc](#).

6.18.2.42 void MSRCXBoot (void) [inline]

MSRCXBoot function. Send the Boot command to an RCX.

Examples:

[ex_MSRCXBoot.nxc](#).

6.18.2.43 void MSRCXCalibrateEvent (const byte *evt*, const byte *low*, const byte *hi*, const byte *hyst*) [inline]

MSRCXCalibrateEvent function. Send the CalibrateEvent command to an RCX.

Parameters:

- evt* The event number.
- low* The low threshold.
- hi* The high threshold.
- hyst* The hysterisis value.

Examples:

[ex_MSRCXCalibrateEvent.nxc](#).

6.18.2.44 void MSRCXClearAllEvents (void) [inline]

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

Examples:

[ex_MSRCXClearAllEvents.nxc](#).

6.18.2.45 void MSRCXClearCounter (const byte *counter*) [inline]

MSRCXClearCounter function. Send the ClearCounter command to an RCX.

Parameters:

- counter* The counter to clear.

Examples:

[ex_MSRCXClearCounter.nxc](#).

6.18.2.46 void MSRCXClearMsg (void) [inline]

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

Examples:

[ex_MSRCXClearMsg.nxc](#).

6.18.2.47 void MSRCXClearSensor (const byte *port*) [inline]

MSRCXClearSensor function. Send the ClearSensor command to an RCX.

Parameters:

port The RCX port number.

Examples:

[ex_MSRCXClearSensor.nxc](#).

6.18.2.48 void MSRCXClearSound (void) [inline]

MSRCXClearSound function. Send the ClearSound command to an RCX.

Examples:

[ex_MSRCXClearSound.nxc](#).

6.18.2.49 void MSRCXClearTimer (const byte *timer*) [inline]

MSRCXClearTimer function. Send the ClearTimer command to an RCX.

Parameters:

timer The timer to clear.

Examples:

[ex_MSRCXClearTimer.nxc](#).

6.18.2.50 void MSRCXCreateDatalog (const unsigned int *size*) [inline]

MSRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

Parameters:

size The new datalog size.

Examples:

[ex_MSRCXCreateDatalog.nxc](#).

6.18.2.51 void MSRCXDecCounter (const byte *counter*) [inline]

MSRCXDecCounter function. Send the DecCounter command to an RCX.

Parameters:

counter The counter to decrement.

Examples:

[ex_MSRCXDecCounter.nxc](#).

6.18.2.52 void MSRCXDeleteSub (const byte *s*) [inline]

MSRCXDeleteSub function. Send the DeleteSub command to an RCX.

Parameters:

s The subroutine number to delete.

Examples:

[ex_MSRCXDeleteSub.nxc](#).

6.18.2.53 void MSRCXDeleteSubs (void) [inline]

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

Examples:

[ex_MSRCXDeleteSubs.nxc](#).

6.18.2.54 void MSRCXDeleteTask (const byte *t*) [inline]

MSRCXDeleteTask function. Send the DeleteTask command to an RCX.

Parameters:

t The task number to delete.

Examples:

[ex_MSRCXDeleteTask.nxc](#).

6.18.2.55 void MSRCXDeleteTasks (void) [inline]

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

Examples:

[ex_MSRCXDeleteTasks.nxc](#).

6.18.2.56 void MSRCXDisableOutput (const byte *outputs*) [inline]

MSRCXDisableOutput function. Send the DisableOutput command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXDisableOutput.nxc](#).

6.18.2.57 void MSRCXDivVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXDivVar function. Send the DivVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXDivVar.nxc](#).

6.18.2.58 void MSRCXEnableOutput (const byte *outputs*) [inline]

MSRCXEnableOutput function. Send the EnableOutput command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXEnableOutput.nxc](#).

6.18.2.59 void MSRCXEvent (const byte *src*, const unsigned int *value*) [inline]

MSRCXEvent function. Send the Event command to an RCX.

Parameters:

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

value The RCX value.

Examples:

[ex_MSRCXEvent.nxc](#).

6.18.2.60 void MSRCXFloat (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXFloat.nxc](#).

6.18.2.61 void MSRCXFwd (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXFwd.nxc](#).

6.18.2.62 void MSRCXIncCounter (const byte *counter*) [inline]

MSRCXIncCounter function. Send the IncCounter command to an RCX.

Parameters:

counter The counter to increment.

Examples:

[ex_MSRCXIncCounter.nxc](#).

6.18.2.63 void MSRCXInvertOutput (const byte *outputs*) [inline]

MSRCXInvertOutput function. Send the InvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXInvertOutput.nxc](#).

6.18.2.64 void MSRCXMulVar (const byte *varnum*, const byte *src*, unsigned int *value*) [inline]

MSRCXMulVar function. Send the MulVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXMulVar.nxc](#).

6.18.2.65 void MSRCXMuteSound (void) [inline]

MSRCXMuteSound function. Send the MuteSound command to an RCX.

Examples:

[ex_MSRCXMuteSound.nxc](#).

6.18.2.66 void MSRCXObvertOutput (const byte *outputs*) [inline]

MSRCXObvertOutput function. Send the ObvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXObvertOutput.nxc](#).

6.18.2.67 void MSRCXOff (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXOff.nxc](#).

6.18.2.68 void MSRCXOn (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXOn.nxc](#).

6.18.2.69 void MSRCXOnFor (const byte *outputs*, const unsigned int *ms*) [inline]

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

Parameters:

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

ms The number of milliseconds to leave the outputs on

Examples:

[ex_MSRCXOnFor.nxc](#).

6.18.2.70 void MSRCXOnFwd (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXOnFwd.nxc](#).

6.18.2.71 void MSRCXOnRev (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXOnRev.nxc](#).

6.18.2.72 void MSRCXOrVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXOrVar function. Send the OrVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXOrVar.nxc](#).

6.18.2.73 void MSRCXPBTurnOff (void) [inline]

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

Examples:

[ex_MSRCXPBTurnOff.nxc](#).

6.18.2.74 void MSRCXPing (void) [inline]

MSRCXPing function. Send the Ping command to an RCX.

Examples:

[ex_MSRCXPing.nxc](#).

6.18.2.75 void MSRCXPlaySound (const byte *snd*) [inline]

MSRCXPlaySound function. Send the PlaySound command to an RCX.

Parameters:

snd The sound number to play.

Examples:

[ex_MSRCXPlaySound.nxc](#).

**6.18.2.76 void MSRCXPlayTone (const unsigned int *freq*, const byte *duration*)
[inline]**

MSRCXPlayTone function. Send the PlayTone command to an RCX.

Parameters:

freq The frequency of the tone to play.
duration The duration of the tone to play.

Examples:

[ex_MSRCXPlayTone.nxc](#).

**6.18.2.77 void MSRCXPlayToneVar (const byte *varnum*, const byte *duration*)
[inline]**

MSRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

Parameters:

varnum The variable containing the tone frequency to play.
duration The duration of the tone to play.

Examples:

[ex_MSRCXPlayToneVar.nxc](#).

6.18.2.78 int MSRCXPoll (const byte *src*, const byte *value*) [inline]

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

Parameters:

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

Returns:

The value read from the specified port and value.

Examples:

[ex_MSRCXPoll.nxc](#).

6.18.2.79 int MSRCXPollMemory (const unsigned int *address*) [inline]

MSRCXPollMemory function. Send the PollMemory command to an RCX.

Parameters:

address The RCX memory address.

Returns:

The value read from the specified address.

Examples:

[ex_MSRCXPollMemory.nxc](#).

6.18.2.80 void MSRCXRemote (unsigned int *cmd*) [inline]

MSRCXRemote function. Send the Remote command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXRemote.nxc](#).

6.18.2.81 void MSRCXReset (void) [inline]

MSRCXReset function. Send the Reset command to an RCX.

Examples:

[ex_MSRCXReset.nxc](#).

6.18.2.82 void MSRCXRev (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXRev.nxc](#).

6.18.2.83 void MSRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]

MSRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

Parameters:

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

value The RCX value.

Examples:

[ex_MSRCXSelectDisplay.nxc](#).

6.18.2.84 void MSRCXSelectProgram (const byte *prog*) [inline]

MSRCXSelectProgram function. Send the SelectProgram command to an RCX.

Parameters:

prog The program number to select.

Examples:

[ex_MSRCXSelectProgram.nxc](#).

**6.18.2.85 void MSRCXSendSerial (const byte *first*, const byte *count*)
[inline]**

MSRCXSendSerial function. Send the SendSerial command to an RCX.

Parameters:

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

Examples:

[ex_MSRCXSendSerial.nxc](#).

**6.18.2.86 void MSRCXSet (const byte *dtsrc*, const byte *dstval*, const byte *src*,
unsigned int *value*) [inline]**

MSRCXSet function. Send the Set command to an RCX.

Parameters:

dtsrc The RCX destination source. See [RCX and Scout source constants](#).
dstval The RCX destination value.
src The RCX source. See [RCX and Scout source constants](#).
value The RCX value.

Examples:

[ex_MSRCXSet.nxc](#).

**6.18.2.87 void MSRCXSetDirection (const byte *outputs*, const byte *dir*)
[inline]**

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

Parameters:

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

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

Examples:

[ex_MSRCXSetDirection.nxc](#).

**6.18.2.88 void MSRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)
[inline]**

MSRCXSetEvent function. Send the SetEvent command to an RCX.

Parameters:

evt The event number to set.

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

type The event type.

Examples:

[ex_MSRCXSetEvent.nxc](#).

**6.18.2.89 void MSRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)
[inline]**

MSRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

Parameters:

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

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

Examples:

[ex_MSRCXSetGlobalDirection.nxc](#).

**6.18.2.90 void MSRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)
[inline]**

MSRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

Parameters:

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

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

Examples:

[ex_MSRCXSetGlobalOutput.nxc](#).

**6.18.2.91 void MSRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*,
const byte *pwrval*) [inline]**

MSRCXSetMaxPower function. Send the SetMaxPower command to an RCX.

Parameters:

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

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

pwrval The RCX value.

Examples:

[ex_MSRCXSetMaxPower.nxc](#).

6.18.2.92 void MSRCXSetMessage (const byte *msg*) [inline]

MSRCXSetMessage function. Send the SetMessage command to an RCX.

Parameters:

msg The numeric message to send.

Examples:

[ex_MSRCXSetMessage.nxc](#).

**6.18.2.93 void MSRCXSetNRLinkPort (const byte *port*, const byte *i2caddr*)
[inline]**

MSRCXSetIRLinkPort 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 Lowspeed port before using any of the mindsensors RCX and Scout NRLink functions.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Examples:

[ex_MSRCXSetNRLinkPort.nxc](#).

**6.18.2.94 void MSRCXSetOutput (const byte *outputs*, const byte *mode*)
[inline]**

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

Parameters:

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

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

Examples:

[ex_MSRCXSetOutput.nxc](#).

6.18.2.95 void MSRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]

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

Parameters:

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

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

pwrval The RCX value.

Examples:

[ex_MSRCXSetPower.nxc](#).

6.18.2.96 void MSRCXSetPriority (const byte *p*) [inline]

MSRCXSetPriority function. Send the SetPriority command to an RCX.

Parameters:

p The new task priority.

Examples:

[ex_MSRCXSetPriority.nxc](#).

6.18.2.97 void MSRCXSetSensorMode (const byte *port*, const byte *mode*) [inline]

MSRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

Parameters:

port The RCX sensor port.

mode The RCX sensor mode.

Examples:

[ex_MSRCXSetSensorMode.nxc](#).

6.18.2.98 void MSRCXSetSensorType (const byte *port*, const byte *type*) [inline]

MSRCXSetSensorType function. Send the SetSensorType command to an RCX.

Parameters:

port The RCX sensor port.

type The RCX sensor type.

Examples:

[ex_MSRCXSetSensorType.nxc](#).

6.18.2.99 void MSRCXSetSleepTime (const byte *t*) [inline]

MSRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

Parameters:

t The new sleep time value.

Examples:

[ex_MSRCXSetSleepTime.nxc](#).

6.18.2.100 void MSRCXSetTxPower (const byte *pwr*) [inline]

MSRCXSetTxPower function. Send the SetTxPower command to an RCX.

Parameters:

pwr The IR transmit power level.

Examples:

[ex_MSRCXSetTxPower.nxc](#).

6.18.2.101 void MSRCXSetUserDisplay (const byte *src*, const unsigned int *value*, const byte *precision*) [inline]

MSRCXSetUserDisplay function. Send the SetUserDisplay command to an RCX.

Parameters:

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

value The RCX value.

precision The number of digits of precision.

Examples:

[ex_MSRCXSetUserDisplay.nxc](#).

6.18.2.102 void MSRCXSetVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSetVar function. Send the SetVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXSetVar.nxc](#).

6.18.2.103 void MSRCXSetWatch (const byte *hours*, const byte *minutes*) [inline]

MSRCXSetWatch function. Send the SetWatch command to an RCX.

Parameters:

hours The new watch time hours value.

minutes The new watch time minutes value.

Examples:

[ex_MSRCXSetWatch.nxc](#).

6.18.2.104 void MSRCXSgnVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSgnVar function. Send the SgnVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXSgnVar.nxc](#).

6.18.2.105 void MSRCXStartTask (const byte *t*) [inline]

MSRCXStartTask function. Send the StartTask command to an RCX.

Parameters:

t The task number to start.

Examples:

[ex_MSRCXStartTask.nxc](#).

6.18.2.106 void MSRCXStopAllTasks (void) [inline]

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

Examples:

[ex_MSRCXStopAllTasks.nxc](#).

6.18.2.107 void MSRCXStopTask (const byte *t*) [inline]

MSRCXStopTask function. Send the StopTask command to an RCX.

Parameters:

t The task number to stop.

Examples:

[ex_MSRCXStopTask.nxc](#).

6.18.2.108 void MSRCXSubVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSubVar function. Send the SubVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXSubVar.nxc](#).

6.18.2.109 void MSRCXSumVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSumVar function. Send the SumVar command to an RCX.

Parameters:

varnum The variable number to change.

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

value The RCX value.

Examples:

[ex_MSRCXSumVar.nxc](#).

6.18.2.110 void MSRCXToggle (const byte *outputs*) [inline]

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

Parameters:

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

Examples:

[ex_MSRCXToggle.nxc](#).

6.18.2.111 void MSRCXUnlock (void) [inline]

MSRCXUnlock function. Send the Unlock command to an RCX.

Examples:

[ex_MSRCXUnlock.nxc](#).

6.18.2.112 void MSRCXUnmuteSound (void) [inline]

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

Examples:

[ex_MSRCXUnmuteSound.nxc](#).

6.18.2.113 int MSReadValue (const byte *port*, const byte *i2caddr*, const byte *reg*, const byte *numbytes*) [inline]

Read a mindsensors device value. Read a one 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](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

reg The device register to read.

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

Returns:

The function call result.

Examples:

[ex_MSReadValue.nxc](#).

6.18.2.114 void MSScoutCalibrateSensor (void) [inline]

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

Examples:

[ex_MSScoutCalibrateSensor.nxc](#).

6.18.2.115 void MSScoutMuteSound (void) [inline]

MSScoutMuteSound function. Send the MuteSound command to a Scout.

Examples:

[ex_MSScoutMuteSound.nxc](#).

6.18.2.116 void MSScoutSelectSounds (const byte grp) [inline]

MSScoutSelectSounds function. Send the SelectSounds command to a Scout.

Parameters:

grp The Scout sound group to select.

Examples:

[ex_MSScoutSelectSounds.nxc](#).

**6.18.2.117 void MSScoutSendVLL (const byte *src*, const unsigned int *value*)
[inline]**

MSScoutSendVLL function. Send the SendVLL command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_MSScoutSendVLL.nxc](#).

6.18.2.118 void MSScoutSetCounterLimit (const byte *ctr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetCounterLimit function. Send the SetCounterLimit command to a Scout.

Parameters:

ctr The counter for which to set the limit.

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

value The Scout value.

Examples:

[ex_MSScoutSetCounterLimit.nxc](#).

6.18.2.119 void MSScoutSetEventFeedback (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_MSScoutSetEventFeedback.nxc](#).

6.18.2.120 void MSScoutSetLight (const byte *x*) [inline]

MSScoutSetLight function. Send the SetLight command to a Scout.

Parameters:

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

Examples:

[ex_MSScoutSetLight.nxc](#).

6.18.2.121 void MSScoutSetScoutMode (const byte *mode*) [inline]

MSScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

Parameters:

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

Examples:

[ex_MSScoutSetScoutMode.nxc](#).

6.18.2.122 void MSScoutSetScoutRules (const byte *m*, const byte *t*, const byte *l*, const byte *tm*, const byte *fx*) [inline]

MSScoutSetScoutRules function. Send the SetScoutRules command to a Scout.

Parameters:

m Scout motion rule. See [Scout motion rule constants](#).

t Scout touch rule. See [Scout touch rule constants](#).

l Scout light rule. See [Scout light rule constants](#).

tm Scout transmit rule. See [Scout transmit rule constants](#).

fx Scout special effects rule. See [Scout special effect constants](#).

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

6.18.2.123 void MSScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

Parameters:

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

Examples:

[ex_MSScoutSetSensorClickTime.nxc](#).

6.18.2.124 void MSScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

Parameters:

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

Examples:

[ex_MSScoutSetSensorHysteresis.nxc](#).

6.18.2.125 void MSScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

Parameters:

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

Examples:

[ex_MSScoutSetSensorLowerLimit.nxc](#).

6.18.2.126 void MSScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

Parameters:

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

value The Scout value.

Examples:

[ex_MSScoutSetSensorUpperLimit.nxc](#).

6.18.2.127 void MSScoutSetTimerLimit (const byte *tmr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetTimerLimit function. Send the SetTimerLimit command to a Scout.

Parameters:

tmr The timer for which to set a limit.

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

value The Scout value.

Examples:

[ex_MSScoutSetTimerLimit.nxc](#).

6.18.2.128 void MSScoutUnmuteSound (void) [inline]

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

Examples:

[ex_MSScoutUnmuteSound.nxc](#).

6.18.2.129 char NRLink2400 (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in 2400 baud mode. Configure the mindsensors NRLink device in 2400 baud mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLink2400.nxc](#).

6.18.2.130 char NRLink4800 (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in 4800 baud mode. Configure the mindsensors NRLink device in 4800 baud mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLink4800.nxc](#).

6.18.2.131 char NRLinkFlush (const byte *port*, const byte *i2caddr*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkFlush.nxc](#).

**6.18.2.132 char NRLinkIRLong (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkIRLong.nxc](#).

**6.18.2.133 char NRLinkIRShort (const byte *port*, const byte *i2caddr*)
[inline]**

Configure NRLink in IR short mode. Configure the mindsensors NRLink device in IR short mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkIRShort.nxc](#).

**6.18.2.134 char NRLinkSetPF (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkSetPF.nxc](#).

**6.18.2.135 char NRLinkSetRCX (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkSetRCX.nxc](#).

**6.18.2.136 char NRLinkSetTrain (const byte *port*, const byte *i2caddr*)
[inline]**

Configure NRLink in IR train mode. Configure the mindsensors NRLink device in IR train mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkSetTrain.nxc](#).

**6.18.2.137 byte NRLinkStatus (const byte *port*, const byte *i2caddr*)
[inline]**

Read NRLink status. Read the status of the mindsensors NRLink device. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The mindsensors NRLink status.

Examples:

[ex_NRLinkStatus.nxc](#).

**6.18.2.138 char NRLinkTxRaw (const byte *port*, const byte *i2caddr*)
[inline]**

Configure NRLink in raw IR transmit mode. Configure the mindsensors NRLink device in raw IR transmit mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkTxRaw.nxc](#).

**6.18.2.139 char PSPNxAnalog (const byte & *port*, const byte & *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_PSPNxAnalog.nxc](#).

**6.18.2.140 char PSPNxDigital (const byte & *port*, const byte & *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_PSPNxDigital.nxc](#).

6.18.2.141 bool ReadNRLinkBytes (const byte *port*, const byte *i2caddr*, byte & *data*[]) [inline]

Read data from NRLink. Read data from the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

data A byte array that will contain the data read from the device on output.

Returns:

The function call result.

Examples:

[ex_ReadNRLinkBytes.nxc](#).

6.18.2.142 bool ReadSensorMSAccel (const byte *port*, const byte *i2caddr*, int & *x*, int & *y*, int & *z*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

x The output x-axis acceleration.

y The output y-axis acceleration.

z The output z-axis acceleration.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSAccel.nxc](#).

**6.18.2.143 bool ReadSensorMSPlayStation (const byte *port*, const byte *i2caddr*,
byte & *btnset1*, byte & *btnset2*, byte & *xleft*, byte & *yleft*, byte &
xright, byte & *yright*) [inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

btnset1 The button set 1 values. See [MindSensors PSP-Nx button set 1 constants](#).

btnset2 The button set 2 values. See [MindSensors PSP-Nx button set 2 constants](#).

xleft The left joystick x value.

yleft The left joystick y value.

xright The right joystick x value.

yright The right joystick y value.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSPlayStation.nxc](#).

6.18.2.144 bool ReadSensorMSRTClock (const byte *port*, byte & *sec*, byte & *min*, byte & *hrs*, byte & *dow*, byte & *date*, byte & *month*, byte & *year*) [inline]

Read mindsensors RTClock values. Read real-time clock values from the Mindsensors RTClock sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

sec The seconds.

min The minutes.

hrs The hours.

dow The day of week number.

date The day.

month The month.

year The year.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSRTClock.nxc](#).

6.18.2.145 bool ReadSensorMSTilt (const byte *port*, const byte *i2caddr*, byte & *x*, byte & *y*, byte & *z*) [inline]

Read mindsensors tilt values. Read X, Y, and Z axis tilt values from the mindsensors tilt sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

- x* The output x-axis tilt.
- y* The output y-axis tilt.
- z* The output z-axis tilt.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSTilt.nxc](#).

6.18.2.146 char RunNRLinkMacro (const byte *port*, const byte *i2caddr*, const byte *macro*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

macro The address of the macro to execute.

Returns:

The function call result.

Examples:

[ex_RunNRLinkMacro.nxc](#).

6.18.2.147 int SensorMSCompass (const byte & *port*, const byte *i2caddr*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The mindsensors compass value

Examples:

[ex_SensorMSCompass.nxc](#).

6.18.2.148 int SensorMSDROD (const byte & *port*) [inline]

Read mindsensors DROD value. Return the Mindsensors DROD sensor value.

Parameters:

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

Returns:

The mindsensors DROD value

Examples:

[ex_SensorMSDROD.nxc](#).

6.18.2.149 int SensorMSPressure (const byte & *port*) [inline]

Read mindsensors pressure sensor. Read the pressure sensor value of the mindsensors pressure sensor on the specified port.

Parameters:

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

Returns:

The pressure reading.

Examples:

[ex_SensorMSPressure.nxc](#).

6.18.2.150 int SensorMSPressureRaw (const byte & *port*) [inline]

Read mindsensors raw pressure value. Return the Mindsensors pressure sensor raw value.

Parameters:

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

Returns:

The mindsensors raw pressure value

Examples:

[ex_SensorMSPressureRaw.nxc](#).

6.18.2.151 char SetACCLNxSensitivity (const byte *port*, const byte *i2caddr*, byte *slevel*) [inline]

Set ACCL-Nx sensitivity. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

slevel The sensitivity level. See [MindSensors ACCL-Nx sensitivity level constants](#).

Returns:

The function call result.

Examples:

[ex_SetACCLNxSensitivity.nxc](#).

6.18.2.152 void SetSensorMSDROD (const byte & *port*, bool *bActive*) [inline]

Configure a mindsensors DROD sensor. Configure the specified port for a mindsensors DROD sensor.

Parameters:

port The port to configure. See [Input port constants](#).

bActive A flag indicating whether to configure the sensor in active or inactive mode.

Examples:

[ex_SetSensorMSDROD.nxc](#).

6.18.2.153 void SetSensorMSPressure (const byte & *port*) [inline]

Configure a mindsensors pressure sensor. Configure the specified port for a mindsensors pressure sensor.

Parameters:

port The port to configure. See [Input port constants](#).

Examples:

[ex_SetSensorMSPressure.nxc](#).

6.18.2.154 char WriteNRLinkBytes (const byte *port*, const byte *i2caddr*, const byte *data*[]) [inline]

Write data to NRLink. Write data to the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

data A byte array containing the data to write.

Returns:

The function call result.

Examples:

[ex_WriteNRLinkBytes.nxc](#).

6.19 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 **RICOOpDescription**(*_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 **RICOOpCopyBits**(*_CopyOptions*, *_DataAddr*, *_SrcRect*, *_DstPoint*) 18,
0, 3, 0, (*_CopyOptions*)&0xFF, (*_CopyOptions*)>>8, (*_DataAddr*)&0xFF, (*_DataAddr*)>>8, *_SrcRect*, *_DstPoint*
Output an RIC CopyBits opcode.
- #define **RICOOpPixel**(*_CopyOptions*, *_Point*, *_Value*) 10, 0, 4, 0, (*_CopyOptions*)&0xFF, (*_CopyOptions*)>>8, *_Point*, (*_Value*)&0xFF, (*_Value*)>>8
Output an RIC Pixel opcode.
- #define **RICOOpLine**(*_CopyOptions*, *_Point1*, *_Point2*) 12, 0, 5, 0, (*_CopyOptions*)&0xFF, (*_CopyOptions*)>>8, *_Point1*, *_Point2*
Output an RIC Line opcode.
- #define **RICOOpRect**(*_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 **RICOOpCircle**(*_CopyOptions*, *_Point*, *_Radius*) 10, 0, 7, 0, (*_CopyOptions*)&0xFF, (*_CopyOptions*)>>8, *_Point*, (*_Radius*)&0xFF, (*_Radius*)>>8

Output an RIC Circle opcode.

- #define **RICOOpNumBox**(_CopyOptions, _Point, _Value) 10, 0, 8, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_Value)&0xFF, (_Value)>>8

Output an RIC NumBox opcode.

- #define **RICOOpSprite**(_DataAddr, _Rows, _BytesPerRow, _SpriteData) ((-_Rows*_BytesPerRow)+(_Rows*_BytesPerRow)%2)+8)>>8, ((-_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 **RICOOpVarMap**(_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 **RICOOpPolygon**(_CopyOptions, _Count, _ThePoints) ((-_Count*4)+6)&0xFF, ((-_Count*4)+6)>>8, 10, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, (_Count)&0xFF, (_Count)>>8, _ThePoints

Output an RIC Polygon opcode.

- #define **RICPolygonPoints**(_pPoint1, _pPoint2,...) _pPoint1, _pPoint2, __VA_ARGS__

Output RIC polygon points.

- #define **RICOPEllipse**(_CopyOptions, _Point, _RadiusX, _RadiusY) 12, 0, 9, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_RadiusX)&0xFF, (_RadiusX)>>8, (_RadiusY)&0xFF, (_RadiusY)>>8

Output an RIC Ellipse opcode.

6.19.1 Detailed Description

Macro wrappers for use in defining RIC byte arrays.

6.19.2 Define Documentation

6.19.2.1 #define **RICArg**(_arg) ((_arg)|0x1000)

Output an RIC parameterized argument.

Parameters:

_arg The argument that you want to parameterize.

Examples:

[ex_dispgaoutex.nxc](#).

6.19.2.2 #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](#).

**6.19.2.3 #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](#).

**6.19.2.4 #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.

**6.19.2.5 #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.

**6.19.2.6 #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](#).

**6.19.2.7 #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.

**6.19.2.8 #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](#).

```
6.19.2.9 #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](#).

```
6.19.2.10 #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.

```
6.19.2.11 #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](#).

**6.19.2.12 #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.

**6.19.2.13 #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).

**6.19.2.14 #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](#).

```
6.19.2.15 #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.

```
6.19.2.16 #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](#).

```
6.19.2.17 #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](#).

**6.19.2.18 #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](#).

**6.19.2.19 #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

6.19.2.20 #define RICSpriteData(...) __VA_ARGS__

Output RIC sprite data.

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

6.20 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"

6.20.1 Detailed Description

Constant string names for all the NXT firmware modules.

6.20.2 Define Documentation**6.20.2.1 #define ButtonModuleName "Button.mod"**

The button module name

6.20.2.2 #define CommandModuleName "Command.mod"

The command module name

Examples:

[ex_systiomapread.nxc](#).

6.20.2.3 #define CommModuleName "Comm.mod"

The Comm module name

6.20.2.4 #define DisplayModuleName "Display.mod"

The display module name

6.20.2.5 #define InputModuleName "Input.mod"

The input module name.

6.20.2.6 #define IOCtrlModuleName "IOCtrl.mod"

The IOCtrl module name

6.20.2.7 #define LoaderModuleName "Loader.mod"

The Loader module name

6.20.2.8 #define LowSpeedModuleName "Low Speed.mod"

The low speed module name

6.20.2.9 #define OutputModuleName "Output.mod"

The output module name

6.20.2.10 #define SoundModuleName "Sound.mod"

The sound module name

Examples:

[ex_systiomapwrite.nxc](#).

6.20.2.11 #define UIModuleName "Ui.mod"

The Ui module name

6.21 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
- #defineInputModuleID 0x00030001
- #define OutputModuleID 0x00020001
- #define LowSpeedModuleID 0x000B0001
- #define DisplayModuleID 0x000A0001
- #define CommModuleID 0x00050001

6.21.1 Detailed Description

Constant numeric IDs for all the NXT firmware modules.

6.21.2 Define Documentation

6.21.2.1 #define ButtonModuleID 0x00040001

The button module ID

6.21.2.2 #define CommandModuleID 0x00010001

The command module ID

Examples:

[ex_readdressof.nxc](#), and [ex_systomapreadbyid.nxc](#).

6.21.2.3 #define CommModuleID 0x00050001

The Comm module ID

6.21.2.4 #define DisplayModuleID 0x000A0001

The display module ID

6.21.2.5 #defineInputModuleID 0x00030001

The input module ID

6.21.2.6 #define IOCtrlModuleID 0x00060001

The IOCtrl module ID

6.21.2.7 #define LoaderModuleID 0x00090001

The Loader module ID

6.21.2.8 #define LowSpeedModuleID 0x000B0001

The low speed module ID

6.21.2.9 #define OutputModuleID 0x00020001

The output module ID

6.21.2.10 #define SoundModuleID 0x00080001

The sound module ID

Examples:

[ex_sysiomapwritebyid.nxc](#).

6.21.2.11 #define UIModuleID 0x000C0001

The Ui module ID

6.22 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

6.22.1 Detailed Description

Miscellaneous constants for use in NBC and NXC.

6.22.2 Define Documentation**6.22.2.1 #define DEGREES_PER_RADIAN 180/PI**

Used for converting from radians to degrees

6.22.2.2 #define FALSE 0

A false value

6.22.2.3 #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](#).

6.22.2.4 #define PI 3.141593

A constant for PI

Examples:

[ex_dispfnout.nxc](#), and [ex_string.nxc](#).

6.22.2.5 #define RADIANS_PER_DEGREE PI/180

Used for converting from degrees to radians

Examples:

[ex_sin_cos.nxc](#).

6.22.2.6 #define TRUE 1

A true value

Examples:

[ex_SysCommBTConnection.nxc](#).

6.23 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.

6.23.1 Detailed Description

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

6.24 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.

6.24.1 Detailed Description

Documentation for various Standard-C library routines.

6.25 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.

6.25.1 Detailed Description

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

6.25.2 Function Documentation

6.25.2.1 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](#).

6.25.2.2 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](#).

6.25.2.3 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.

6.25.2.4 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.

6.25.2.5 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](#).

6.25.2.6 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.

6.25.2.7 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](#).

6.25.2.8 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](#).

6.25.2.9 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](#).

6.25.2.10 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.

6.25.2.11 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](#).

6.25.2.12 void glEnd () [inline]

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

6.25.2.13 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.

6.25.2.14 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](#).

6.25.2.15 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](#).

6.25.2.16 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](#).

6.25.2.17 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).

6.25.2.18 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](#).

6.25.2.19 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](#).

6.25.2.20 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.

6.25.2.21 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.

6.25.2.22 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.

6.26 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

6.26.1 Detailed Description

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

6.26.2 Define Documentation**6.26.2.1 #define s16 int**

Signed 16 bit type

6.26.2.2 #define s32 long

Signed 32 bit type

6.26.2.3 #define s8 char

Signed 8 bit type

6.26.2.4 #define u16 unsigned int

Unsigned 16 bit type

6.26.2.5 #define u32 unsigned long

Unsigned 32 bit type

6.26.2.6 #define u8 unsigned char

Unsigned 8 bit type

6.27 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

6.27.1 Detailed Description

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

6.27.2 Define Documentation

6.27.2.1 #define S1 0

Input port 1

Examples:

```
ex_ACCLNxCalibrateX.nxc,      ex_ACCLNxCalibrateXEnd.nxc,      ex-
ACCLNxCalibrateY.nxc,          ex_ACCLNxCalibrateYEnd.nxc,      ex-
ACCLNxCalibrateZ.nxc,          ex_ACCLNxCalibrateZEnd.nxc,      ex-
ACCLNxResetCalibration.nxc,   ex_ACCLNxSensitivity.nxc,       ex-
ACCLNxXOffset.nxc,            ex_ACCLNxXRange.nxc,           ex_ACCLNxYOffset.nxc,
ex_ACCLNxYRange.nxc,          ex_ACCLNxZOffset.nxc,           ex_ACCLNxZRange.nxc,
ex_ClearSensor.nxc,           ex_ColorADRaw.nxc,             ex_ColorBoolean.nxc,
ex_ColorCalibration.nxc,      ex_ColorCalibrationState.nxc,  ex-
ColorCalLimits.nxc,           ex_ColorSensorRaw.nxc,           ex_ColorSensorValue.nxc,
ex_CustomSensorActiveStatus.nxc, ex_CustomSensorPercentFullScale.nxc,
ex_CustomSensorZeroOffset.nxc,  ex_DISTNxDistance.nxc,         ex-
DISTNxGP2D12.nxc,             ex_DISTNxGP2D120.nxc,           ex_DISTNxGP2YA02.nxc,
ex_DISTNxGP2YA21.nxc,         ex_DISTNxMaxDistance.nxc,     ex-
DISTNxMinDistance.nxc,        ex_DISTNxModuleType.nxc,       ex-
DISTNxNumPoints.nxc,          ex_DISTNxVoltage.nxc,           ex_GetInput.nxc,    ex-
GetLSInputBuffer.nxc,          ex_GetLSSOutputBuffer.nxc,      ex_HTIITrain.nxc,
ex_HTPFComboDirect.nxc,       ex_HTPFComboPWM.nxc,           ex-
HTPFRawOutput.nxc,            ex_HTPFRepeat.nxc,             ex_HTPFSingleOutputCST.nxc,
ex_HTPFSingleOutputPWM.nxc,   ex_HTPFSinglePin.nxc,           ex_HTPFTrain.nxc,
ex_HTRCXAddToDatalog.nxc,    ex_HTRCXClearSensor.nxc,       ex-
HTRCXSetIRLinkPort.nxc,      ex_HTRCXSetSensorMode.nxc,     ex-
HTRCXSetSensorType.nxc,       ex_I2CBytesReady.nxc,           ex_I2CCheckStatus.nxc,
ex_I2CDeviceId.nxc,          ex_I2CDeviceInfo.nxc,           ex_I2CRead.nxc,    ex-
I2CSendCommand.nxc,          ex_I2CStatus.nxc,              ex_I2CVendorId.nxc,  ex-
I2CVersion.nxc,              ex_i2cwrite.nxc,                ex_LowspeedBytesReady.nxc,
ex_LowspeedCheckStatus.nxc,   ex_LowspeedRead.nxc,            ex-
lowspeedstatus.nxc,           ex_LowspeedWrite.nxc,           ex_LSChannelState.nxc,
ex_LSErrorType.nxc,          ex_LSInputBufferBytesToRx.nxc,  ex-
LSInputBufferInPtr.nxc,       ex_LSInputBufferOutPtr.nxc,      ex_LSMODE.nxc,
ex_LSSOutputBufferBytesToRx.nxc, ex_LSSOutputBufferInPtr.nxc,  ex-
LSSOutputBufferOutPtr.nxc,   ex_MSADPAOff.nxc,               ex_MSADPAOn.nxc,
ex_MSDeenergize.nxc,          ex_MSEnergize.nxc,             ex_MSIRTrain.nxc,
ex_MSPFComboDirect.nxc,      ex_MSPFComboPWM.nxc,           ex-
MSPFRawOutput.nxc,            ex_MSPFRepeat.nxc,             ex_MSPFSingleOutputCST.nxc,
ex_MSPFSingleOutputPWM.nxc,   ex_MSPFSinglePin.nxc,           ex-
MSPFTrain.nxc,               ex_MSRCXAddToDatalog.nxc,      ex_MSRCXClearSensor.nxc,
ex_MSRCXSetNRLLinkPort.nxc,   ex_MSRCXSetSensorMode.nxc,     ex-
MSRCXSetSensorType.nxc,       ex_MSRCXSumVar.nxc,            ex-
MSReadValue.nxc,              ex_NRLink2400.nxc,             ex_NRLink4800.nxc,  ex-
NRLinkFlush.nxc,              ex_NRLinkIRLong.nxc,            ex_NRLinkIRShort.nxc, ex-
NRLinkSetPF.nxc,              ex_NRLinkSetRCX.nxc,            ex_NRLinkSetTrain.nxc, ex-
NRLinkStatus.nxc,              ex_NRLinkTxRaw.nxc,             ex_PSPNxAnalog.nxc,  ex-
PSPNxDigital.nxc,             ex_readi2cregister.nxc,        ex_ReadNRLinkBytes.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_ReadSensorMSPlayStation.nxc,
ex_ReadSensorMSRTClock.nxc,     ex_ReadSensorMSTilt.nxc,         ex_-
readsensorusex.nxc,            ex_RemoteResetScaledValue.nxc,   ex_-
RemoteSetInputMode.nxc,         ex_RunNRLLinkMacro.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_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_-
SetACCLNxSensitivity.nxc,      ex_SetCustomSensorActiveStatus.nxc, ex_-
SetCustomSensorPercentFullScale.nxc, ex_SetCustomSensorZeroOffset.nxc,
ex_SetHTColor2Mode.nxc,         ex_SetHTIRSeeker2Mode.nxc,        ex_SetInput.nxc,
ex_SetSensor.nxc,               ex_SetSensorBoolean.nxc,         ex_SetSensorColorBlue.nxc,
ex_SetSensorColorFull.nxc,      ex_SetSensorColorGreen.nxc,       ex_-
SetSensorColorNone.nxc,         ex_SetSensorColorRed.nxc,         ex_-
SetSensorDigiPinsDirection.nxc, ex_SetSensorDigiPinsOutputLevel.nxc,
ex_SetSensorDigiPinsStatus.nxc, ex_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,        and      ex_-
WriteNRLLinkBytes.nxc.

```

6.27.2.2 #define S2 1

Input port 2

6.27.2.3 #define S3 2

Input port 3

6.27.2.4 #define S4 3

Input port 4

Examples:

[ex_I2CBytes.nxc](#), and [ex_sensorus.nxc](#).

6.28 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

6.28.1 Detailed Description

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

See also:

[SetSensorType\(\)](#)

6.28.2 Define Documentation**6.28.2.1 #define SENSOR_TYPE_COLORBLUE IN_TYPE_COLORBLUE**

NXT 2.0 color sensor with blue light

6.28.2.2 #define SENSOR_TYPE_COLORFULL IN_TYPE_COLORFULL

NXT 2.0 color sensor in full color mode

**6.28.2.3 #define SENSOR_TYPE_COLORGREEN IN_TYPE_-
COLORGREEN**

NXT 2.0 color sensor with green light

6.28.2.4 #define SENSOR_TYPE_COLORNONE IN_TYPE_COLORNONE

NXT 2.0 color sensor with no light

6.28.2.5 #define SENSOR_TYPE_COLORRED IN_TYPE_COLORRED

NXT 2.0 color sensor with red light

6.28.2.6 #define SENSOR_TYPE_CUSTOM IN_TYPE_CUSTOM

NXT custom sensor

6.28.2.7 #define SENSOR_TYPE_HIGHSPEED IN_TYPE_HISPEED

NXT Hi-speed port (only S4)

6.28.2.8 #define SENSOR_TYPE_LIGHT IN_TYPE_REFLECTION

RCX light sensor

**6.28.2.9 #define SENSOR_TYPE_LIGHT_ACTIVE IN_TYPE_LIGHT_-
ACTIVE**

NXT light sensor with light

**6.28.2.10 #define SENSOR_TYPE_LIGHT_INACTIVE IN_TYPE_LIGHT_-
INACTIVE**

NXT light sensor without light

6.28.2.11 #define SENSOR_TYPE_LOWSPEED IN_TYPE_LOWSPEED

NXT I2C digital sensor

Examples:

[ex_RemoteSetInputModule.nxc](#).

**6.28.2.12 #define SENSOR_TYPE_LOWSPEED_9V IN_TYPE_LOWSPEED_-
9V**

NXT I2C digital sensor with 9V power

6.28.2.13 #define SENSOR_TYPE_NONE IN_TYPE_NO_SENSOR

No sensor configured

6.28.2.14 #define SENSOR_TYPE_ROTATION IN_TYPE_ANGLE

RCX rotation sensor

6.28.2.15 #define SENSOR_TYPE_SOUND_DB IN_TYPE_SOUND_DB

NXT sound sensor with dB scaling

Examples:

[ex_SetInput.nxc](#).

6.28.2.16 #define SENSOR_TYPE_SOUND_DBA IN_TYPE_SOUND_DBA

NXT sound sensor with dBA scaling

**6.28.2.17 #define SENSOR_TYPE_TEMPERATURE IN_TYPE_-
TEMPERATURE**

RCX temperature sensor

6.28.2.18 #define SENSOR_TYPE_TOUCH IN_TYPE_SWITCH

NXT or RCX touch sensor

Examples:

[ex_HTRCXSetSensorType.nxc](#), [ex_MSRCXSetSensorType.nxc](#), and [ex_SetSensorType.nxc](#).

6.29 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

6.29.1 Detailed Description

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

See also:

[SetSensorMode\(\)](#)

6.29.2 Define Documentation**6.29.2.1 #define SENSOR_MODE_BOOL IN_MODE_BOOLEAN**

Boolean value (0 or 1)

Examples:

[ex_HTRCXSetSensorMode.nxc](#), and [ex_MSRCXSetSensorMode.nxc](#).

6.29.2.2 #define SENSOR_MODE_CELSIUS IN_MODE_CELSIUS

RCX temperature sensor value in degrees celcius

6.29.2.3 #define SENSOR_MODE_EDGE IN_MODE_TRANSITIONCNT

Counts the number of boolean transitions

6.29.2.4 #define SENSOR_MODE_FAHRENHEIT IN_MODE_FAHRENHEIT

RCX temperature sensor value in degrees fahrenheit

6.29.2.5 #define SENSOR_MODE_PERCENT IN_MODE_PCTFULLSCALE

Scaled value from 0 to 100

6.29.2.6 #define SENSOR_MODE_PULSE IN_MODE_PERIODCOUNTER

Counts the number of boolean periods

6.29.2.7 #define SENSOR_MODE_RAW IN_MODE_RAW

Raw value from 0 to 1023

Examples:

[ex_RemoteSetInputMode.nxc](#), and [ex_SetSensorMode.nxc](#).

6.29.2.8 #define SENSOR_MODE_ROTATION IN_MODE_ANGLESTEP

RCX rotation sensor (16 ticks per revolution)

6.30 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)`

6.30.1 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\(\)](#)

6.30.2 Define Documentation

6.30.2.1 #define _SENSOR_CFG(_type, _mode) (((_type)<<8)+(_mode))

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

**6.30.2.2 #define SENSOR_CELSIUS_SENSOR_CFG(SENSOR_TYPE_-
TEMPERATURE, SENSOR_MODE_CELSIUS)**

RCX temperature sensor in celcius mode

**6.30.2.3 #define SENSOR_COLORBLUE_SENSOR_CFG(SENSOR_TYPE_-
COLORBLUE, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (blue) in percent mode

**6.30.2.4 #define SENSOR_COLORFULL_SENSOR_CFG(SENSOR_TYPE_-
COLORFULL, SENSOR_MODE_RAW)**

NXT 2.0 color sensor (full) in raw mode

**6.30.2.5 #define SENSOR_COLORGREEN_SENSOR_CFG(SENSOR_TYPE_-
COLORGREEN, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

**6.30.2.6 #define SENSOR_COLORNONE_SENSOR_CFG(SENSOR_TYPE_-
COLORNONE, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

**6.30.2.7 #define SENSOR_COLORRED_SENSOR_CFG(SENSOR_TYPE_-
COLORRED, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (red) in percent mode

**6.30.2.8 #define SENSOR_EDGE_SENSOR_CFG(SENSOR_TYPE_TOUCH,
SENSOR_MODE_EDGE)**

Touch sensor in edge mode

6.30.2.9 #define SENSOR_FAHRENHEIT_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE, SENSOR_MODE_FAHRENHEIT)

RCX temperature sensor in fahrenheit mode

6.30.2.10 #define SENSOR_LIGHT_SENSOR_CFG(SENSOR_TYPE_LIGHT, SENSOR_MODE_PERCENT)

Light sensor in percent mode

6.30.2.11 #define SENSOR_LOWSPEED_SENSOR_CFG(SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW)

NXT I2C sensor without 9V power in raw mode

6.30.2.12 #define SENSOR_LOWSPEED_9V_SENSOR_CFG(SENSOR_TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)

NXT I2C sensor with 9V power in raw mode

6.30.2.13 #define SENSOR_NXTLIGHT_SENSOR_CFG(SENSOR_TYPE_LIGHT_ACTIVE, SENSOR_MODE_PERCENT)

NXT light sensor in active mode

6.30.2.14 #define SENSOR_PULSE_SENSOR_CFG(SENSOR_TYPE_TOUCH, SENSOR_MODE_PULSE)

Touch sensor in pulse mode

6.30.2.15 #define SENSOR_ROTATION_SENSOR_CFG(SENSOR_TYPE_ROTATION, SENSOR_MODE_ROTATION)

RCX rotation sensor in rotation mode

6.30.2.16 #define SENSOR_SOUND_SENSOR_CFG(SENSOR_TYPE_SOUND_DB, SENSOR_MODE_PERCENT)

NXT sound sensor (dB) in percent mode

**6.30.2.17 #define SENSOR_TOUCH_SENSOR_CFG(SENSOR_TYPE_-
TOUCH, SENSOR_MODE_BOOL)**

Touch sensor in boolean mode

Examples:

[ex_SetSensor.nxc](#).

6.31 Input module types

Types used by various input module functions.

Data Structures

- struct [ColorSensorReadType](#)

Parameters for the ColorSensorRead system call.

6.31.1 Detailed Description

Types used by various input module functions.

6.32 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 &pct-FullScale)
Set percent full scale.
- void **SetCustomSensorZeroOffset** (const byte port, const int &zeroOffset)
Set custom 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 &output-Level)
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.

6.32.1 Detailed Description

Functions for accessing and modifying input module features.

6.32.2 Function Documentation

6.32.2.1 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](#).

6.32.2.2 `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](#).

6.32.2.3 `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](#).

6.32.2.4 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](#).

6.32.2.5 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](#).

6.32.2.6 `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](#).

6.32.2.7 `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](#).

6.32.2.8 `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](#).

6.32.2.9 `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](#).

6.32.2.10 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](#).

6.32.2.11 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](#).

6.32.2.12 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](#).

6.32.2.13 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](#).

6.32.2.14 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](#).

6.32.2.15 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](#).

6.32.2.16 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](#).

6.32.2.17 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](#).

6.32.2.18 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](#).

6.32.2.19 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](#).

6.32.2.20 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](#).

6.32.2.21 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](#).

6.32.2.22 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](#).

6.32.2.23 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](#).

6.32.2.24 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](#).

6.32.2.25 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](#).

6.32.2.26 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](#).

6.32.2.27 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](#).

6.32.2.28 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](#).

6.32.2.29 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](#).

6.32.2.30 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](#).

6.32.2.31 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](#).

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

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

Parameters:

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

zeroOffset The new zero offset value.

Examples:

[ex_SetCustomSensorZeroOffset.nxc](#).

6.32.2.33 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](#).

**6.32.2.34 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](#).

**6.32.2.35 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.

6.32.2.36 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](#).

6.32.2.37 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](#).

6.32.2.38 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](#).

6.32.2.39 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](#).

6.32.2.40 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](#).

6.32.2.41 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](#).

6.32.2.42 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](#).

6.32.2.43 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](#).

**6.32.2.44 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](#).

**6.32.2.45 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_I2CDeviceInfo.nxc](#),
[ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), and [ex_SetSensorLowspeed.nxc](#).

6.32.2.46 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](#).

**6.32.2.47 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](#).

6.32.2.48 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_ReadSensorHTouchMultiplexer.nxc](#), and [ex_SetSensorTouch.nxc](#).

6.32.2.49 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 [Reset-Sensor](#) to give the firmware time to reconfigure the sensor port.

See also:

[SetSensorMode\(\)](#), [SetSensor\(\)](#)

Parameters:

port The port to configure. See [Input port constants](#).

type The desired sensor type. See [Sensor type constants](#).

Examples:

[ex_SetSensorType.nxc](#).

6.32.2.50 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](#).

6.33 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)

6.33.1 Detailed Description

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

6.33.2 Define Documentation**6.33.2.1 #define SENSOR_1 Sensor(S1)**

Read the value of the analog sensor on port S1

6.33.2.2 #define SENSOR_2 Sensor(S2)

Read the value of the analog sensor on port S2

6.33.2.3 #define SENSOR_3 Sensor(S3)

Read the value of the analog sensor on port S3

6.33.2.4 #define SENSOR_4 Sensor(S4)

Read the value of the analog sensor on port S4

6.34 Output module functions

Functions for accessing and modifying output module features.

Functions

- void **SetMotorPwnFreq** (byte n)
Set motor PWN 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.

6.34.1 Detailed Description

Functions for accessing and modifying output module features.

6.34.2 Function Documentation

6.34.2.1 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](#).

6.34.2.2 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](#).

6.34.2.3 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](#).

6.34.2.4 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](#).

6.34.2.5 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](#).

6.34.2.6 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](#).

6.34.2.7 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](#).

6.34.2.8 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](#).

6.34.2.9 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](#).

6.34.2.10 byte MotorPwnFreq () [inline]

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

Returns:

The motor pwn frequency

Examples:

[ex_motorpwnfreq.nxc](#).

6.34.2.11 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](#).

6.34.2.12 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](#).

6.34.2.13 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](#).

6.34.2.14 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](#).

6.34.2.15 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](#).

6.34.2.16 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](#).

6.34.2.17 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](#).

6.34.2.18 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](#).

6.34.2.19 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](#).

6.34.2.20 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](#).

6.34.2.21 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](#).

6.34.2.22 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](#).

6.34.2.23 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](#).

6.34.2.24 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](#).

6.34.2.25 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](#).

6.34.2.26 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](#).

6.34.2.27 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](#).

6.34.2.28 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](#).

6.34.2.29 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](#).

6.34.2.30 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](#).

6.34.2.31 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](#).

6.34.2.32 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](#).

6.34.2.33 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](#).

6.34.2.34 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](#).

6.34.2.35 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](#).

6.34.2.36 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](#).

6.34.2.37 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](#).**6.34.2.38 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](#).**6.34.2.39 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](#).

6.34.2.40 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](#).

6.34.2.41 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.

turnpect Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onrevsyncpid.nxc](#).

6.34.2.42 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](#).

6.34.2.43 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](#).

6.34.2.44 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](#).

6.34.2.45 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](#).

6.34.2.46 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](#).

6.34.2.47 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](#).

6.34.2.48 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](#).

6.34.2.49 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](#).

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

Set motor PWN frequency. Set the motor PWN frequency.

Parameters:

n The motor pwn frequency

Examples:

[ex_setmotorpwnfreq.nxc](#).

6.34.2.51 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](#).

6.35 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.

6.35.1 Detailed Description

Types used by various display module functions.

6.36 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 fontaddr)
Set the display font memory address.
- void **SetDisplayDisplay** (unsigned long dispaddr)
Set the display memory address.
- void **SetDisplayEraseMask** (unsigned long eraseMask)
Set the display erase mask.

- void [SetDisplayFlags](#) (byte flags)
Set the display flags.
- void [SetDisplayTextLinesCenterFlags](#) (byte ctrFlags)
Set the display text lines center flags.
- void [SetDisplayUpdateMask](#) (unsigned long updateMask)
Set the display update mask.
- void [SetDisplayContrast](#) (byte contrast)
Set the display contrast.

6.36.1 Detailed Description

Functions for accessing and modifying display module features.

6.36.2 Function Documentation

6.36.2.1 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](#), and [ex_file_system.nxc](#).

6.36.2.2 void ClearLine (byte *line*) [inline]

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

Parameters:

line The line you want to clear. See [Line number constants](#).

Examples:

[ex_ClearLine.nxc](#).

6.36.2.3 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](#).

6.36.2.4 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](#).

6.36.2.5 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](#).

6.36.2.6 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](#).

6.36.2.7 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](#).

6.36.2.8 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_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_DisplayFont.nxc](#), and [ex_SetDisplayFont.nxc](#).

6.36.2.9 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_dispmiss.nxc](#).

6.36.2.10 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_dispmiss.nxc](#).

6.36.2.11 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](#).

6.36.2.12 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](#).

6.36.2.13 `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](#).

6.36.2.14 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](#).

6.36.2.15 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](#).

6.36.2.16 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](#).

6.36.2.17 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](#).

6.36.2.18 `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_dispout.nxc](#), and [ex_GraphicOut.nxc](#).

**6.36.2.19 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_dispoutex.nxc](#), and [ex_GraphicOutEx.nxc](#).

**6.36.2.20 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](#).

6.36.2.21 `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_dispgaout.nxc](#), [ex_dispgout.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_FlattenVar.nxc](#), [ex_getchar.nxc](#), [ex_isNAN.nxc](#), [ex_labs.nxc](#), [ex_ldiv.nxc](#), [ex_memcmp.nxc](#), [ex_NumOut.nxc](#), [ex_readdressof.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](#), [ex_SysWriteSemData.nxc](#), and [ex_UnflattenVar.nxc](#).

6.36.2.22 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](#), [ex_sin_cos.nxc](#), and [ex_sind_cosd.nxc](#).

6.36.2.23 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](#).

6.36.2.24 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](#).

6.36.2.25 void ResetScreen () [inline]

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

Examples:

[ex_ResetScreen.nxc](#).

6.36.2.26 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](#).

6.36.2.27 void SetDisplayDisplay (unsigned long *dispaddr*) [inline]

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

Parameters:

dispaddr The new display memory address.

Examples:

[ex_dispmiss.nxc](#), and [ex_SetDisplayDisplay.nxc](#).

6.36.2.28 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_dispmiss.nxc](#), and [ex_SetDisplayEraseMask.nxc](#).

6.36.2.29 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_dispmiss.nxc](#), and [ex_SetDisplayFlags.nxc](#).

6.36.2.30 void SetDisplayFont (unsigned long *fontaddr*) [inline]

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

Parameters:

fontaddr The new display font memory address.

Examples:

[ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_DisplayFont.nxc](#), and [ex_SetDisplayFont.nxc](#).

6.36.2.31 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](#).

6.36.2.32 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](#).

6.36.2.33 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_dispmiss.nxc](#), and [ex_SetDisplayTextLinesCenterFlags.nxc](#).

6.36.2.34 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_dispmiss.nxc](#), and [ex_SetDisplayUpdateMask.nxc](#).

6.36.2.35 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](#).

6.36.2.36 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](#).

6.36.2.37 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](#).

6.36.2.38 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_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

6.36.2.39 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](#).

6.36.2.40 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](#).

6.36.2.41 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](#).

6.36.2.42 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](#).

6.36.2.43 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](#).

6.36.2.44 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](#).

6.36.2.45 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](#).

6.36.2.46 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](#).

6.36.2.47 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_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressoffex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#),

`ex_asin.nxc`, `ex_asind.nxc`, `ex_atan.nxc`, `ex_atan2.nxc`, `ex_atan2d.nxc`, `ex_atand.nxc`, `ex_ClearLine.nxc`, `ex_Copy.nxc`, `ex_ctype.nxc`, `ex_delete_data_file.nxc`, `ex_dispgout.nxc`, `ex_DisplayFont.nxc`, `ex_file_system.nxc`, `ex_GetBrickDataAddress.nxc`, `ex_I2CDeviceId.nxc`, `ex_I2CDeviceInfo.nxc`, `ex_I2CVendorId.nxc`, `ex_I2CVersion.nxc`, `ex_isNAN.nxc`, `ex_labs.nxc`, `ex_leftstr.nxc`, `ex_midstr.nxc`, `ex_ReadSensorHTTouchMultiplexer.nxc`, `ex_readdressof.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_SysCommBTOff.nxc`, `ex_SysCommHSCheckStatus.nxc`, `ex_SysCommHSControl.nxc`, `ex_SysCommHSRead.nxc`, `ex_SysComputeCalibValue.nxc`, `ex_SysDatalogWrite.nxc`, `ex_sysfilefindfirst.nxc`, `ex_sysfilefindnext.nxc`, `ex_sysfileread.nxc`, `ex_syslistfiles.nxc`, `ex_sysmessageread.nxc`, `ex_tan.nxc`, `ex_tand.nxc`, `ex_TextOut.nxc`, `util_battery_1.nxc`, `util_battery_2.nxc`, and `util_rpm.nxc`.

6.37 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.

6.37.1 Detailed Description

Types used by various sound module functions.

6.38 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.

6.38.1 Detailed Description

Functions for accessing and modifying sound module features.

6.38.2 Function Documentation

6.38.2.1 `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](#).

6.38.2.2 `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](#).

6.38.2.3 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](#).

<i>aCode</i>	Resulting Sound
SOUND_CLICK	key click sound
SOUND_DOUBLE_BEEP	double beep
SOUND_DOWN	sweep down
SOUND_UP	sweep up
SOUND_LOW_BEEP	error sound
SOUND_FAST_UP	fast sweep up

Parameters:

aCode The system sound to play. See [RCX and Scout sound constants](#).

Examples:

[ex_playsound.nxc](#).

**6.38.2.4 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_file_system.nxc](#), [ex_PlayTone.nxc](#), and [ex_yield.nxc](#).

6.38.2.5 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](#).

6.38.2.6 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](#).

6.38.2.7 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](#).

6.38.2.8 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](#).

6.38.2.9 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](#).

6.38.2.10 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](#).

6.38.2.11 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](#).

6.38.2.12 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](#).

6.38.2.13 void SetSoundVolume (byte *volume*) [inline]

Set sound volume. Set the sound volume.

See also:

[SoundVolume\(\)](#)

Parameters:

volume The new volume

Examples:

[ex_SetSoundVolume.nxc](#).

6.38.2.14 unsigned int SoundDuration () [inline]

Get sound duration. Return the current sound duration.

See also:

[SetSoundDuration\(\)](#)

Returns:

The current sound duration.

Examples:

[ex_SoundDuration.nxc](#).

6.38.2.15 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](#).

6.38.2.16 unsigned int SoundFrequency () [inline]

Get sound frequency. Return the current sound frequency.

See also:

[SetSoundFrequency\(\)](#)

Returns:

The current sound frequency.

Examples:

[ex_SoundFrequency.nxc](#).

6.38.2.17 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](#).

6.38.2.18 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](#).

6.38.2.19 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](#).

6.38.2.20 byte SoundVolume () [inline]

Get volume. Return the current sound volume.

See also:

[SetSoundVolume\(\)](#)

Returns:

The current sound volume.

Examples:

[ex_SoundVolume.nxc](#).

6.38.2.21 byte StopSound () [inline]

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

Returns:

The result

Todo

?

Examples:

[ex_StopSound.nxc](#).

6.38.2.22 void SysSoundGetState (SoundGetType & args) [inline]

Get sound state. This function lets you retrieve information about the sound module state via the [SoundGetType](#) structure.

Parameters:

args The [SoundGetType](#) structure containing the needed parameters.

Examples:

[ex_syssoundgetstate.nxc](#).

6.38.2.23 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](#).

6.38.2.24 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](#).

6.38.2.25 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](#).

6.39 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.

6.39.1 Detailed Description

Types used by various low speed module functions.

6.40 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 [ReadSensorEMeter](#) (const byte port, float &vIn, float &aIn, float &vOut, float &aOut, int &joules, float &wIn, float &wOut)

Read the LEGO EMeter values.

- long **LowspeedStatus** (const byte port, byte &bytesready)
Get lowspeed status.
- long **LowspeedCheckStatus** (const byte port)
Check lowspeed status.
- byte **LowspeedBytesReady** (const byte port)
Get lowspeed bytes ready.
- long **LowspeedWrite** (const byte port, byte retlen, byte buffer[])
Write lowspeed data.
- long **LowspeedRead** (const byte port, byte buflen, byte &buffer[])
Read lowspeed data.
- long **I2CStatus** (const byte port, byte &bytesready)
Get I2C status.
- long **I2CCheckStatus** (const byte port)
Check I2C status.
- byte **I2CBytesReady** (const byte port)
Get I2C bytes ready.
- long **I2CWrite** (const byte port, byte retlen, byte buffer[])
Write I2C data.
- long **I2CRead** (const byte port, byte buflen, byte &buffer[])
Read I2C data.
- long **I2CBytes** (const byte port, byte inbuf[], byte &count, byte &outbuf[])
Perform an I2C write/read transaction.
- char **ReadI2CRegister** (byte port, byte i2caddr, byte reg, byte &out)
Read I2C register.
- char **WriteI2CRegister** (byte port, byte i2caddr, byte reg, byte val)
Write I2C register.
- string **I2CDeviceInfo** (byte port, byte i2caddr, byte info)

Read I2C device information.

- string **I2CVersion** (byte port, byte i2caddr)

Read I2C device version.

- string **I2CVendorId** (byte port, byte i2caddr)

Read I2C device vendor.

- string **I2CDeviceId** (byte port, byte i2caddr)

Read I2C device identifier.

- long **I2CSendCommand** (byte port, byte i2caddr, byte cmd)

Send an I2C command.

6.40.1 Detailed Description

Functions for accessing and modifying low speed module features.

6.40.2 Function Documentation

6.40.2.1 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](#).

6.40.2.2 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](#).

6.40.2.3 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](#).

6.40.2.4 string I2CDeviceId (byte port, byte i2caddr) [inline]

Read I2C device identifier. Read standard I2C device identifier. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device identifier.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

6.40.2.5 string I2CDeviceInfo (byte *port*, byte *i2caddr*, byte *info*) [inline]

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

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

Returns:

A string containing the requested device information.

Examples:

[ex_I2CDeviceInfo.nxc](#).

6.40.2.6 long I2CRead (const byte *port*, byte *buflen*, byte & *buffer*[]) [inline]

Read I2C data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

buflen The initial size of the output buffer.

buffer A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_I2CRead.nxc](#).

6.40.2.7 long I2CSendCommand (byte *port*, byte *i2caddr*, byte *cmd*) [inline]

Send an I2C command. Send a command to an I2C device at the standard command register: [I2C_REG_CMD](#). The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

cmd The command to send to the I2C device.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

Examples:

[ex_I2CSendCommand.nxc](#).

6.40.2.8 long I2CStatus (const byte *port*, byte & *bytesready*) [inline]

Get I2C status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

bytesready The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible return values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while I2CStatus returns [STAT_COMM_PENDING](#).

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [LowspeedStatus](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

Examples:

[ex_I2CStatus.nxc](#).

6.40.2.9 string I2CVendorId (byte *port*, byte *i2caddr*) [inline]

Read I2C device vendor. Read standard I2C device vendor. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device vendor.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

6.40.2.10 string I2CVersion (byte *port*, byte *i2caddr*) [inline]

Read I2C device version. Read standard I2C device version. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device version.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

6.40.2.11 long I2CWrite (const byte *port*, byte *retlen*, byte *buffer*[]) [inline]

Write I2C data. This method starts a transaction to write the bytes contained in the array buffer to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used

where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

retlen The number of bytes that should be returned by the I2C device.

buffer A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_i2cwrite.nxc](#).

6.40.2.12 byte LowspeedBytesReady (const byte *port*) [inline]

Get lowspeed bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful [LowspeedWrite](#) call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedBytesReady.nxc](#).

6.40.2.13 long LowspeedCheckStatus (const byte *port*) [inline]

Check lowspeed status. This method checks the status of the I2C communication on the specified port.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while [LowspeedCheckStatus](#) returns [STAT_COMM_PENDING](#).

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedCheckStatus.nxc](#).

6.40.2.14 long LowspeedRead (const byte *port*, byte *buflen*, byte & *buffer*[]) [inline]

Read lowspeed data. Read the specified number of bytes from the I2C device on the specified port and store the bytes read in the byte array buffer provided. The maximum number of bytes that can be written or read is 16.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used

where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

buflen The initial size of the output buffer.

buffer A byte array that contains the data read from the internal I2C buffer. If the return value is negative then the output buffer will be empty.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSReadType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedRead.nxc](#).

6.40.2.15 long LowspeedStatus (const byte *port*, byte & *bytesready*) [inline]

Get lowspeed status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

bytesready The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while LowspeedStatus returns [STAT_COMM_PENDING](#).

See also:

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [I2CCheckStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

Examples:

[ex_lowspeedstatus.nxc](#).

6.40.2.16 long LowspeedWrite (const byte *port*, byte *retlen*, byte *buffer*[]) [inline]

Write lowspeed data. This method starts a transaction to write the bytes contained in the array *buffer* to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

retlen The number of bytes that should be returned by the I2C device.

buffer A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedWrite.nxc](#).

**6.40.2.17 char ReadI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte & *out*)
[inline]**

Read I2C register. Read a single byte from an I2C device register.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.
i2caddr The I2C device address.
reg The I2C device register from which to read a single byte.
out The single byte read from the I2C device.

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

Examples:

[ex_readi2cregister.nxc](#).

**6.40.2.18 char ReadSensorEMeter (const byte *port*, float & *vIn*, float & *aIn*,
float & *vOut*, float & *aOut*, int & *joules*, float & *wIn*, float & *wOut*)
[inline]**

Read the LEGO EMeter values. Read all the LEGO EMeter register values. They must all be read at once to ensure data coherency.

Parameters:

port The port to which the LEGO EMeter sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.
vIn Input voltage
aIn Input current
vOut Output voltage
aOut Output current
joules The number of joules stored in the EMeter
wIn The number of watts generated
wOut The number of watts consumed

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

6.40.2.19 char ReadSensorUSEx (const byte *port*, byte & *values*[]) [inline]

Read multiple ultrasonic sensor values. Return eight ultrasonic sensor distance values.

Parameters:

port The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

values An array of bytes that will contain the 8 distance values read from the ultrasonic sensor.

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

Examples:

[ex_readsensorusex.nxc](#).

6.40.2.20 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_sensorsus.nxc](#).

**6.40.2.21 char WriteI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte *val*)
[inline]**

Write I2C register. Write a single byte to an I2C device register.

Parameters:

- port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.
- i2caddr* The I2C device address.
- reg* The I2C device register to which to write a single byte.
- val* The byte to write to the I2C device.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

Examples:

[ex_writei2cregister.nxc](#).

6.41 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 **LSSOutputBufferInPtr** (const byte port)
Get I2C output buffer in-pointer.
- byte **LSSOutputBufferOutPtr** (const byte port)
Get I2C output buffer out-pointer.
- byte **LSSOutputBufferBytesToRx** (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.

6.41.1 Detailed Description

Low level functions for accessing low speed module features.

6.41.2 Function Documentation

6.41.2.1 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](#).

6.41.2.2 void GetLSSOutputBuffer (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_GetLSSOutputBuffer.nxc](#).

6.41.2.3 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_LSCChannelState.nxc](#).**6.41.2.4 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](#).**6.41.2.5 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](#).**6.41.2.6 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](#).

6.41.2.7 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](#).

6.41.2.8 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](#).

6.41.2.9 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](#).

6.41.2.10 byte LSOOutputBufferBytesToRx (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_LSOOutputBufferBytesToRx.nxc](#).

6.41.2.11 byte LSOOutputBufferInPtr (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_LSOOutputBufferInPtr.nxc](#).

6.41.2.12 byte LSOOutputBufferOutPtr (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_LSOOutputBufferOutPtr.nxc](#).

6.41.2.13 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](#).

6.41.2.14 byte LSState () [inline]

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

Returns:

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

Examples:

[ex_LSSState.nxc](#).

6.42 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).

6.42.1 Detailed Description

System call functions for accessing low speed module features.

6.42.2 Function Documentation

6.42.2.1 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](#).

6.42.2.2 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](#).

6.42.2.3 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](#).

6.42.2.4 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](#).

6.43 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.

6.43.1 Detailed Description

Types used by various Command module functions.

6.44 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)
Release a mutex.
- void **StartTask** (task t)
Start a task.
- void **StopTask** (task t)
Stop a task.
- void **SetIOMapBytes** (string moduleName, unsigned int offset, unsigned int count, byte data[])
Set IOMap bytes by name.
- void **SetIOMapValue** (string moduleName, unsigned int offset, variant value)
Set IOMap value by name.
- void **GetIOMapBytes** (string moduleName, unsigned int offset, unsigned int count, byte &data[])
Get IOMap bytes by name.
- void **GetIOMapValue** (string moduleName, unsigned int offset, variant &value)
Get IOMap value by name.
- void **GetLowSpeedModuleBytes** (unsigned int offset, unsigned int count, byte &data[])
Get Lowspeed module IOMap bytes.
- void **GetDisplayModuleBytes** (unsigned int offset, unsigned int count, byte &data[])

Get Display module IOMap bytes.

- void **GetCommModuleBytes** (unsigned int offset, unsigned int count, byte &data[])

Get Comm module IOMap bytes.

- void **GetCommandModuleBytes** (unsigned int offset, unsigned int count, byte &data[])

Get Command module IOMap bytes.

- void **SetCommandModuleBytes** (unsigned int offset, unsigned int count, byte data[])

Set Command module IOMap bytes.

- void **SetLowSpeedModuleBytes** (unsigned int offset, unsigned int count, byte data[])

Set Lowspeed module IOMap bytes.

- void **SetDisplayModuleBytes** (unsigned int offset, unsigned int count, byte data[])

Set Display module IOMap bytes.

- void **SetCommModuleBytes** (unsigned int offset, unsigned int count, byte data[])

Set Comm module IOMap bytes.

- void **SetIOMapBytesByID** (unsigned long moduleId, unsigned int offset, unsigned int count, byte data[])

Set IOMap bytes by ID.

- void **SetIOMapValueByID** (unsigned long moduleId, unsigned int offset, variant value)

Set IOMap value by ID.

- void **GetIOMapBytesByID** (unsigned long moduleId, unsigned int offset, unsigned int count, byte &data[])

Get IOMap bytes by ID.

- void **GetIOMapValueByID** (unsigned long moduleId, unsigned int offset, variant &value)

Get IOMap value by ID.

- void **SetCommandModuleValue** (unsigned int offset, variant value)

Set Command module IOMap value.

- void **SetIOCctrlModuleValue** (unsigned int offset, variant value)

Set IOCctrl module IOMap value.

- void **SetLoaderModuleValue** (unsigned int offset, variant value)

Set Loader module IOMap value.

- void **SetUIModuleValue** (unsigned int offset, variant value)

Set Ui module IOMap value.

- void **SetSoundModuleValue** (unsigned int offset, variant value)

Set Sound module IOMap value.

- void **SetButtonModuleValue** (unsigned int offset, variant value)

Set Button module IOMap value.

- void **SetInputModuleValue** (unsigned int offset, variant value)

Set Input module IOMap value.

- void **SetOutputModuleValue** (unsigned int offset, variant value)

Set Output module IOMap value.

- void **SetLowSpeedModuleValue** (unsigned int offset, variant value)

Set Lowspeed module IOMap value.

- void **SetDisplayModuleValue** (unsigned int offset, variant value)

Set Display module IOMap value.

- void **SetCommModuleValue** (unsigned int offset, variant value)

Set Comm module IOMap value.

- void **GetCommandModuleValue** (unsigned int offset, variant &value)

Get Command module IOMap value.

- void **GetLoaderModuleValue** (unsigned int offset, variant &value)

Get Loader module IOMap value.

- void **GetSoundModuleValue** (unsigned int offset, variant &value)

Get Sound module IOMap value.

- void **GetButtonModuleValue** (unsigned int offset, variant &value)

Get Button module IOMap value.

- void [GetUIModuleValue](#) (unsigned int offset, variant &value)
Get Ui module IOMap value.
- void [GetInputModuleValue](#) (unsigned int offset, variant &value)
Get Input module IOMap value.
- void [GetOutputModuleValue](#) (unsigned int offset, variant &value)
Get Output module IOMap value.
- void [GetLowSpeedModuleValue](#) (unsigned int offset, variant &value)
Get LowSpeed module IOMap value.
- void [GetDisplayModuleValue](#) (unsigned int offset, variant &value)
Get Display module IOMap value.
- void [GetCommModuleValue](#) (unsigned int offset, variant &value)
Get Comm module IOMap value.

6.44.1 Detailed Description

Functions for accessing and modifying Command module features.

6.44.2 Function Documentation

6.44.2.1 void Acquire (*mutex m*) [inline]

Acquire a mutex. Acquire the specified mutex variable. If another task already has acquired the mutex then the current task will be suspended until the mutex is released by the other task. This function is used to ensure that the current task has exclusive access to a shared resource, such as the display or a motor. After the current task has finished using the shared resource the program should call Release to allow other tasks to acquire the mutex.

Parameters:

m The mutex to acquire.

Examples:

[ex_Acquire.nxc](#), and [ex_Release.nxc](#).

6.44.2.2 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](#).

6.44.2.3 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](#).

6.44.2.4 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](#).

6.44.2.5 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](#).

6.44.2.6 void GetButtonModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Button module IOMap value. Read a value from the Button module IOMap structure. You provide the offset into the Button module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

- offset* The number of bytes offset from the start of the IOMap structure where the value should be read. See [Button module IOMAP offsets](#).
- value* A variable that will contain the value read from the IOMap.

6.44.2.7 void GetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Command module IOMap bytes. Read one or more bytes of data from Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the data should be read. See [Command module IOMAP offsets](#).

count The number of bytes to read from the specified Command module IOMap offset.

data A byte array that will contain the data read from the Command module IOMap.

**6.44.2.8 void GetCommandModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Command module IOMap value. Read a value from the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Command module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**6.44.2.9 void GetCommModuleBytes (unsigned int *offset*, unsigned int *count*,
byte & *data*[]) [inline]**

Get Comm module IOMap bytes. Read one or more bytes of data from Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the data should be read. See [Comm module IOMAP offsets](#).

count The number of bytes to read from the specified Comm module IOMap offset.

data A byte array that will contain the data read from the Comm module IOMap.

**6.44.2.10 void GetCommModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Comm module IOMap value. Read a value from the Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Comm module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

6.44.2.11 void GetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Display module IOMap bytes. Read one or more bytes of data from Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the data should be read. See [Display module IOMAP offsets](#).

count The number of bytes to read from the specified Display module IOMap offset.

data A byte array that will contain the data read from the Display module IOMap.

**6.44.2.12 void GetDisplayModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Display module IOMap value. Read a value from the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Display module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**6.44.2.13 void GetInputModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Input module IOMap value. Read a value from the Input module IOMap structure. You provide the offset into the Input module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Input module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**6.44.2.14 void GetIOMapBytes (string *moduleName*, unsigned int *offset*,
unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by name. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

moduleName The module name of the IOMap. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be read

count The number of bytes to read from the specified IOMap offset.

data A byte array that will contain the data read from the IOMap

6.44.2.15 void GetIOMapBytesByID (unsigned long *moduleId*, unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get IOMap bytes by ID. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

- moduleId*** The module ID of the IOMap. See [NXT firmware module IDs](#).
- offset*** The number of bytes offset from the start of the IOMap structure where the data should be read.
- count*** The number of bytes to read from the specified IOMap offset.
- data*** A byte array that will contain the data read from the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

6.44.2.16 void GetIOMapValue (string *moduleName*, unsigned int *offset*, variant & *value*) [inline]

Get IOMap value by name. Read a value from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

Parameters:

- moduleName*** The module name of the IOMap. See [NXT firmware module names](#).
- offset*** The number of bytes offset from the start of the IOMap structure where the value should be read
- value*** A variable that will contain the value read from the IOMap

6.44.2.17 void GetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant & *value*) [inline]

Get IOMap value by ID. Read a value from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

Parameters:

moduleId The module ID of the IOMap. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the value should be read.

value A variable that will contain the value read from the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

**6.44.2.18 void GetLoaderModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Loader module IOMap value. Read a value from the Loader module IOMap structure. You provide the offset into the Loader module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Loader module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**6.44.2.19 void GetLowSpeedModuleBytes (unsigned int *offset*, unsigned int
count, byte & *data*[]) [inline]**

Get Lowspeed module IOMap bytes. Read one or more bytes of data from Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be read. See [Low speed module IOMAP offsets](#).

count The number of bytes to read from the specified Lowspeed module IOMap offset.

data A byte array that will contain the data read from the Lowspeed module IOMap.

6.44.2.20 void GetLowSpeedModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get LowSpeed module IOMap value. Read a value from the LowSpeed module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Low speed module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

6.44.2.21 void GetOutputModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Output module IOMap value. Read a value from the Output module IOMap structure. You provide the offset into the Output module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Output module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

6.44.2.22 void GetSoundModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Sound module IOMap value. Read a value from the Sound module IOMap structure. You provide the offset into the Sound module IOMap structure where you want

to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Sound module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**6.44.2.23 void GetUIModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Ui module IOMap value. Read a value from the Ui module IOMap structure. You provide the offset into the Ui module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Ui module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

6.44.2.24 void Precedes (task *task1*, task *task2*, ..., task *taskN*) [inline]

Declare tasks that this task precedes. Schedule the listed tasks for execution once the current task has completed executing. The tasks will all execute simultaneously unless other dependencies prevent them from doing so. This statement should be used once within a task - preferably at the start of the task definition. Any number of tasks may be listed in the Precedes statement.

Parameters:

task1 The first task to start executing after the current task ends.

task2 The second task to start executing after the current task ends.

taskN The last task to start executing after the current task ends.

Examples:

[alternating_tasks.nxc](#), [ex_Precedes.nxc](#), and [ex_yield.nxc](#).

6.44.2.25 void Release (mutex *m*) [inline]

Acquire a mutex. Release the specified mutex variable. Use this to relinquish a mutex so that it can be acquired by another task. Release should always be called after a matching call to Acquire and as soon as possible after a shared resource is no longer needed.

Parameters:

m The mutex to release.

Examples:

[ex_Acquire.nxc](#), and [ex_Release.nxc](#).

6.44.2.26 long ResetSleepTimer () [inline]

Reset the sleep timer. This function lets you reset the sleep timer.

Returns:

The result of resetting the sleep timer.

Examples:

[ex_ResetSleepTimer.nxc](#).

6.44.2.27 void SetButtonModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Button module IOMap value. Set one of the fields of the Button module IOMap structure to a new value. You provide the offset into the Button module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Button module IOMap structure where the new value should be written. See [Button module IOMAP offsets](#).

value A variable containing the new value to write to the Button module IOMap.

6.44.2.28 void SetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Command module IOMap bytes. Modify one or more bytes of data in the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the data should be written. See [Command module IOMAP offsets](#).

count The number of bytes to write at the specified Command module IOMap offset.

data The byte array containing the data to write to the Command module IOMap.

6.44.2.29 void SetCommandModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Command module IOMap value. Set one of the fields of the Command module IOMap structure to a new value. You provide the offset into the Command module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the new value should be written. See [Command module IOMAP offsets](#).

value A variable containing the new value to write to the Command module IOMap.

6.44.2.30 void SetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Comm module IOMap bytes. Modify one or more bytes of data in an IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the data should be written. See [Comm module IOMAP offsets](#).

count The number of bytes to write at the specified Comm module IOMap offset.

data The byte array containing the data to write to the Comm module IOMap.

**6.44.2.31 void SetCommModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Comm module IOMap value. Set one of the fields of the Comm module IOMap structure to a new value. You provide the offset into the Comm module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the new value should be written. See [Comm module IOMAP offsets](#).

value A variable containing the new value to write to the Comm module IOMap.

**6.44.2.32 void SetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*,
byte *data*[]) [inline]**

Set Display module IOMap bytes. Modify one or more bytes of data in the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the data should be written. See [Display module IOMAP offsets](#).

count The number of bytes to write at the specified Display module IOMap offset.

data The byte array containing the data to write to the Display module IOMap.

**6.44.2.33 void SetDisplayModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Display module IOMap value. Set one of the fields of the Display module IOMap structure to a new value. You provide the offset into the Display module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the new value should be written. See [Display module IOMAP offsets](#).

value A variable containing the new value to write to the Display module IOMap.

**6.44.2.34 void SetInputModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Input module IOMap value. Set one of the fields of the Input module IOMap structure to a new value. You provide the offset into the Input module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Input module IOMap structure where the new value should be written. See [Input module IOMAP offsets](#).

value A variable containing the new value to write to the Input module IOMap.

**6.44.2.35 void SetIOCctrlModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set IOCctrl module IOMap value. Set one of the fields of the IOCctrl module IOMap structure to a new value. You provide the offset into the IOCctrl module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the IOCctrl module IOMap structure where the new value should be written. See [IOCctrl module IOMAP offsets](#).

value A variable containing the new value to write to the IOCctrl module IOMap.

**6.44.2.36 void SetIOMapBytes (string *moduleName*, unsigned int *offset*,
unsigned int *count*, byte *data*[])** [inline]

Set IOMap bytes by name. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

moduleName The module name of the IOMap to modify. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be written

count The number of bytes to write at the specified IOMap offset.

data The byte array containing the data to write to the IOMap

**6.44.2.37 void SetIOMapBytesByID (unsigned long *moduleId*, unsigned int
offset, unsigned int *count*, byte *data*[])** [inline]

Set IOMap bytes by ID. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

moduleId The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be written.

count The number of bytes to write at the specified IOMap offset.

data The byte array containing the data to write to the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

6.44.2.38 void SetIOMapValue (string *moduleName*, unsigned int *offset*, variant *value*) [inline]

Set IOMap value by name. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

moduleName The module name of the IOMap to modify. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the new value should be written

value A variable containing the new value to write to the IOMap

6.44.2.39 void SetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant *value*) [inline]

Set IOMap value by ID. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

moduleId The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the new value should be written.

value A variable containing the new value to write to the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

6.44.2.40 void SetLoaderModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Loader module IOMap value. Set one of the fields of the Loader module IOMap structure to a new value. You provide the offset into the Loader module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Loader module IOMap structure where the new value should be written. See [Loader module IOMAP offsets](#).

value A variable containing the new value to write to the Loader module IOMap.

6.44.2.41 void SetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Lowspeed module IOMap bytes. Modify one or more bytes of data in the Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be written. See [Low speed module IOMAP offsets](#).

count The number of bytes to write at the specified Lowspeed module IOMap offset.

data The byte array containing the data to write to the Lowspeed module IOMap.

6.44.2.42 void SetLowSpeedModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Lowspeed module IOMap value. Set one of the fields of the Lowspeed module IOMap structure to a new value. You provide the offset into the Lowspeed module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the new value should be written. See [Low speed module IOMAP offsets](#).

value A variable containing the new value to write to the Lowspeed module IOMap.

**6.44.2.43 void SetOutputModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Output module IOMap value. Set one of the fields of the Output module IOMap structure to a new value. You provide the offset into the Output module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Output module IOMap structure where the new value should be written. See [Output module IOMAP offsets](#).

value A variable containing the new value to write to the Output module IOMap.

**6.44.2.44 void SetSoundModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Sound module IOMap value. Set one of the fields of the Sound module IOMap structure to a new value. You provide the offset into the Sound module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Sound module IOMap structure where the new value should be written. See [Sound module IOMAP offsets](#).

value A variable containing the new value to write to the Sound module IOMap.

**6.44.2.45 void SetUIModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Ui module IOMap value. Set one of the fields of the Ui module IOMap structure to a new value. You provide the offset into the Ui module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Ui module IOMap structure where the new value should be written. See [Ui module IOMAP offsets](#).
value A variable containing the new value to write to the Ui module IOMap.

6.44.2.46 void StartTask (task *t*) [inline]

Start a task. Start the specified task.

Parameters:

t The task to start.

Examples:

[ex_starttask.nxc](#).

6.44.2.47 void Stop (bool *bvalue*) [inline]

Stop the running program. Stop the running program if *bvalue* is true. This will halt the program completely, so any code following this command will be ignored.

Parameters:

bvalue If this value is true the program will stop executing.

Examples:

[ex_file_system.nxc](#), and [ex_stop.nxc](#).

6.44.2.48 void StopAllTasks () [inline]

Stop all tasks. Stop all currently running tasks. This will halt the program completely, so any code following this command will be ignored.

Examples:

[ex_stopalltasks.nxc](#).

6.44.2.49 void StopTask (task *t*) [inline]

Stop a task. Stop the specified task.

Parameters:

t The task to stop.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_stoptask.nxc](#).

6.44.2.50 void SysCall (byte *funcID*, variant & *args*) [inline]

Call any system function. This generic macro can be used to call any system function. No type checking is performed so you need to make sure you use the correct structure type given the selected system function ID. This is, however, the fastest possible way to call a system function in NXC.

Valid function ID constants are defined in the [System Call function constants](#) group.

Parameters:

funcID The function ID constant corresponding to the function to be called.

args The structure containing the needed parameters.

Examples:

[ex_dispgout.nxc](#), and [ex_syscall.nxc](#).

6.44.2.51 void SysComputeCalibValue (ComputeCalibValueType & *args*) [inline]

Compute calibration values. This function lets you compute calibration values using the values specified via the [ComputeCalibValueType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [ComputeCalibValueType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysComputeCalibValue.nxc](#).

**6.44.2.52 void SysDatalogGetTimes (DatalogGetTimesType & *args*)
[inline]**

Get datalog times. This function lets you get datalog times using the values specified via the [DatalogGetTimesType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [DatalogGetTimesType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysDatalogGetTimes.nxc](#).

6.44.2.53 void SysDatalogWrite (DatalogWriteType & *args*) [inline]

Write to the datalog. This function lets you write to the datalog using the values specified via the [DatalogWriteType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [DatalogWriteType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysDatalogWrite.nxc](#).

6.44.2.54 void SysGetStartTick (GetStartTickType & *args*) [inline]

Get start tick. This function lets you obtain the tick value at the time your program began executing via the [GetStartTickType](#) structure.

Parameters:

args The [GetStartTickType](#) structure receiving results.

Examples:

[ex_sysgetstarttick.nxc](#).

6.44.2.55 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_systiomapread.nxc](#).

**6.44.2.56 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_readdressof.nxc](#), and [ex_systiomapreadbyid.nxc](#).

6.44.2.57 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_systiomapwrite.nxc](#).

**6.44.2.58 void SysIOMapWriteByID (IOMapWriteByIDType & args)
[inline]**

Write to IOMap by identifier. This function lets you write data to a firmware module's IOMap using the values specified via the [IOMapWriteByIDType](#) structure. This function can be as much as three times faster than using SysIOMapWrite since it does not have to do a string lookup using the ModuleName.

Parameters:

args The [IOMapWriteByIDType](#) structure containing the needed parameters.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapwritebyid.nxc](#).

6.44.2.59 void SysKeepAlive (KeepAliveType & args) [inline]

Keep alive. This function lets you reset the sleep timer via the [KeepAliveType](#) structure.

Parameters:

args The [KeepAliveType](#) structure receiving results.

Examples:

[ex_syskeepalive.nxc](#).

6.44.2.60 void SysReadSemData (ReadSemDataType & args) [inline]

Read semaphore data. This function lets you read global motor semaphore data using the values specified via the [ReadSemDataType](#) structure.

Parameters:

args The [ReadSemDataType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysReadSemData.nxc](#).

6.44.2.61 void SysUpdateCalibCacheInfo (UpdateCalibCacheInfoType & args) [inline]

Update calibration cache information. This function lets you update calibration cache information using the values specified via the [UpdateCalibCacheInfoType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [UpdateCalibCacheInfoType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysUpdateCalibCacheInfo.nxc](#).

6.44.2.62 void SysWriteSemData (WriteSemDataType & args) [inline]

Write semaphore data. This function lets you write global motor semaphore data using the values specified via the [WriteSemDataType](#) structure.

Parameters:

args The [WriteSemDataType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysWriteSemData.nxc](#).

6.44.2.63 void Wait (unsigned long ms) [inline]

Wait some milliseconds. Make a task sleep for specified amount of time (in 1000ths of a second).

Parameters:

ms The number of milliseconds to sleep.

Examples:

[alternating_tasks.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_atof.nxc](#), [ex_atoi.nxc](#), [ex_atol.nxc](#), [ex_CircleOut.nxc](#), [ex_ClearLine.nxc](#), [ex_ClearScreen.nxc](#), [ex_contrast.nxc](#), [ex_Copy.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_disptfout.nxc](#), [ex_dispfunc.nxc](#), [ex_dispgaout.nxc](#), [ex_dispgout.nxc](#), [ex_dispgoutex.nxc](#), [ex_DisplayFont.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_file_system.nxc](#), [ex_FlattenVar.nxc](#), [ex_getchar.nxc](#), [ex_I2CDeviceInfo.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_reladdressof.nxc](#), [ex_rightstr.nxc](#), [ex_SensorHTGyro.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [ex_StrCatOld.nxc](#), [ex_StrIndex.nxc](#), [ex_string.nxc](#), [ex_StrLenOld.nxc](#), [ex_StrReplace.nxc](#), [ex strtod.nxc](#), [ex strtol.nxc](#), [ex strtoul.nxc](#), [ex_SubStr.nxc](#), [ex_SysCommBTConnection.nxc](#), [ex_SysCommHSControl.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_SysDatalogGetTimes.nxc](#), [ex_sysdrawfont.nxc](#), [ex_sysdrawgraphicarray.nxc](#), [ex_sysdrawpolygon.nxc](#), [ex_syslistfiles.nxc](#), [ex_UnflattenVar.nxc](#), [ex_wait.nxc](#), [ex_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

6.44.2.64 void Yield () [inline]

Yield to another task. Make a task yield to another concurrently running task.

Examples:

[ex_yield.nxc](#).

6.45 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.

6.45.1 Detailed Description

Functions for use with NXC array types.

6.45.2 Function Documentation

6.45.2.1 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](#).

6.45.2.2 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](#).

6.45.2.3 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](#), [ex_atan2.nxc](#), [ex_atan2d.nxc](#), [ex_syslistfiles.nxc](#), [ex_tan.nxc](#), and [ex_tand.nxc](#).

6.45.2.4 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](#).

6.45.2.5 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](#).

6.45.2.6 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](#).

6.45.2.7 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](#).

6.45.2.8 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](#).

6.45.2.9 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](#).

6.45.2.10 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](#).

6.45.2.11 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](#).

6.45.2.12 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](#).

6.46 IOCtrl module types

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

6.47 IOCtrl module functions

Functions for accessing and modifying IOCtrl module features.

Functions

- void [PowerDown \(\)](#)
Power down the NXT.
- void [SleepNow \(\)](#)
Put the brick to sleep immediately.
- void [RebootInFirmwareMode \(\)](#)
Reboot the NXT in firmware download mode.

6.47.1 Detailed Description

Functions for accessing and modifying IOCtrl module features.

6.47.2 Function Documentation

6.47.2.1 void PowerDown () [inline]

Power down the NXT. This function powers down the NXT. The running program will terminate as a result of this action.

Examples:

[ex_powerdown.nxc](#).

6.47.2.2 void RebootInFirmwareMode () [inline]

Reboot the NXT in firmware download mode. This function lets you reboot the NXT into SAMBA or firmware download mode. The running program will terminate as a result of this action.

Examples:

[ex_rebootinfirmwaremode.nxc](#).

6.47.2.3 void SleepNow () [inline]

Put the brick to sleep immediately. This function lets you immediately put the NXT to sleep. The running program will terminate as a result of this action.

Examples:

[ex_sleepnow.nxc](#).

6.48 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.

6.48.1 Detailed Description

Types used by various Comm module functions.

6.49 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 SetInputModule 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 **HSSoutputBufferInPtr** (void)
Get hi-speed port output buffer in-pointer.
- byte **HSSoutputBufferOutPtr** (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 **SetBTOOutputBuffer** (const byte offset, byte cnt, byte data[])
Set bluetooth output buffer data.
- void **SetBTOOutputBufferInPtr** (byte n)
Set bluetooth output buffer in-pointer.
- void **SetBTOOutputBufferOutPtr** (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.

6.49.1 Detailed Description

Functions for accessing and modifying Comm module features.

6.49.2 Function Documentation

6.49.2.1 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](#).

6.49.2.2 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](#).

6.49.2.3 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](#).

6.49.2.4 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](#).

6.49.2.5 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](#).

6.49.2.6 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](#).

6.49.2.7 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](#).

6.49.2.8 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](#).

6.49.2.9 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](#).

6.49.2.10 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](#).

6.49.2.11 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](#).

6.49.2.12 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](#).

6.49.2.13 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](#).

6.49.2.14 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](#).

6.49.2.15 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](#).

6.49.2.16 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](#).

6.49.2.17 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](#).

6.49.2.18 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](#).

6.49.2.19 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](#).

6.49.2.20 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](#).

6.49.2.21 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](#).

6.49.2.22 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](#).

6.49.2.23 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](#).

6.49.2.24 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](#).

6.49.2.25 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](#).

6.49.2.26 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](#).

6.49.2.27 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](#).

6.49.2.28 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](#).

6.49.2.29 void GetHSSOutputBuffer (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_HSOutputBuffer.nxc](#).

6.49.2.30 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_USBInputBuffer.nxc](#).

6.49.2.31 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_USBOutputBuffer.nxc](#).

6.49.2.32 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](#).

6.49.2.33 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](#).

6.49.2.34 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](#).

6.49.2.35 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](#).

6.49.2.36 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](#).

6.49.2.37 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](#).

6.49.2.38 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](#).

6.49.2.39 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](#).

6.49.2.40 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](#).

**6.49.2.41 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.

**6.49.2.42 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](#).

6.49.2.43 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](#).

6.49.2.44 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.

**6.49.2.45 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](#).

6.49.2.46 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](#).

6.49.2.47 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](#).

6.49.2.48 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](#).

**6.49.2.49 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](#).

6.49.2.50 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](#).

**6.49.2.51 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](#).

6.49.2.52 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](#).

6.49.2.53 char RemoteSetInputMode (byte *conn*, byte *port*, byte *type*, byte *mode*) [inline]

Send a SetInputModule message. Send the SetInputModule 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_RemoteSetInputModule.nxc](#).

6.49.2.54 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](#).

6.49.2.55 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](#).

6.49.2.56 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](#).

6.49.2.57 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](#).

6.49.2.58 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 SysCommH-SControl 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](#).

6.49.2.59 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](#).

6.49.2.60 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](#).

6.49.2.61 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](#).

6.49.2.62 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](#).

6.49.2.63 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](#).

**6.49.2.64 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](#).

6.49.2.65 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](#).

6.49.2.66 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](#).

6.49.2.67 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](#).

6.49.2.68 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](#).

**6.49.2.69 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](#).

**6.49.2.70 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](#).

6.49.2.71 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](#).

6.49.2.72 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](#).

6.49.2.73 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](#).

6.49.2.74 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](#).

6.49.2.75 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](#).

6.49.2.76 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](#).

6.49.2.77 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](#).

6.49.2.78 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](#).

6.49.2.79 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](#).

6.49.2.80 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](#).

6.49.2.81 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](#).

6.49.2.82 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](#).

6.49.2.83 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](#).

6.49.2.84 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](#).

6.49.2.85 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](#).

6.49.2.86 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](#).

6.49.2.87 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](#).

6.49.2.88 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](#).

6.49.2.89 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](#).

6.49.2.90 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](#).

6.49.2.91 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](#).

6.49.2.92 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](#).

6.49.2.93 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](#).

6.49.2.94 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](#).

6.49.2.95 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](#).

6.49.2.96 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](#).

6.49.2.97 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](#).

6.49.2.98 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](#).

6.49.2.99 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](#).

6.49.2.100 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](#).

6.49.2.101 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](#).

6.49.2.102 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](#).

6.49.2.103 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](#).

6.49.2.104 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](#).

6.49.2.105 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](#).

6.49.2.106 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](#).

6.49.2.107 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_syscommexecufunction.nxc](#).

6.49.2.108 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](#).

**6.49.2.109 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](#).

**6.49.2.110 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](#).

**6.49.2.111 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](#).

6.49.2.112 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](#).

6.49.2.113 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](#).

6.49.2.114 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](#).

6.49.2.115 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](#).

6.49.2.116 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](#).

6.49.2.117 byte USBOOutputBufferOutPtr (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_USBOOutputBufferOutPtr.nxc](#).

6.49.2.118 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](#).

6.49.2.119 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](#).

6.49.2.120 byte USBState (void) [inline]

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

Returns:

The USB state.

6.50 Button module types

Types used by various Button module functions.

Data Structures

- struct [ReadButtonType](#)
Parameters for the ReadButton system call.

6.50.1 Detailed Description

Types used by various Button module functions.

6.51 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.

6.51.1 Detailed Description

Functions for accessing and modifying Button module features.

6.51.2 Function Documentation

6.51.2.1 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](#).

6.51.2.2 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](#).

6.51.2.3 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](#).

6.51.2.4 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](#).

6.51.2.5 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](#).

6.51.2.6 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](#).

6.51.2.7 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](#).

6.51.2.8 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.](#)

**6.51.2.9 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.](#)

**6.51.2.10 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.](#)

**6.51.2.11 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](#).

6.51.2.12 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](#).

**6.51.2.13 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](#).

6.51.2.14 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](#).

6.51.2.15 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](#).

6.51.2.16 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](#).

6.52 Ui module types

Types used by various Ui module functions.

Data Structures

- struct [SetSleepTimeoutType](#)
Parameters for the SetSleepTimeout system call.

6.52.1 Detailed Description

Types used by various Ui module functions.

6.53 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.

6.53.1 Detailed Description

Functions for accessing and modifying Ui module features.

6.53.2 Function Documentation

6.53.2.1 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](#).

6.53.2.2 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](#).

6.53.2.3 byte BatteryState (void) [inline]

Get battery state. Return battery state information (0..4).

Returns:

The battery state (0..4)

Examples:

[ex_BatteryState.nxc](#).

6.53.2.4 byte BluetoothState (void) [inline]

Get bluetooth state. Return the bluetooth state.

Returns:

The bluetooth state. See [BluetoothState constants](#).

Examples:

[ex_BluetoothState.nxc](#).

6.53.2.5 byte CommandFlags (void) [inline]

Get command flags. Return the command flags.

Returns:

Command flags. See [CommandFlags constants](#)

Examples:

[ex_CommandFlags.nxc](#).

6.53.2.6 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](#).

6.53.2.7 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](#).

6.53.2.8 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](#).

6.53.2.9 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](#).

6.53.2.10 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](#).

6.53.2.11 void SetBatteryState (byte *state*) [inline]

Set battery state. Set battery state information.

Parameters:

state The desired battery state (0..4).

Examples:

[ex_SetBatteryState.nxc](#).

6.53.2.12 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](#).

6.53.2.13 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](#).

6.53.2.14 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](#).

6.53.2.15 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](#).

6.53.2.16 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](#).

6.53.2.17 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](#).

6.53.2.18 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](#).

6.53.2.19 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](#).

6.53.2.20 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](#).

6.53.2.21 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](#).

6.53.2.22 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](#).

6.53.2.23 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](#).

6.53.2.24 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](#).

6.53.2.25 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](#).

6.53.2.26 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](#).

6.53.2.27 byte UIButton (void) [inline]

Read UI button. Return user interface button information.

Returns:

A UI button value. See [UIButton constants](#).

Examples:

[ex_UIButton.nxc](#).

6.53.2.28 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](#).

6.53.2.29 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](#).

6.53.2.30 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](#).

6.53.2.31 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](#).

6.54 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 [ListFileType](#)

Parameters for the ListFiles system call.

6.54.1 Detailed Description

Types used by various Loader module functions.

6.55 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.

6.55.1 Detailed Description

Functions for accessing and modifying Loader module features.

6.55.2 Function Documentation

6.55.2.1 `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](#), and [ex_file_system.nxc](#).

6.55.2.2 `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](#), and [ex_file_system.nxc](#).

6.55.2.3 `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](#).

6.55.2.4 `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](#).

6.55.2.5 `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_delete_data_file.nxc](#), and [ex_DeleteFile.nxc](#).

**6.55.2.6 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](#).

**6.55.2.7 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](#).

6.55.2.8 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](#).

6.55.2.9 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_file_system.nxc](#), and [ex_OpenFileAppend.nxc](#).

6.55.2.10 `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_file_system.nxc](#), and [ex_OpenFileRead.nxc](#).

6.55.2.11 `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](#).

6.55.2.12 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_file_system.nxc](#), and [ex_Read.nxc](#).

6.55.2.13 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](#).

6.55.2.14 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](#).

6.55.2.15 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](#).

**6.55.2.16 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.

Parameters:

oldname The old filename.

newname The new filename.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_RenameFile.nxc](#).

**6.55.2.17 unsigned int ResizeFile (string *fname*, const unsigned int *newsiz*e)
[inline]**

Resize a file. Resize the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

Parameters:

fname The name of the file to resize.

*newsiz*e The new size for the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:[ex_ResizeFile.nxc.](#)**6.55.2.18 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.](#)**6.55.2.19 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.](#)

6.55.2.20 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](#).

6.55.2.21 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](#).

6.55.2.22 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](#).

6.55.2.23 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](#).

6.55.2.24 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](#).

6.55.2.25 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](#).

6.55.2.26 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](#).

6.55.2.27 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](#).

6.55.2.28 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](#).

6.55.2.29 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](#).

6.55.2.30 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](#).

6.55.2.31 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](#).

6.55.2.32 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](#).

6.55.2.33 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](#).

6.55.2.34 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](#).

6.55.2.35 void SysListFiles (ListFilesType & args) [inline]

List files. This function lets you retrieve a list of files on the NXT using the values specified via the [ListFilesType](#) structure.

Parameters:

args The [ListFilesType](#) structure containing the needed parameters.

Examples:

[ex_syslistfiles.nxc](#).

6.55.2.36 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](#).

6.55.2.37 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_file_system.nxc](#), and [ex_Write.nxc](#).

6.55.2.38 `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](#).

6.55.2.39 `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](#).

**6.55.2.40 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](#).

**6.55.2.41 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](#).

6.55.2.42 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](#).

6.56 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 __RETNVAL__, _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 { 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.

6.56.1 Detailed Description

Standard C cmath API functions.

6.56.2 Define Documentation

6.56.2.1 #define Acos(_X) asm { acos __FLTRETRVAL__, _X }

Compute arc cosine. Computes the arc cosine of _X. Only constants or variables allowed (no expressions).

Deprecated

Use [acos\(\)](#) instead.

Parameters:

_X Floating point value.

Returns:

Arc cosine of _X.

6.56.2.2 #define AcosD(_X) asm {acosd __FLTRETRVAL__, _X }

Compute arc cosine (degrees). Computes the arc cosine of _X. Only constants or variables allowed (no expressions).

Deprecated

Use [acosd\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Arc cosine of $_X$.

6.56.2.3 #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 $_X$.

6.56.2.4 #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 $_X$.

6.56.2.5 #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 $_X$.

6.56.2.6 #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.

6.56.2.7 #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 $_Y/_X$.

6.56.2.8 #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 $_X$.

6.56.2.9 #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 $_X$.

6.56.2.10 #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 $_X$.

6.56.2.11 #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 $_X$.

6.56.2.12 #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 $_X$.

6.56.2.13 #define CoshD(_X) asm { coshd __FLTRETRVAL__, _X }

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [coshd\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Hyperbolic cosine of $_X$.

6.56.2.14 #define Exp(_X) asm { exp __FLTRETRVAL__, _X }

Compute exponential function . Computes the base-e exponential function of $_X$, which is the e number raised to the power $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [exp\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Exponential value of $_X$.

6.56.2.15 #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 $_X$.

6.56.2.16 #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 $_X$.

6.56.2.17 #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 $_X$.

6.56.2.18 #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 $_X$.

6.56.2.19 #define MulDiv32(_A, _B, _C) asm { muldiv __RETNVAL__, _A, _B, _C }

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value. Only constants or variables allowed (no expressions).

Deprecated

Use [muldiv32\(\)](#) instead.

Parameters:

$_A$ 32-bit long value.

_B 32-bit long value.

_C 32-bit long value.

Returns:

The result of multiplying *_A* times *_B* and dividing by *_C*.

6.56.2.20 #define Pow(_Base, _Exponent) asm { pow __FLTRETRVAL__, _Base, _Exponent }

Raise to power. Computes *_Base* raised to the power *_Exponent*. Only constants or variables allowed (no expressions).

Deprecated

Use [pow\(\)](#) instead.

Parameters:

_Base Floating point value.

_Exponent Floating point value.

Returns:

The result of raising *_Base* to the power *_Exponent*.

6.56.2.21 #define Sin(_X) asm { sin __FLTRETRVAL__, _X }

Compute sine. Computes the sine of *_X*. Only constants or variables allowed (no expressions).

Deprecated

Use [sin\(\)](#) instead.

Parameters:

_X Floating point value.

Returns:

Sine of *_X*.

6.56.2.22 #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 $_X$.

6.56.2.23 #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 $_X$.

6.56.2.24 #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 $_X$.

6.56.2.25 #define Sqrt(_X) asm { sqrt __FLTRETRVAL__, _X }

Compute square root. Computes the square root of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [sqrt\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Square root of $_X$.

6.56.2.26 #define Tan(_X) asm { tan __FLTRETRVAL__, _X }

Compute tangent. Computes the tangent of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [tan\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Tangent of $_X$.

6.56.2.27 #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 $_X$.

6.56.2.28 #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 $_X$.

6.56.2.29 #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 $_X$.

6.56.2.30 #define Trunc(_X) asm { trunc __RETRVAL__, _X }

Compute integral part. Computes the integral part of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [trunc\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Integral part of $_X$.

6.56.3 Function Documentation**6.56.3.1 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](#).

6.56.3.2 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](#).

6.56.3.3 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](#).

6.56.3.4 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](#).

6.56.3.5 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](#).

6.56.3.6 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](#).

6.56.3.7 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](#).

6.56.3.8 float atan (float *x*) [inline]

Compute arc tangent (degrees). Computes the principal value of the arc tangent of *x*, expressed in degrees. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use atan2d if you need to determine the quadrant.

Parameters:

x Floating point value.

Returns:

Arc tangent of *x*, in the interval [-90,+90] degrees.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_atand.nxc](#).

6.56.3.9 byte bcd2dec (byte *bcd*) [inline]

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

Parameters:

bcd The value you want to convert from bcd to decimal.

Returns:

The decimal equivalent of the binary coded decimal byte.

Examples:

[ex_bcd2dec.nxc](#).

6.56.3.10 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](#).

6.56.3.11 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_sin_cos.nxc](#).

6.56.3.12 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_sind_cosd.nxc](#).

6.56.3.13 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](#).

6.56.3.14 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.

6.56.3.15 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](#).

6.56.3.16 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](#).

6.56.3.17 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](#).

6.56.3.18 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](#).

6.56.3.19 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](#).

6.56.3.20 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](#).

6.56.3.21 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](#).

6.56.3.22 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.](#)

6.56.3.23 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.](#)

6.56.3.24 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_cos.nxc](#).

6.56.3.25 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_cosd.nxc](#).

6.56.3.26 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](#).

6.56.3.27 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.

6.56.3.28 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](#).

6.56.3.29 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](#).

6.56.3.30 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](#).

6.56.3.31 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](#).

6.56.3.32 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.

6.56.3.33 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_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), and [ex_trunc.nxc](#).

6.57 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.

6.57.1 Detailed Description

Standard C cstdio API functions.

6.57.2 Define Documentation

6.57.2.1 #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](#).

6.57.2.2 #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](#).

6.57.3 Function Documentation

6.57.3.1 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](#).

6.57.3.2 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](#).

6.57.3.3 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](#).

6.57.3.4 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](#).

6.57.3.5 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](#).

6.57.3.6 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](#).

6.57.3.7 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](#).

6.57.3.8 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](#).

6.57.3.9 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](#).

6.57.3.10 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](#).

6.57.3.11 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.](#)**6.57.3.12 int getchar () [inline]**

Get character from stdin. Returns the next character from the standard input (stdin). It is equivalent to getc with stdin as its argument. On the NXT this means wait for a button press and return the value of the button pressed.

Returns:

The pressed button. See [Button name constants](#).

Examples:[ex_getchar.nxc.](#)**6.57.3.13 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.](#)**6.57.3.14 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.

6.57.3.15 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](#).

6.57.3.16 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](#).

6.57.3.17 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](#).

6.58 fseek origin constants

Constants for use in calls to fseek.

Defines

- #define SEEK_SET 0
- #define SEEK_CUR 1
- #define SEEK_END 2

6.58.1 Detailed Description

Constants for use in calls to fseek.

6.58.2 Define Documentation**6.58.2.1 #define SEEK_CUR 1**

Seek from the current file position

Examples:

[ex_fseek.nxc](#).

6.58.2.2 #define SEEK_END 2

Seek from the end of the file

6.58.2.3 #define SEEK_SET 0

Seek from the beginning of the file

Examples:

[ex_sysfileseek.nxc](#).

6.59 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.

6.59.1 Detailed Description

Standard C cstdlib API functions and types.

6.59.2 Function Documentation

6.59.2.1 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](#).

6.59.2.2 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](#).

6.59.2.3 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](#).

6.59.2.4 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](#).

6.59.2.5 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](#).

6.59.2.6 `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](#).

6.59.2.7 long labs (long *n*) [inline]

Absolute value. Return the absolute value of parameter *n*.

Parameters:

n Integral value.

Returns:

The absolute value of *n*.

6.59.2.8 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](#).

6.59.2.9 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](#).

6.59.2.10 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_dspgoutex.nxc](#), [ex_EllipseOut.nxc](#), [ex_file_system.nxc](#), [ex_Random.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [ex_string.nxc](#), [ex_sysdrawellipse.nxc](#), and [ex_wait.nxc](#).

6.59.2.11 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](#).

**6.59.2.12 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](#).

**6.59.2.13 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](#).

6.59.2.14 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](#).

6.60 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.

6.60.1 Detailed Description

Standard C cstdlib API types.

6.61 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 **addressOf** (variant data)

Get the absolute address of a variable.

- unsigned long [reladdressOf](#) (variant data)

Get the relative address of a variable.

- unsigned long [addressOfEx](#) (variant data, bool relative)

Get the absolute or relative address of a variable.

6.61.1 Detailed Description

Standard C cstring API functions.

6.61.2 Function Documentation

6.61.2.1 unsigned long [addressOf](#) (variant *data*) [inline]

Get the absolute address of a variable. Get the absolute address of a variable and return it to the calling routine as an unsigned long value.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

Returns:

The absolute address of the variable.

Examples:

[ex_addressof.nxc](#).

6.61.2.2 unsigned long [addressOfEx](#) (variant *data*, bool *relative*) [inline]

Get the absolute or relative address of a variable. Get the absolute or relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

relative A boolean flag indicating whether you want to get the relative or absolute address.

Returns:

The absolute or relative address of the variable.

Examples:

[ex_addressofex.nxc](#).

6.61.2.3 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](#).

6.61.2.4 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](#).

6.61.2.5 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](#).

6.61.2.6 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](#).

6.61.2.7 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](#), [ex_string.nxc](#), and [ex_UnflattenVar.nxc](#).

6.61.2.8 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_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atan2.nxc](#), [ex_atan2d.nxc](#), [ex_atand.nxc](#), [ex_delete_data_file.nxc](#), [ex_DisplayFont.nxc](#), [ex_file_system.nxc](#), [ex_FormatNum.nxc](#), [ex_GetBrickDataAddress.nxc](#), [ex_readdressof.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_string.nxc](#), [ex_tan.nxc](#), [ex_tand.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

6.61.2.9 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](#).

6.61.2.10 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](#).

6.61.2.11 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](#).

6.61.2.12 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](#).

**6.61.2.13 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](#).

6.61.2.14 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](#).

6.61.2.15 unsigned long readdressOf (variant *data*) [inline]

Get the relative address of a variable. Get the relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

Returns:

The relative address of the variable.

Examples:

[ex_reladdressof.nxc](#).

6.61.2.16 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](#).

6.61.2.17 string streat (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_streat.nxc](#).

6.61.2.18 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_GetBrickDataAddress.nxc](#), [ex_StrCatOld.nxc](#), [ex_string.nxc](#), [ex_StrReplace.nxc](#), and [util_battery_1.nxc](#).

6.61.2.19 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](#).

6.61.2.20 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](#).

6.61.2.21 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](#).

6.61.2.22 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](#).

6.61.2.23 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](#).

**6.61.2.24 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](#).

6.61.2.25 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](#).

**6.61.2.26 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](#).

**6.61.2.27 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](#).

6.61.2.28 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](#).

6.61.2.29 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](#).

**6.61.2.30 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](#).

6.61.2.31 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_FlattenVar.nxc](#), [ex_string.nxc](#), and [ex_UnflattenVar.nxc](#).

6.62 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 **iscntrl** (int c)
Check if character is a control character.
- int **isprint** (int c)
Check if character is printable.
- int **isgraph** (int c)
Check if character has graphical representation.
- int **ispunct** (int c)
Check if character is a punctuation.
- int **isxdigit** (int c)
Check if character is hexadecimal digit.
- int **toupper** (int c)
Convert lowercase letter to uppercase.

- int **tolower** (int c)

Convert uppercase letter to lowercase.

6.62.1 Detailed Description

Standard C ctype API functions.

6.62.2 Function Documentation

6.62.2.1 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](#).

6.62.2.2 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](#).

6.62.2.3 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](#).

6.62.2.4 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](#).

6.62.2.5 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](#).

6.62.2.6 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](#).

6.62.2.7 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](#).

6.62.2.8 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](#).

6.62.2.9 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](#).

6.62.2.10 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](#).

6.62.2.11 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](#).

6.62.2.12 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](#).

6.62.2.13 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](#).

6.63 Array operation constants

Constants for use with the NXC ArrayOp function and the NBC arrop statement.

Defines

- #define [OPARR_SUM](#) 0x00
- #define [OPARR_MEAN](#) 0x01
- #define [OPARR_SUMSQR](#) 0x02
- #define [OPARR_STD](#) 0x03
- #define [OPARR_MIN](#) 0x04
- #define [OPARR_MAX](#) 0x05
- #define [OPARR_SORT](#) 0x06

6.63.1 Detailed Description

Constants for use with the NXC ArrayOp function and the NBC arrop statement.

6.63.2 Define Documentation**6.63.2.1 #define OPARR_MAX 0x05**

Calculate the maximum value of the elements in the numeric input array

Examples:

[ex_ArrayOp.nxc.](#)

6.63.2.2 #define OPARR_MEAN 0x01

Calculate the mean value for the elements in the numeric input array

6.63.2.3 #define OPARR_MIN 0x04

Calculate the minimum value of the elements in the numeric input array

6.63.2.4 #define OPARR_SORT 0x06

Sort the elements in the numeric input array

6.63.2.5 #define OPARR_STD 0x03

Calculate the standard deviation of the elements in the numeric input array

6.63.2.6 #define OPARR_SUM 0x00

Calculate the sum of the elements in the numeric input array

6.63.2.7 #define OPARR_SUMSQR 0x02

Calculate the sum of the squares of the elements in the numeric input array

6.64 System Call function constants

Constants for use in the [SysCall\(\)](#) function or NBC syscall statement.

Defines

- #define `FileOpenRead` 0
- #define `FileOpenWrite` 1
- #define `FileOpenAppend` 2
- #define `FileRead` 3
- #define `FileWrite` 4
- #define `FileClose` 5
- #define `FileResolveHandle` 6
- #define `FileRename` 7
- #define `FileDelete` 8
- #define `SoundPlayFile` 9
- #define `SoundPlayTone` 10
- #define `SoundGetState` 11
- #define `SoundSetState` 12
- #define `DrawText` 13
- #define `DrawPoint` 14
- #define `DrawLine` 15
- #define `DrawCircle` 16
- #define `DrawRect` 17
- #define `DrawGraphic` 18
- #define `SetScreenMode` 19
- #define `ReadButton` 20
- #define `CommLSWrite` 21
- #define `CommLSRead` 22
- #define `CommLSCheckStatus` 23
- #define `RandomNumber` 24
- #define `GetStartTick` 25
- #define `MessageWrite` 26
- #define `MessageRead` 27
- #define `CommBTCheckStatus` 28
- #define `CommBTWrite` 29
- #define `CommBTRead` 30
- #define `KeepAlive` 31
- #define `IOMapRead` 32
- #define `IOMapWrite` 33
- #define `ColorSensorRead` 34
- #define `CommBTOnOff` 35
- #define `CommBTConnection` 36
- #define `CommHSWrite` 37
- #define `CommHSRead` 38
- #define `CommHSCheckStatus` 39
- #define `ReadSemData` 40

- #define WriteSemData 41
- #define ComputeCalibValue 42
- #define UpdateCalibCacheInfo 43
- #define DatalogWrite 44
- #define DatalogGetTimes 45
- #define SetSleepTimeoutVal 46
- #define ListFiles 47
- #define 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

6.64.1 Detailed Description

Constants for use in the `SysCall()` function or NBC syscall statement.

6.64.2 Define Documentation

6.64.2.1 #define ColorSensorRead 34

Read data from the NXT 2.0 color sensor

6.64.2.2 #define CommBTCheckStatus 28

Check the bluetooth status

6.64.2.3 #define CommBTConnection 36

Connect or disconnect to a known bluetooth device

6.64.2.4 #define CommBTOnOff 35

Turn the bluetooth radio on or off

6.64.2.5 #define CommBTRead 30

Read from a bluetooth connection

6.64.2.6 #define CommBTWrite 29

Write to a bluetooth connections

6.64.2.7 #define CommExecuteFunction 81

Execute one of the Comm module's internal functions

6.64.2.8 #define CommHSCheckStatus 39

Check the status of the hi-speed port

6.64.2.9 #define CommHSControl 88

Control the hi-speed port

6.64.2.10 #define CommHSRead 38

Read data from the hi-speed port

6.64.2.11 #define CommHSWrite 37

Write data to the hi-speed port

6.64.2.12 #define CommLSCheckStatus 23

Check the status of a lowspeed (aka I2C) device

6.64.2.13 #define CommLSRead 22

Read from a lowspeed (aka I2C) device

6.64.2.14 #define CommLSWrite 21

Write to a lowspeed (aka I2C) device

6.64.2.15 #define CommLSWriteEx 89

Write to a lowspeed (aka I2C) device with optional restart on read

6.64.2.16 #define ComputeCalibValue 42

Compute a calibration value

6.64.2.17 #define DatalogGetTimes 45

Get datalog timing information

6.64.2.18 #define DatalogWrite 44

Write to the datalog

6.64.2.19 #define DisplayExecuteFunction 80

Execute one of the Display module's internal functions

6.64.2.20 #define DrawCircle 16

Draw a circle on the LCD screen

6.64.2.21 #define DrawEllipse 94

Draw an ellipse on the LCD screen

6.64.2.22 #define DrawFont 95

Draw text using a custom RIC-based font to the LCD screen

6.64.2.23 #define DrawGraphic 18

Draw a graphic image on the LCD screen

6.64.2.24 #define DrawGraphicArray 92

Draw a graphic image from a byte array to the LCD screen

Examples:

[ex_dispout.nxc](#).

6.64.2.25 #define DrawLine 15

Draw a line on the LCD screen

6.64.2.26 #define DrawPoint 14

Draw a single pixel on the LCD screen

6.64.2.27 #define DrawPolygon 93

Draw a polygon on the LCD screen

6.64.2.28 #define DrawRect 17

Draw a rectangle on the LCD screen

6.64.2.29 #define DrawText 13

Draw text to one of 8 LCD lines

Examples:

[ex_syscall.nxc](#).

6.64.2.30 #define FileClose 5

Close the specified file

6.64.2.31 #define FileDelete 8

Delete a file

6.64.2.32 #define FileFindFirst 83

Start a search for a file using a filename pattern

6.64.2.33 #define FileFindNext 84

Continue searching for a file

6.64.2.34 #define FileOpenAppend 2

Open a file for appending to the end of the file

6.64.2.35 #define FileOpenRead 0

Open a file for reading

6.64.2.36 #define FileOpenReadLinear 87

Open a linear file for reading

6.64.2.37 #define FileOpenWrite 1

Open a file for writing (creates a new file)

6.64.2.38 #define FileOpenWriteLinear 85

Open a linear file for writing

6.64.2.39 #define FileOpenWriteNonLinear 86

Open a non-linear file for writing

6.64.2.40 #define FileRead 3

Read from the specified file

6.64.2.41 #define FileRename 7

Rename a file

6.64.2.42 #define FileResize 91

Resize a file (not yet implemented)

6.64.2.43 #define FileResolveHandle 6

Get a file handle for the specified filename if it is already open

6.64.2.44 #define FileSeek 90

Seek to a specific position in an open file

6.64.2.45 #define FileWrite 4

Write to the specified file

6.64.2.46 #define GetStartTick 25

Get the current system tick count

6.64.2.47 #define IOMapRead 32

Read data from one of the firmware module's IOMap structures using the module's name

6.64.2.48 #define IOMapReadByID 78

Read data from one of the firmware module's IOMap structures using the module's ID

6.64.2.49 #define IOMapWrite 33

Write data to one of the firmware module's IOMap structures using the module's name

6.64.2.50 #define IOMapWriteByID 79

Write data to one of the firmware module's IOMap structures using the module's ID

6.64.2.51 #define KeepAlive 31

Reset the NXT sleep timer

6.64.2.52 #define ListFiles 47

List files that match the specified filename pattern

6.64.2.53 #define LoaderExecuteFunction 82

Execute one of the Loader module's internal functions

6.64.2.54 #define MessageRead 27

Read a message from a mailbox

6.64.2.55 #define MessageWrite 26

Write a message to a mailbox

6.64.2.56 #define RandomNumber 24

Generate a random number

6.64.2.57 #define ReadButton 20

Read the current button state

6.64.2.58 #define ReadSemData 40

Read motor semaphore data

6.64.2.59 #define SetScreenMode 19

Set the screen mode

6.64.2.60 #define SetSleepTimeoutVal 46

Set the NXT sleep timeout value

6.64.2.61 #define SoundGetState 11

Get the current sound module state

6.64.2.62 #define SoundPlayFile 9

Play a sound or melody file

6.64.2.63 #define SoundPlayTone 10

Play a simple tone with the specified frequency and duration

6.64.2.64 #define SoundSetState 12

Set the sound module state

6.64.2.65 #define UpdateCalibCacheInfo 43

Update sensor calibration cache information

6.64.2.66 #define WriteSemData 41

Write motor semaphore data

6.65 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

6.65.1 Detailed Description

Line numbers for use with DrawText system function.

See also:

[SysDrawText\(\)](#), [TextOut\(\)](#), [NumOut\(\)](#)

6.65.2 Define Documentation

6.65.2.1 #define LCD_LINE1 56

The 1st line of the LCD screen

Examples:

```
ex_acos.nxc,      ex_acosd.nxc,      ex_addressof.nxc,      ex_addressofex.nxc,  
ex_ArrayMax.nxc,   ex_ArrayMean.nxc,   ex_ArrayMin.nxc,   ex_-  
ArrayOp.nxc,     ex_ArraySort.nxc,   ex_ArrayStd.nxc,   ex_ArraySum.nxc,  
ex_ArraySumSqr.nxc, ex_asin.nxc,     ex_asind.nxc,     ex_atan.nxc,     ex_-  
atand.nxc,       ex_atof.nxc,     ex_atoi.nxc,     ex_atol.nxc,     ex_ButtonPressed.nxc,  
ex_ClearLine.nxc, ex_contrast.nxc,  ex_Copy.nxc,    ex_ctype.nxc,   ex_delete_-  
data_file.nxc,   ex_dispgaout.nxc, ex_dispout.nxc,  ex_DisplayFont.nxc,  
ex_dispmisc.nxc, ex_div.nxc,      ex_file_system.nxc, ex_FlattenVar.nxc,  
ex_GetBrickDataAddress.nxc, ex_getchar.nxc,  ex_I2CDeviceId.nxc,  
ex_I2CDeviceInfo.nxc, ex_I2CVendorId.nxc, ex_I2CVersion.nxc, ex_-  
isNAN.nxc,       ex_labs.nxc,     ex_ldiv.nxc,    ex_leftstr.nxc,  ex_memcmp.nxc,  
ex_midstr.nxc,   ex_NumOut.nxc,   ex_ReadSensorHTTouchMultiplexer.nxc,  
ex_readdressof.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_SysCommBTOFF.nxc,           ex_SysCommHSCheckStatus.nxc,  
ex_SysCommHSControl.nxc,       ex_SysCommHSRead.nxc,  ex_-  
SysComputeCalibValue.nxc,     ex_SysDatalogWrite.nxc, ex_sysdrawtext.nxc,  
ex_sysfilefindfirst.nxc,     ex_sysfilefindnext.nxc, ex_sysfileread.nxc, ex_-  
sysfilewrite.nxc,  ex_sysmessageread.nxc,  ex_SysReadSemData.nxc, ex_-  
SysUpdateCalibCacheInfo.nxc, ex_SysWriteSemData.nxc,  and ex_-  
UnflattenVar.nxc.
```

6.65.2.2 #define LCD_LINE2 48

The 2nd line of the LCD screen

Examples:

`ex_acos.nxc, ex_acosd.nxc, ex_addressof.nxc, ex_addressofex.nxc, ex_ArrayMax.nxc, ex_ArrayMean.nxc, ex_ArrayMin.nxc, ex_ArrayOp.nxc, ex_ArraySort.nxc, ex_ArrayStd.nxc, ex_ArraySum.nxc, ex_ArraySumSqr.nxc, ex_asin.nxc, ex_asind.nxc, ex_atan.nxc, ex_atand.nxc, ex_ButtonPressed.nxc, ex_ctype.nxc, ex_DisplayFont.nxc, ex_dispmisc.nxc, ex_div.nxc, ex_file_system.nxc, ex_FlattenVar.nxc, ex_I2CDeviceId.nxc, ex_I2CDeviceInfo.nxc, ex_I2CVendorId.nxc, ex_I2CVersion.nxc, ex_isNAN.nxc, ex_labs.nxc, ex_ldiv.nxc, ex_memcmp.nxc, ex_ReadSensorHTTouchMultiplexer.nxc, ex_reladdressof.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, ex_UnflattenVar.nxc, util_battery_1.nxc, util_battery_2.nxc, and util_rpm.nxc.`

6.65.2.3 #define LCD_LINE3 40

The 3rd line of the LCD screen

Examples:

`ex_acos.nxc, ex_acosd.nxc, ex_ArraySort.nxc, ex_asin.nxc, ex_asind.nxc, ex_atan.nxc, ex_atand.nxc, ex_ButtonPressed.nxc, ex_ctype.nxc, ex_dispmisc.nxc, ex_FlattenVar.nxc, ex_I2CDeviceId.nxc, ex_I2CDeviceInfo.nxc, ex_I2CVendorId.nxc, ex_I2CVersion.nxc, ex_memcmp.nxc, ex_ReadSensorHTTouchMultiplexer.nxc, ex_reladdressof.nxc, ex_StrCatOld.nxc, ex_string.nxc, ex strtod.nxc, ex strtol.nxc, ex strtoul.nxc, ex_SysCommBTConnection.nxc, ex_TextOut.nxc, and ex_UnflattenVar.nxc.`

6.65.2.4 #define LCD_LINE4 32

The 4th line of the LCD screen

Examples:

`ex_acos.nxc, ex_acosd.nxc, ex_addressof.nxc, ex_addressofex.nxc, ex_arraybuild.nxc, ex_ArraySort.nxc, ex_asin.nxc, ex_asind.nxc, ex_atan.nxc, ex_atand.nxc, ex_ButtonPressed.nxc, ex_ctype.nxc, ex_DisplayFont.nxc, ex_dispmisc.nxc, ex_FlattenVar.nxc, ex_ReadSensorHTTouchMultiplexer.nxc, ex_reladdressof.nxc, ex_SetDisplayFont.nxc, ex_string.nxc, ex_StrReplace.nxc, ex_SysDatalogGetTimes.nxc, and ex_UnflattenVar.nxc.`

6.65.2.5 #define LCD_LINE5 24

The 5th line of the LCD screen

Examples:

[ex_arraybuild.nxc](#), [ex_ArraySort.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_ctype.nxc](#), [ex_dispmisc.nxc](#), [ex_StrIndex.nxc](#), [ex_string.nxc](#), and [ex_SysDatalogGetTimes.nxc](#).

6.65.2.6 #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](#).

6.65.2.7 #define LCD_LINE7 8

The 7th line of the LCD screen

Examples:

[ex_ArraySort.nxc](#), [ex_ctype.nxc](#), and [ex_string.nxc](#).

6.65.2.8 #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](#).

6.66 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

6.66.1 Detailed Description

Constants for use with the [Wait\(\)](#) function.

See also:

[Wait\(\)](#)

6.66.2 Define Documentation

6.66.2.1 #define MIN_1 60000

1 minute

Examples:

[ex_SysSetSleepTimeout.nxc](#).

6.66.2.2 #define MS_1 1

1 millisecond

6.66.2.3 #define MS_10 10

10 milliseconds

6.66.2.4 #define MS_100 100

100 milliseconds

Examples:

[ex_PolyOut.nxc](#), and [ex_sysdrawpolygon.nxc](#).

6.66.2.5 #define MS_150 150

150 milliseconds

6.66.2.6 #define MS_2 2

2 milliseconds

6.66.2.7 #define MS_20 20

20 milliseconds

Examples:

[ex_dispgaout.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).

6.66.2.8 #define MS_200 200

200 milliseconds

Examples:

[ex_dispgoutex.nxc](#), and [ex_playtones.nxc](#).

6.66.2.9 #define MS_250 250

250 milliseconds

6.66.2.10 #define MS_3 3

3 milliseconds

6.66.2.11 #define MS_30 30

30 milliseconds

6.66.2.12 #define MS_300 300

300 milliseconds

6.66.2.13 #define MS_350 350

350 milliseconds

6.66.2.14 #define MS_4 4

4 milliseconds

6.66.2.15 #define MS_40 40

40 milliseconds

6.66.2.16 #define MS_400 400

400 milliseconds

6.66.2.17 #define MS_450 450

450 milliseconds

6.66.2.18 #define MS_5 5

5 milliseconds

Examples:[ex_getchar.nxc](#).**6.66.2.19 #define MS_50 50**

50 milliseconds

Examples:[ex_CircleOut.nxc](#), and [ex_playtones.nxc](#).**6.66.2.20 #define MS_500 500**

500 milliseconds

Examples:[alternating_tasks.nxc](#), [ex_dispgout.nxc](#), [ex_playsound.nxc](#), [ex_yield.nxc](#), and [util_rpm.nxc](#).**6.66.2.21 #define MS_6 6**

6 milliseconds

6.66.2.22 #define MS_60 60

60 milliseconds

6.66.2.23 #define MS_600 600

600 milliseconds

6.66.2.24 #define MS_7 7

7 milliseconds

6.66.2.25 #define MS_70 70

70 milliseconds

6.66.2.26 #define MS_700 700

700 milliseconds

6.66.2.27 #define MS_8 8

8 milliseconds

6.66.2.28 #define MS_80 80

80 milliseconds

6.66.2.29 #define MS_800 800

800 milliseconds

6.66.2.30 #define MS_9 9

9 milliseconds

6.66.2.31 #define MS_90 90

90 milliseconds

6.66.2.32 #define MS_900 900

900 milliseconds

6.66.2.33 #define SEC_1 1000

1 second

Examples:

[alternating_tasks.nxc](#), [ex_dispmissc.nxc](#), [ex_file_system.nxc](#), [ex_playsound.nxc](#),
[ex_playtones.nxc](#), [ex_PolyOut.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_sysdrawpolygon.nxc](#), [ex_wait.nxc](#), and [ex_yield.nxc](#).

6.66.2.34 #define SEC_10 10000

10 seconds

Examples:

[ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ClearScreen.nxc](#), [ex_DisplayFont.nxc](#),
[ex_I2CDeviceInfo.nxc](#), [ex_reladdressof.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_string.nxc](#), [ex_SysCommBTConnection.nxc](#), and [ex_SysCommHSControl.nxc](#).

6.66.2.35 #define SEC_15 15000

15 seconds

Examples:

[ex_dispfunc.nxc](#), and [ex_memcmp.nxc](#).

6.66.2.36 #define SEC_2 2000

2 seconds

Examples:

[ex_CircleOut.nxc](#), [ex_dispmissc.nxc](#), [ex_file_system.nxc](#), [ex_LineOut.nxc](#), [ex_PolyOut.nxc](#), and [ex_sysdrawpolygon.nxc](#).

6.66.2.37 #define SEC_20 20000

20 seconds

6.66.2.38 #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](#).

6.66.2.39 #define SEC_30 30000

30 seconds

6.66.2.40 #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](#).

6.66.2.41 #define SEC_5 5000

5 seconds

Examples:

[ex_atof.nxc](#), [ex_atoi.nxc](#), [ex_atol.nxc](#), [ex_ClearLine.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_dispftout.nxc](#), [ex_dispgout.nxc](#), [ex_FlattenVar.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](#), [ex_UnflattenVar.nxc](#), and [ex_wait.nxc](#).

6.66.2.42 #define SEC_6 6000

6 seconds

Examples:[ex strtod.nxc](#), [ex strtol.nxc](#), and [ex strtoul.nxc](#).**6.66.2.43 #define SEC_7 7000**

7 seconds

6.66.2.44 #define SEC_8 8000

8 seconds

Examples:[ex_file_system.nxc](#).**6.66.2.45 #define SEC_9 9000**

9 seconds

Examples:[ex_SensorHTGyro.nxc](#).**6.67 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

6.67.1 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\(\)](#)

6.67.2 Define Documentation

6.67.2.1 #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](#).

6.67.2.2 #define MAILBOX10 9

Mailbox number 10

6.67.2.3 #define MAILBOX2 1

Mailbox number 2

6.67.2.4 #define MAILBOX3 2

Mailbox number 3

6.67.2.5 #define MAILBOX4 3

Mailbox number 4

6.67.2.6 #define MAILBOX5 4

Mailbox number 5

6.67.2.7 #define MAILBOX6 5

Mailbox number 6

6.67.2.8 #define MAILBOX7 6

Mailbox number 7

6.67.2.9 #define MAILBOX8 7

Mailbox number 8

6.67.2.10 #define MAILBOX9 8

Mailbox number 9

6.68 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

6.68.1 Detailed Description

Constants defining possible VM states.

6.68.2 Define Documentation**6.68.2.1 #define BREAKOUT_REQ 3**

VM should break out of current thread

6.68.2.2 #define CLUMP_DONE 1

VM has finished executing thread

6.68.2.3 #define CLUMP_SUSPEND 2

VM should suspend thread

6.68.2.4 #define ROTATE_QUEUE 5

VM should rotate queue

6.68.2.5 #define STOP_REQ 4

VM should stop executing program

6.68.2.6 #define TIMES_UP 6

VM time is up

6.69 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

6.69.1 Detailed Description

Constants defining various fatal error conditions.

6.69.2 Define Documentation

6.69.2.1 #define `ERR_ARG` -1

0xFF Bad arguments

6.69.2.2 #define `ERR_BAD_POOL_SIZE` -10

0xF6 VarsCmd.PoolSize > POOL_MAX_SIZE

6.69.2.3 #define `ERR_BAD_PTR` -6

0xFA Someone passed us a bad pointer!

6.69.2.4 #define `ERR_CLUMP_COUNT` -7

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT_A_CLUMP)

6.69.2.5 #define `ERR_DEFAULT_OFFSETS` -14

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

6.69.2.6 #define `ERR_FILE` -3

0xFD Malformed file contents

6.69.2.7 #define ERR_INSANE_OFFSET -9

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount * 2)

6.69.2.8 #define ERR_INSTR -2

0xFE Illegal bytecode instruction

6.69.2.9 #define ERR_LOADER_ERR -11

0xF5 LOADER_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

6.69.2.10 #define ERR_MEM -5

0xFB Insufficient memory available

6.69.2.11 #define ERR_MEMMGR_FAIL -15

0xF1 (UBYTE *)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV_ARRAY[0].Offset

6.69.2.12 #define ERR_NO_ACTIVE_CLUMP -13

0xF3 VarsCmd.RunQ.Head == NOT_A_CLUMP

6.69.2.13 #define ERR_NO_CODE -8

0xF8 VarsCmd.CodespaceCount == 0

6.69.2.14 #define ERR_NON_FATAL -16

Fatal errors are greater than this value

6.69.2.15 #define ERR_SPOTCHECK_FAIL -12

0xF4 ((UBYTE*)(VarsCmd.pCodespace) < pData) (c_cmd.c 1893)

6.69.2.16 #define ERR_VER -4

0xFC Version mismatch between firmware and compiler

6.70 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

6.70.1 Detailed Description

Constants defining general error conditions.

6.70.2 Define Documentation

6.70.2.1 #define `ERR_INVALID_FIELD` -17

0xEF Attempted to access invalid field of a structure

6.70.2.2 #define `ERR_INVALID_PORT` -16

0xF0 Bad input or output port specified

6.70.2.3 #define `ERR_INVALID_QUEUE` -18

0xEE Illegal queue ID specified

6.70.2.4 #define `ERR_INVALID_SIZE` -19

0xED Illegal size specified

6.70.2.5 #define `ERR_NO_PROG` -20

0xEC No active program

6.71 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

6.71.1 Detailed Description

Constants defining communication error conditions.

6.71.2 Define Documentation

6.71.2.1 #define `ERR_COMM_BUFFER_FULL` -34

0xDE No room in comm buffer

6.71.2.2 #define `ERR_COMM_BUS_ERR` -35

0xDD Something went wrong on the communications bus

6.71.2.3 #define `ERR_COMM_CHAN_INVALID` -33

0xDF Specified channel/connection is not valid

6.71.2.4 #define `ERR_COMM_CHAN_NOT_READY` -32

0xE0 Specified channel/connection not configured or busy

6.72 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

6.72.1 Detailed Description

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

6.72.2 Define Documentation**6.72.2.1 #define ERR_RC_BAD_PACKET -65**

0xBF Clearly insane packet

6.72.2.2 #define ERR_RC_FAILED -67

0xBD Request failed (i.e. specified file not found)

6.72.2.3 #define ERR_RC_ILLEGAL_VAL -64

0xC0 Data contains out-of-range values

6.72.2.4 #define ERR_RC_UNKNOWN_CMD -66

0xBE Unknown command opcode

6.73 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

6.73.1 Detailed Description

Constants defining various states of the command module virtual machine.

6.73.2 Define Documentation

6.73.2.1 #define PROG_ABORT 4

Program has been aborted

6.73.2.2 #define PROG_ERROR 3

A program error has occurred

6.73.2.3 #define PROG_IDLE 0

Program state is idle

6.73.2.4 #define PROG_OK 1

Program state is okay

6.73.2.5 #define PROG_RESET 5

Program has been reset

6.73.2.6 #define PROG_RUNNING 2

Program is running

6.74 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

6.74.1 Detailed Description

Constant offsets into the Command module IOMAP structure.

6.74.2 Define Documentation**6.74.2.1 #define `CommandOffsetActivateFlag` 30**

Offset to the activate flag

6.74.2.2 #define `CommandOffsetAwake` 29

Offset to the VM's awake state

6.74.2.3 #define `CommandOffsetDeactivateFlag` 31

Offset to the deactivate flag

6.74.2.4 #define `CommandOffsetFileName` 32

Offset to the running program's filename

6.74.2.5 #define `CommandOffsetFormatString` 0

Offset to the format string

6.74.2.6 #define CommandOffsetMemoryPool 52

Offset to the VM's memory pool

Examples:

[ex_reladdressof.nxc](#).

6.74.2.7 #define CommandOffsetOffsetDS 24

Offset to the running program's data space (DS)

6.74.2.8 #define CommandOffsetOffsetDVA 26

Offset to the running program's DOPE vector address (DVA)

6.74.2.9 #define CommandOffsetPRCHandler 16

Offset to the RC Handler function pointer

6.74.2.10 #define CommandOffsetProgStatus 28

Offset to the running program's status

6.74.2.11 #define CommandOffsetSyncTick 32824

Offset to the VM sync tick

6.74.2.12 #define CommandOffsetSyncTime 32820

Offset to the VM sync time

6.74.2.13 #define CommandOffsetTick 20

Offset to the VM's current tick

Examples:

[ex_systiompread.nxc](#), and [ex_systiompreadbyid.nxc](#).

6.75 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.

6.75.1 Detailed Description

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

6.76 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

6.76.1 Detailed Description

Use these constants to power down the NXT or boot it into SAMBA (aka firmware download) mode.

6.76.2 Define Documentation

6.76.2.1 #define IOCTRL_BOOT 0xA55A

Reboot the NXT into SAMBA mode

6.76.2.2 #define IOCTRL_POWERDOWN 0x5A00

Power down the NXT

6.77 IOCtrl module IOMAP offsets

Constant offsets into the IOCtrl module IOMAP structure.

Defines

- #define [IOCtrlOffsetPowerOn](#) 0

6.77.1 Detailed Description

Constant offsets into the IOCtrl module IOMAP structure.

6.77.2 Define Documentation

6.77.2.1 #define IOCtrlOffsetPowerOn 0

Offset to power on field

6.78 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

6.78.1 Detailed Description

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

6.78.2 Define Documentation**6.78.2.1 #define EOF -1**

A constant representing end of file

6.78.2.2 #define NULL 0

A constant representing NULL

6.79 Loader module IOMAP offsets

Constant offsets into the Loader module IOMAP structure.

Defines

- #define `LoaderOffsetPFunc` 0
- #define `LoaderOffsetFreeUserFlash` 4

6.79.1 Detailed Description

Constant offsets into the Loader module IOMAP structure.

6.79.2 Define Documentation**6.79.2.1 #define LoaderOffsetFreeUserFlash 4**

Offset to the amount of free user flash

6.79.2.2 #define LoaderOffsetPFunc 0

Offset to the Loader module function pointer

6.80 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_FILENOFOUND` 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_OUTOFCBOUNDARY` 0x9100
- #define `LDR_ILLEGALFILENAME` 0x9200
- #define `LDR_ILLEGALHANDLE` 0x9300
- #define `LDR_BTBUSY` 0x9400
- #define `LDR_BTCONNECTFAIL` 0x9500
- #define `LDR_BT TIMEOUT` 0x9600
- #define `LDR_FILETX_TIMEOUT` 0x9700
- #define `LDR_FILETX_DSTEXISTS` 0x9800
- #define `LDR_FILETX_SRCMISSING` 0x9900
- #define `LDR_FILETX_STREAMERROR` 0x9A00
- #define `LDR_FILETX_CLOSEERROR` 0x9B00
- #define `LDR_INVALIDSEEK` 0x9C00

6.80.1 Detailed Description

Error codes returned by functions in the Loader module (file access).

6.80.2 Define Documentation

6.80.2.1 #define LDR_APPENDNOTPOSSIBLE 0x8D00

Only datafiles can be appended to.

6.80.2.2 #define LDR_BTBUSY 0x9400

The bluetooth system is busy.

6.80.2.3 #define LDR_BTCONNECTFAIL 0x9500

Bluetooth connection attempt failed.

6.80.2.4 #define LDR_BTTIMEOUT 0x9600

A timeout in the bluetooth system has occurred.

6.80.2.5 #define LDR_ENDOFFILE 0x8500

The end of the file has been reached.

Examples:

[ex_file_system.nxc](#).

6.80.2.6 #define LDR_EOFEXPECTED 0x8400

EOF expected.

Examples:

[ex_file_system.nxc](#).

6.80.2.7 #define LDR_FILEEXISTS 0x8F00

A file with the same name already exists.

Examples:

[ex_file_system.nxc](#).

6.80.2.8 #define LDR_FILEISBUSY 0x8B00

The file is already being used.

6.80.2.9 #define LDR_FILEISFULL 0x8E00

The allocated file size has been filled.

Examples:

[ex_file_system.nxc](#).

6.80.2.10 #define LDR_FILENOFOUND 0x8700

No files matched the search criteria.

6.80.2.11 #define LDR_FILETX_CLOSEERROR 0x9B00

Error transmitting file: attempt to close file failed.

6.80.2.12 #define LDR_FILETX_DSTEXISTS 0x9800

Error transmitting file: destination file exists.

6.80.2.13 #define LDR_FILETX_SRCMISSING 0x9900

Error transmitting file: source file is missing.

6.80.2.14 #define LDR_FILETX_STREAMERROR 0x9A00

Error transmitting file: a stream error occurred.

6.80.2.15 #define LDR_FILETX_TIMEOUT 0x9700

Error transmitting file: a timeout occurred.

6.80.2.16 #define LDR_HANDLEALREADYCLOSED 0x8800

The file handle has already been closed.

6.80.2.17 #define LDR_ILLEGALFILENAME 0x9200

Filename length too long or attempted open a system file (*.rxe, *.rtm, or *.sys) for writing as a datafile.

6.80.2.18 #define LDR_ILLEGALHANDLE 0x9300

Invalid file handle.

6.80.2.19 #define LDR_INPROGRESS 0x0001

The function is executing but has not yet completed.

6.80.2.20 #define LDR_INVALIDSEEK 0x9C00

Invalid file seek operation.

6.80.2.21 #define LDR_MODULENOTFOUND 0x9000

No modules matched the specified search criteria.

6.80.2.22 #define LDR_NOLINEARSPACE 0x8900

Not enough linear flash memory is available.

6.80.2.23 #define LDR_NOMOREFILES 0x8300

The maximum number of files has been reached.

6.80.2.24 #define LDR_NOMOREHANDLES 0x8100

All available file handles are in use.

6.80.2.25 #define LDR_NOSPACE 0x8200

Not enough free flash memory for the specified file size.

6.80.2.26 #define LDR_NOTLINEARFILE 0x8600

The specified file is not linear.

6.80.2.27 #define LDR_NOWRITEBUFFERS 0x8C00

No more write buffers are available.

6.80.2.28 #define LDR_OUTOFCBOUNDARY 0x9100

Specified IOMap offset is outside the bounds of the IOMap.

6.80.2.29 #define LDR_REQPIN 0x0002

A PIN exchange request is in progress.

6.80.2.30 #define LDR_SUCCESS 0x0000

The function completed successfully.

Examples:

[ex_file_system.nxc](#), [ex_syscommbtcheckstatus.nxc](#), [ex_-SysCommBTConnection.nxc](#), [ex_sysfilerename.nxc](#), and [ex_-sysfileresolvehandle.nxc](#).

6.80.2.31 #define LDR_UNDEFINEDERROR 0x8A00

An undefined error has occurred.

6.81 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

6.81.1 Detailed Description

Constants defining the functions provided by the Loader module.

6.81.2 Define Documentation**6.81.2.1 #define LDR_CMD_BOOTCMD 0x97**

Reboot the NXT into SAMBA mode

6.81.2.2 #define LDR_CMD_BTFACTORYRESET 0xA4

Reset bluetooth configuration to factory defaults

6.81.2.3 #define LDR_CMD_BTGETADR 0x9A

Get the NXT's bluetooth brick address

6.81.2.4 #define LDR_CMD_CLOSE 0x84

Close a file handle

6.81.2.5 #define LDR_CMD_CLOSEMODHANDLE 0x92

Close a module handle

6.81.2.6 #define LDR_CMD_CROPDATAFILE 0x8D

Crop a data file to its used space

6.81.2.7 #define LDR_CMD_DELETE 0x85

Delete a file

6.81.2.8 #define LDR_CMD_DELETEUSERFLASH 0xA0

Delete all files from user flash memory

6.81.2.9 #define LDR_CMD_DEVICEINFO 0x9B

Read device information

6.81.2.10 #define LDR_CMD_FINDFIRST 0x86

Find the first file matching the specified pattern

6.81.2.11 #define LDR_CMD_FINDFIRSTMODULE 0x90

Find the first module matching the specified pattern

6.81.2.12 #define LDR_CMD_FINDNEXT 0x87

Find the next file matching the specified pattern

6.81.2.13 #define LDR_CMD_FINDNEXTMODULE 0x91

Find the next module matching the specified pattern

6.81.2.14 #define LDR_CMD_IOMAPREAD 0x94

Read data from a module IOMAP

6.81.2.15 #define LDR_CMD_IOMAPWRITE 0x95

Write data to a module IOMAP

6.81.2.16 #define LDR_CMD_OPENAPPENDDATA 0x8C

Open a data file for appending

6.81.2.17 #define LDR_CMD_OPENREAD 0x80

Open a file for reading

6.81.2.18 #define LDR_CMD_OPENREADLINEAR 0x8A

Open a linear file for reading

6.81.2.19 #define LDR_CMD_OPENWRITE 0x81

Open a file for writing

6.81.2.20 #define LDR_CMD_OPENWRITEDATA 0x8B

Open a data file for writing

6.81.2.21 #define LDR_CMD_OPENWITELINEAR 0x89

Open a linear file for writing

6.81.2.22 #define LDR_CMD_POLLCMD 0xA2

Poll command

6.81.2.23 #define LDR_CMD_POLLCMDLEN 0xA1

Read poll command length

6.81.2.24 #define LDR_CMD_READ 0x82

Read from a file

6.81.2.25 #define LDR_CMD_RENAMEFILE 0xA3

Rename a file

6.81.2.26 #define LDR_CMD_RESIZEDATAFILE 0xD0

Resize a data file

6.81.2.27 #define LDR_CMD_SEEKFROMCURRENT 0xD2

Seek from the current position

6.81.2.28 #define LDR_CMD_SEEKFROMEND 0xD3

Seek from the end of the file

6.81.2.29 #define LDR_CMD_SEEKFROMSTART 0xD1

Seek from the start of the file

6.81.2.30 #define LDR_CMD_SETBRICKNAME 0x98

Set the NXT's brick name

6.81.2.31 #define LDR_CMD_VERSIONS 0x88

Read firmware version information

6.81.2.32 #define LDR_CMD_WRITE 0x83

Write to a file

6.82 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.

6.82.1 Detailed Description

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

6.83 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

6.83.1 Detailed Description

Constants for use with the [SoundFlags\(\)](#) function.

See also:

[SoundFlags\(\)](#)

6.83.2 Define Documentation

6.83.2.1 #define SOUND_FLAGS_IDLE 0x00

R - Sound is idle

6.83.2.2 #define SOUND_FLAGS_RUNNING 0x02

R - Currently processing a tone or file

6.83.2.3 #define SOUND_FLAGS_UPDATE 0x01

W - Make changes take effect

Examples:

[ex_SetSoundFlags.nxc](#).

6.84 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

6.84.1 Detailed Description

Constants for use with the [SoundState\(\)](#) function.

See also:

[SoundState\(\)](#)

6.84.2 Define Documentation

6.84.2.1 #define SOUND_STATE_FILE 0x02

R - Processing a file of sound/melody data

6.84.2.2 #define SOUND_STATE_IDLE 0x00

R - Idle, ready for start sound (SOUND_UPDATE)

Examples:

[ex_syssoundgetstate.nxc](#).

6.84.2.3 #define SOUND_STATE_STOP 0x04

W - Stop sound immediately and close hardware

Examples:

[ex_SetSoundModuleState.nxc](#), and [ex_syssoundsetstate.nxc](#).

6.84.2.4 #define SOUND_STATE_TONE 0x03

R - Processing a play tone request

6.85 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

6.85.1 Detailed Description

Constants for use with the [SoundMode\(\)](#) function.

See also:

[SoundMode\(\)](#)

6.85.2 Define Documentation

6.85.2.1 #define SOUND_MODE_LOOP 0x01

W - Play file until writing SOUND_STATE_STOP into SoundState

6.85.2.2 #define SOUND_MODE_ONCE 0x00

W - Only play file once

Examples:

[ex_SetSoundMode.nxc](#).

6.85.2.3 #define SOUND_MODE_TONE 0x02

W - Play tone specified in Frequency for Duration ms

6.86 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

6.86.1 Detailed Description

Constant offsets into the Sound module IOMAP structure.

6.86.2 Define Documentation

6.86.2.1 #define SoundOffsetDuration 2

RW - Tone duration [mS] (2 bytes)

6.86.2.2 #define SoundOffsetFlags 26

RW - Play flag - described above (1 byte) [SoundFlags constants](#)

6.86.2.3 #define SoundOffsetFreq 0

RW - [Tone](#) frequency [Hz] (2 bytes)

6.86.2.4 #define SoundOffsetMode 28

RW - Play mode - described above (1 byte) [SoundMode constants](#)

6.86.2.5 #define SoundOffsetSampleRate 4

RW - Sound file sample rate [2000..16000] (2 bytes)

Examples:

[ex_systiomapwrite.nxc](#), and [ex_systiomapwritebyid.nxc](#).

6.86.2.6 #define SoundOffsetSoundFilename 6

RW - Sound/melody filename (20 bytes)

6.86.2.7 #define SoundOffsetState 27

RW - Play state - described above (1 byte) [SoundState constants](#)

6.86.2.8 #define SoundOffsetVolume 29

RW - Sound/melody volume [0..4] 0 = off (1 byte)

6.87 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

6.87.1 Detailed Description

Constants defining miscellaneous sound module aspects.

6.87.2 Define Documentation

6.87.2.1 #define FREQUENCY_MAX 14080

Maximum frequency [Hz]

6.87.2.2 #define FREQUENCY_MIN 220

Minimum frequency [Hz]

6.87.2.3 #define SAMPLERATE_DEFAULT 8000

Default sample rate [sps]

6.87.2.4 #define SAMPLERATE_MAX 16000

Max sample rate [sps]

6.87.2.5 #define SAMPLERATE_MIN 2000

Min sample rate [sps]

6.88 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

6.88.1 Detailed Description

Constants for use in the [SoundPlayTone\(\)](#) API function.

See also:

[SoundPlayTone\(\)](#)

6.88.2 Define Documentation

6.88.2.1 #define TONE_A3 220

Third octave A

6.88.2.2 #define TONE_A4 440

Fourth octave A

Examples:

[ex_yield.nxc](#).

6.88.2.3 #define TONE_A5 880

Fifth octave A

6.88.2.4 #define TONE_A6 1760

Sixth octave A

6.88.2.5 #define TONE_A7 3520

Seventh octave A

6.88.2.6 #define TONE_AS3 233

Third octave A sharp

6.88.2.7 #define TONE_AS4 466

Fourth octave A sharp

6.88.2.8 #define TONE_AS5 932

Fifth octave A sharp

6.88.2.9 #define TONE_AS6 1865

Sixth octave A sharp

6.88.2.10 #define TONE_AS7 3729

Seventh octave A sharp

6.88.2.11 #define TONE_B3 247

Third octave B

6.88.2.12 #define TONE_B4 494

Fourth octave B

6.88.2.13 #define TONE_B5 988

Fifth octave B

6.88.2.14 #define TONE_B6 1976

Sixth octave B

6.88.2.15 #define TONE_B7 3951

Seventh octave B

6.88.2.16 #define TONE_C4 262

Fourth octave C

Examples:[alternating_tasks.nxc](#), and [ex_playtones.nxc](#).**6.88.2.17 #define TONE_C5 523**

Fifth octave C

Examples:[ex_file_system.nxc](#), and [ex_playtones.nxc](#).**6.88.2.18 #define TONE_C6 1047**

Sixth octave C

Examples:[alternating_tasks.nxc](#), and [ex_playtones.nxc](#).**6.88.2.19 #define TONE_C7 2093**

Seventh octave C

6.88.2.20 #define TONE_CS4 277

Fourth octave C sharp

6.88.2.21 #define TONE_CS5 554

Fifth octave C sharp

6.88.2.22 #define TONE_CS6 1109

Sixth octave C sharp

6.88.2.23 #define TONE_CS7 2217

Seventh octave C sharp

6.88.2.24 #define TONE_D4 294

Fourth octave D

6.88.2.25 #define TONE_D5 587

Fifth octave D

6.88.2.26 #define TONE_D6 1175

Sixth octave D

6.88.2.27 #define TONE_D7 2349

Seventh octave D

6.88.2.28 #define TONE_DS4 311

Fourth octave D sharp

6.88.2.29 #define TONE_DS5 622

Fifth octave D sharp

6.88.2.30 #define TONE_DS6 1245

Sixth octave D sharp

6.88.2.31 #define TONE_DS7 2489

Seventh octave D sharp

6.88.2.32 #define TONE_E4 330

Fourth octave E

Examples:

[ex_playtones.nxc.](#)

6.88.2.33 #define TONE_E5 659

Fifth octave E

Examples:

[ex_playtones.nxc.](#)

6.88.2.34 #define TONE_E6 1319

Sixth octave E

6.88.2.35 #define TONE_E7 2637

Seventh octave E

6.88.2.36 #define TONE_F4 349

Fourth octave F

6.88.2.37 #define TONE_F5 698

Fifth octave F

6.88.2.38 #define TONE_F6 1397

Sixth octave F

6.88.2.39 #define TONE_F7 2794

Seventh octave F

6.88.2.40 #define TONE_FS4 370

Fourth octave F sharp

6.88.2.41 #define TONE_FS5 740

Fifth octave F sharp

6.88.2.42 #define TONE_FS6 1480

Sixth octave F sharp

6.88.2.43 #define TONE_FS7 2960

Seventh octave F sharp

6.88.2.44 #define TONE_G4 392

Fourth octave G

Examples:

[ex_playtones.nxc.](#)

6.88.2.45 #define TONE_G5 784

Fifth octave G

Examples:

[ex_playtones.nxc.](#)

6.88.2.46 #define TONE_G6 1568

Sixth octave G

6.88.2.47 #define TONE_G7 3136

Seventh octave G

6.88.2.48 #define TONE_GS4 415

Fourth octave G sharp

6.88.2.49 #define TONE_GS5 831

Fifth octave G sharp

6.88.2.50 #define TONE_GS6 1661

Sixth octave G sharp

6.88.2.51 #define TONE_GS7 3322

Seventh octave G sharp

6.89 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.

6.89.1 Detailed Description

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

6.90 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 **BTNEXTIT** BTN1
- #define **BTNRIGHT** BTN2
- #define **BTNLEFT** BTN3
- #define **BTNCENTER** BTN4
- #define **NO_OF_BTNS** 4

6.90.1 Detailed Description

Constants to specify which button to use with button module functions.

See also:

[ButtonPressed\(\)](#), [ButtonState\(\)](#), [ButtonCount\(\)](#), [ReadButtonEx\(\)](#), [SysReadButton\(\)](#), [ReadButtonType](#)

6.90.2 Define Documentation

6.90.2.1 #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](#).

6.90.2.2 #define **BTN2** 1

The right button.

6.90.2.3 #define **BTN3** 2

The left button.

6.90.2.4 #define BTN4 3

The enter button.

6.90.2.5 #define BTNCENTER BTN4

The enter button.

Examples:

[ex_ButtonPressed.nxc](#).

6.90.2.6 #define BTNEXIT BTN1

The exit button.

Examples:

[ex_ButtonPressed.nxc](#).

6.90.2.7 #define BTNLEFT BTN3

The left button.

Examples:

[ex_ButtonPressed.nxc](#).

6.90.2.8 #define BTNRIGHT BTN2

The right button.

Examples:

[ex_ButtonPressed.nxc](#), and [ex_sysreadbutton.nxc](#).

6.90.2.9 #define NO_OF_BTNS 4

The number of NXT buttons.

6.91 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

6.91.1 Detailed Description

Constants for use with the [ButtonState\(\)](#) function. The _EV values can be combined together using a bitwise OR operation.

See also:

[ButtonState\(\)](#)

6.91.2 Define Documentation

6.91.2.1 #define BTNSTATE_LONG_PRESSED_EV 0x04

Button is in the long pressed state.

Examples:

[ex_SetAbortFlag.nxc](#).

6.91.2.2 #define BTNSTATE_LONG_RELEASED_EV 0x08

Button is in the long released state.

6.91.2.3 #define BTNSTATE_NONE 0x10

The default button state.

6.91.2.4 #define BTNSTATE_PRESSED_EV 0x01

Button is in the pressed state.

Examples:

[ex_SetButtonState.nxc](#).

6.91.2.5 #define BTNSTATE_PRESSED_STATE 0x80

A bitmask for the button pressed state

6.91.2.6 #define BTNSTATE_SHORT_RELEASED_EV 0x02

Button is in the short released state.

6.92 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)

6.92.1 Detailed Description

Constant offsets into the Button module IOMAP structure.

6.92.2 Define Documentation**6.92.2.1 #define ButtonOffsetLongPressCnt(b) (((b)*8)+1)**

Offset to the LongPressCnt field. This field stores the long press count.

6.92.2.2 #define ButtonOffsetLongRelCnt(b) (((b)*8)+3)

Offset to the LongRelCnt field. This field stores the long release count.

6.92.2.3 #define ButtonOffsetPressedCnt(b) (((b)*8)+0)

Offset to the PressedCnt field. This field stores the press count.

6.92.2.4 #define ButtonOffsetRelCnt(b) (((b)*8)+4)

Offset to the RelCnt field. This field stores the release count.

6.92.2.5 #define ButtonOffsetShortRelCnt(b) (((b)*8)+2)

Offset to the ShortRelCnt field. This field stores the short release count.

6.92.2.6 #define ButtonOffsetState(b) ((b)+32)

Offset to the State field. This field stores the current button state.

6.93 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.

6.93.1 Detailed Description

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

6.94 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

6.94.1 Detailed Description

Constants for use with the [CommandFlags\(\)](#) function.

See also:

[CommandFlags\(\)](#)

6.94.2 Define Documentation

6.94.2.1 #define UI_FLAGS_BUSY 0x40

R - UI busy running or datalogging (popup disabled)

6.94.2.2 #define UI_FLAGS_DISABLE_EXIT 0x04

RW - Disable exit button

6.94.2.3 #define UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER 0x02

RW - Disable left, right and enter button

6.94.2.4 #define UI_FLAGS_ENABLE_STATUS_UPDATE 0x80

W - Enable status line to be updated

6.94.2.5 #define UI_FLAGS_EXECUTE_LMS_FILE 0x20

W - Execute LMS file in "LMSfilename" (Try It)

6.94.2.6 #define UI_FLAGS_REDRAW_STATUS 0x08

W - Redraw entire status line

Examples:

[ex_SetCommandFlags.nxc](#).

6.94.2.7 #define UI_FLAGS_RESET_SLEEP_TIMER 0x10

W - Reset sleep timeout timer

6.94.2.8 #define UI_FLAGS_UPDATE 0x01

W - Make changes take effect

6.95 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

6.95.1 Detailed Description

Constants for use with the [UIState\(\)](#) function.

See also:

[UIState\(\)](#)

6.95.2 Define Documentation

6.95.2.1 #define `UI_STATE_BT_ERROR` 16

R - BT error

6.95.2.2 #define `UI_STATE_CONNECT_REQUEST` 12

RW - Request for connection accept

6.95.2.3 #define `UI_STATE_DRAW_MENU` 6

RW - Execute function and draw menu icons

6.95.2.4 #define `UI_STATE_ENTER_PRESSED` 10

RW - Load selected function and next menu id

6.95.2.5 #define `UI_STATE_EXECUTE_FILE` 13

RW - Execute file in "LMSfilename"

6.95.2.6 #define UI_STATE_EXECUTING_FILE 14

R - Executing file in "LMSfilename"

6.95.2.7 #define UI_STATE_EXIT_PRESSED 11

RW - Load selected function and next menu id

6.95.2.8 #define UI_STATE_INIT_DISPLAY 0

RW - Init display and load font, menu etc.

6.95.2.9 #define UI_STATE_INIT_INTRO 2

R - Display intro

6.95.2.10 #define UI_STATE_INIT_LOW_BATTERY 1

R - Low battery voltage at power on

6.95.2.11 #define UI_STATE_INIT_MENU 4

RW - Init menu system

6.95.2.12 #define UI_STATE_INIT_WAIT 3

RW - Wait for initialization end

6.95.2.13 #define UI_STATE_LEFT_PRESSED 8

RW - Load selected function and next menu id

6.95.2.14 #define UI_STATE_LOW_BATTERY 15

R - Low battery at runtime

Examples:

[ex_SetUIState.nxc](#).

6.95.2.15 #define UI_STATE_NEXT_MENU 5

RW - Next menu icons ready for drawing

6.95.2.16 #define UI_STATE_RIGHT_PRESSED 9

RW - Load selected function and next menu id

6.95.2.17 #define UI_STATE_TEST_BUTTONS 7

RW - Wait for buttons to be pressed

6.96 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

6.96.1 Detailed Description

Constants for use with the [UIButton\(\)](#) function.

See also:

[UIButton\(\)](#)

6.96.2 Define Documentation

6.96.2.1 #define UI_BUTTON_ENTER 2

W - Insert enter button

Examples:

[ex_SetUIButton.nxc](#).

6.96.2.2 #define UI_BUTTON_EXIT 4

W - Insert exit button

6.96.2.3 #define UI_BUTTON_LEFT 1

W - Insert left arrow button

6.96.2.4 #define UI_BUTTON_NONE 0

R - Button inserted are executed

6.96.2.5 #define UI_BUTTON_RIGHT 3

W - Insert right arrow button

6.97 BluetoothState constantsConstants for use with the [BluetoothState\(\)](#) function.**Defines**

- #define [UI_BT_STATE_VISIBLE](#) 0x01
- #define [UI_BT_STATE_CONNECTED](#) 0x02
- #define [UI_BT_STATE_OFF](#) 0x04
- #define [UI_BT_ERROR_ATTENTION](#) 0x08
- #define [UI_BT_CONNECT_REQUEST](#) 0x40
- #define [UI_BT_PIN_REQUEST](#) 0x80

6.97.1 Detailed DescriptionConstants for use with the [BluetoothState\(\)](#) function.**See also:**[BluetoothState\(\)](#)**6.97.2 Define Documentation****6.97.2.1 #define UI_BT_CONNECT_REQUEST 0x40**

RW - BT get connect accept in progress

6.97.2.2 #define UI_BT_ERROR_ATTENTION 0x08

W - BT error attention

6.97.2.3 #define UI_BT_PIN_REQUEST 0x80

RW - BT get pin code

6.97.2.4 #define UI_BT_STATE_CONNECTED 0x02

RW - BT connected to something

6.97.2.5 #define UI_BT_STATE_OFF 0x04

RW - BT power off

Examples:[ex_SetBluetoothState.nxc.](#)**6.97.2.6 #define UI_BT_STATE_VISIBLE 0x01**

RW - BT visible

6.98 VM run state constantsConstants for use with the [VMRunState\(\)](#) function.**Defines**

- #define [UI_VM_IDLE](#) 0
- #define [UI_VM_RUN_FREE](#) 1
- #define [UI_VM_RUN_SINGLE](#) 2
- #define [UI_VM_RUN_PAUSE](#) 3
- #define [UI_VM_RESET1](#) 4
- #define [UI_VM_RESET2](#) 5

6.98.1 Detailed Description

Constants for use with the [VMRunState\(\)](#) function.

See also:

[VMRunState\(\)](#)

6.98.2 Define Documentation

6.98.2.1 #define UI_VM_IDLE 0

VM_IDLE: Just sitting around. Request to run program will lead to ONE of the VM_RUN* states.

6.98.2.2 #define UI_VM_RESET1 4

VM_RESET2: Final clean up and return to IDLE

6.98.2.3 #define UI_VM_RESET2 5

VM_RESET1: Initialize state variables and some I/O devices -- executed when programs end

6.98.2.4 #define UI_VM_RUN_FREE 1

VM_RUN_FREE: Attempt to run as many instructions as possible within our timeslice

6.98.2.5 #define UI_VM_RUN_PAUSE 3

VM_RUN_PAUSE: Program still "active", but someone has asked us to pause

6.98.2.6 #define UI_VM_RUN_SINGLE 2

VM_RUN_SINGLE: Run exactly one instruction per timeslice

6.99 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

6.99.1 Detailed Description

Constant offsets into the Ui module IOMAP structure.

6.99.2 Define Documentation**6.99.2.1 #define `UIOffsetAbortFlag` 40**

RW - Long Abort (true == use long press to abort) (1 byte)

6.99.2.2 #define `UIOffsetBatteryState` 30

W - Battery state (0..4 capacity) (1 byte)

6.99.2.3 #define `UIOffsetBatteryVoltage` 4

R - Battery voltage in millivolts (2 bytes)

6.99.2.4 #define `UIOffsetBluetoothState` 31

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)

6.99.2.5 #define UIOffsetButton 28

RW - Insert button (buttons enumerated above) (1 byte)

6.99.2.6 #define UIOffsetError 37

W - Error code (1 byte)

6.99.2.7 #define UIOffsetFlags 26

RW - Update command flags (flags enumerated above) (1 byte)

6.99.2.8 #define UIOffsetForceOff 39

W - Force off (> 0 = off) (1 byte)

6.99.2.9 #define UIOffsetLMSfilename 6

W - LMS filename to execute (Try It) (20 bytes)

6.99.2.10 #define UIOffsetOBPPointer 38

W - Actual OBP step (0 - 4) (1 byte)

6.99.2.11 #define UIOffsetPMenu 0

W - Pointer to menu file (4 bytes)

6.99.2.12 #define UIOffsetRechargeable 35

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

6.99.2.13 #define UIOffsetRunState 29

W - VM Run state (0 = stopped, 1 = running) (1 byte)

6.99.2.14 #define UIOffsetSleepTimeout 33

RW - Sleep timeout time (min) (1 byte)

6.99.2.15 #define UIOffsetSleepTimer 34

RW - Sleep timer (min) (1 byte)

6.99.2.16 #define UIOffsetState 27

RW - UI state (states enumerated above) (1 byte)

6.99.2.17 #define UIOffsetUsbState 32

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

6.99.2.18 #define UIOffsetVolume 36

RW - Volume used in UI (0 - 4) (1 byte)

6.100 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

6.100.1 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](#)**6.100.2 Define Documentation****6.100.2.1 #define IN_1 0x00**

Input port 1

6.100.2.2 #define IN_2 0x01

Input port 2

6.100.2.3 #define IN_3 0x02

Input port 3

6.100.2.4 #define IN_4 0x03

Input port 4

6.101 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

6.101.1 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\(\)](#)

6.101.2 Define Documentation

6.101.2.1 `#define IN_TYPE_ANGLE 0x04`

RCX rotation sensor

6.101.2.2 `#define IN_TYPE_COLORBLUE 0x10`

NXT 2.0 color sensor with blue light

6.101.2.3 `#define IN_TYPE_COLOREXIT 0x12`

NXT 2.0 color sensor internal state

6.101.2.4 `#define IN_TYPE_COLORFULL 0x0D`

NXT 2.0 color sensor in full color mode

6.101.2.5 `#define IN_TYPE_COLORGREEN 0x0F`

NXT 2.0 color sensor with green light

6.101.2.6 `#define IN_TYPE_COLORNONE 0x11`

NXT 2.0 color sensor with no light

6.101.2.7 `#define IN_TYPE_COLORRED 0x0E`

NXT 2.0 color sensor with red light

6.101.2.8 #define IN_TYPE_CUSTOM 0x09

NXT custom sensor

6.101.2.9 #define IN_TYPE_HISPEED 0x0C

NXT Hi-speed port (only S4)

6.101.2.10 #define IN_TYPE_LIGHT_ACTIVE 0x05

NXT light sensor with light

6.101.2.11 #define IN_TYPE_LIGHT_INACTIVE 0x06

NXT light sensor without light

6.101.2.12 #define IN_TYPE_LOWSPEED 0x0A

NXT I2C digital sensor

6.101.2.13 #define IN_TYPE_LOWSPEED_9V 0x0B

NXT I2C digital sensor with 9V power

6.101.2.14 #define IN_TYPE_NO_SENSOR 0x00

No sensor configured

6.101.2.15 #define IN_TYPE_REFLECTION 0x03

RCX light sensor

6.101.2.16 #define IN_TYPE_SOUND_DB 0x07

NXT sound sensor with dB scaling

6.101.2.17 #define IN_TYPE_SOUND_DBA 0x08

NXT sound sensor with dBA scaling

6.101.2.18 #define IN_TYPE_SWITCH 0x01

NXT or RCX touch sensor

6.101.2.19 #define IN_TYPE_TEMPERATURE 0x02

RCX temperature sensor

6.102 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

6.102.1 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\(\)](#)**6.102.2 Define Documentation****6.102.2.1 #define IN_MODE_ANGLESTEP 0xE0**

RCX rotation sensor (16 ticks per revolution)

6.102.2.2 #define IN_MODE_BOOLEAN 0x20

Boolean value (0 or 1)

6.102.2.3 #define IN_MODE_CELSIUS 0xA0

RCX temperature sensor value in degrees celcius

6.102.2.4 #define IN_MODE_FAHRENHEIT 0xC0

RCX temperature sensor value in degrees fahrenheit

6.102.2.5 #define IN_MODE_MODEMASK 0xE0

Mask for the mode without any slope value

6.102.2.6 #define IN_MODE_PCTFULLSCALE 0x80

Scaled value from 0 to 100

6.102.2.7 #define IN_MODE_PERIODCOUNTER 0x60

Counts the number of boolean periods

6.102.2.8 #define IN_MODE_RAW 0x00

Raw value from 0 to 1023

6.102.2.9 #define IN_MODE_SLOPEMASK 0x1F

Mask for slope parameter added to mode

6.102.2.10 #define IN_MODE_TRANSITIONCNT 0x40

Counts the number of boolean transitions

6.103 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

6.103.1 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.

6.103.2 Define Documentation**6.103.2.1 #define InputMode 1**

Input mode field. Contains one of the sensor mode constants. Read/write.

6.103.2.2 #define InvalidData 5

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

6.103.2.3 #define NormalizedValue 3

Normalized value field. Contains the current normalized analog sensor value. Read only.

6.103.2.4 #define RawValue 2

Raw value field. Contains the current raw analog sensor value. Read only.

6.103.2.5 #define ScaledValue 4

Scaled value field. Contains the current scaled analog sensor value. Read/write.

6.103.2.6 #define Type 0

Type field. Contains one of the sensor type constants. Read/write.

Examples:

[ex_GetInput.nxc](#), and [ex_SetInput.nxc](#).

6.104 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

6.104.1 Detailed Description

Constants for use with color sensor value arrays to index RGB and blank return values.

See also:

[ReadSensorColorEx\(\)](#), [ReadSensorColorRaw\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

6.104.2 Define Documentation

6.104.2.1 #define INPUT_BLANK 3

Access the blank value from color sensor value arrays

6.104.2.2 #define INPUT_BLUE 2

Access the blue value from color sensor value arrays

6.104.2.3 #define INPUT_GREEN 1

Access the green value from color sensor value arrays

6.104.2.4 #define INPUT_NO_OF_COLORS 4

The number of entries in the color sensor value arrays

6.104.2.5 #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](#).

6.105 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

6.105.1 Detailed Description

Constants for use with the ColorValue returned by the color sensor in full color mode.

See also:

[SensorValue\(\)](#), [SysColorSensorRead\(\)](#), [ColorSensorReadType](#)

6.105.2 Define Documentation**6.105.2.1 #define INPUT_BLACKCOLOR 1**

The color value is black

6.105.2.2 #define INPUT_BLUECOLOR 2

The color value is blue

6.105.2.3 #define INPUT_GREENCOLOR 3

The color value is green

6.105.2.4 #define INPUT_REDCOLOR 5

The color value is red

6.105.2.5 #define INPUT_WHITECOLOR 6

The color value is white

6.105.2.6 #define INPUT_YELLOWCOLOR 4

The color value is yellow

6.106 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

6.106.1 Detailed Description

Constants for use with the color calibration state function.

See also:

[ColorCalibrationState\(\)](#)

6.106.2 Define Documentation**6.106.2.1 #define INPUT_RESETCAL 0x80**

Unused calibration state constant

6.106.2.2 #define INPUT_RUNNINGCAL 0x20

Unused calibration state constant

6.106.2.3 #define INPUT_SENSORCAL 0x01

The state returned while the color sensor is calibrating

6.106.2.4 #define INPUT_SENSOROFF 0x02

The state returned once calibration has completed

6.106.2.5 #define INPUT_STARTCAL 0x40

Unused calibration state constant

6.107 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

6.107.1 Detailed Description

Constants for use with the color calibration functions.

See also:

[ColorCalibration\(\)](#), [ColorCalLimits\(\)](#)

6.107.2 Define Documentation**6.107.2.1 #define INPUT_CAL_POINT_0 0**

Calibration point 0

Examples:

[ex_ColorCalibration.nxc](#), and [ex_ColorCalLimits.nxc](#).

6.107.2.2 #define INPUT_CAL_POINT_1 1

Calibration point 1

6.107.2.3 #define INPUT_CAL_POINT_2 2

Calibration point 2

6.107.2.4 #define INPUT_NO_OF_POINTS 3

The number of calibration points

6.108 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)

6.108.1 Detailed Description

Constant offsets into the Input module IOMAP structure.

6.108.2 Define Documentation

6.108.2.1 #define InputOffsetADRaw(p) (((p)*20)+2)

Read the AD raw sensor value (2 bytes) uword

6.108.2.2 #define InputOffsetColorADRaw(p, nc) (80+((p)*84)+52+((nc)*2))

Read AD raw color sensor values

6.108.2.3 #define InputOffsetColorCalibration(p, np, nc) (80+((p)*84)+0+((np)*16)+((nc)*4))

Read/write color calibration point values

6.108.2.4 #define InputOffsetColorCalibrationState(p) (80+((p)*84)+80)

Read color sensor calibration state

6.108.2.5 #define InputOffsetColorCalLimits(p, np) (80+((p)*84)+48+((np)*2))

Read/write color calibration limits

6.108.2.6 #define InputOffsetColorSensorBoolean(p, nc) (80+((p)*84)+76+((nc)*2))

Read color sensor boolean values

6.108.2.7 #define InputOffsetColorSensorRaw(p, nc) (80+((p)*84)+60+((nc)*2))

Read raw color sensor values

6.108.2.8 #define InputOffsetColorSensorValue(p, nc) (80+((p)*84)+68+((nc)*2))

Read scaled color sensor values

6.108.2.9 #define InputOffsetCustomActiveStatus(p) (((p)*20)+15)

Read/write the active or inactive state of the custom sensor

6.108.2.10 #define InputOffsetCustomPctFullScale(p) (((p)*20)+14)

Read/write the Pct full scale of the custom sensor

6.108.2.11 #define InputOffsetCustomZeroOffset(p) (((p)*20)+0)

Read/write the zero offset of a custom sensor (2 bytes) uword

6.108.2.12 #define InputOffsetDigiPinsDir(p) (((p)*20)+11)

Read/write the direction of the Digital pins (1 is output, 0 is input)

6.108.2.13 #define InputOffsetDigiPinsIn(p) (((p)*20)+12)

Read/write the status of the digital pins

6.108.2.14 #define InputOffsetDigiPinsOut(p) (((p)*20)+13)

Read/write the output level of the digital pins

6.108.2.15 #define InputOffsetInvalidData(p) (((p)*20)+16)

Indicates whether data is invalid (1) or valid (0)

6.108.2.16 #define InputOffsetSensorBoolean(p) (((p)*20)+10)

Read the sensor boolean value

6.108.2.17 #define InputOffsetSensorMode(p) (((p)*20)+9)

Read/write the sensor mode

6.108.2.18 #define InputOffsetSensorRaw(p) (((p)*20)+4)

Read the raw sensor value (2 bytes) uword

6.108.2.19 #define InputOffsetSensorType(p) (((p)*20)+8)

Read/write the sensor type

6.108.2.20 #define InputOffsetSensorValue(p) (((p)*20)+6)

Read/write the scaled sensor value (2 bytes) sword

6.109 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

6.109.1 Detailed Description

Output port constants are used when calling motor control API functions.

6.109.2 Define Documentation

6.109.2.1 #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_-  
motorregvalue.nxc, ex_motorregpvalue.nxc, ex_motorregulation.nxc, ex_-  
motorrotationcount.nxc, ex_motorrunstate.nxc, ex_motortachocount.nxc,  
ex_motortacholimit.nxc, ex_motorturnratio.nxc, ex_off.nxc, ex_offex.nxc,  
ex_onfwd.nxc, ex_onfwdex.nxc, ex_onfwdreg.nxc, ex_onfwdregex.nxc, ex_-  
onfwdregexpid.nxc, ex_onfwdregpid.nxc, ex_onrev.nxc, ex_onrevex.nxc,
```

`ex_onrevreg.nxc`, `ex_onrevregex.nxc`, `ex_onrevregpid.nxc`, `ex_onrevregpid.nxc`,
`ex_RemoteResetMotorPosition.nxc`, `ex_RemoteSetOutputState.nxc`, `ex_-`
`rotatemotor.nxc`, `ex_rotatemotorpid.nxc`, and `ex_yield.nxc`.

6.109.2.2 #define OUT_AB 0x03

Output ports A and B

Examples:

`ex_onfwdsync.nxc`, `ex_onfwdsyncex.nxc`, `ex_onfwdsyncexpid.nxc`,
`ex_onfwdsynccpid.nxc`, `ex_onrevsync.nxc`, `ex_onrevsyncex.nxc`, `ex_-`
`onrevsyncexpid.nxc`, `ex_onrevsyncexpid.nxc`, `ex_resetalltachocounts.nxc`, `ex_-`
`resetblocktachocount.nxc`, `ex_resetrotationcount.nxc`, `ex_resettachocount.nxc`,
`ex_rotatemotorex.nxc`, `ex_rotatemotorexpid.nxc`, and `ex_setoutput.nxc`.

6.109.2.3 #define OUT_ABC 0x06

Output ports A, B, and C

6.109.2.4 #define OUT_AC 0x04

Output ports A and C

6.109.2.5 #define OUT_B 0x01

Output port B

6.109.2.6 #define OUT_BC 0x05

Output ports B and C

6.109.2.7 #define OUT_C 0x02

Output port C

6.110 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

6.110.1 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\(\)](#)

6.110.2 Define Documentation**6.110.2.1 #define PID_0 0**

PID zero

6.110.2.2 #define PID_1 32

PID one

6.110.2.3 #define PID_2 64

PID two

6.110.2.4 #define PID_3 96

PID three

6.110.2.5 #define PID_4 128

PID four

6.110.2.6 #define PID_5 160

PID five

6.110.2.7 #define PID_6 192

PID six

6.110.2.8 #define PID_7 224

PID seven

6.111 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

6.111.1 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\(\)](#)

6.111.2 Define Documentation**6.111.2.1 #define UF_PENDING_UPDATES 0x80**

Are there any pending motor updates?

6.111.2.2 #define UF_UPDATE_MODE 0x01

Commits changes to the OutputMode output property

6.111.2.3 #define UF_UPDATE_PID_VALUES 0x10

Commits changes to the PID motor regulation properties

6.111.2.4 #define UF_UPDATE_RESET_BLOCK_COUNT 0x20

Resets the NXT-G block-relative rotation counter

6.111.2.5 #define UF_UPDATE_RESET_COUNT 0x08

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

6.111.2.6 #define UF_UPDATE_RESET_ROTATION_COUNT 0x40

Resets the program-relative (user) rotation counter

6.111.2.7 #define UF_UPDATE_SPEED 0x02

Commits changes to the Power output property

6.111.2.8 #define UF_UPDATE_TACHO_LIMIT 0x04

Commits changes to the TachoLimit output property

6.112 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

6.112.1 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...

6.112.2 Define Documentation**6.112.2.1 #define RESET_ALL 0x68**

Reset all three tachometer counters

6.112.2.2 #define RESET_BLOCK_COUNT 0x20

Reset the NXT-G block tachometer counter

6.112.2.3 #define RESET_BLOCKANDTACHO 0x28

Reset both the internal counter and the NXT-G block counter

6.112.2.4 #define RESET_COUNT 0x08

Reset the internal tachometer counter

6.112.2.5 #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](#).

6.112.2.6 #define RESET_ROTATION_COUNT 0x40

Reset the rotation counter

6.113 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

6.113.1 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\(\)](#)

6.113.2 Define Documentation**6.113.2.1 #define OUT_MODE_BRAKE 0x02**

Uses electronic braking to outputs

6.113.2.2 #define OUT_MODE_COAST 0x00

No power and no braking so motors rotate freely.

6.113.2.3 #define OUT_MODE_MOTORON 0x01

Enables PWM power to the outputs given the Power setting

Examples:

[ex_RemoteSetOutputState.nxc](#).

6.113.2.4 #define OUT_MODE_REGMETHOD 0xF0

Mask for unimplemented regulation mode

6.113.2.5 #define OUT_MODE_REGULATED 0x04

Enables active power regulation using the RegMode value

6.114 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

6.114.1 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\(\)](#)

6.114.2 Define Documentation**6.114.2.1 #define OUT_OPTION_HOLDATLIMIT 0x10**

Option to have the firmware hold the motor when it reaches the tachometer limit

6.114.2.2 #define OUT_OPTION_RAMPDOWNTOLIMIT 0x20

Option to have the firmware rampdown the motor power as it approaches the tachometer limit (not implemented yet)

6.115 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

6.115.1 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\(\)](#)

6.115.2 Define Documentation

6.115.2.1 #define OUT_RUNSTATE_HOLD 0x60

Set motor run state to hold at the current position.

6.115.2.2 #define OUT_RUNSTATE_IDLE 0x00

Disable all power to motors.

6.115.2.3 #define OUT_RUNSTATE_RAMPDOWN 0x40

Enable ramping down from a current Power to a new (lower) Power over a specified TachoLimit goal.

6.115.2.4 #define OUT_RUNSTATE_RAMPUP 0x10

Enable ramping up from a current Power to a new (higher) Power over a specified TachoLimit goal.

6.115.2.5 #define OUT_RUNSTATE_RUNNING 0x20

Enable power to motors at the specified Power level.

Examples:

[ex_RemoteSetOutputState.nxc](#).

6.116 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

6.116.1 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\(\)](#)

6.116.2 Define Documentation**6.116.2.1 #define OUT_REGMODE_IDLE 0**

No motor regulation.

Examples:

[ex_RemoteSetOutputState.nxc](#).

6.116.2.2 #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](#).

6.116.2.3 #define OUT_REGMODE_SYNC 2

Synchronize the rotation of two motors.

6.117 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.

6.117.1 Detailed Description

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

See also:

[SetOutput\(\)](#), [GetOutput\(\)](#)

6.117.2 Define Documentation

6.117.2.1 #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.

6.117.2.2 #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.

6.117.2.3 #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.

6.117.2.4 #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).

6.117.2.5 #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](#).

6.117.2.6 #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.

6.117.2.7 #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.

6.117.2.8 #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.

6.117.2.9 #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 sync 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.

6.117.2.10 #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 algorithm. Set [UF_UPDATE_PID_VALUES](#) to commit changes to RegPValue, RegIValue, and RegDValue simultaneously.

6.117.2.11 #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.

6.117.2.12 #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](#).

6.117.2.13 #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.

6.117.2.14 #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](#).

6.117.2.15 #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.

6.117.2.16 #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.

6.118 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`

6.118.1 Detailed Description

Constant offsets into the Output module IOMAP structure.

6.118.2 Define Documentation

6.118.2.1 `#define OutputOffsetActualSpeed(p) (((p)*32)+21)`

R - Holds the current motor speed (1 byte) sbyte

6.118.2.2 `#define OutputOffsetBlockTachoCount(p) (((p)*32)+4)`

R - Holds current number of counts for the current output block (4 bytes) slong

6.118.2.3 #define OutputOffsetFlags(p) (((p)*32)+18)

RW - Holds flags for which data should be updated (1 byte) ubyte

6.118.2.4 #define OutputOffsetMode(p) (((p)*32)+19)

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

6.118.2.5 #define OutputOffsetMotorRPM(p) (((p)*32)+16)

Not updated, will be removed later !! (2 bytes) sword

6.118.2.6 #define OutputOffsetOptions(p) (((p)*32)+29)

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

6.118.2.7 #define OutputOffsetOverloaded(p) (((p)*32)+27)

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

6.118.2.8 #define OutputOffsetPwnFreq 96

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

6.118.2.9 #define OutputOffsetRegDParameter(p) (((p)*32)+24)

RW - Holds the D-constant used in the regulation (1 byte) ubyte

6.118.2.10 #define OutputOffsetRegIParameter(p) (((p)*32)+23)

RW - Holds the I-constant used in the regulation (1 byte) ubyte

6.118.2.11 #define OutputOffsetRegMode(p) (((p)*32)+26)

RW - Tells which regulation mode should be used (1 byte) ubyte

6.118.2.12 #define OutputOffsetRegPParameter(p) (((p)*32)+22)

RW - Holds the P-constant used in the regulation (1 byte) ubyte

6.118.2.13 #define OutputOffsetRotationCount(p) (((p)*32)+8)

R - Holds current number of counts for the rotation counter to the output (4 bytes)
slong

6.118.2.14 #define OutputOffsetRunState(p) (((p)*32)+25)

RW - Holds the current RunState in the output module (1 byte) ubyte

6.118.2.15 #define OutputOffsetSpeed(p) (((p)*32)+20)

RW - Holds the wanted speed (1 byte) sbyte

6.118.2.16 #define OutputOffsetSyncTurnParameter(p) (((p)*32)+28)

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

6.118.2.17 #define OutputOffsetTachoCount(p) (((p)*32)+0)

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes)
slong

6.118.2.18 #define OutputOffsetTachoLimit(p) (((p)*32)+12)

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

6.119 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.

- [LEGO I2C address constants](#)

Constants for LEGO I2C device addresses.

- [Ultrasonic sensor constants](#)

Constants for use with the ultrasonic sensor.

- [LEGO temperature sensor constants](#)

Constants for use with the LEGO temperature sensor.

- [E-Meter sensor constants](#)

Constants for use with the e-meter sensor.

6.119.1 Detailed Description

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

6.120 LSState constants

Constants for the low speed module LSState function.

Defines

- #define COM_CHANNEL_NONE_ACTIVE 0x00
- #define COM_CHANNEL_ONE_ACTIVE 0x01
- #define COM_CHANNEL_TWO_ACTIVE 0x02
- #define COM_CHANNEL_THREE_ACTIVE 0x04
- #define COM_CHANNEL_FOUR_ACTIVE 0x08

6.120.1 Detailed Description

Constants for the low speed module LSState function. These values are combined together using a bitwise OR operation.

See also:

[LSState\(\)](#)

6.120.2 Define Documentation**6.120.2.1 #define COM_CHANNEL_FOUR_ACTIVE 0x08**

Low speed channel 4 is active

6.120.2.2 #define COM_CHANNEL_NONE_ACTIVE 0x00

None of the low speed channels are active

6.120.2.3 #define COM_CHANNEL_ONE_ACTIVE 0x01

Low speed channel 1 is active

6.120.2.4 #define COM_CHANNEL_THREE_ACTIVE 0x04

Low speed channel 3 is active

6.120.2.5 #define COM_CHANNEL_TWO_ACTIVE 0x02

Low speed channel 2 is active

6.121 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

6.121.1 Detailed Description

Constants for the low speed module LSChannelState function.

See also:

[LSChannelState\(\)](#)

6.121.2 Define Documentation**6.121.2.1 #define LOWSPEED_COMMUNICATING 3**

Channel is actively communicating

6.121.2.2 #define LOWSPEED_DONE 5

Channel is done communicating

6.121.2.3 #define LOWSPEED_ERROR 4

Channel is in an error state

6.121.2.4 #define LOWSPEED_IDLE 0

Channel is idle

Examples:

[ex_syscommlscheckstatus.nxc](#).

6.121.2.5 #define LOWSPEED_INIT 1

Channel is being initialized

6.121.2.6 #define LOWSPEED_LOAD_BUFFER 2

Channel buffer is loading

6.122 LMode constants

Constants for the low speed module LMode function.

Defines

- #define [LOWSPEED_TRANSMITTING](#) 1
- #define [LOWSPEED RECEIVING](#) 2
- #define [LOWSPEED_DATA_RECEIVED](#) 3

6.122.1 Detailed Description

Constants for the low speed module LMode function.

See also:

[LMode\(\)](#)

6.122.2 Define Documentation**6.122.2.1 #define LOWSPEED_DATA_RECEIVED 3**

Lowspeed port is in data received mode

6.122.2.2 #define LOWSPEED_RECEIVING 2

Lowspeed port is in receiving mode

6.122.2.3 #define LOWSPEED_TRANSMITTING 1

Lowspeed port is in transmitting mode

6.123 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

6.123.1 Detailed Description

Constants for the low speed module LSErrorType function.

See also:

[LSErrorType\(\)](#)

6.123.2 Define Documentation**6.123.2.1 #define LOWSPEED_CH_NOT_READY 1**

Lowspeed port is not ready

6.123.2.2 #define LOWSPEED_NO_ERROR 0

Lowspeed port has no error

6.123.2.3 #define LOWSPEED_RX_ERROR 3

Lowspeed port encountered an error while receiving data

6.123.2.4 #define LOWSPEED_TX_ERROR 2

Lowspeed port encountered an error while transmitting data

6.124 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`

6.124.1 Detailed Description

Constant offsets into the low speed module IOMAP structure.

6.124.2 Define Documentation

6.124.2.1 #define `LowSpeedOffsetChannelState(p) ((p)+156)`

R - Lowspeed channel state (1 byte)

6.124.2.2 #define `LowSpeedOffsetErrorType(p) ((p)+160)`

R - Lowspeed port error type (1 byte)

6.124.2.3 #define `LowSpeedOffsetInBufBuf(p) (((p)*19)+0)`

RW - Input buffer data buffer field offset (16 bytes)

6.124.2.4 #define `LowSpeedOffsetInBufBytesToRx(p) (((p)*19)+18)`

RW - Input buffer bytes to receive field offset (1 byte)

6.124.2.5 #define `LowSpeedOffsetInBufInPtr(p) (((p)*19)+16)`

RW - Input buffer in pointer field offset (1 byte)

6.124.2.6 #define LowSpeedOffsetInBufOutPtr(p) (((p)*19)+17)

RW - Input buffer out pointer field offset (1 byte)

6.124.2.7 #define LowSpeedOffsetMode(p) ((p)+152)

R - Lowspeed port mode (1 byte)

6.124.2.8 #define LowSpeedOffsetNoRestartOnRead 166

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

6.124.2.9 #define LowSpeedOffsetOutBufBuf(p) (((p)*19)+76)

RW - Output buffer data buffer field offset (16 bytes)

6.124.2.10 #define LowSpeedOffsetOutBufBytesToRx(p) (((p)*19)+94)

RW - Output buffer bytes to receive field offset (1 byte)

6.124.2.11 #define LowSpeedOffsetOutBufInPtr(p) (((p)*19)+92)

RW - Output buffer in pointer field offset (1 byte)

6.124.2.12 #define LowSpeedOffsetOutBufOutPtr(p) (((p)*19)+93)

RW - Output buffer out pointer field offset (1 byte)

6.124.2.13 #define LowSpeedOffsetSpeed 165

R - Lowspeed speed (unused)

6.124.2.14 #define LowSpeedOffsetState 164

R - Lowspeed state (all channels)

6.125 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

6.125.1 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\(\)](#)

6.125.2 Define Documentation

6.125.2.1 #define LSREAD_NO_RESTART_1 0x01

No restart on read for channel 1

6.125.2.2 #define LSREAD_NO_RESTART_2 0x02

No restart on read for channel 2

6.125.2.3 #define LSREAD_NO_RESTART_3 0x04

No restart on read for channel 3

6.125.2.4 #define LSREAD_NO_RESTART_4 0x08

No restart on read for channel 4

6.125.2.5 #define LSREAD_NO_RESTART_MASK 0x10

No restart mask

6.125.2.6 #define LSREAD_RESTART_ALL 0x00

Restart on read for all channels (default)

6.125.2.7 #define LSREAD_RESTART_NONE 0x0F

No restart on read for all channels

6.126 Standard I2C constants

Constants for use with standard I2C devices.

Defines

- #define I2C_ADDR_DEFAULT 0x02
- #define I2C_REG_VERSION 0x00
- #define I2C_REG_VENDOR_ID 0x08
- #define I2C_REG_DEVICE_ID 0x10
- #define I2C_REG_CMD 0x41

6.126.1 Detailed Description

Constants for use with standard I2C devices.

6.126.2 Define Documentation**6.126.2.1 #define I2C_ADDR_DEFAULT 0x02**

Standard NXT I2C device address

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CSendCommand.nxc](#),
[ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_MSDenergize.nxc](#), [ex_-MSEnergize.nxc](#), [ex_MSIRTrain.nxc](#), [ex_MSPFComboDirect.nxc](#), [ex_-MSPFComboPWM.nxc](#), [ex_MSPFRawOutput.nxc](#), [ex_MSPFRepeat.nxc](#),

[ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), [ex_MSPFSinglePin.nxc](#), [ex_MSPFTrain.nxc](#), [ex_MSReadValue.nxc](#), [ex_readi2cregister.nxc](#), and [ex_writei2cregister.nxc](#).

6.126.2.2 #define I2C_REG_CMD 0x41

Standard NXT I2C device command register

Examples:

[ex_MSReadValue.nxc](#), [ex_readi2cregister.nxc](#), and [ex_writei2cregister.nxc](#).

6.126.2.3 #define I2C_REG_DEVICE_ID 0x10

Standard NXT I2C device ID register

Examples:

[ex_I2CDeviceInfo.nxc](#).

6.126.2.4 #define I2C_REG_VENDOR_ID 0x08

Standard NXT I2C vendor ID register

Examples:

[ex_I2CDeviceInfo.nxc](#).

6.126.2.5 #define I2C_REG_VERSION 0x00

Standard NXT I2C version register

Examples:

[ex_I2CDeviceInfo.nxc](#).

6.127 LEGO I2C address constants

Constants for LEGO I2C device addresses.

Defines

- #define **LEGO_ADDR_US** 0x02
- #define **LEGO_ADDR_TEMP** 0x98
- #define **LEGO_ADDR_EMETER** 0x04

6.127.1 Detailed Description

Constants for LEGO I2C device addresses.

6.127.2 Define Documentation**6.127.2.1 #define LEGO_ADDR_EMETER 0x04**

The LEGO e-meter sensor's I2C address

6.127.2.2 #define LEGO_ADDR_TEMP 0x98

The LEGO temperature sensor's I2C address

6.127.2.3 #define LEGO_ADDR_US 0x02

The LEGO ultrasonic sensor's I2C address

6.128 Ultrasonic sensor constants

Constants for use with the ultrasonic sensor.

Defines

- #define **US_CMD_OFF** 0x00
- #define **US_CMD_SINGLESHT** 0x01
- #define **US_CMD_CONTINUOUS** 0x02
- #define **US_CMD_EVENTCAPTURE** 0x03
- #define **US_CMD_WARMRESET** 0x04
- #define **US_REG_CM_INTERVAL** 0x40
- #define **US_REG_ACTUAL_ZERO** 0x50
- #define **US_REG_SCALE_FACTOR** 0x51
- #define **US_REG_SCALE_DIVISOR** 0x52
- #define **US_REG_FACTORY_ACTUAL_ZERO** 0x11

- #define **US_REG_FACTORY_SCALE_FACTOR** 0x12
- #define **US_REG_FACTORY_SCALE_DIVISOR** 0x13
- #define **US_REG_MEASUREMENT_UNITS** 0x14

6.128.1 Detailed Description

Constants for use with the ultrasonic sensor.

6.128.2 Define Documentation

6.128.2.1 #define **US_CMD_CONTINUOUS** 0x02

Command to put the ultrasonic sensor into continuous polling mode (default)

6.128.2.2 #define **US_CMD_EVENTCAPTURE** 0x03

Command to put the ultrasonic sensor into event capture mode

6.128.2.3 #define **US_CMD_OFF** 0x00

Command to turn off the ultrasonic sensor

Examples:

[ex_writei2cregister.nxc](#).

6.128.2.4 #define **US_CMD_SINGLESHOT** 0x01

Command to put the ultrasonic sensor into single shot mode

6.128.2.5 #define **US_CMD_WARMRESET** 0x04

Command to warm reset the ultrasonic sensor

6.128.2.6 #define **US_REG_ACTUAL_ZERO** 0x50

The register address used to store the actual zero value

6.128.2.7 #define **US_REG_CM_INTERVAL** 0x40

The register address used to store the CM interval

6.128.2.8 #define US_REG_FACTORY_ACTUAL_ZERO 0x11

The register address containing the factory setting for the actual zero value

6.128.2.9 #define US_REG_FACTORY_SCALE_DIVISOR 0x13

The register address containing the factory setting for the scale divisor value

6.128.2.10 #define US_REG_FACTORY_SCALE_FACTOR 0x12

The register address containing the factory setting for the scale factor value

6.128.2.11 #define US_REG_MEASUREMENT_UNITS 0x14

The register address containing the measurement units (degrees C or F)

6.128.2.12 #define US_REG_SCALE_DIVISOR 0x52

The register address used to store the scale divisor value

6.128.2.13 #define US_REG_SCALE_FACTOR 0x51

The register address used to store the scale factor value

6.129 LEGO temperature sensor constants

Constants for use with the LEGO 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_REG_TEMP 0x00
- #define TEMP_REG_CONFIG 0x01
- #define TEMP_REG_TLOW 0x02
- #define TEMP_REG_THIGH 0x03

6.129.1 Detailed Description

Constants for use with the LEGO temperature sensor.

6.129.2 Define Documentation

6.129.2.1 #define TEMP_FQ_1 0x00

Set fault queue 1

6.129.2.2 #define TEMP_FQ_2 0x08

Set fault queue 2

6.129.2.3 #define TEMP_FQ_4 0x10

Set fault queue 4

6.129.2.4 #define TEMP_FQ_6 0x18

Set fault queue 6

6.129.2.5 #define TEMP_OS_ONESHOT 0x80

Set the sensor into oneshot mode

6.129.2.6 #define TEMP_POL_HIGH 0x04

Set polarity to high

6.129.2.7 #define TEMP_POL_LOW 0x00

Set polarity to low

6.129.2.8 #define TEMP_REG_CONFIG 0x01

The register for reading/writing sensor configuration values

6.129.2.9 #define TEMP_REG_TEMP 0x00

The register where temperature values can be read

6.129.2.10 #define TEMP_REG_THIGH 0x03

The register where temperature high values can be read

6.129.2.11 #define TEMP_REG_TLOW 0x02

The register where temperature low values can be read

6.129.2.12 #define TEMP_RES_10BIT 0x20

Set the resolution to 10 bit

6.129.2.13 #define TEMP_RES_11BIT 0x40

Set the resolution to 11 bit

6.129.2.14 #define TEMP_RES_12BIT 0x60

Set the resolution to 12 bit

6.129.2.15 #define TEMP_RES_9BIT 0x00

Set the resolution to 9 bit

6.129.2.16 #define TEMP_SD_CONTINUOUS 0x00

Set the mode to continuous

6.129.2.17 #define TEMP_SD_SHUTDOWN 0x01

Set the mode to shutdown

6.129.2.18 #define TEMP_TM_COMPARATOR 0x00

Set the thermostat mode to comparator

6.129.2.19 #define TEMP_TM_INTERRUPT 0x02

Set the thermostat mode to interrupt

6.130 E-Meter sensor constants

Constants for use with the e-meter sensor.

Defines

- #define EMETER_REG_VIN 0x0a
- #define EMETER_REG_AIN 0x0c
- #define EMETER_REG_VOUT 0x0e
- #define EMETER_REG_AOUT 0x10
- #define EMETER_REG_JOULES 0x12
- #define EMETER_REG_WIN 0x14
- #define EMETER_REG_WOUT 0x16

6.130.1 Detailed Description

Constants for use with the e-meter sensor.

6.130.2 Define Documentation**6.130.2.1 #define EMETER_REG_AIN 0x0c**

The register address for amps in

6.130.2.2 #define EMETER_REG_AOUT 0x10

The register address for amps out

6.130.2.3 #define EMETER_REG_JOULES 0x12

The register address for joules

6.130.2.4 #define EMETER_REG_VIN 0x0a

The register address for voltage in

6.130.2.5 #define EMETER_REG_VOUT 0x0e

The register address for voltage out

6.130.2.6 #define EMETER_REG_WIN 0x14

The register address for watts in

6.130.2.7 #define EMETER_REG_WOUT 0x16

The register address for watts out

6.131 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_OFFSET 7
- #define DISPLAY_MENUICONS_X_DIFF 31
- #define MENUICON_LEFT 0
- #define MENUICON_CENTER 1
- #define MENUICON_RIGHT 2
- #define MENUICONS 3
- #define FRAME_SELECT 0
- #define STATUSTEXT 1
- #define MENUTEXT 2
- #define STEPLINE 3
- #define TOPLINE 4
- #define SPECIALS 5
- #define STATUSICON_BLUETOOTH 0
- #define STATUSICON_USB 1
- #define STATUSICON_VM 2
- #define STATUSICON_BATTERY 3
- #define STATUSICONS 4
- #define SCREEN_BACKGROUND 0
- #define SCREEN_LARGE 1
- #define SCREEN_SMALL 2
- #define SCREENS 3
- #define BITMAP_1 0
- #define BITMAP_2 1
- #define BITMAP_3 2
- #define BITMAP_4 3
- #define BITMAPS 4

- #define STEPICON_1 0
- #define STEPICON_2 1
- #define STEPICON_3 2
- #define STEPICON_4 3
- #define STEPICON_5 4
- #define STEPICONS 5

6.131.1 Detailed Description

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

6.131.2 Define Documentation

6.131.2.1 #define BITMAP_1 0

Bitmap 1

6.131.2.2 #define BITMAP_2 1

Bitmap 2

6.131.2.3 #define BITMAP_3 2

Bitmap 3

6.131.2.4 #define BITMAP_4 3

Bitmap 4

6.131.2.5 #define BITMAPS 4

The number of bitmap bits

6.131.2.6 #define DISPLAY_HEIGHT 64

The height of the LCD screen in pixels

Examples:

[ex_LineOut.nxc](#).

6.131.2.7 #define DISPLAY_MENUICONS_X_DIFF 31

6.131.2.8 #define DISPLAY_MENUICONS_X_OFFSET 7

6.131.2.9 #define DISPLAY_MENUICONS_Y 40

6.131.2.10 #define DISPLAY_WIDTH 100

The width of the LCD screen in pixels

Examples:

[ex_LineOut.nxc](#).

6.131.2.11 #define FRAME_SELECT 0

Center icon select frame

6.131.2.12 #define MENUICON_CENTER 1

Center icon

6.131.2.13 #define MENUICON_LEFT 0

Left icon

6.131.2.14 #define MENUICON_RIGHT 2

Right icon

6.131.2.15 #define MENUICONS 3

The number of menu icons

6.131.2.16 #define MENUTEXT 2

Center icon text

6.131.2.17 #define SCREEN_BACKGROUND 0

Entire screen

6.131.2.18 #define SCREEN_LARGE 1

Entire screen except status line

6.131.2.19 #define SCREEN_MODE_CLEAR 0x01

Clear the screen

See also:

[SetScreenMode\(\)](#)

6.131.2.20 #define SCREEN_MODE_RESTORE 0x00

Restore the screen

See also:

[SetScreenMode\(\)](#)

6.131.2.21 #define SCREEN_SMALL 2

Screen between menu icons and status line

6.131.2.22 #define SCREENS 3

The number of screen bits

6.131.2.23 #define SPECIALS 5

The number of special bit values

6.131.2.24 #define STATUSICON_BATTERY 3

Battery status icon collection

6.131.2.25 #define STATUSICON_BLUETOOTH 0

BlueTooth status icon collection

6.131.2.26 #define STATUSICON_USB 1

USB status icon collection

6.131.2.27 #define STATUSICON_VM 2

VM status icon collection

6.131.2.28 #define STATUSICONS 4

The number of status icons

6.131.2.29 #define STATUSTEXT 1

Status text (BT name)

6.131.2.30 #define STEPICON_1 0

Left most step icon

6.131.2.31 #define STEPICON_2 1

6.131.2.32 #define STEPICON_3 2

6.131.2.33 #define STEPICON_4 3

6.131.2.34 #define STEPICON_5 4

Right most step icon

6.131.2.35 #define STEPICONS 5**6.131.2.36 #define STEPLINE 3**

Step collection lines

6.131.2.37 #define TOPLINE 4

Top status underline

6.132 DisplayExecuteFunction constants

Constants that are for use with the DisplayExecuteFunction system call.

Defines

- #define DISPLAY_ERASE_ALL 0x00
- #define DISPLAY_PIXEL 0x01
- #define DISPLAY_HORIZONTAL_LINE 0x02
- #define DISPLAY_VERTICAL_LINE 0x03
- #define DISPLAY_CHAR 0x04
- #define DISPLAY_ERASE_LINE 0x05
- #define DISPLAY_FILL_REGION 0x06
- #define DISPLAY_FRAME 0x07

6.132.1 Detailed Description

Constants that are for use with the DisplayExecuteFunction system call.

6.132.2 Define Documentation

6.132.2.1 #define DISPLAY_CHAR 0x04

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

6.132.2.2 #define DISPLAY_ERASE_ALL 0x00

W - erase entire screen (CMD,x,x,x,x,x)

Examples:

[ex_sysdisplayexecutefunction.nxc](#).

6.132.2.3 #define DISPLAY_ERASE_LINE 0x05

W - erase a single line (CMD,x,LINE,x,x,x)

6.132.2.4 #define DISPLAY_FILL_REGION 0x06

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

6.132.2.5 #define DISPLAY_FRAME 0x07

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

6.132.2.6 #define DISPLAY_HORIZONTAL_LINE 0x02

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

Examples:

[ex_dispfunc.nxc](#).

6.132.2.7 #define DISPLAY_PIXEL 0x01

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

6.132.2.8 #define DISPLAY_VERTICAL_LINE 0x03

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

6.133 Drawing option constants

Constants that are for specifying drawing options in several display module API functions.

Modules

- [Font drawing option constants](#)

These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.

Defines

- #define [DRAW_OPT_NORMAL](#) (0x0000)
- #define [DRAW_OPT_CLEAR_WHOLE_SCREEN](#) (0x0001)
- #define [DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN](#) (0x0002)
- #define [DRAW_OPT_CLEAR_PIXELS](#) (0x0004)
- #define [DRAW_OPT_CLEAR](#) (0x0004)
- #define [DRAW_OPT_INVERT](#) (0x0004)
- #define [DRAW_OPT_LOGICAL_COPY](#) (0x0000)
- #define [DRAW_OPT_LOGICAL_AND](#) (0x0008)
- #define [DRAW_OPT_LOGICAL_OR](#) (0x0010)
- #define [DRAW_OPT_LOGICAL_XOR](#) (0x0018)
- #define [DRAW_OPT_FILL_SHAPE](#) (0x0020)
- #define [DRAW_OPT_CLEAR_SCREEN_MODES](#) (0x0003)
- #define [DRAW_OPT_LOGICAL_OPERATIONS](#) (0x0018)
- #define [DRAW_OPT_POLYGON_POLYLINE](#) (0x0400)

6.133.1 Detailed Description

Constants that are for specifying drawing options in several display module API functions. Bits 0 & 1 (values 0,1,2,3) control screen clearing behaviour (Not within RIC files). Bit 2 (value 4) controls the NOT operation, i.e. draw in white or invert text/graphics. Bits 3 & 4 (values 0,8,16,24) control pixel logical combinations (COPY/AND/OR/XOR). Bit 5 (value 32) controls shape filling, or overrides text/graphic bitmaps with set pixels. These may be ORed together for the full instruction (e.g., DRAW_OPT_NORMAL|DRAW_OPT_LOGICAL_XOR) These operations are resolved into the separate, common parameters defined in 'c_display.iom' before any drawing function is called. Note that when drawing a RIC file, the initial 'DrawingOptions' parameter supplied in the drawing instruction controls screen clearing, but nothing else. The 'CopyOptions' parameter from each instruction in the RIC file then controls graphic operations, but the screen-clearing bits are ignored.

See also:

[TextOut\(\)](#), [NumOut\(\)](#), [PointOut\(\)](#), [LineOut\(\)](#), [CircleOut\(\)](#), [RectOut\(\)](#), [PolyOut\(\)](#),
[EllipseOut\(\)](#), [FontTextOut\(\)](#), [FontNumOut\(\)](#), [GraphicOut\(\)](#), [GraphicArrayOut\(\)](#)

6.133.2 Define Documentation**6.133.2.1 #define DRAW_OPT_CLEAR (0x0004)**

Clear pixels while drawing (aka draw in white)

**6.133.2.2 #define DRAW_OPT_CLEAR_EXCEPT_STATUS_-
SCREEN (0x0002)**

Clear the screen except for the status line before drawing

6.133.2.3 #define DRAW_OPT_CLEAR_PIXELS (0x0004)

Clear pixels while drawing (aka draw in white)

6.133.2.4 #define DRAW_OPT_CLEAR_SCREEN_MODES (0x0003)

Bit mask for the clear screen modes

6.133.2.5 #define DRAW_OPT_CLEAR_WHOLE_SCREEN (0x0001)

Clear the entire screen before drawing

Examples:

[ex_dispoutex.nxc](#).

6.133.2.6 #define DRAW_OPT_FILL_SHAPE (0x0020)

Fill the shape while drawing (rectangle, circle, ellipses, and polygon)

Examples:

[ex_CircleOut.nxc](#), [ex_EllipseOut.nxc](#), [ex_PolyOut.nxc](#), [ex_sysdrawellipse.nxc](#),
and [ex_sysdrawpolygon.nxc](#).

6.133.2.7 #define DRAW_OPT_INVERT (0x0004)

Invert text or graphics

Examples:

[ex_dispftout.nxc](#).

6.133.2.8 #define DRAW_OPT_LOGICAL_AND (0x0008)

Draw pixels using a logical AND operation

Examples:

[ex_dispftout.nxc](#).

6.133.2.9 #define DRAW_OPT_LOGICAL_COPY (0x0000)

Draw pixels using a logical copy operation

6.133.2.10 #define DRAW_OPT_LOGICAL_OPERATIONS (0x0018)

Bit mask for the logical drawing operations

6.133.2.11 #define DRAW_OPT_LOGICAL_OR (0x0010)

Draw pixels using a logical OR operation

Examples:

[ex_dispftout.nxc](#).

6.133.2.12 #define DRAW_OPT_LOGICAL_XOR (0x0018)

Draw pixels using a logical XOR operation

Examples:

[ex_CircleOut.nxc](#), [ex_EllipseOut.nxc](#), [ex_LineOut.nxc](#), [ex_PolyOut.nxc](#), [ex_sysdrawellipse.nxc](#), and [ex_sysdrawpolygon.nxc](#).

6.133.2.13 #define DRAW_OPT_NORMAL (0x0000)

Normal drawing

Examples:

[ex_CircleOut.nxc](#), [ex_disptout.nxc](#), [ex_dispfunc.nxc](#), and [ex_sysdrawfont.nxc](#).

6.133.2.14 #define DRAW_OPT_POLYGON_POLYLINE (0x0400)

When drawing polygons, do not close (i.e., draw a polyline instead)

6.134 Font drawing option constants

These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.

Defines

- #define [DRAW_OPT_FONT_DIRECTIONS](#) (0x01C0)
- #define [DRAW_OPT_FONT_WRAP](#) (0x0200)
- #define [DRAW_OPT_FONT_DIR_L2RB](#) (0x0000)
- #define [DRAW_OPT_FONT_DIR_L2RT](#) (0x0040)
- #define [DRAW_OPT_FONT_DIR_R2LB](#) (0x0080)
- #define [DRAW_OPT_FONT_DIR_R2LT](#) (0x00C0)
- #define [DRAW_OPT_FONT_DIR_B2TL](#) (0x0100)
- #define [DRAW_OPT_FONT_DIR_B2TR](#) (0x0140)
- #define [DRAW_OPT_FONT_DIR_T2BL](#) (0x0180)
- #define [DRAW_OPT_FONT_DIR_T2BR](#) (0x01C0)

6.134.1 Detailed Description

These addition drawing option constants are only for use when drawing text and numbers on the LCD using an RIC-based font.

See also:

[FontTextOut\(\)](#), [FontNumOut\(\)](#)

6.134.2 Define Documentation**6.134.2.1 #define DRAW_OPT_FONT_DIR_B2TL (0x0100)**

Font bottom to top left align

6.134.2.2 #define DRAW_OPT_FONT_DIR_B2TR (0x0140)

Font bottom to top right align

6.134.2.3 #define DRAW_OPT_FONT_DIR_L2RB (0x0000)

Font left to right bottom align

Examples:

[ex_dispftout.nxc](#).

6.134.2.4 #define DRAW_OPT_FONT_DIR_L2RT (0x0040)

Font left to right top align

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

6.134.2.5 #define DRAW_OPT_FONT_DIR_R2LB (0x0080)

Font right to left bottom align

6.134.2.6 #define DRAW_OPT_FONT_DIR_R2LT (0x00C0)

Font right to left top align

6.134.2.7 #define DRAW_OPT_FONT_DIR_T2BL (0x0180)

Font top to bottom left align

Examples:

[ex_dispftout.nxc](#).

6.134.2.8 #define DRAW_OPT_FONT_DIR_T2BR (0x01C0)

Font top to bottom right align

6.134.2.9 #define DRAW_OPT_FONT_DIRECTIONS (0x01C0)

Bit mask for the font direction bits

6.134.2.10 #define DRAW_OPT_FONT_WRAP (0x0200)

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

Examples:

[ex_dispftout.nxc](#).

6.135 Display flags

Constants that are for use with the display flags functions.

Defines

- #define [DISPLAY_ON](#) 0x01
- #define [DISPLAY_REFRESH](#) 0x02
- #define [DISPLAY_POPUP](#) 0x08
- #define [DISPLAY_REFRESH_DISABLED](#) 0x40
- #define [DISPLAY_BUSY](#) 0x80

6.135.1 Detailed Description

Constants that are for use with the display flags functions.

See also:

[SetDisplayFlags\(\)](#), [DisplayFlags\(\)](#)

6.135.2 Define Documentation

6.135.2.1 #define DISPLAY_BUSY 0x80

R - Refresh in progress

6.135.2.2 #define DISPLAY_ON 0x01

W - Display on

6.135.2.3 #define DISPLAY_POPUP 0x08

W - Use popup display memory

Examples:

[ex_dispmiss.nxc](#).

6.135.2.4 #define DISPLAY_REFRESH 0x02

W - Enable refresh

6.135.2.5 #define DISPLAY_REFRESH_DISABLED 0x40

R - Refresh disabled

6.136 Display contrast constants

Constants that are for use with the display contrast API functions.

Defines

- #define [DISPLAY_CONTRAST_DEFAULT](#) 0x5A
- #define [DISPLAY_CONTRAST_MAX](#) 0x7F

6.136.1 Detailed Description

Constants that are for use with the display contrast API functions.

See also:

[SetDisplayContrast\(\)](#), [DisplayContrast\(\)](#)

6.136.2 Define Documentation

6.136.2.1 #define DISPLAY_CONTRAST_DEFAULT 0x5A

Default display contrast value

Examples:

[ex_contrast.nxc](#), and [ex_SetDisplayContrast.nxc](#).

6.136.2.2 #define DISPLAY_CONTRAST_MAX 0x7F

Maximum display contrast value

Examples:

[ex_contrast.nxc](#).

6.137 Text line constants

Constants that are for use with getting/setting display data.

Defines

- #define TEXTLINE_1 0
- #define TEXTLINE_2 1
- #define TEXTLINE_3 2
- #define TEXTLINE_4 3
- #define TEXTLINE_5 4
- #define TEXTLINE_6 5
- #define TEXTLINE_7 6
- #define TEXTLINE_8 7
- #define TEXTLINES 8

6.137.1 Detailed Description

Constants that are for use with getting/setting display data.

See also:

[SetDisplayNormal\(\)](#), [GetDisplayNormal\(\)](#), [SetDisplayPopup\(\)](#), [GetDisplayPopup\(\)](#)

6.137.2 Define Documentation

6.137.2.1 #define TEXTLINE_1 0

Text line 1

Examples:

[ex_GetDisplayNormal.nxc](#), [ex_GetDisplayPopup.nxc](#), [ex_SetDisplayNormal.nxc](#),
and [ex_SetDisplayPopup.nxc](#).

6.137.2.2 #define TEXTLINE_2 1

Text line 2

6.137.2.3 #define TEXTLINE_3 2

Text line 3

6.137.2.4 #define TEXTLINE_4 3

Text line 4

6.137.2.5 #define TEXTLINE_5 4

Text line 5

6.137.2.6 #define TEXTLINE_6 5

Text line 6

6.137.2.7 #define TEXTLINE_7 6

Text line 7

6.137.2.8 #define TEXTLINE_8 7

Text line 8

6.137.2.9 #define TEXTLINES 8

The number of text lines on the LCD

6.138 Display module IOMAP offsets

Constant offsets into the display module IOMAP structure.

Defines

- #define `DisplayOffsetPFunc` 0
- #define `DisplayOffsetEraseMask` 4
- #define `DisplayOffsetUpdateMask` 8
- #define `DisplayOffsetPFont` 12
- #define `DisplayOffsetPTextLines`(*p*) (((*p*)*4)+16)
- #define `DisplayOffsetPStatusText` 48
- #define `DisplayOffsetPStatusIcons` 52
- #define `DisplayOffsetPScreens`(*p*) (((*p*)*4)+56)
- #define `DisplayOffsetPBitmaps`(*p*) (((*p*)*4)+68)
- #define `DisplayOffsetPMenuText` 84
- #define `DisplayOffsetPMenuIcons`(*p*) (((*p*)*4)+88)
- #define `DisplayOffsetPStepIcons` 100
- #define `DisplayOffsetDisplay` 104
- #define `DisplayOffsetStatusIcons`(*p*) ((*p*)+108)
- #define `DisplayOffsetStepIcons`(*p*) ((*p*)+112)
- #define `DisplayOffsetFlags` 117
- #define `DisplayOffsetTextLinesCenterFlags` 118
- #define `DisplayOffsetNormal`(*l*, *w*) (((*l*)*100)+(*w*)+119)
- #define `DisplayOffsetPopup`(*l*, *w*) (((*l*)*100)+(*w*)+919)
- #define `DisplayOffsetContrast` 1719

6.138.1 Detailed Description

Constant offsets into the display module IOMAP structure.

6.138.2 Define Documentation

6.138.2.1 #define `DisplayOffsetContrast` 1719

Adjust the display contrast with this field

6.138.2.2 #define `DisplayOffsetDisplay` 104

Display content copied to physical display every 17 mS

6.138.2.3 #define `DisplayOffsetEraseMask` 4

Section erase mask (executed first)

6.138.2.4 #define DisplayOffsetFlags 117

Update flags enumerated above

6.138.2.5 #define DisplayOffsetNormal(l, w) (((l)*100)+(w)+119)

Raw display memory for normal screen

6.138.2.6 #define DisplayOffsetPBitmaps(p) (((p)*4)+68)

Pointer to free bitmap files

6.138.2.7 #define DisplayOffsetPFont 12

Pointer to font file

6.138.2.8 #define DisplayOffsetPFunc 0

Simple draw entry

6.138.2.9 #define DisplayOffsetPMenuIcons(p) (((p)*4)+88)

Pointer to menu icon images (NULL == none)

6.138.2.10 #define DisplayOffsetPMenuText 84

Pointer to menu icon text (NULL == none)

6.138.2.11 #define DisplayOffsetPopup(l, w) (((l)*100)+(w)+919)

Raw display memory for popup screen

6.138.2.12 #define DisplayOffsetPScreens(p) (((p)*4)+56)

Pointer to screen bitmap file

6.138.2.13 #define DisplayOffsetPStatusIcons 52

Pointer to status icon collection file

6.138.2.14 #define DisplayOffsetPStatusText 48

Pointer to status text string

6.138.2.15 #define DisplayOffsetPStepIcons 100

Pointer to step icon collection file

6.138.2.16 #define DisplayOffsetPTextLines(p) (((p)*4)+16)

Pointer to text strings

6.138.2.17 #define DisplayOffsetStatusIcons(p) ((p)+108)

Index in status icon collection file (index = 0 -> none)

6.138.2.18 #define DisplayOffsetStepIcons(p) ((p)+112)

Index in step icon collection file (index = 0 -> none)

6.138.2.19 #define DisplayOffsetTextLinesCenterFlags 118

Mask to center TextLines

6.138.2.20 #define DisplayOffsetUpdateMask 8

Section update mask (executed next)

6.139 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.

6.139.1 Detailed Description

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

6.140 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

6.140.1 Detailed Description

Miscellaneous constants related to the Comm module.

6.140.2 Define Documentation

6.140.2.1 #define BT_CMD_BYTE 1

Size of Bluetooth command

6.140.2.2 #define BT_DEFAULT_INQUIRY_MAX 0

Bluetooth default inquiry Max (0 == unlimited)

6.140.2.3 #define BT_DEFAULT_INQUIRY_TIMEOUT_LO 15

Bluetooth inquiry timeout (15*1.28 sec = 19.2 sec)

6.140.2.4 #define MAX_BT_MSG_SIZE 60000

Max Bluetooth Message Size

6.140.2.5 #define SIZE_OF_BDADDR 7

Size of Bluetooth Address

6.140.2.6 #define SIZE_OF_BRICK_NAME 8

Size of NXT Brick name

6.140.2.7 #define SIZE_OF_BT_CONNECT_TABLE 4

Size of Bluetooth connection table -- Index 0 is always incoming connection

6.140.2.8 #define SIZE_OF_BT_DEVICE_TABLE 30

Size of Bluetooth device table

6.140.2.9 #define SIZE_OF_BT_NAME 16

Size of Bluetooth name

6.140.2.10 #define SIZE_OF_BT_PINCODE 16

Size of Bluetooth PIN

6.140.2.11 #define SIZE_OF_BTBUF 128

Size of Bluetooth buffer

6.140.2.12 #define SIZE_OF_CLASS_OF_DEVICE 4

Size of class of device

6.140.2.13 #define SIZE_OF_HSBUF 128

Size of High Speed Port 4 buffer

6.140.2.14 #define SIZE_OF_USBBUF 64

Size of USB Buffer in bytes

6.140.2.15 #define SIZE_OF_USBDATA 62

Size of USB Buffer available for data

6.140.2.16 #define USB_PROTOCOL_OVERHEAD 2

Size of USB Overhead in bytes -- Command type byte + Command

6.141 Bluetooth State constants

Constants related to the bluetooth state.

Defines

- #define BT_ARM_OFF 0
- #define BT_ARM_CMD_MODE 1
- #define BT_ARM_DATA_MODE 2

6.141.1 Detailed Description

Constants related to the bluetooth state.

6.141.2 Define Documentation**6.141.2.1 #define BT_ARM_CMD_MODE 1**

BtState constant bluetooth command mode

6.141.2.2 #define BT_ARM_DATA_MODE 2

BtState constant bluetooth data mode

6.141.2.3 #define BT_ARM_OFF 0

BtState constant bluetooth off

6.142 Bluetooth state status constants

Constants related to the bluetooth state status.

Defines

- #define BT_BRICK_VISIBILITY 0x01
- #define BT_BRICK_PORT_OPEN 0x02

- #define BT_CONNECTION_0_ENABLE 0x10
- #define BT_CONNECTION_1_ENABLE 0x20
- #define BT_CONNECTION_2_ENABLE 0x40
- #define BT_CONNECTION_3_ENABLE 0x80

6.142.1 Detailed Description

Constants related to the bluetooth state status.

6.142.2 Define Documentation

6.142.2.1 #define BT_BRICK_PORT_OPEN 0x02

BtStateStatus port open bit

6.142.2.2 #define BT_BRICK_VISIBILITY 0x01

BtStateStatus brick visibility bit

6.142.2.3 #define BT_CONNECTION_0_ENABLE 0x10

BtStateStatus connection 0 enable/disable bit

6.142.2.4 #define BT_CONNECTION_1_ENABLE 0x20

BtStateStatus connection 1 enable/disable bit

6.142.2.5 #define BT_CONNECTION_2_ENABLE 0x40

BtStateStatus connection 2 enable/disable bit

6.142.2.6 #define BT_CONNECTION_3_ENABLE 0x80

BtStateStatus connection 3 enable/disable bit

6.143 Bluetooth hardware status constants

Constants related to the bluetooth hardware status.

Defines

- #define BT_ENABLE 0x00
- #define BT_DISABLE 0x01

6.143.1 Detailed Description

Constants related to the bluetooth hardware status.

6.143.2 Define Documentation**6.143.2.1 #define BT_DISABLE 0x01**

BtHwStatus bluetooth disable

6.143.2.2 #define BT_ENABLE 0x00

BtHwStatus bluetooth enable

6.144 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.

6.144.1 Detailed Description

Constants related to the hi-speed port.

6.145 Hi-speed port flags constants

Constants related to the hi-speed port flags.

Defines

- #define HS_UPDATE 1

6.145.1 Detailed Description

Constants related to the hi-speed port flags.

6.145.2 Define Documentation

6.145.2.1 #define HS_UPDATE 1

HsFlags high speed update required

6.146 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

6.146.1 Detailed Description

Constants related to the hi-speed port state.

6.146.2 Define Documentation

6.146.2.1 #define HS_DISABLE 4

HsState disable

6.146.2.2 #define HS_ENABLE 5

HsState enable

6.146.2.3 #define HS_INIT_RECEIVER 2

HsState initialize receiver

6.146.2.4 #define HS_INITIALISE 1

HsState initialize

6.146.2.5 #define HS_SEND_DATA 3

HsState send data

6.147 Hi-speed port SysCommHSControl constants

Constants for use with the SysCommHSControl API function.

Defines

- #define HS_CTRL_INIT 0
- #define HS_CTRL_UART 1
- #define HS_CTRL_EXIT 2

6.147.1 Detailed Description

Constants for use with the SysCommHSControl API function.

See also:

[SysCommHSControl\(\)](#)

6.147.2 Define Documentation

6.147.2.1 #define HS_CTRL_EXIT 2

Ddisable the high speed port

6.147.2.2 #define HS_CTRL_INIT 0

Enable the high speed port

Examples:

[ex_SysCommHSControl.nxc](#).

6.147.2.3 #define HS_CTRL_UART 1

Setup the high speed port UART configuration

Examples:

[ex_RS485Control.nxc](#).

6.148 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

6.148.1 Detailed Description

Constants for configuring the hi-speed port baud rate.

6.148.2 Define Documentation

6.148.2.1 #define HS_BAUD_115200 12

HsMode 115200 Baud

Examples:

[ex_RS485Uart.nxc](#).

6.148.2.2 #define HS_BAUD_1200 0

HsMode 1200 Baud

6.148.2.3 #define HS_BAUD_14400 6

HsMode 14400 Baud

6.148.2.4 #define HS_BAUD_19200 7

HsMode 19200 Baud

6.148.2.5 #define HS_BAUD_230400 13

HsMode 230400 Baud

6.148.2.6 #define HS_BAUD_2400 1

HsMode 2400 Baud

6.148.2.7 #define HS_BAUD_28800 8

HsMode 28800 Baud

6.148.2.8 #define HS_BAUD_3600 2

HsMode 3600 Baud

6.148.2.9 #define HS_BAUD_38400 9

HsMode 38400 Baud

6.148.2.10 #define HS_BAUD_460800 14

HsMode 460800 Baud

6.148.2.11 #define HS_BAUD_4800 3

HsMode 4800 Baud

6.148.2.12 #define HS_BAUD_57600 10

HsMode 57600 Baud

6.148.2.13 #define HS_BAUD_7200 4

HsMode 7200 Baud

6.148.2.14 #define HS_BAUD_76800 11

HsMode 76800 Baud

6.148.2.15 #define HS_BAUD_921600 15

HsMode 921600 Baud

6.148.2.16 #define HS_BAUD_9600 5

HsMode 9600 Baud

Examples:[ex_RS485Control.nxc.](#)**6.149 Hi-speed port data bits constants**

Constants referring to HsMode (number of data bits).

Defines

- #define HS_MODE_5_DATA 0x0000
- #define HS_MODE_6_DATA 0x0040
- #define HS_MODE_7_DATA 0x0080
- #define HS_MODE_8_DATA 0x00C0

6.149.1 Detailed Description

Constants referring to HsMode (number of data bits).

6.149.2 Define Documentation**6.149.2.1 #define HS_MODE_5_DATA 0x0000**

HsMode 5 data bits

6.149.2.2 #define HS_MODE_6_DATA 0x0040

HsMode 6 data bits

6.149.2.3 #define HS_MODE_7_DATA 0x0080

HsMode 7 data bits

6.149.2.4 #define HS_MODE_8_DATA 0x00C0

HsMode 8 data bits

6.150 Hi-speed port stop bits constants

Constants referring to HsMode (number of stop bits).

Defines

- #define HS_MODE_10_STOP 0x0000
- #define HS_MODE_15_STOP 0x1000
- #define HS_MODE_20_STOP 0x2000

6.150.1 Detailed Description

Constants referring to HsMode (number of stop bits).

6.150.2 Define Documentation

6.150.2.1 #define HS_MODE_10_STOP 0x0000

HsMode 1 stop bit

6.150.2.2 #define HS_MODE_15_STOP 0x1000

HsMode 1.5 stop bits

6.150.2.3 #define HS_MODE_20_STOP 0x2000

HsMode 2 stop bits

6.151 Hi-speed port parity constants

Constants referring to HsMode (parity).

Defines

- #define HS_MODE_E_PARITY 0x0000
- #define HS_MODE_O_PARITY 0x0200
- #define HS_MODE_S_PARITY 0x0400
- #define HS_MODE_M_PARITY 0x0600
- #define HS_MODE_N_PARITY 0x0800

6.151.1 Detailed Description

Constants referring to HsMode (parity).

6.151.2 Define Documentation

6.151.2.1 #define HS_MODE_E_PARITY 0x0000

HsMode Even parity

6.151.2.2 #define HS_MODE_M_PARITY 0x0600

HsMode Mark parity

6.151.2.3 #define HS_MODE_N_PARITY 0x0800

HsMode No parity

6.151.2.4 #define HS_MODE_O_PARITY 0x0200

HsMode Odd parity

6.151.2.5 #define HS_MODE_S_PARITY 0x0400

HsMode Space parity

6.152 Hi-speed port combined UART constants

Constants that combine data bits, parity, and stop bits into a single value.

Defines

- #define **HS_MODE_8N1** (HS_MODE_8_DATA|HS_MODE_N_PARITY|HS_MODE_10_STOP)
- #define **HS_MODE_7E1** (HS_MODE_7_DATA|HS_MODE_E_PARITY|HS_MODE_10_STOP)

6.152.1 Detailed Description

Constants that combine data bits, parity, and stop bits into a single value.

6.152.2 Define Documentation

**6.152.2.1 #define HS_MODE_7E1 (HS_MODE_7_DATA|HS_MODE_E_-
PARITY|HS_MODE_10_STOP)**

HsMode 7 data bits, even parity, 1 stop bit

**6.152.2.2 #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](#).

6.153 Device status constants

Constants referring to DeviceStatus within DeviceTable.

Defines

- #define **BT_DEVICE_EMPTY** 0x00
- #define **BT_DEVICE_UNKNOWN** 0x01
- #define **BT_DEVICE_KNOWN** 0x02
- #define **BT_DEVICE_NAME** 0x40
- #define **BT_DEVICE_AWAY** 0x80

6.153.1 Detailed Description

Constants referring to DeviceStatus within DeviceTable.

6.153.2 Define Documentation

6.153.2.1 #define BT_DEVICE_AWAY 0x80

Bluetooth device away

6.153.2.2 #define BT_DEVICE_EMPTY 0x00

Bluetooth device table empty

6.153.2.3 #define BT_DEVICE_KNOWN 0x02

Bluetooth device known

6.153.2.4 #define BT_DEVICE_NAME 0x40

Bluetooth device name

6.153.2.5 #define BT_DEVICE_UNKNOWN 0x01

Bluetooth device unknown

6.154 Comm module interface function constants

Constants for all the Comm module interface functions executable via SysCommExecuteFunction.

Defines

- #define [INTF_SENDFILE](#) 0
- #define [INTF_SEARCH](#) 1
- #define [INTF_STOPSEARCH](#) 2
- #define [INTF_CONNECT](#) 3
- #define [INTF_DISCONNECT](#) 4
- #define [INTF_DISCONNECTALL](#) 5
- #define [INTF_REMOVEDevice](#) 6
- #define [INTF_VISIBILITY](#) 7
- #define [INTF_SETCMDMODE](#) 8
- #define [INTF_OPENSTREAM](#) 9
- #define [INTF_SENDDATA](#) 10
- #define [INTF_FACTORYRESET](#) 11
- #define [INTF_BTON](#) 12
- #define [INTF_BTOFF](#) 13
- #define [INTF_SETBTNAME](#) 14
- #define [INTF_EXTREAD](#) 15
- #define [INTF_PINREQ](#) 16
- #define [INTF_CONNECTREQ](#) 17
- #define [INTF_CONNECTBYNAME](#) 18

6.154.1 Detailed Description

Constants for all the Comm module interface functions executable via SysCommExecuteFunction.

See also:

[SysCommExecuteFunction\(\)](#)

6.154.2 Define Documentation

6.154.2.1 #define INTF_BTOFF 13

Turn off the bluetooth radio

Examples:

[ex_syscommexecufunction.nxc](#).

6.154.2.2 #define INTF_BTTON 12

Turn on the bluetooth radio

6.154.2.3 #define INTF_CONNECT 3

Connect to one of the known devices

6.154.2.4 #define INTF_CONNECTBYNAME 18

Connect to a bluetooth device by name

6.154.2.5 #define INTF_CONNECTREQ 17

Connection request from another device

6.154.2.6 #define INTF_DISCONNECT 4

Disconnect from one of the connected devices

6.154.2.7 #define INTF_DISCONNECTALL 5

Disconnect all devices

6.154.2.8 #define INTF_EXTREAD 15

External read request

6.154.2.9 #define INTF_FACTORYRESET 11

Reset bluetooth settings to factory values

6.154.2.10 #define INTF_OPENSTREAM 9

Open a bluetooth stream

6.154.2.11 #define INTF_PINREQ 16

Bluetooth PIN request

6.154.2.12 #define INTF_REMOVEDevice 6

Remove a device from the known devices table

6.154.2.13 #define INTF_SEARCH 1

Search for bluetooth devices

6.154.2.14 #define INTF_SENDDATA 10

Send data over a bluetooth connection

6.154.2.15 #define INTF_SENDFILE 0

Send a file via bluetooth to another device

6.154.2.16 #define INTF_SETBTNAME 14

Set the bluetooth name

6.154.2.17 #define INTF_SETCMDMODE 8

Set bluetooth into command mode

6.154.2.18 #define INTF_STOPSEARCH 2

Stop searching for bluetooth devices

6.154.2.19 #define INTF_VISIBILITY 7

Set the bluetooth visibility on or off

6.155 Comm module status code constants

Constants for Comm module status codes.

Defines

- #define **LR_SUCCESS** 0x50
- #define **LR_COULD_NOT_SAVE** 0x51
- #define **LR_STORE_IS_FULL** 0x52
- #define **LR_ENTRY_REMOVED** 0x53
- #define **LR_UNKNOWN_ADDR** 0x54
- #define **USB_CMD_READY** 0x01
- #define **BT_CMD_READY** 0x02
- #define **HS_CMD_READY** 0x04

6.155.1 Detailed Description

Constants for Comm module status codes.

6.155.2 Define Documentation**6.155.2.1 #define BT_CMD_READY 0x02**

A constant representing bluetooth direct command

6.155.2.2 #define HS_CMD_READY 0x04

A constant representing high speed direct command

6.155.2.3 #define LR_COULD_NOT_SAVE 0x51

Bluetooth list result could not save

6.155.2.4 #define LR_ENTRY_REMOVED 0x53

Bluetooth list result entry removed

6.155.2.5 #define LR_STORE_IS_FULL 0x52

Bluetooth list result store is full

6.155.2.6 #define LR_SUCCESS 0x50

Bluetooth list result success

6.155.2.7 #define LR_UNKNOWN_ADDR 0x54

Bluetooth list result unknown address

6.155.2.8 #define USB_CMD_READY 0x01

A constant representing usb direct command

6.156 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

6.156.1 Detailed Description

Constant offsets into the Comm module IOMAP structure.

6.156.2 Define Documentation

6.156.2.1 #define CommOffsetBrickDataBdAddr 1144

Offset to Bluetooth address (7 bytes)

6.156.2.2 #define CommOffsetBrickDataBluecoreVersion 1142

Offset to Bluecore version (2 bytes)

6.156.2.3 #define CommOffsetBrickDataBtHwStatus 1152

Offset to BtHwStatus (1 byte)

6.156.2.4 #define CommOffsetBrickDataBtStateStatus 1151

Offset to BtStateStatus (1 byte)

6.156.2.5 #define CommOffsetBrickDataName 1126

Offset to brick name (16 bytes)

6.156.2.6 #define CommOffsetBrickDataTimeOutValue 1153

Offset to data timeout value (1 byte)

6.156.2.7 #define CommOffsetBtConnectTableBdAddr(p) (((p)*47)+974)

Offset to Bluetooth connect table address (7 bytes)

6.156.2.8 #define CommOffsetBtConnectTableClassOfDevice(p) (((p)*47)+954)

Offset to Bluetooth connect table device class (4 bytes)

6.156.2.9 #define CommOffsetBtConnectTableHandleNr(p) (((p)*47)+981)

Offset to Bluetooth connect table handle (1 byte)

6.156.2.10 #define CommOffsetBtConnectTableLinkQuality(p) (((p)*47)+983)

Offset to Bluetooth connect table link quality (1 byte)

6.156.2.11 #define CommOffsetBtConnectTableName(p) (((p)*47)+938)

Offset to Bluetooth connect table name (16 bytes)

6.156.2.12 #define CommOffsetBtConnectTablePinCode(p) (((p)*47)+958)

Offset to Bluetooth connect table pin code (16 bytes)

6.156.2.13 #define CommOffsetBtConnectTableStreamStatus(p) (((p)*47)+982)

Offset to Bluetooth connect table stream status (1 byte)

6.156.2.14 #define CommOffsetBtDeviceCnt 1889

Offset to Bluetooth device count (1 byte)

6.156.2.15 #define CommOffsetBtDeviceNameCnt 1890

Offset to Bluetooth device name count (1 byte)

6.156.2.16 #define CommOffsetBtDeviceTableBdAddr(p) (((p)*31)+28)

Offset to Bluetooth device table address (7 bytes)

6.156.2.17 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)*31)+24)

Offset to Bluetooth device table device class (4 bytes)

6.156.2.18 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)*31)+35)

Offset to Bluetooth device table status (1 byte)

6.156.2.19 #define CommOffsetBtDeviceTableName(p) (((p)*31)+8)

Offset to BT device table name (16 bytes)

6.156.2.20 #define CommOffsetBtInBufBuf 1157

Offset to Bluetooth input buffer data (128 bytes)

6.156.2.21 #define CommOffsetBtInBufInPtr 1285

Offset to Bluetooth input buffer front pointer (1 byte)

6.156.2.22 #define CommOffsetBtInBufOutPtr 1286

Offset to Bluetooth output buffer back pointer (1 byte)

6.156.2.23 #define CommOffsetBtOutBufBuf 1289

Offset to Bluetooth output buffer offset data (128 bytes)

6.156.2.24 #define CommOffsetBtOutBufInPtr 1417

Offset to Bluetooth output buffer front pointer (1 byte)

6.156.2.25 #define CommOffsetBtOutBufOutPtr 1418

Offset to Bluetooth output buffer back pointer (1 byte)

6.156.2.26 #define CommOffsetHsFlags 1891

Offset to High Speed flags (1 byte)

6.156.2.27 #define CommOffsetHsInBufBuf 1421

Offset to High Speed input buffer data (128 bytes)

6.156.2.28 #define CommOffsetHsInBufInPtr 1549

Offset to High Speed input buffer front pointer (1 byte)

6.156.2.29 #define CommOffsetHsInBufOutPtr 1550

Offset to High Speed input buffer back pointer (1 byte)

6.156.2.30 #define CommOffsetHsMode 1895

Offset to High Speed mode (2 bytes)

6.156.2.31 #define CommOffsetHsOutBufBuf 1553

Offset to High Speed output buffer data (128 bytes)

6.156.2.32 #define CommOffsetHsOutBufInPtr 1681

Offset to High Speed output buffer front pointer (1 byte)

6.156.2.33 #define CommOffsetHsOutBufOutPtr 1682

Offset to High Speed output buffer back pointer (1 byte)

6.156.2.34 #define CommOffsetHsSpeed 1892

Offset to High Speed speed (1 byte)

6.156.2.35 #define CommOffsetHsState 1893

Offset to High Speed state (1 byte)

6.156.2.36 #define CommOffsetPFunc 0

Offset to the Comm module first function pointer (4 bytes)

6.156.2.37 #define CommOffsetPFuncTwo 4

Offset to the Comm module second function pointer (4 bytes)

6.156.2.38 #define CommOffsetUsbInBufBuf 1685

Offset to Usb input buffer data (64 bytes)

6.156.2.39 #define CommOffsetUsbInBufInPtr 1749

Offset to Usb input buffer front pointer (1 byte)

6.156.2.40 #define CommOffsetUsbInBufOutPtr 1750

Offset to Usb input buffer back pointer (1 byte)

6.156.2.41 #define CommOffsetUsbOutBufBuf 1753

Offset to Usb output buffer data (64 bytes)

6.156.2.42 #define CommOffsetUsbOutBufInPtr 1817

Offset to Usb output buffer front pointer (1 byte)

6.156.2.43 #define CommOffsetUsbOutBufOutPtr 1818

Offset to Usb output buffer back pointer (1 byte)

6.156.2.44 #define CommOffsetUsbPollBufBuf 1821

Offset to Usb Poll buffer data (64 bytes)

6.156.2.45 #define CommOffsetUsbPollBufInPtr 1885

Offset to Usb Poll buffer front pointer (1 byte)

6.156.2.46 #define CommOffsetUsbPollBufOutPtr 1886

Offset to Usb Poll buffer back pointer (1 byte)

6.156.2.47 #define CommOffsetUsbState 1894

Offset to Usb State (1 byte)

6.157 RCX constants

Constants that are for use with devices that communicate with the RCX or Scout programmable bricks via IR such as the HiTechnic IRLink or the MindSensors nRLink.

Modules

- [RCX output constants](#)

Constants for use when choosing RCX outputs.

- [RCX output mode constants](#)

Constants for use when configuring RCX output mode.

- [RCX output direction constants](#)

Constants for use when configuring RCX output direction.

- [RCX output power constants](#)

Constants for use when configuring RCX output power.

- [RCX IR remote constants](#)

Constants for use when simulating RCX IR remote messages.

- [RCX and Scout sound constants](#)

Constants for use when playing standard RCX and Scout sounds.

- [Scout constants](#)

Constants for use when controlling the Scout brick.

- [RCX and Scout source constants](#)

Constants for use when specifying RCX and Scout sources.

- [RCX and Scout opcode constants](#)

Constants for use when specifying RCX and Scout opcodes.

6.157.1 Detailed Description

Constants that are for use with devices that communicate with the RCX or Scout programmable bricks via IR such as the HiTechnic IRLink or the MindSensors nRLink.

6.158 RCX output constants

Constants for use when choosing RCX outputs.

Defines

- #define [RCX_OUT_A](#) 0x01
- #define [RCX_OUT_B](#) 0x02
- #define [RCX_OUT_C](#) 0x04
- #define [RCX_OUT_AB](#) 0x03
- #define [RCX_OUT_AC](#) 0x05
- #define [RCX_OUT_BC](#) 0x06
- #define [RCX_OUT_ABC](#) 0x07

6.158.1 Detailed Description

Constants for use when choosing RCX outputs.

6.158.2 Define Documentation**6.158.2.1 #define RCX_OUT_A 0x01**

RCX Output A

Examples:

[ex_HTRCXDisableOutput.nxc](#), [ex_HTRCXEnableOutput.nxc](#), [ex_HTRCXFloat.nxc](#), [ex_HTRCxFwd.nxc](#), [ex_HTRCXInvertOutput.nxc](#),
[ex_HTRCXObvertOutput.nxc](#), [ex_HTRCXOff.nxc](#), [ex_HTRCXOn.nxc](#), [ex_HTRCXOnFor.nxc](#), [ex_HTRCXOnFwd.nxc](#), [ex_HTRCXOnRev.nxc](#), [ex_HTRCXRev.nxc](#), [ex_HTRCXSetDirection.nxc](#),
[ex_HTRCXSetGlobalDirection.nxc](#), [ex_HTRCXSetGlobalOutput.nxc](#), [ex_HTRCXSetMaxPower.nxc](#), [ex_HTRCXSetOutput.nxc](#), [ex_HTRCXSetPower.nxc](#), [ex_HTRCXToggle.nxc](#), [ex_MSRCXDisableOutput.nxc](#),
[ex_MSRCXEnableOutput.nxc](#), [ex_MSRCXFloat.nxc](#), [ex_MSRCXFwd.nxc](#), [ex_MSRCXInvertOutput.nxc](#), [ex_MSRCXObvertOutput.nxc](#), [ex_MSRCXOff.nxc](#),
[ex_MSRCXOn.nxc](#), [ex_MSRCXOnFor.nxc](#), [ex_MSRCXOnFwd.nxc](#), [ex_MSRCXOnRev.nxc](#), [ex_MSRCXRev.nxc](#), [ex_MSRCXSetDirection.nxc](#),
[ex_MSRCXSetGlobalDirection.nxc](#), [ex_MSRCXSetGlobalOutput.nxc](#), [ex_MSRCXSetMaxPower.nxc](#), [ex_MSRCXSetOutput.nxc](#), [ex_MSRCXSetPower.nxc](#), and [ex_MSRCXToggle.nxc](#).

6.158.2.2 #define RCX_OUT_AB 0x03

RCX Outputs A and B

6.158.2.3 #define RCX_OUT_ABC 0x07

RCX Outputs A, B, and C

6.158.2.4 #define RCX_OUT_AC 0x05

RCX Outputs A and C

6.158.2.5 #define RCX_OUT_B 0x02

RCX Output B

6.158.2.6 #define RCX_OUT_BC 0x06

RCX Outputs B and C

6.158.2.7 #define RCX_OUT_C 0x04

RCX Output C

6.159 RCX output mode constants

Constants for use when configuring RCX output mode.

Defines

- #define **RCX_OUT_FLOAT** 0
- #define **RCX_OUT_OFF** 0x40
- #define **RCX_OUT_ON** 0x80

6.159.1 Detailed Description

Constants for use when configuring RCX output mode.

6.159.2 Define Documentation**6.159.2.1 #define RCX_OUT_FLOAT 0**

Set RCX output to float

6.159.2.2 #define RCX_OUT_OFF 0x40

Set RCX output to off

6.159.2.3 #define RCX_OUT_ON 0x80

Set RCX output to on

Examples:

[ex_HTRCXSetGlobalOutput.nxc](#), [ex_HTRCXSetOutput.nxc](#), [ex_MSRCXSetGlobalOutput.nxc](#), and [ex_MSRCXSetOutput.nxc](#).

6.160 RCX output direction constants

Constants for use when configuring RCX output direction.

Defines

- #define **RCX_OUT_REV** 0
- #define **RCX_OUT_TOGGLE** 0x40
- #define **RCX_OUT_FWD** 0x80

6.160.1 Detailed Description

Constants for use when configuring RCX output direction.

6.160.2 Define Documentation**6.160.2.1 #define RCX_OUT_FWD 0x80**

Set RCX output direction to forward

Examples:

[ex_HTRCXSetDirection.nxc](#), [ex_HTRCXSetGlobalDirection.nxc](#), [ex_MSRCXSetDirection.nxc](#), and [ex_MSRCXSetGlobalDirection.nxc](#).

6.160.2.2 #define RCX_OUT_REV 0

Set RCX output direction to reverse

6.160.2.3 #define RCX_OUT_TOGGLE 0x40

Set RCX output direction to toggle

6.161 RCX output power constants

Constants for use when configuring RCX output power.

Defines

- #define **RCX_OUT_LOW** 0
- #define **RCX_OUT_HALF** 3
- #define **RCX_OUT_FULL** 7

6.161.1 Detailed Description

Constants for use when configuring RCX output power.

6.161.2 Define Documentation

6.161.2.1 #define RCX_OUT_FULL 7

Set RCX output power level to full

Examples:

[ex_HTRCXSetPower.nxc](#), and [ex_MSRCXSetPower.nxc](#).

6.161.2.2 #define RCX_OUT_HALF 3

Set RCX output power level to half

6.161.2.3 #define RCX_OUT_LOW 0

Set RCX output power level to low

6.162 RCX IR remote constants

Constants for use when simulating RCX IR remote messages.

Defines

- #define [RCX_RemoteKeysReleased](#) 0x0000
- #define [RCX_RemotePBMessag1](#) 0x0100
- #define [RCX_RemotePBMessag2](#) 0x0200
- #define [RCX_RemotePBMessag3](#) 0x0400
- #define [RCX_RemoteOutAForward](#) 0x0800
- #define [RCX_RemoteOutBForward](#) 0x1000
- #define [RCX_RemoteOutCForward](#) 0x2000
- #define [RCX_RemoteOutABackward](#) 0x4000
- #define [RCX_RemoteOutBBackward](#) 0x8000
- #define [RCX_RemoteOutCBackward](#) 0x0001
- #define [RCX_RemoteSelProgram1](#) 0x0002
- #define [RCX_RemoteSelProgram2](#) 0x0004
- #define [RCX_RemoteSelProgram3](#) 0x0008

- #define **RCX_RemoteSelProgram4** 0x0010
- #define **RCX_RemoteSelProgram5** 0x0020
- #define **RCX_RemoteStopOutOff** 0x0040
- #define **RCX_RemotePlayASound** 0x0080

6.162.1 Detailed Description

Constants for use when simulating RCX IR remote messages.

6.162.2 Define Documentation

6.162.2.1 #define **RCX_RemoteKeysReleased** 0x0000

All remote keys have been released

6.162.2.2 #define **RCX_RemoteOutABackward** 0x4000

Set output A backward

6.162.2.3 #define **RCX_RemoteOutAForward** 0x0800

Set output A forward

6.162.2.4 #define **RCX_RemoteOutBBackward** 0x8000

Set output B backward

6.162.2.5 #define **RCX_RemoteOutBForward** 0x1000

Set output B forward

6.162.2.6 #define **RCX_RemoteOutCBackward** 0x0001

Set output C backward

6.162.2.7 #define **RCX_RemoteOutCForward** 0x2000

Set output C forward

6.162.2.8 #define RCX_RemotePBMessage1 0x0100

Send PB message 1

6.162.2.9 #define RCX_RemotePBMessage2 0x0200

Send PB message 2

6.162.2.10 #define RCX_RemotePBMessage3 0x0400

Send PB message 3

6.162.2.11 #define RCX_RemotePlayASound 0x0080

Play a sound

Examples:

[ex_HTRCXRemote.nxc](#), and [ex_MSRCXRemote.nxc](#).

6.162.2.12 #define RCX_RemoteSelProgram1 0x0002

Select program 1

6.162.2.13 #define RCX_RemoteSelProgram2 0x0004

Select program 2

6.162.2.14 #define RCX_RemoteSelProgram3 0x0008

Select program 3

6.162.2.15 #define RCX_RemoteSelProgram4 0x0010

Select program 4

6.162.2.16 #define RCX_RemoteSelProgram5 0x0020

Select program 5

6.162.2.17 #define RCX_RemoteStopOutOff 0x0040

Stop and turn off outputs

6.163 RCX and Scout sound constants

Constants for use when playing standard RCX and Scout sounds.

Defines

- #define SOUND_CLICK 0
- #define SOUND_DOUBLE_BEEP 1
- #define SOUND_DOWN 2
- #define SOUND_UP 3
- #define SOUND_LOW_BEEP 4
- #define SOUND_FAST_UP 5

6.163.1 Detailed Description

Constants for use when playing standard RCX and Scout sounds.

6.163.2 Define Documentation

6.163.2.1 #define SOUND_CLICK 0

Play the standard key click sound

6.163.2.2 #define SOUND_DOUBLE_BEEP 1

Play the standard double beep sound

6.163.2.3 #define SOUND_DOWN 2

Play the standard sweep down sound

Examples:

`ex_playsound.nxc.`

6.163.2.4 #define SOUND_FAST_UP 5

Play the standard fast up sound

Examples:

[ex_playsound.nxc](#).

6.163.2.5 #define SOUND_LOW_BEEP 4

Play the standard low beep sound

Examples:

[ex_playsound.nxc](#).

6.163.2.6 #define SOUND_UP 3

Play the standard sweep up sound

Examples:

[ex_playsound.nxc](#).

6.164 Scout constants

Constants for use when controlling the Scout brick.

Modules

- [Scout light constants](#)

Constants for use when controlling the Scout light settings.

- [Scout sound constants](#)

Constants for use when playing standard Scout sounds.

- [Scout sound set constants](#)

Constants for use when choosing standard Scout sound sets.

- [Scout mode constants](#)

Constants for use when setting the scout mode.

- [Scout motion rule constants](#)

Constants for use when setting the scout motion rule.

- [Scout touch rule constants](#)

Constants for use when setting the scout touch rule.

- [Scout light rule constants](#)

Constants for use when setting the scout light rule.

- [Scout transmit rule constants](#)

Constants for use when setting the scout transmit rule.

- [Scout special effect constants](#)

Constants for use when setting the scout special effect.

6.164.1 Detailed Description

Constants for use when controlling the Scout brick.

6.165 Scout light constants

Constants for use when controlling the Scout light settings.

Defines

- #define [SCOUT_LIGHT_ON](#) 0x80
- #define [SCOUT_LIGHT_OFF](#) 0

6.165.1 Detailed Description

Constants for use when controlling the Scout light settings.

6.165.2 Define Documentation

6.165.2.1 #define SCOUT_LIGHT_OFF 0

Turn off the scout light

6.165.2.2 #define SCOUT_LIGHT_ON 0x80

Turn on the scout light

Examples:

[ex_HTScoutSetLight.nxc](#).

6.166 Scout sound constants

Constants for use when playing standard Scout sounds.

Defines

- #define SCOUT_SOUND_REMOTE 6
- #define SCOUT_SOUND_ENTERSA 7
- #define SCOUT_SOUND_KEYERROR 8
- #define SCOUT_SOUND_NONE 9
- #define SCOUT_SOUND_TOUCH1_PRES 10
- #define SCOUT_SOUND_TOUCH1_REL 11
- #define SCOUT_SOUND_TOUCH2_PRES 12
- #define SCOUT_SOUND_TOUCH2_REL 13
- #define SCOUT_SOUND_ENTER_BRIGHT 14
- #define SCOUT_SOUND_ENTER_NORMAL 15
- #define SCOUT_SOUND_ENTER_DARK 16
- #define SCOUT_SOUND_1_BLINK 17
- #define SCOUT_SOUND_2_BLINK 18
- #define SCOUT_SOUND_COUNTER1 19
- #define SCOUT_SOUND_COUNTER2 20
- #define SCOUT_SOUND_TIMER1 21
- #define SCOUT_SOUND_TIMER2 22
- #define SCOUT_SOUND_TIMER3 23
- #define SCOUT_SOUND_MAIL_RECEIVED 24
- #define SCOUT_SOUND_SPECIAL1 25
- #define SCOUT_SOUND_SPECIAL2 26
- #define SCOUT_SOUND_SPECIAL3 27

6.166.1 Detailed Description

Constants for use when playing standard Scout sounds.

6.166.2 Define Documentation**6.166.2.1 #define SCOUT_SOUND_1_BLINK 17**

Play the Scout 1 blink sound

6.166.2.2 #define SCOUT_SOUND_2_BLINK 18

Play the Scout 2 blink sound

6.166.2.3 #define SCOUT_SOUND_COUNTER1 19

Play the Scout counter 1 sound

6.166.2.4 #define SCOUT_SOUND_COUNTER2 20

Play the Scout counter 2 sound

6.166.2.5 #define SCOUT_SOUND_ENTER_BRIGHT 14

Play the Scout enter bright sound

6.166.2.6 #define SCOUT_SOUND_ENTER_DARK 16

Play the Scout enter dark sound

6.166.2.7 #define SCOUT_SOUND_ENTER_NORMAL 15

Play the Scout enter normal sound

6.166.2.8 #define SCOUT_SOUND_ENTERSA 7

Play the Scout enter standalone sound

6.166.2.9 #define SCOUT_SOUND_KEYERROR 8

Play the Scout key error sound

6.166.2.10 #define SCOUT_SOUND_MAIL_RECEIVED 24

Play the Scout mail received sound

6.166.2.11 #define SCOUT_SOUND_NONE 9

Play the Scout none sound

6.166.2.12 #define SCOUT_SOUND_REMOTE 6

Play the Scout remote sound

6.166.2.13 #define SCOUT_SOUND_SPECIAL1 25

Play the Scout special 1 sound

6.166.2.14 #define SCOUT_SOUND_SPECIAL2 26

Play the Scout special 2 sound

6.166.2.15 #define SCOUT_SOUND_SPECIAL3 27

Play the Scout special 3 sound

6.166.2.16 #define SCOUT_SOUND_TIMER1 21

Play the Scout timer 1 sound

6.166.2.17 #define SCOUT_SOUND_TIMER2 22

Play the Scout timer 2 sound

6.166.2.18 #define SCOUT_SOUND_TIMER3 23

Play the Scout timer 3 sound

6.166.2.19 #define SCOUT_SOUND_TOUCH1_PRES 10

Play the Scout touch 1 pressed sound

6.166.2.20 #define SCOUT_SOUND_TOUCH1_REL 11

Play the Scout touch 1 released sound

6.166.2.21 #define SCOUT_SOUND_TOUCH2_PRES 12

Play the Scout touch 2 pressed sound

6.166.2.22 #define SCOUT_SOUND_TOUCH2_REL 13

Play the Scout touch 2 released sound

6.167 Scout sound set constants

Constants for use when choosing standard Scout sound sets.

Defines

- #define SCOUT_SNDSET_NONE 0
- #define SCOUT_SNDSET_BASIC 1
- #define SCOUT_SNDSET_BUG 2
- #define SCOUT_SNDSET_ALARM 3
- #define SCOUT_SNDSET_RANDOM 4
- #define SCOUT_SNDSET_SCIENCE 5

6.167.1 Detailed Description

Constants for use when choosing standard Scout sound sets.

6.167.2 Define Documentation**6.167.2.1 #define SCOUT_SNDSET_ALARM 3**

Set sound set to alarm

6.167.2.2 #define SCOUT_SNDSET_BASIC 1

Set sound set to basic

6.167.2.3 #define SCOUT_SNDSET_BUG 2

Set sound set to bug

6.167.2.4 #define SCOUT_SNDSET_NONE 0

Set sound set to none

6.167.2.5 #define SCOUT_SNDSET_RANDOM 4

Set sound set to random

6.167.2.6 #define SCOUT_SNDSET_SCIENCE 5

Set sound set to science

6.168 Scout mode constants

Constants for use when setting the scout mode.

Defines

- #define SCOUT_MODE_STANDALONE 0
- #define SCOUT_MODE_POWER 1

6.168.1 Detailed Description

Constants for use when setting the scout mode.

6.168.2 Define Documentation**6.168.2.1 #define SCOUT_MODE_POWER 1**

Enter power mode

Examples:

[ex_HTScoutSetScoutMode.nxc](#), and [ex_MSScoutSetScoutMode.nxc](#).

6.168.2.2 #define SCOUT_MODE_STANDALONE 0

Enter stand alone mode

6.169 Scout motion rule constants

Constants for use when setting the scout motion rule.

Defines

- #define SCOUT_MR_NO_MOTION 0
- #define SCOUT_MR_FORWARD 1
- #define SCOUT_MR_ZIGZAG 2
- #define SCOUT_MR_CIRCLE_RIGHT 3
- #define SCOUT_MR_CIRCLE_LEFT 4
- #define SCOUT_MR_LOOP_A 5
- #define SCOUT_MR_LOOP_B 6
- #define SCOUT_MR_LOOP_AB 7

6.169.1 Detailed Description

Constants for use when setting the scout motion rule.

6.169.2 Define Documentation**6.169.2.1 #define SCOUT_MR_CIRCLE_LEFT 4**

Motion rule circle left

6.169.2.2 #define SCOUT_MR_CIRCLE_RIGHT 3

Motion rule circle right

6.169.2.3 #define SCOUT_MR_FORWARD 1

Motion rule forward

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

6.169.2.4 #define SCOUT_MR_LOOP_A 5

Motion rule loop A

6.169.2.5 #define SCOUT_MR_LOOP_AB 7

Motion rule loop A then B

6.169.2.6 #define SCOUT_MR_LOOP_B 6

Motion rule loop B

6.169.2.7 #define SCOUT_MR_NO_MOTION 0

Motion rule none

6.169.2.8 #define SCOUT_MR_ZIGZAG 2

Motion rule zigzag

6.170 Scout touch rule constants

Constants for use when setting the scout touch rule.

Defines

- #define SCOUT_TR_IGNORE 0
- #define SCOUT_TR_REVERSE 1
- #define SCOUT_TR_AVOID 2
- #define SCOUT_TR_WAIT_FOR 3
- #define SCOUT_TR_OFF_WHEN 4

6.170.1 Detailed Description

Constants for use when setting the scout touch rule.

6.170.2 Define Documentation**6.170.2.1 #define SCOUT_TR_AVOID 2**

Touch rule avoid

6.170.2.2 #define SCOUT_TR_IGNORE 0

Touch rule ignore

6.170.2.3 #define SCOUT_TR_OFF_WHEN 4

Touch rule off when

6.170.2.4 #define SCOUT_TR_REVERSE 1

Touch rule reverse

Examples:[ex_MSScoutSetScoutRules.nxc.](#)**6.170.2.5 #define SCOUT_TR_WAIT_FOR 3**

Touch rule wait for

6.171 Scout light rule constants

Constants for use when setting the scout light rule.

Defines

- #define [SCOUT_LR_IGNORE](#) 0
- #define [SCOUT_LR_SEEK_LIGHT](#) 1
- #define [SCOUT_LR_SEEK_DARK](#) 2
- #define [SCOUT_LR_AVOID](#) 3
- #define [SCOUT_LR_WAIT_FOR](#) 4
- #define [SCOUT_LR_OFF_WHEN](#) 5

6.171.1 Detailed Description

Constants for use when setting the scout light rule.

6.171.2 Define Documentation

6.171.2.1 #define SCOUT_LR_AVOID 3

Light rule avoid

6.171.2.2 #define SCOUT_LR_IGNORE 0

Light rule ignore

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

6.171.2.3 #define SCOUT_LR_OFF_WHEN 5

Light rule off when

6.171.2.4 #define SCOUT_LR_SEEK_DARK 2

Light rule seek dark

6.171.2.5 #define SCOUT_LR_SEEK_LIGHT 1

Light rule seek light

6.171.2.6 #define SCOUT_LR_WAIT_FOR 4

Light rule wait for

6.172 Scout transmit rule constants

Constants for use when setting the scout transmit rule.

Defines

- #define SCOUT_TGS_SHORT 0
- #define SCOUT_TGS_MEDIUM 1
- #define SCOUT_TGS_LONG 2

6.172.1 Detailed Description

Constants for use when setting the scout transmit rule.

6.172.2 Define Documentation**6.172.2.1 #define SCOUT_TGS_LONG 2**

Transmit level long

6.172.2.2 #define SCOUT_TGS_MEDIUM 1

Transmit level medium

6.172.2.3 #define SCOUT_TGS_SHORT 0

Transmit level short

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

6.173 Scout special effect constants

Constants for use when setting the scout special effect.

Defines

- #define SCOUT_FXR_NONE 0
- #define SCOUT_FXR_BUG 1
- #define SCOUT_FXR_ALARM 2
- #define SCOUT_FXR_RANDOM 3
- #define SCOUT_FXR_SCIENCE 4

6.173.1 Detailed Description

Constants for use when setting the scout special effect.

6.173.2 Define Documentation

6.173.2.1 #define SCOUT_FXR_ALARM 2

Alarm special effects

6.173.2.2 #define SCOUT_FXR_BUG 1

Bug special effects

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

6.173.2.3 #define SCOUT_FXR_NONE 0

No special effects

6.173.2.4 #define SCOUT_FXR_RANDOM 3

Random special effects

6.173.2.5 #define SCOUT_FXR_SCIENCE 4

Science special effects

6.174 RCX and Scout source constants

Constants for use when specifying RCX and Scout sources.

Defines

- #define [RCX_VariableSrc](#) 0
- #define [RCX_TimerSrc](#) 1
- #define [RCX_ConstantSrc](#) 2
- #define [RCX_OutputStatusSrc](#) 3

- #define `RCX_RandomSrc` 4
- #define `RCX_ProgramSlotSrc` 8
- #define `RCX_InputValueSrc` 9
- #define `RCX_InputTypeSrc` 10
- #define `RCX_InputModeSrc` 11
- #define `RCX_InputRawSrc` 12
- #define `RCX_InputBooleanSrc` 13
- #define `RCX_WatchSrc` 14
- #define `RCX_MessageSrc` 15
- #define `RCX_GlobalMotorStatusSrc` 17
- #define `RCX_ScoutRulesSrc` 18
- #define `RCX_ScoutLightParamsSrc` 19
- #define `RCX_ScoutTimerLimitSrc` 20
- #define `RCX_CounterSrc` 21
- #define `RCX_ScoutCounterLimitSrc` 22
- #define `RCX_TaskEventsSrc` 23
- #define `RCX_ScoutEventFBSrc` 24
- #define `RCX_EventStateSrc` 25
- #define `RCX_TenMSTimerSrc` 26
- #define `RCX_ClickCounterSrc` 27
- #define `RCX_UpperThresholdSrc` 28
- #define `RCX_LowerThresholdSrc` 29
- #define `RCX_HysteresisSrc` 30
- #define `RCX_DurationSrc` 31
- #define `RCX_UARTSetupSrc` 33
- #define `RCX_BatteryLevelSrc` 34
- #define `RCX_FirmwareVersionSrc` 35
- #define `RCX_IndirectVarSrc` 36
- #define `RCX_DatalogSrcIndirectSrc` 37
- #define `RCX_DatalogSrcDirectSrc` 38
- #define `RCX_DatalogValueIndirectSrc` 39
- #define `RCX_DatalogValueDirectSrc` 40
- #define `RCX_DatalogRawIndirectSrc` 41
- #define `RCX_DatalogRawDirectSrc` 42

6.174.1 Detailed Description

Constants for use when specifying RCX and Scout sources.

6.174.2 Define Documentation**6.174.2.1 #define RCX_BatteryLevelSrc 34**

The RCX battery level source

6.174.2.2 #define RCX_ClickCounterSrc 27

The RCX event click counter source

6.174.2.3 #define RCX_ConstantSrc 2

The RCX constant value source

Examples:

[ex_HTRCXEvent.nxc](#), [ex_HTRCXSetEvent.nxc](#), [ex_HTRCXSetMaxPower.nxc](#),
[ex_HTRCXSetPower.nxc](#), [ex_HTSentVLL.nxc](#), [ex_HTSentEventFeedback.nxc](#), [ex_HTSentSensorClickTime.nxc](#),
[ex_HTSentSensorHysteresis.nxc](#), [ex_MSRCXAndVar.nxc](#), [ex_MSRCXDivVar.nxc](#), [ex_MSRCXEvent.nxc](#), [ex_MSRCXOrVar.nxc](#), [ex_MSRCXSetEvent.nxc](#), [ex_MSRCXSetMaxPower.nxc](#), [ex_MSRCXSetPower.nxc](#),
[ex_MSSentVLL.nxc](#), [ex_MSSentCounterLimit.nxc](#), [ex_MSSentEventFeedback.nxc](#), [ex_MSSentSensorClickTime.nxc](#), [ex_MSSentSensorHysteresis.nxc](#), and [ex_MSSentTimerLimit.nxc](#).

6.174.2.4 #define RCX_CounterSrc 21

The RCX counter source

6.174.2.5 #define RCX_DatalogRawDirectSrc 42

The RCX direct datalog raw source

6.174.2.6 #define RCX_DatalogRawIndirectSrc 41

The RCX indirect datalog raw source

6.174.2.7 #define RCX_DatalogSrcDirectSrc 38

The RCX direct datalog source source

6.174.2.8 #define RCX_DatalogSrcIndirectSrc 37

The RCX indirect datalog source source

6.174.2.9 #define RCX_DatalogValueDirectSrc 40

The RCX direct datalog value source

6.174.2.10 #define RCX_DatalogValueIndirectSrc 39

The RCX indirect datalog value source

6.174.2.11 #define RCX_DurationSrc 31

The RCX event duration source

6.174.2.12 #define RCX_EventStateSrc 25

The RCX event static source

6.174.2.13 #define RCX_FirmwareVersionSrc 35

The RCX firmware version source

6.174.2.14 #define RCX_GlobalMotorStatusSrc 17

The RCX global motor status source

6.174.2.15 #define RCX_HysteresisSrc 30

The RCX event hysteresis source

6.174.2.16 #define RCX_IndirectVarSrc 36

The RCX indirect variable source

6.174.2.17 #define RCX_InputBooleanSrc 13

The RCX input boolean source

6.174.2.18 #define RCX_InputModeSrc 11

The RCX input mode source

6.174.2.19 #define RCX_InputRawSrc 12

The RCX input raw source

6.174.2.20 #define RCX_InputTypeSrc 10

The RCX input type source

6.174.2.21 #define RCX_InputValueSrc 9

The RCX input value source

Examples:

[ex_HTRCXAddToDatalog.nxc](#), [ex_MSRCXAddToDatalog.nxc](#), and [ex-MSRCXSumVar.nxc](#).

6.174.2.22 #define RCX_LowerThresholdSrc 29

The RCX event lower threshold source

6.174.2.23 #define RCX_MessageSrc 15

The RCX message source

6.174.2.24 #define RCX_OutputStatusSrc 3

The RCX output status source

6.174.2.25 #define RCX_ProgramSlotSrc 8

The RCX program slot source

6.174.2.26 #define RCX_RandomSrc 4

The RCX random number source

Examples:

[ex_MSRCXSet.nxc](#), and [ex_MSRCXSubVar.nxc](#).

6.174.2.27 #define RCX_ScoutCounterLimitSrc 22

The Scout counter limit source

6.174.2.28 #define RCX_ScoutEventFBSrc 24

The Scout event feedback source

6.174.2.29 #define RCX_ScoutLightParamsSrc 19

The Scout light parameters source

6.174.2.30 #define RCX_ScoutRulesSrc 18

The Scout rules source

6.174.2.31 #define RCX_ScoutTimerLimitSrc 20

The Scout timer limit source

6.174.2.32 #define RCX_TaskEventsSrc 23

The RCX task events source

6.174.2.33 #define RCX_TenMSTimerSrc 26

The RCX 10ms timer source

6.174.2.34 #define RCX_TimerSrc 1

The RCX timer source

6.174.2.35 #define RCX_UARTSetupSrc 33

The RCX UART setup source

6.174.2.36 #define RCX_UpperThresholdSrc 28

The RCX event upper threshold source

6.174.2.37 #define RCX_VariableSrc 0

The RCX variable source

Examples:

`ex_HTRCXPoll.nxc, ex_HTRCXSelectDisplay.nxc, ex_-
HTScoutSetSensorLowerLimit.nxc, ex_HTS ScoutSetSensorUpperLimit.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.`

6.174.2.38 #define RCX_WatchSrc 14

The RCX watch source

6.175 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_LSSupperThreshOp` 0xb3
- #define `RCX_LSLowerThreshOp` 0xc3
- #define `RCX_LSHysteresisOp` 0xd3
- #define `RCX_LSBlinkTimeOp` 0xe3
- #define `RCX_CalibrateEventOp` 0x04
- #define `RCX_SetVarOp` 0x14
- #define `RCX_SumVarOp` 0x24
- #define `RCX_SubVarOp` 0x34
- #define `RCX_DivVarOp` 0x44
- #define `RCX_MulVarOp` 0x54

- #define **RCX_SgnVarOp** 0x64
- #define **RCX_AbsVarOp** 0x74
- #define **RCX_AndVarOp** 0x84
- #define **RCX_OrVarOp** 0x94
- #define **RCX_UploadDatalogOp** 0xa4
- #define **RCX_SetTimerLimitOp** 0xc4
- #define **RCX_SetCounterOp** 0xd4
- #define **RCX_SetSourceValueOp** 0x05
- #define **RCX_UnlockOp** 0x15
- #define **RCX_BootModeOp** 0x65
- #define **RCX_UnlockFirmOp** 0xa5
- #define **RCX_ScoutRulesOp** 0xd5
- #define **RCX_ViewSourceValOp** 0xe5
- #define **RCX_ScoutOp** 0x47
- #define **RCX_SoundOp** 0x57
- #define **RCX_GOutputModeOp** 0x67
- #define **RCX_GOutputDirOp** 0x77
- #define **RCX_LightOp** 0x87
- #define **RCX_IncCounterOp** 0x97
- #define **RCX_DecCounterOp** 0xa7
- #define **RCX_ClearCounterOp** 0xb7
- #define **RCX_SetPriorityOp** 0xd7
- #define **RCX_MessageOp** 0xf7

6.175.1 Detailed Description

Constants for use when specifying RCX and Scout opcodes.

6.175.2 Define Documentation

6.175.2.1 #define **RCX_AbsVarOp** 0x74

Absolute value function

6.175.2.2 #define **RCX_AndVarOp** 0x84

AND function

6.175.2.3 #define **RCX_AutoOffOp** 0xb1

Set auto off timer

6.175.2.4 #define RCX_BatteryLevelOp 0x30

Read the battery level

6.175.2.5 #define RCX_BootModeOp 0x65

Set into book mode

6.175.2.6 #define RCX_CalibrateEventOp 0x04

Calibrate event

6.175.2.7 #define RCX_ClearAllEventsOp 0x06

Clear all events

6.175.2.8 #define RCX_ClearCounterOp 0xb7

Clear a counter

6.175.2.9 #define RCX_ClearMsgOp 0x90

Clear message

6.175.2.10 #define RCX_ClearSensorOp 0xd1

Clear a sensor

6.175.2.11 #define RCX_ClearSoundOp 0x80

Clear sound

6.175.2.12 #define RCX_ClearTimerOp 0xa1

Clear a timer

6.175.2.13 #define RCX_DatalogOp 0x62

Datalog the specified source/value

6.175.2.14 #define RCX_DecCounterOp 0xa7

Decrement a counter

6.175.2.15 #define RCX_DeleteSubOp 0xc1

Delete a subroutine

6.175.2.16 #define RCX_DeleteSubsOp 0x70

Delete subroutines

6.175.2.17 #define RCX_DeleteTaskOp 0x61

Delete a task

6.175.2.18 #define RCX_DeleteTasksOp 0x40

Delete tasks

6.175.2.19 #define RCX_DirectEventOp 0x03

Fire an event

6.175.2.20 #define RCX_DisplayOp 0x33

Set LCD display value

6.175.2.21 #define RCX_DivVarOp 0x44

Divide function

6.175.2.22 #define RCX_GOutputDirOp 0x77

Set global motor direction

6.175.2.23 #define RCX_GOutputModeOp 0x67

Set global motor mode

6.175.2.24 #define RCX_GOutputPowerOp 0xa3

Set global motor power levels

6.175.2.25 #define RCX_IncCounterOp 0x97

Increment a counter

6.175.2.26 #define RCX_InputModeOp 0x42

Set the input mode

6.175.2.27 #define RCX_InputTypeOp 0x32

Set the input type

6.175.2.28 #define RCX_IRModeOp 0x31

Set the IR transmit mode

6.175.2.29 #define RCX_LightOp 0x87

Light opcode

6.175.2.30 #define RCX_LSBlinkTimeOp 0xe3

Set the light sensor blink time

6.175.2.31 #define RCX_LSCalibrateOp 0xc0

Calibrate the light sensor

6.175.2.32 #define RCX_LSHysteresisOp 0xd3

Set the light sensor hysteresis

6.175.2.33 #define RCX_LSLowerThreshOp 0xc3

Set the light sensor lower threshold

6.175.2.34 #define RCX_LSUppThreshOp 0xb3

Set the light sensor upper threshold

6.175.2.35 #define RCX_MessageOp 0xf7

Set message

6.175.2.36 #define RCX_MulVarOp 0x54

Multiply function

6.175.2.37 #define RCX_MuteSoundOp 0xd0

Mute sound

6.175.2.38 #define RCX_OnOffFloatOp 0x21

Control motor state - on, off, float

6.175.2.39 #define RCX_OrVarOp 0x94

OR function

6.175.2.40 #define RCX_OutputDirOp 0xe1

Set the motor direction

6.175.2.41 #define RCX_OutputPowerOp 0x13

Set the motor power level

6.175.2.42 #define RCX_PBTurnOffOp 0x60

Turn off the brick

6.175.2.43 #define RCX_PingOp 0x10

Ping the brick

6.175.2.44 #define RCX_PlaySoundOp 0x51

Play a sound

6.175.2.45 #define RCX_PlayToneOp 0x23

Play a tone

6.175.2.46 #define RCX_PlayToneVarOp 0x02

Play a tone using a variable

6.175.2.47 #define RCX_PollMemoryOp 0x63

Poll a memory location

6.175.2.48 #define RCX_PollOp 0x12

Poll a source/value combination

6.175.2.49 #define RCX_RemoteOp 0xd2

Execute simulated remote control buttons

6.175.2.50 #define RCX_ScoutOp 0x47

Scout opcode

6.175.2.51 #define RCX_ScoutRulesOp 0xd5

Set Scout rules

6.175.2.52 #define RCX_SelectProgramOp 0x91

Select a program slot

6.175.2.53 #define RCX_SendUARTDataOp 0xc2

Send data via IR using UART settings

6.175.2.54 #define RCX_SetCounterOp 0xd4

Set counter value

6.175.2.55 #define RCX_SetDatalogOp 0x52

Set the datalog size

6.175.2.56 #define RCX_SetEventOp 0x93

Set an event

6.175.2.57 #define RCX_SetFeedbackOp 0x83

Set Scout feedback

6.175.2.58 #define RCX_SetPriorityOp 0xd7

Set task priority

6.175.2.59 #define RCX_SetSourceValueOp 0x05

Set a source/value

6.175.2.60 #define RCX_SetTimerLimitOp 0xc4

Set timer limit

6.175.2.61 #define RCX_SetVarOp 0x14

Set function

6.175.2.62 #define RCX_SetWatchOp 0x22

Set the watch source/value

6.175.2.63 #define RCX_SgnVarOp 0x64

Sign function

6.175.2.64 #define RCX_SoundOp 0x57

Sound opcode

6.175.2.65 #define RCX_StartTaskOp 0x71

Start a task

6.175.2.66 #define RCX_StopAllTasksOp 0x50

Stop all tasks

6.175.2.67 #define RCX_StopTaskOp 0x81

Stop a task

6.175.2.68 #define RCX_SubVarOp 0x34

Subtract function

6.175.2.69 #define RCX_SumVarOp 0x24

Sum function

6.175.2.70 #define RCX_UnlockFirmOp 0xa5

Unlock the firmware

6.175.2.71 #define RCX_UnlockOp 0x15

Unlock the brick

6.175.2.72 #define RCX_UnmuteSoundOp 0xe0

Unmute sound

6.175.2.73 #define RCX_UploadDatalogOp 0xa4

Upload datalog contents

6.175.2.74 #define RCX_ViewSourceValOp 0xe5[View a source/value](#)**6.175.2.75 #define RCX_VLLOp 0xe2**

Send visual light link (VLL) data

6.176 HiTechnic/mindsensors Power Function/IR Train constants

Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.

Modules

- [Power Function command constants](#)

Constants that are for sending Power Function commands.

- [Power Function channel constants](#)

Constants that are for specifying Power Function channels.

- [Power Function mode constants](#)

Constants that are for choosing Power Function modes.

- [PF/IR Train function constants](#)

Constants that are for sending PF/IR Train functions.

- [IR Train channel constants](#)

Constants that are for specifying IR Train channels.

- [Power Function output constants](#)

Constants that are for choosing a Power Function output.

- [Power Function pin constants](#)

Constants that are for choosing a Power Function pin.

- [Power Function single pin function constants](#)

Constants that are for sending Power Function single pin functions.

- [Power Function CST options constants](#)

Constants that are for specifying Power Function CST options.

- Power Function PWM option constants

Constants that are for specifying Power Function PWM options.

6.176.1 Detailed Description

Constants that are for use with the HiTechnic IRLink or mindsensors nRLink in Power Function or IR Train mode.

6.177 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

6.177.1 Detailed Description

Constants that are for sending Power Function commands.

6.177.2 Define Documentation

6.177.2.1 #define PF_CMD_BRAKE 3

Power function command brake

6.177.2.2 #define PF_CMD_FWD 1

Power function command forward

Examples:

[ex_HTPFComboDirect.nxc](#), and [ex_MSPFComboDirect.nxc](#).

6.177.2.3 #define PF_CMD_REV 2

Power function command reverse

6.177.2.4 #define PF_CMD_STOP 0

Power function command stop

Examples:

[ex_HTPFComboDirect.nxc](#), and [ex_MSPFComboDirect.nxc](#).

6.178 Power Function channel constants

Constants that are for specifying Power Function channels.

Defines

- #define [PF_CHANNEL_1](#) 0
- #define [PF_CHANNEL_2](#) 1
- #define [PF_CHANNEL_3](#) 2
- #define [PF_CHANNEL_4](#) 3

6.178.1 Detailed Description

Constants that are for specifying Power Function channels.

6.178.2 Define Documentation**6.178.2.1 #define PF_CHANNEL_1 0**

Power function channel 1

Examples:

[ex_HTPFComboDirect.nxc](#), [ex_HTPFComboPWM.nxc](#), [ex_HTPFSingleOutputCST.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#), [ex_HTPFSinglePin.nxc](#), [ex_HTPFTrain.nxc](#), [ex_MSPFComboDirect.nxc](#), [ex_MSPFComboPWM.nxc](#), [ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), [ex_MSPFSinglePin.nxc](#), and [ex_MSPFTrain.nxc](#).

6.178.2.2 #define PF_CHANNEL_2 1

Power function channel 2

6.178.2.3 #define PF_CHANNEL_3 2

Power function channel 3

6.178.2.4 #define PF_CHANNEL_4 3

Power function channel 4

6.179 Power Function mode constants

Constants that are for choosing Power Function modes.

Defines

- #define PF_MODE_TRAIN 0
- #define PF_MODE_COMBO_DIRECT 1
- #define PF_MODE_SINGLE_PIN_CONT 2
- #define PF_MODE_SINGLE_PIN_TIME 3
- #define PF_MODE_COMBO_PWM 4
- #define PF_MODE_SINGLE_OUTPUT_PWM 4
- #define PF_MODE_SINGLE_OUTPUT_CST 6

6.179.1 Detailed Description

Constants that are for choosing Power Function modes.

6.179.2 Define Documentation**6.179.2.1 #define PF_MODE_COMBO_DIRECT 1**

Power function mode combo direct

6.179.2.2 #define PF_MODE_COMBO_PWM 4

Power function mode combo pulse width modulation (PWM)

6.179.2.3 #define PF_MODE_SINGLE_OUTPUT_CST 6

Power function mode single output clear, set, toggle (CST)

6.179.2.4 #define PF_MODE_SINGLE_OUTPUT_PWM 4

Power function mode single output pulse width modulation (PWM)

6.179.2.5 #define PF_MODE_SINGLE_PIN_CONT 2

Power function mode single pin continuous

6.179.2.6 #define PF_MODE_SINGLE_PIN_TIME 3

Power function mode single pin timed

6.179.2.7 #define PF_MODE_TRAIN 0

Power function mode IR Train

6.180 PF/IR Train function constants

Constants that are for sending PF/IR Train functions.

Defines

- #define TRAIN_FUNC_STOP 0
- #define TRAIN_FUNC_INCR_SPEED 1
- #define TRAIN_FUNC_DECR_SPEED 2
- #define TRAIN_FUNC_TOGGLE_LIGHT 4

6.180.1 Detailed Description

Constants that are for sending PF/IR Train functions.

6.180.2 Define Documentation**6.180.2.1 #define TRAIN_FUNC_DECR_SPEED 2**

PF/IR Train function decrement speed

6.180.2.2 #define TRAIN_FUNC_INCR_SPEED 1

PF/IR Train function increment speed

Examples:

[ex_HTIRTrain.nxc](#), [ex_HTPFTrain.nxc](#), [ex_MSIRTrain.nxc](#), and [ex_MSPFTrain.nxc](#).

6.180.2.3 #define TRAIN_FUNC_STOP 0

PF/IR Train function stop

6.180.2.4 #define TRAIN_FUNC_TOGGLE_LIGHT 4

PF/IR Train function toggle light

6.181 IR Train channel constants

Constants that are for specifying IR Train channels.

Defines

- #define [TRAIN_CHANNEL_1](#) 0
- #define [TRAIN_CHANNEL_2](#) 1
- #define [TRAIN_CHANNEL_3](#) 2
- #define [TRAIN_CHANNEL_ALL](#) 3

6.181.1 Detailed Description

Constants that are for specifying IR Train channels.

6.181.2 Define Documentation**6.181.2.1 #define TRAIN_CHANNEL_1 0**

IR Train channel 1

Examples:

[ex_HTIRTrain.nxc](#), and [ex_MSIRTrain.nxc](#).

6.181.2.2 #define TRAIN_CHANNEL_2 1

IR Train channel 2

6.181.2.3 #define TRAIN_CHANNEL_3 2

IR Train channel 3

6.181.2.4 #define TRAIN_CHANNEL_ALL 3

IR Train channel all

6.182 Power Function output constants

Constants that are for choosing a Power Function output.

Defines

- #define PF_OUT_A 0
- #define PF_OUT_B 1

6.182.1 Detailed Description

Constants that are for choosing a Power Function output.

6.182.2 Define Documentation**6.182.2.1 #define PF_OUT_A 0**

Power function output A

Examples:

[ex_HTPFSingleOutputCST.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#),
[ex_HTPFSinglePin.nxc](#), [ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), and [ex_MSPFSinglePin.nxc](#).

6.182.2.2 #define PF_OUT_B 1

Power function output B

6.183 Power Function pin constants

Constants that are for choosing a Power Function pin.

Defines

- #define PF_PIN_C1 0
- #define PF_PIN_C2 1

6.183.1 Detailed Description

Constants that are for choosing a Power Function pin.

6.183.2 Define Documentation

6.183.2.1 #define PF_PIN_C1 0

Power function pin C1

Examples:

[ex_HTPFSinglePin.nxc](#), and [ex_MSPFSinglePin.nxc](#).

6.183.2.2 #define PF_PIN_C2 1

Power function pin C2

6.184 Power Function single pin function constants

Constants that are for sending Power Function single pin functions.

Defines

- #define PF_FUNC_NOCHANGE 0
- #define PF_FUNC_CLEAR 1
- #define PF_FUNC_SET 2
- #define PF_FUNC_TOGGLE 3

6.184.1 Detailed Description

Constants that are for sending Power Function single pin functions.

6.184.2 Define Documentation**6.184.2.1 #define PF_FUNC_CLEAR 1**

Power function single pin - clear

6.184.2.2 #define PF_FUNC_NOCHANGE 0

Power function single pin - no change

6.184.2.3 #define PF_FUNC_SET 2

Power function single pin - set

Examples:

[ex_HTPFSinglePin.nxc](#), and [ex_MSPFSinglePin.nxc](#).

6.184.2.4 #define PF_FUNC_TOGGLE 3

Power function single pin - toggle

6.185 Power Function CST options constants

Constants that are for specifying Power Function CST options.

Defines

- #define PF_CST_CLEAR1_CLEAR2 0
- #define PF_CST_SET1_CLEAR2 1
- #define PF_CST_CLEAR1_SET2 2
- #define PF_CST_SET1_SET2 3
- #define PF_CST_INCREMENT_PWM 4
- #define PF_CST_DECREMENT_PWM 5
- #define PF_CST_FULL_FWD 6
- #define PF_CST_FULL_REV 7
- #define PF_CST_TOGGLE_DIR 8

6.185.1 Detailed Description

Constants that are for specifying Power Function CST options.

6.185.2 Define Documentation**6.185.2.1 #define PF_CST_CLEAR1_CLEAR2 0**

Power function CST clear 1 and clear 2

6.185.2.2 #define PF_CST_CLEAR1_SET2 2

Power function CST clear 1 and set 2

6.185.2.3 #define PF_CST_DECREMENT_PWM 5

Power function CST decrement PWM

6.185.2.4 #define PF_CST_FULL_FWD 6

Power function CST full forward

6.185.2.5 #define PF_CST_FULL_REV 7

Power function CST full reverse

6.185.2.6 #define PF_CST_INCREMENT_PWM 4

Power function CST increment PWM

6.185.2.7 #define PF_CST_SET1_CLEAR2 1

Power function CST set 1 and clear 2

6.185.2.8 #define PF_CST_SET1_SET2 3

Power function CST set 1 and set 2

Examples:

[ex_HTPFSingleOutputCST.nxc](#), and [ex_MSPPFSingleOutputCST.nxc](#).

6.185.2.9 #define PF_CST_TOGGLE_DIR 8

Power function CST toggle direction

6.186 Power Function PWM option constants

Constants that are for specifying Power Function PWM options.

Defines

- #define PF_PWM_FLOAT 0
- #define PF_PWM_FWD1 1
- #define PF_PWM_FWD2 2
- #define PF_PWM_FWD3 3
- #define PF_PWM_FWD4 4
- #define PF_PWM_FWD5 5
- #define PF_PWM_FWD6 6
- #define PF_PWM_FWD7 7
- #define PF_PWM_BRAKE 8
- #define PF_PWM_REV7 9
- #define PF_PWM_REV6 10
- #define PF_PWM_REV5 11
- #define PF_PWM_REV4 12
- #define PF_PWM_REV3 13
- #define PF_PWM_REV2 14
- #define PF_PWM_REV1 15

6.186.1 Detailed Description

Constants that are for specifying Power Function PWM options.

6.186.2 Define Documentation

6.186.2.1 #define PF_PWM_BRAKE 8

Power function PWM brake

6.186.2.2 #define PF_PWM_FLOAT 0

Power function PWM float

6.186.2.3 #define PF_PWM_FWD1 1

Power function PWM foward level 1

6.186.2.4 #define PF_PWM_FWD2 2

Power function PWM foward level 2

6.186.2.5 #define PF_PWM_FWD3 3

Power function PWM foward level 3

6.186.2.6 #define PF_PWM_FWD4 4

Power function PWM foward level 4

6.186.2.7 #define PF_PWM_FWD5 5

Power function PWM foward level 5

Examples:

[ex_HTPFComboPWM.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#), [ex_MSPFComboPWM.nxc](#), and [ex_MSPFSingleOutputPWM.nxc](#).

6.186.2.8 #define PF_PWM_FWD6 6

Power function PWM foward level 6

6.186.2.9 #define PF_PWM_FWD7 7

Power function PWM foward level 7

6.186.2.10 #define PF_PWM_REV1 15

Power function PWM reverse level 1

6.186.2.11 #define PF_PWM_REV2 14

Power function PWM reverse level 2

6.186.2.12 #define PF_PWM_REV3 13

Power function PWM reverse level 3

6.186.2.13 #define PF_PWM_REV4 12

Power function PWM reverse level 4

Examples:

[ex_HTPFComboPWM.nxc](#), and [ex_MSPFComboPWM.nxc](#).

6.186.2.14 #define PF_PWM_REV5 11

Power function PWM reverse level 5

6.186.2.15 #define PF_PWM_REV6 10

Power function PWM reverse level 6

6.186.2.16 #define PF_PWM_REV7 9

Power function PWM reverse level 7

6.187 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.

Defines

- #define [HT_ADDR_IRSEEKER](#) 0x02
- #define [HT_ADDR_IRSEEKER2](#) 0x10
- #define [HT_ADDR_IRRECEIVER](#) 0x02

- #define HT_ADDR_COMPASS 0x02
- #define HT_ADDR_ACCEL 0x02
- #define HT_ADDR_COLOR 0x02
- #define HT_ADDR_COLOR2 0x02
- #define HT_ADDR_IRLINK 0x02

6.187.1 Detailed Description

Constants that are for use with HiTechnic devices.

6.187.2 Define Documentation

6.187.2.1 #define HT_ADDR_ACCEL 0x02

HiTechnic Accel I2C address

6.187.2.2 #define HT_ADDR_COLOR 0x02

HiTechnic Color I2C address

6.187.2.3 #define HT_ADDR_COLOR2 0x02

HiTechnic Color2 I2C address

6.187.2.4 #define HT_ADDR_COMPASS 0x02

HiTechnic Compass I2C address

6.187.2.5 #define HT_ADDR_IRLINK 0x02

HiTechnic IRLink I2C address

6.187.2.6 #define HT_ADDR_IRRECEIVER 0x02

HiTechnic IRReceiver I2C address

6.187.2.7 #define HT_ADDR_IRSEEKER 0x02

HiTechnic IRSeeker I2C address

6.187.2.8 #define HT_ADDR_IRSEEKER2 0x10

HiTechnic IRSeeker2 I2C address

6.188 HiTechnic IRSeeker2 constants

Constants that are for use with the HiTechnic IRSeeker2 device.

Defines

- #define HTIR2_MODE_1200 0
- #define HTIR2_MODE_600 1
- #define HTIR2_REG_MODE 0x41
- #define HTIR2_REG_DCDIR 0x42
- #define HTIR2_REG_DC01 0x43
- #define HTIR2_REG_DC02 0x44
- #define HTIR2_REG_DC03 0x45
- #define HTIR2_REG_DC04 0x46
- #define HTIR2_REG_DC05 0x47
- #define HTIR2_REG_DCAVG 0x48
- #define HTIR2_REG_ACDIR 0x49
- #define HTIR2_REG_AC01 0x4A
- #define HTIR2_REG_AC02 0x4B
- #define HTIR2_REG_AC03 0x4C
- #define HTIR2_REG_AC04 0x4D
- #define HTIR2_REG_AC05 0x4E

6.188.1 Detailed Description

Constants that are for use with the HiTechnic IRSeeker2 device.

6.188.2 Define Documentation

6.188.2.1 #define HTIR2_MODE_1200 0

Set IRSeeker2 to 1200 mode

Examples:

[ex_SetHTIRSeeker2Mode.nxc](#), and [ex_SetSensorBoolean.nxc](#).

6.188.2.2 #define HTIR2_MODE_600 1

Set IRSeeker2 to 600 mode

6.188.2.3 #define HTIR2_REG_AC01 0x4A

IRSeeker2 AC 01 register

6.188.2.4 #define HTIR2_REG_AC02 0x4B

IRSeeker2 AC 02 register

6.188.2.5 #define HTIR2_REG_AC03 0x4C

IRSeeker2 AC 03 register

6.188.2.6 #define HTIR2_REG_AC04 0x4D

IRSeeker2 AC 04 register

6.188.2.7 #define HTIR2_REG_AC05 0x4E

IRSeeker2 AC 05 register

6.188.2.8 #define HTIR2_REG_ACDIR 0x49

IRSeeker2 AC direction register

6.188.2.9 #define HTIR2_REG_DC01 0x43

IRSeeker2 DC 01 register

6.188.2.10 #define HTIR2_REG_DC02 0x44

IRSeeker2 DC 02 register

6.188.2.11 #define HTIR2_REG_DC03 0x45

IRSeeker2 DC 03 register

6.188.2.12 #define HTIR2_REG_DC04 0x46

IRSeeker2 DC 04 register

6.188.2.13 #define HTIR2_REG_DC05 0x47

IRSeeker2 DC 05 register

6.188.2.14 #define HTIR2_REG_DCAVG 0x48

IRSeeker2 DC average register

Examples:[ex_SensorHTIRSeeker2Addr.nxc](#).**6.188.2.15 #define HTIR2_REG_DCDIR 0x42**

IRSeeker2 DC direction register

6.188.2.16 #define HTIR2_REG_MODE 0x41

IRSeeker2 mode register

6.189 HiTechnic IRReceiver constants

Constants that are for use with the HiTechnic IRReceiver device.

Defines

- #define [HT_CH1_A](#) 0
- #define [HT_CH1_B](#) 1
- #define [HT_CH2_A](#) 2
- #define [HT_CH2_B](#) 3
- #define [HT_CH3_A](#) 4
- #define [HT_CH3_B](#) 5
- #define [HT_CH4_A](#) 6
- #define [HT_CH4_B](#) 7

6.189.1 Detailed Description

Constants that are for use with the HiTechnic IRReceiver device.

6.189.2 Define Documentation

6.189.2.1 `#define HT_CH1_A 0`

Use IRReceiver channel 1 output A

Examples:

[ex_ReadSensorHTIRReceiverEx.nxc](#).

6.189.2.2 `#define HT_CH1_B 1`

Use IRReceiver channel 1 output B

6.189.2.3 `#define HT_CH2_A 2`

Use IRReceiver channel 2 output A

6.189.2.4 `#define HT_CH2_B 3`

Use IRReceiver channel 2 output B

6.189.2.5 `#define HT_CH3_A 4`

Use IRReceiver channel 3 output A

6.189.2.6 `#define HT_CH3_B 5`

Use IRReceiver channel 3 output B

6.189.2.7 `#define HT_CH4_A 6`

Use IRReceiver channel 4 output A

6.189.2.8 #define HT_CH4_B 7

Use IRReceiver channel 4 output B

6.190 HiTechnic Color2 constants

Constants that are for use with the HiTechnic Color2 device.

Defines

- #define HT_CMD_COLOR2_ACTIVE 0x00
- #define HT_CMD_COLOR2_PASSIVE 0x01
- #define HT_CMD_COLOR2_RAW 0x03
- #define HT_CMD_COLOR2_50HZ 0x35
- #define HT_CMD_COLOR2_60HZ 0x36
- #define HT_CMD_COLOR2_BLCAL 0x42
- #define HT_CMD_COLOR2_WBCAL 0x43
- #define HT_CMD_COLOR2_FAR 0x46
- #define HT_CMD_COLOR2_LED_HI 0x48
- #define HT_CMD_COLOR2_LED_LOW 0x4C
- #define HT_CMD_COLOR2_NEAR 0x4E

6.190.1 Detailed Description

Constants that are for use with the HiTechnic Color2 device.

6.190.2 Define Documentation**6.190.2.1 #define HT_CMD_COLOR2_50HZ 0x35**

Set the Color2 sensor to 50Hz mode

6.190.2.2 #define HT_CMD_COLOR2_60HZ 0x36

Set the Color2 sensor to 60Hz mode

6.190.2.3 #define HT_CMD_COLOR2_ACTIVE 0x00

Set the Color2 sensor to active mode

Examples:

[ex_I2CSendCommand.nxc](#), and [ex_SetHTColor2Mode.nxc](#).

6.190.2.4 #define HT_CMD_COLOR2_BLCAL 0x42

Set the Color2 sensor to black level calibration mode

6.190.2.5 #define HT_CMD_COLOR2_FAR 0x46

Set the Color2 sensor to far mode

6.190.2.6 #define HT_CMD_COLOR2_LED_HI 0x48

Set the Color2 sensor to LED high mode

6.190.2.7 #define HT_CMD_COLOR2_LED_LOW 0x4C

Set the Color2 sensor to LED low mode

6.190.2.8 #define HT_CMD_COLOR2_NEAR 0x4E

Set the Color2 sensor to near mode

6.190.2.9 #define HT_CMD_COLOR2_PASSIVE 0x01

Set the Color2 sensor to passive mode

6.190.2.10 #define HT_CMD_COLOR2_RAW 0x03

Set the Color2 sensor to raw mode

6.190.2.11 #define HT_CMD_COLOR2_WBCAL 0x43

Set the Color2 sensor to white level calibration mode

6.191 MindSensors device constants

Constants that are for use with MindSensors devices.

Modules

- [MindSensors DIST-Nx constants](#)

Constants that are for use with the MindSensors DIST-Nx device.

- [MindSensors PSP-Nx constants](#)

Constants that are for use with the MindSensors PSP-Nx device.

- [MindSensors nRLink constants](#)

Constants that are for use with the MindSensors nRLink device.

- [MindSensors ACCL-Nx constants](#)

Constants that are for use with the MindSensors ACCL-Nx device.

Defines

- #define [MS_CMD_ENERGIZED](#) 0x45
- #define [MS_CMD_DEENERGIZED](#) 0x44
- #define [MS_CMD_ADPA_ON](#) 0x4E
- #define [MS_CMD_ADPA_OFF](#) 0x4F
- #define [MS_ADDR_RTCLOCK](#) 0xD0
- #define [MS_ADDR_DISTNX](#) 0x02
- #define [MS_ADDR_NRLINK](#) 0x02
- #define [MS_ADDR_ACCLNX](#) 0x02
- #define [MS_ADDR_CMPSNX](#) 0x02
- #define [MS_ADDR_PSPNX](#) 0x02
- #define [MS_ADDR_LINELDR](#) 0x02
- #define [MS_ADDR_NXTCAM](#) 0x02
- #define [MS_ADDR_NXTHID](#) 0x04
- #define [MS_ADDR_NXTSERVO](#) 0xB0
- #define [MS_ADDR_PFMATE](#) 0x48
- #define [MS_ADDR_MTRMUX](#) 0xB4
- #define [MS_ADDR_NXTMMX](#) 0x06
- #define [MS_ADDR_IVSENS](#) 0x12
- #define [MS_ADDR_RXMUX](#) 0x7E

6.191.1 Detailed Description

Constants that are for use with MindSensors devices.

6.191.2 Define Documentation**6.191.2.1 #define MS_ADDR_ACCLNX 0x02**

MindSensors ACCL-Nx I2C address

Examples:

[ex_ACCLNxCalibrateX.nxc](#), [ex_ACCLNxCalibrateXEnd.nxc](#), [ex_ACCLNxCalibrateY.nxc](#), [ex_ACCLNxCalibrateYEnd.nxc](#), [ex_ACCLNxCalibrateZ.nxc](#), [ex_ACCLNxCalibrateZEnd.nxc](#), [ex_ACCLNxResetCalibration.nxc](#), [ex_ACCLNxSensitivity.nxc](#), [ex_ACCLNxXOffset.nxc](#), [ex_ACCLNxXRange.nxc](#), [ex_ACCLNxYOffset.nxc](#), [ex_ACCLNxYRange.nxc](#), [ex_ACCLNxZOffset.nxc](#), [ex_ACCLNxZRange.nxc](#), [ex_ReadSensorMSAccel.nxc](#), [ex_ReadSensorMSTilt.nxc](#), and [ex_SetACCLNxSensitivity.nxc](#).

6.191.2.2 #define MS_ADDR_CMPSNX 0x02

MindSensors CMPS-Nx I2C address

Examples:

[ex_SensorMSCompass.nxc](#).

6.191.2.3 #define MS_ADDR_DISTNX 0x02

MindSensors DIST-Nx I2C address

Examples:

[ex_DISTNxDistance.nxc](#), [ex_DISTNxGP2D12.nxc](#), [ex_DISTNxGP2D120.nxc](#), [ex_DISTNxGP2YA02.nxc](#), [ex_DISTNxGP2YA21.nxc](#), [ex_DISTNxMaxDistance.nxc](#), [ex_DISTNxMinDistance.nxc](#), [ex_DISTNxModuleType.nxc](#), [ex_DISTNxNumPoints.nxc](#), [ex_DISTNxVoltage.nxc](#), [ex_MSADPAOff.nxc](#), and [ex_MSADPAOn.nxc](#).

6.191.2.4 #define MS_ADDR_IVSENS 0x12

MindSensors IVSens I2C address

6.191.2.5 #define MS_ADDR_LINELDR 0x02

MindSensors LineLdr I2C address

6.191.2.6 #define MS_ADDR_MTRMUX 0xB4

MindSensors MTRMux I2C address

6.191.2.7 #define MS_ADDR_NRLINK 0x02

MindSensors NRLink I2C address

Examples:

[ex_MSRCXSetNRLinkPort.nxc](#), [ex_NRLink2400.nxc](#), [ex_NRLink4800.nxc](#),
[ex_NRLinkFlush.nxc](#), [ex_NRLinkIRLong.nxc](#), [ex_NRLinkIRShort.nxc](#),
[ex_NRLinkSetPF.nxc](#), [ex_NRLinkSetRCX.nxc](#), [ex_NRLinkSetTrain.nxc](#),
[ex_NRLinkStatus.nxc](#), [ex_NRLinkTxRaw.nxc](#), [ex_ReadNRLinkBytes.nxc](#),
[ex_RunNRLinkMacro.nxc](#), and [ex_WriteNRLinkBytes.nxc](#).

6.191.2.8 #define MS_ADDR_NXTCAM 0x02

MindSensors NXTCam I2C address

6.191.2.9 #define MS_ADDR_NXTHID 0x04

MindSensors NXTHID I2C address

6.191.2.10 #define MS_ADDR_NXTMMX 0x06

MindSensors NXTMMX I2C address

6.191.2.11 #define MS_ADDR_NXTSERVO 0xB0

MindSensors NXTServo I2C address

6.191.2.12 #define MS_ADDR_PFMATE 0x48

MindSensors PFMate I2C address

6.191.2.13 #define MS_ADDR_PSPNX 0x02

MindSensors PSP-Nx I2C address

Examples:

[ex_PSPNxAnalog.nxc](#), [ex_PSPNxDigital.nxc](#), and [ex_-ReadSensorMSPlayStation.nxc](#).

6.191.2.14 #define MS_ADDR_RTCLOCK 0xD0

MindSensors RTClock I2C address

6.191.2.15 #define MS_ADDR_RXMUX 0x7E

MindSensors RXMux I2C address

6.191.2.16 #define MS_CMD_ADPA_OFF 0x4F

Turn MindSensors ADPA mode off

6.191.2.17 #define MS_CMD_ADPA_ON 0x4E

Turn MindSensors ADPA mode on

6.191.2.18 #define MS_CMD_DEENERGIZED 0x44

De-energize the MindSensors device

6.191.2.19 #define MS_CMD_ENERGIZED 0x45

Energize the MindSensors device

6.192 MindSensors DIST-Nx constants

Constants that are for use with the MindSensors DIST-Nx device.

Defines

- #define [DIST_CMD_GP2D12](#) 0x31
- #define [DIST_CMD_GP2D120](#) 0x32
- #define [DIST_CMD_GP2YA21](#) 0x33
- #define [DIST_CMD_GP2YA02](#) 0x34
- #define [DIST_CMD_CUSTOM](#) 0x35

- #define DIST_REG_DIST 0x42
- #define DIST_REG_VOLT 0x44
- #define DIST_REG_MODULE_TYPE 0x50
- #define DIST_REG_NUM_POINTS 0x51
- #define DIST_REG_DIST_MIN 0x52
- #define DIST_REG_DIST_MAX 0x54
- #define DIST_REG_VOLT1 0x56
- #define DIST_REG_DIST1 0x58

6.192.1 Detailed Description

Constants that are for use with the MindSensors DIST-Nx device.

6.192.2 Define Documentation

6.192.2.1 #define DIST_CMD_CUSTOM 0x35

Set the DIST-Nx to a custom mode

6.192.2.2 #define DIST_CMD_GP2D12 0x31

Set the DIST-Nx to GP2D12 mode

6.192.2.3 #define DIST_CMD_GP2D120 0x32

Set the DIST-Nx to GP2D120 mode

6.192.2.4 #define DIST_CMD_GP2YA02 0x34

Set the DIST-Nx to GP2YA02 mode

6.192.2.5 #define DIST_CMD_GP2YA21 0x33

Set the DIST-Nx to GP2YA21 mode

6.192.2.6 #define DIST_REG_DIST 0x42

The DIST-Nx distance register

6.192.2.7 #define DIST_REG_DIST1 0x58

The DIST-Nx distance 1 register

6.192.2.8 #define DIST_REG_DIST_MAX 0x54

The DIST-Nx maximum distance register

6.192.2.9 #define DIST_REG_DIST_MIN 0x52

The DIST-Nx minimum distance register

6.192.2.10 #define DIST_REG_MODULE_TYPE 0x50

The DIST-Nx module type register

6.192.2.11 #define DIST_REG_NUM_POINTS 0x51

The DIST-Nx number of data points in Custom curve register

6.192.2.12 #define DIST_REG_VOLT 0x44

The DIST-Nx voltage register

6.192.2.13 #define DIST_REG_VOLT1 0x56

The DIST-Nx voltage 1 register

6.193 MindSensors PSP-Nx constants

Constants that are for use with the MindSensors PSP-Nx device.

Modules

- [MindSensors PSP-Nx button set 1 constants](#)

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

- [MindSensors PSP-Nx button set 2 constants](#)

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

Defines

- #define **PSP_CMD_DIGITAL** 0x41
- #define **PSP_CMD_ANALOG** 0x73
- #define **PSP_REG_BTNSET1** 0x42
- #define **PSP_REG_BTNSET2** 0x43
- #define **PSP_REG_XLEFT** 0x44
- #define **PSP_REG_YLEFT** 0x45
- #define **PSP_REG_XRIGHT** 0x46
- #define **PSP_REG_YRIGHT** 0x47

6.193.1 Detailed Description

Constants that are for use with the MindSensors PSP-Nx device.

6.193.2 Define Documentation**6.193.2.1 #define PSP_CMD_ANALOG 0x73**

Set the PSP-Nx to analog mode

6.193.2.2 #define PSP_CMD_DIGITAL 0x41

Set the PSP-Nx to digital mode

6.193.2.3 #define PSP_REG_BTNSET1 0x42

The PSP-Nx button set 1 register

6.193.2.4 #define PSP_REG_BTNSET2 0x43

The PSP-Nx button set 2 register

6.193.2.5 #define PSP_REG_XLEFT 0x44

The PSP-Nx X left register

6.193.2.6 #define PSP_REG_XRIGHT 0x46

The PSP-Nx X right register

6.193.2.7 #define PSP_REG_YLEFT 0x45

The PSP-Nx Y left register

6.193.2.8 #define PSP_REG_YRIGHT 0x47

The PSP-Nx Y right register

6.194 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

6.194.1 Detailed Description

Constants that are for interpreting MindSensors PSP-Nx button set 1 values.

6.194.2 Define Documentation**6.194.2.1 #define PSP_BTNSET1_DOWN 0x02**

The PSP-Nx button set 1 down arrow

6.194.2.2 #define PSP_BTNSET1_L3 0x40

The PSP-Nx button set 1 L3

6.194.2.3 #define PSP_BTNSET1_LEFT 0x01

The PSP-Nx button set 1 left arrow

6.194.2.4 #define PSP_BTNSET1_R3 0x20

The PSP-Nx button set 1 R3

6.194.2.5 #define PSP_BTNSET1_RIGHT 0x04

The PSP-Nx button set 1 right arrow

6.194.2.6 #define PSP_BTNSET1_UP 0x08

The PSP-Nx button set 1 up arrow

6.195 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

6.195.1 Detailed Description

Constants that are for interpreting MindSensors PSP-Nx button set 2 values.

6.195.2 Define Documentation**6.195.2.1 #define PSP_BTNSET2_CIRCLE 0x04**

The PSP-Nx button set 2 circle

6.195.2.2 #define PSP_BTNSET2_CROSS 0x02

The PSP-Nx button set 2 cross

6.195.2.3 #define PSP_BTNSET2_L1 0x20

The PSP-Nx button set 2 L1

6.195.2.4 #define PSP_BTNSET2_L2 0x80

The PSP-Nx button set 2 L2

6.195.2.5 #define PSP_BTNSET2_R1 0x10

The PSP-Nx button set 2 R1

6.195.2.6 #define PSP_BTNSET2_R2 0x40

The PSP-Nx button set 2 R2

6.195.2.7 #define PSP_BTNSET2_SQUARE 0x01

The PSP-Nx button set 2 square

6.195.2.8 #define PSP_BTNSET2_TRIANGLE 0x08

The PSP-Nx button set 2 triangle

6.196 MindSensors nRLink constants

Constants that are for use with the MindSensors nRLink device.

Defines

- #define [NRLINK_CMD_2400](#) 0x44
- #define [NRLINK_CMD_FLUSH](#) 0x46
- #define [NRLINK_CMD_4800](#) 0x48
- #define [NRLINK_CMD_IR_LONG](#) 0x4C
- #define [NRLINK_CMD_IR_SHORT](#) 0x53
- #define [NRLINK_CMD_RUN_MACRO](#) 0x52
- #define [NRLINK_CMD_TX_RAW](#) 0x55
- #define [NRLINK_CMD_SET_RCX](#) 0x58
- #define [NRLINK_CMD_SET_TRAIN](#) 0x54
- #define [NRLINK_CMD_SET_PF](#) 0x50

- #define **NRLINK_REG_BYTES** 0x40
- #define **NRLINK_REG_DATA** 0x42
- #define **NRLINK_REG_EEPROM** 0x50

6.196.1 Detailed Description

Constants that are for use with the MindSensors nRLink device.

6.196.2 Define Documentation

6.196.2.1 #define **NRLINK_CMD_2400** 0x44

Set NRLink to 2400 baud

6.196.2.2 #define **NRLINK_CMD_4800** 0x48

Set NRLink to 4800 baud

6.196.2.3 #define **NRLINK_CMD_FLUSH** 0x46

Flush the NRLink

6.196.2.4 #define **NRLINK_CMD_IR_LONG** 0x4C

Set the NRLink to long range IR

6.196.2.5 #define **NRLINK_CMD_IR_SHORT** 0x53

Set the NRLink to short range IR

6.196.2.6 #define **NRLINK_CMD_RUN_MACRO** 0x52

Run an NRLink macro

6.196.2.7 #define **NRLINK_CMD_SET_PF** 0x50

Set the NRLink to Power Function mode

6.196.2.8 #define NRLINK_CMD_SET_RCX 0x58

Set the NRLink to RCX mode

6.196.2.9 #define NRLINK_CMD_SET_TRAIN 0x54

Set the NRLink to IR Train mode

6.196.2.10 #define NRLINK_CMD_TX_RAW 0x55

Set the NRLink to transmit raw bytes

6.196.2.11 #define NRLINK_REG_BYTES 0x40

The NRLink bytes register

6.196.2.12 #define NRLINK_REG_DATA 0x42

The NRLink data register

6.196.2.13 #define NRLINK_REG_EEPROM 0x50

The NRLink eeprom register

6.197 MindSensors ACCL-Nx constants

Constants that are for use with the MindSensors ACCL-Nx device.

Modules

- [MindSensors ACCL-Nx sensitivity level constants](#)

Constants that are for setting the MindSensors ACCL-Nx sensitivity level.

Defines

- #define [ACCL_CMD_X_CAL](#) 0x58
- #define [ACCL_CMD_Y_CAL](#) 0x59
- #define [ACCL_CMD_Z_CAL](#) 0x5a
- #define [ACCL_CMD_X_CAL_END](#) 0x78

- #define ACCL_CMD_Y_CAL_END 0x79
- #define ACCL_CMD_Z_CAL_END 0x7a
- #define ACCL_CMD_RESET_CAL 0x52
- #define ACCL_REG_SENS_LVL 0x19
- #define ACCL_REG_X_TILT 0x42
- #define ACCL_REG_Y_TILT 0x43
- #define ACCL_REG_Z_TILT 0x44
- #define ACCL_REG_X_ACCEL 0x45
- #define ACCL_REG_Y_ACCEL 0x47
- #define ACCL_REG_Z_ACCEL 0x49
- #define ACCL_REG_X_OFFSET 0x4b
- #define ACCL_REG_X_RANGE 0x4d
- #define ACCL_REG_Y_OFFSET 0x4f
- #define ACCL_REG_Y_RANGE 0x51
- #define ACCL_REG_Z_OFFSET 0x53
- #define ACCL_REG_Z_RANGE 0x55

6.197.1 Detailed Description

Constants that are for use with the MindSensors ACCL-Nx device.

6.197.2 Define Documentation

6.197.2.1 #define ACCL_CMD_RESET_CAL 0x52

Reset to factory calibration

6.197.2.2 #define ACCL_CMD_X_CAL 0x58

Acquire X-axis calibration point

6.197.2.3 #define ACCL_CMD_X_CAL_END 0x78

Acquire X-axis calibration point and end calibration

6.197.2.4 #define ACCL_CMD_Y_CAL 0x59

Acquire Y-axis calibration point

6.197.2.5 #define ACCL_CMD_Y_CAL_END 0x79

Acquire Y-axis calibration point and end calibration

6.197.2.6 #define ACCL_CMD_Z_CAL 0x5a

Acquire Z-axis calibration point

6.197.2.7 #define ACCL_CMD_Z_CAL_END 0x7a

Acquire Z-axis calibration point and end calibration

6.197.2.8 #define ACCL_REG_SENS_LVL 0x19

The current sensitivity

6.197.2.9 #define ACCL_REG_X_ACCEL 0x45

The X-axis acceleration data

6.197.2.10 #define ACCL_REG_X_OFFSET 0x4b

The X-axis offset

6.197.2.11 #define ACCL_REG_X_RANGE 0x4d

The X-axis range

6.197.2.12 #define ACCL_REG_X_TILT 0x42

The X-axis tilt data

6.197.2.13 #define ACCL_REG_Y_ACCEL 0x47

The Y-axis acceleration data

6.197.2.14 #define ACCL_REG_Y_OFFSET 0x4f

The Y-axis offset

6.197.2.15 #define ACCL_REG_Y_RANGE 0x51

The Y-axis range

6.197.2.16 #define ACCL_REG_Y_TILT 0x43

The Y-axis tilt data

6.197.2.17 #define ACCL_REG_Z_ACCEL 0x49

The Z-axis acceleration data

6.197.2.18 #define ACCL_REG_Z_OFFSET 0x53

The Z-axis offset

6.197.2.19 #define ACCL_REG_Z_RANGE 0x55

The Z-axis range

6.197.2.20 #define ACCL_REG_Z_TILT 0x44

The Z-axis tilt data

6.198 MindSensors ACCL-Nx sensitivity level constants

Constants that are for setting the MindSensors ACCL-Nx sensitivity level.

Defines

- #define [ACCL_SENSITIVITY_LEVEL_1](#) 0x31
- #define [ACCL_SENSITIVITY_LEVEL_2](#) 0x32
- #define [ACCL_SENSITIVITY_LEVEL_3](#) 0x33
- #define [ACCL_SENSITIVITY_LEVEL_4](#) 0x34

6.198.1 Detailed Description

Constants that are for setting the MindSensors ACCL-Nx sensitivity level.

6.198.2 Define Documentation**6.198.2.1 #define ACCL_SENSITIVITY_LEVEL_1 0x31**

The ACCL-Nx sensitivity level 1

Examples:

[ex_SetACCLNxSensitivity.nxc.](#)

6.198.2.2 #define ACCL_SENSITIVITY_LEVEL_2 0x32

The ACCL-Nx sensitivity level 2

6.198.2.3 #define ACCL_SENSITIVITY_LEVEL_3 0x33

The ACCL-Nx sensitivity level 3

6.198.2.4 #define ACCL_SENSITIVITY_LEVEL_4 0x34

The ACCL-Nx sensitivity level 4

6.199 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

6.199.1 Detailed Description

Constants that define various data type limits.

6.199.2 Define Documentation

6.199.2.1 #define CHAR_BIT 8

The number of bits in the char type

6.199.2.2 #define CHAR_MAX 127

The maximum value of the char type

6.199.2.3 #define CHAR_MIN -127

The minimum value of the char type

6.199.2.4 #define INT_MAX 32767

The maximum value of the int type

6.199.2.5 #define INT_MIN -32767

The minimum value of the int type

6.199.2.6 #define LONG_MAX 2147483647

The maximum value of the long type

6.199.2.7 #define LONG_MIN -2147483647

The minimum value of the long type

6.199.2.8 #define RAND_MAX 32768

The maximum unsigned int random number returned by rand

6.199.2.9 #define SCHAR_MAX 127

The maximum value of the signed char type

6.199.2.10 #define SCHAR_MIN -127

The minimum value of the signed char type

6.199.2.11 #define SHRT_MAX 32767

The maximum value of the short type

6.199.2.12 #define SHRT_MIN -32767

The minimum value of the short type

6.199.2.13 #define UCHAR_MAX 255

The maximum value of the unsigned char type

6.199.2.14 #define UINT_MAX 65535

The maximum value of the unsigned int type

6.199.2.15 #define ULONG_MAX 4294967295

The maximum value of the unsigned long type

6.199.2.16 #define USHRT_MAX 65535

The maximum value of the unsigned short type

6.200 Graphics library begin modes

Constants that are used to specify the polygon surface begin mode.

Defines

- #define **GL_POLYGON** 1
- #define **GL_LINE** 2
- #define **GL_POINT** 3
- #define **GL_CIRCLE** 4

6.200.1 Detailed Description

Constants that are used to specify the polygon surface begin mode.

6.200.2 Define Documentation**6.200.2.1 #define GL_CIRCLE 4**

Use circle mode.

Examples:

[glCircleDemo.nxc](#).

6.200.2.2 #define GL_LINE 2

Use line mode.

6.200.2.3 #define GL_POINT 3

Use point mode.

6.200.2.4 #define GL_POLYGON 1

Use polygon mode.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

6.201 Graphics library actions

Constants that are used to specify a graphics library action.

Defines

- #define `GL_TRANSLATE_X` 1
- #define `GL_TRANSLATE_Y` 2
- #define `GL_TRANSLATE_Z` 3
- #define `GL_ROTATE_X` 4
- #define `GL_ROTATE_Y` 5
- #define `GL_ROTATE_Z` 6
- #define `GL_SCALE_X` 7
- #define `GL_SCALE_Y` 8
- #define `GL_SCALE_Z` 9

6.201.1 Detailed Description

Constants that are used to specify a graphics library action.

6.201.2 Define Documentation**6.201.2.1 #define GL_ROTATE_X 4**

Rotate around the X axis.

Examples:

[glRotateDemo.nxc](#).

6.201.2.2 #define GL_ROTATE_Y 5

Rotate around the Y axis.

Examples:

[glRotateDemo.nxc](#).

6.201.2.3 #define GL_ROTATE_Z 6

Rotate around the Z axis.

6.201.2.4 #define GL_SCALE_X 7

Scale along the X axis.

Examples:

[glScaleDemo.nxc](#).

6.201.2.5 #define GL_SCALE_Y 8

Scale along the Y axis.

6.201.2.6 #define GL_SCALE_Z 9

Scale along the Z axis.

6.201.2.7 #define GL_TRANSLATE_X 1

Translate along the X axis.

Examples:

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

6.201.2.8 #define GL_TRANSLATE_Y 2

Translate along the Y axis.

Examples:

[glTranslateDemo.nxc](#).

6.201.2.9 #define GL_TRANSLATE_Z 3

Translate along the Z axis.

Examples:

[glTranslateDemo.nxc](#).

6.202 Graphics library settings

Constants that are used to configure the graphics library settings.

Defines

- #define `GL_CIRCLE_SIZE` 1
- #define `GL_CULL_MODE` 2
- #define `GL_CAMERA_DEPTH` 3
- #define `GL_ZOOM_FACTOR` 4

6.202.1 Detailed Description

Constants that are used to configure the graphics library settings.

6.202.2 Define Documentation

6.202.2.1 #define `GL_CAMERA_DEPTH` 3

Set the camera depth.

6.202.2.2 #define `GL_CIRCLE_SIZE` 1

Set the circle size.

6.202.2.3 #define `GL_CULL_MODE` 2

Set the cull mode.

Examples:

`glCircleDemo.nxc`, and `glTranslateDemo.nxc`.

6.202.2.4 #define `GL_ZOOM_FACTOR` 4

Set the zoom factor.

6.203 Graphics library cull mode

Constants to use when setting the graphics library cull mode.

Defines

- #define **GL_CULL_BACK** 2
- #define **GL_CULL_FRONT** 3
- #define **GL_CULL_NONE** 4

6.203.1 Detailed Description

Constants to use when setting the graphics library cull mode.

6.203.2 Define Documentation**6.203.2.1 #define GL_CULL_BACK 2**

Cull lines in back.

6.203.2.2 #define GL_CULL_FRONT 3

Cull lines in front.

6.203.2.3 #define GL_CULL_NONE 4

Do not cull any lines.

Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

7 Data Structure Documentation

7.1 ColorSensorReadType Struct Reference

Parameters for the ColorSensorRead system call.

```
#include <NXCDefs.h>
```

Data Fields

- char **Result**
- byte **Port**
- int **ColorValue**

- `unsigned int RawArray []`
- `unsigned int NormalizedArray []`
- `int ScaledArray []`
- `bool Invalid`

7.1.1 Detailed Description

Parameters for the ColorSensorRead system call. This structure is used when calling the [SysColorSensorRead](#) system call function. Choose the sensor port ([Input port constants](#)) and after calling the function read the sensor values from the `ColorValue` field or the raw, normalized, or scaled value arrays.

See also:

[SysColorSensorRead\(\)](#)

Examples:

[ex_SysColorSensorRead.nxc](#).

7.1.2 Field Documentation

7.1.2.1 `int ColorSensorReadType::ColorValue`

The color value returned by the sensor. See the [Color values](#) group.

Examples:

[ex_SysColorSensorRead.nxc](#).

7.1.2.2 `bool ColorSensorReadType::Invalid`

Are the sensor values valid?

7.1.2.3 `unsigned int ColorSensorReadType::NormalizedArray[]`

Normalized color values returned by the sensor. See the [Color sensor array indices](#) group.

7.1.2.4 `byte ColorSensorReadType::Port`

The sensor port. See the constants in the [Input port constants](#) group.

Examples:[ex_SysColorSensorRead.nxc](#).**7.1.2.5 unsigned int ColorSensorReadType::RawArray[]**

Raw color values returned by the sensor. See the [Color sensor array indices](#) group.

7.1.2.6 char ColorSensorReadType::Result

The function call result. [NO_ERR](#) means it succeeded.

Examples:[ex_SysColorSensorRead.nxc](#).**7.1.2.7 int ColorSensorReadType::ScaledArray[]**

Scaled color values returned by the sensor. See the [Color sensor array indices](#) group.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.2 CommBTCheckStatusType Struct Reference

Parameters for the CommBTCheckStatus system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Connection](#)

7.2.1 Detailed Description

Parameters for the CommBTCheckStatus system call. This structure is used when calling the [SysCommBTCheckStatus](#) system call function.

See also:[SysCommBTCheckStatus\(\)](#)

Examples:

[ex_syscommbtcheckstatus.nxc](#).

7.2.2 Field Documentation**7.2.2.1 byte CommBTCheckStatusType::Connection**

The connection to check.

Examples:

[ex_syscommbtcheckstatus.nxc](#).

7.2.2.2 char CommBTCheckStatusType::Result

The function call result. Possible values include [ERR_-INVALID_PORT](#), [STAT_COMM_PENDING](#), [ERR_COMM_CHAN_NOT_READY](#), and [LDR_SUCCESS](#).

Examples:

[ex_syscommbtcheckstatus.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.3 CommBTConnectionType Struct Reference

Parameters for the CommBTConnection system call.

```
#include <NXCDefs.h>
```

Data Fields

- [unsigned int Result](#)
- [byte Action](#)
- [string Name](#)
- [byte ConnectionSlot](#)

7.3.1 Detailed Description

Parameters for the CommBTConnection system call. This structure is used when calling the [SysCommBTConnection](#) system call function.

See also:

[SysCommBTConnection\(\)](#)

Examples:

[ex_SysCommBTConnection.nxc](#).

7.3.2 Field Documentation

7.3.2.1 byte CommBTConnectionType::Action

The connection action (connect or disconnect).

Examples:

[ex_SysCommBTConnection.nxc](#).

7.3.2.2 byte CommBTConnectionType::ConnectionSlot

The connection slot to connect or disconnect.

Examples:

[ex_SysCommBTConnection.nxc](#).

7.3.2.3 string CommBTConnectionType::Name

The name of the device to connect or disconnect.

Examples:

[ex_SysCommBTConnection.nxc](#).

7.3.2.4 unsigned int CommBTConnectionType::Result

The function call result.

Examples:

[ex_SysCommBTConnection.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.4 CommBTOnOffType Struct Reference

Parameters for the CommBTOnOff system call.

```
#include <NXCDefs.h>
```

Data Fields

- `unsigned int Result`
- `bool PowerState`

7.4.1 Detailed Description

Parameters for the CommBTOnOff system call. This structure is used when calling the [SysCommBTOnOff](#) system call function.

See also:

[SysCommBTOnOff\(\)](#)

Examples:

[ex_SysCommBTOnOff.nxc](#).

7.4.2 Field Documentation

7.4.2.1 bool CommBTOnOffType::PowerState

If true then turn on bluetooth, otherwise, turn it off.

Examples:

[ex_SysCommBTOnOff.nxc](#).

7.4.2.2 unsigned int CommBTOffType::Result

The function call result.

Examples:

[ex_SysCommBTOff.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.5 CommBTWriteType Struct Reference

Parameters for the CommBTWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Connection](#)
- byte [Buffer](#) []

7.5.1 Detailed Description

Parameters for the CommBTWrite system call. This structure is used when calling the [SysCommBTWrite](#) system call function.

See also:

[SysCommBTWrite\(\)](#)

Examples:

[ex_syscommbtwrite.nxc](#).

7.5.2 Field Documentation

7.5.2.1 byte CommBTWriteType::Buffer[]

The data to write to the connection.

Examples:

[ex_syscommbtwrite.nxc](#).

7.5.2.2 byte CommBTWriteType::Connection

The connection to use.

Examples:

[ex_syscommbtwrite.nxc](#).

7.5.2.3 char CommBTWriteType::Result

The function call result. Possible values include [ERR_COMM_CHAN_NOT_READY](#) and [STAT_COMM_PENDING](#) (write accepted).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.6 CommExecuteFunctionType Struct Reference

Parameters for the CommExecuteFunction system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int [Result](#)
- byte [Cmd](#)
- byte [Param1](#)
- byte [Param2](#)
- byte [Param3](#)
- string [Name](#)
- unsigned int [RetVal](#)

7.6.1 Detailed Description

Parameters for the CommExecuteFunction system call. This structure is used when calling the [SysCommExecuteFunction](#) system call function.

The fields usage depends on the requested command and are documented in the table below. If a field member is shown as 'x' it is ignored by the specified command.

Cmd	Meaning	(Param1,Param2,Param3,Name)
INTF_SENDFILE	Send a file over a Bluetooth connection	(Connection,x,x,Filename)
INTF_SEARCH	Search for Bluetooth devices	(x,x,x,x)
INTF_STOPSEARCH	Stop searching for Bluetooth devices	(x,x,x,x)
INTF_CONNECT	Connect to a Bluetooth device	(DeviceIndex,Connection,x,x)
INTF_DISCONNECT	Disconnect a Bluetooth device	(Connection,x,x,x)
INTF_DISCONNECTALL	Disconnect all Bluetooth devices	(x,x,x,x)
INTF_REMOVEDEVICE	Remove device from My Contacts	(DeviceIndex,x,x,x)
INTF_VISIBILITY	Set Bluetooth visibility	(true/false,x,x,x)
INTF_SETCMDMODE	Set command mode	(x,x,x,x)
INTF_OPENSTREAM	Open a stream	(x,Connection,x,x)
INTF_SENDDATA	Send data	(Length, Connection, WaitForIt, Buffer)
INTF_FACTORYRESET	Bluetooth factory reset	(x,x,x,x)
INTF_BTON	Turn Bluetooth on	(x,x,x,x)
INTF_BTOFF	Turn Bluetooth off	(x,x,x,x)
INTF_SETBTNAME	Set Bluetooth name	(x,x,x,x)
INTF_EXTREAD	Handle external? read	(x,x,x,x)
INTF_PINREQ	Handle Blueooth PIN request	(x,x,x,x)
INTF_CONNECTREQ	Handle Bluetooth connect request	(x,x,x,x)

See also:

[SysCommExecuteFunction\(\)](#)

Examples:

[ex_syscommexecufunction.nxc](#).

7.6.2 Field Documentation

7.6.2.1 byte CommExecuteFunctionType::Cmd

The command to execute.

Examples:

[ex_syscommexecutefunction.nxc](#).

7.6.2.2 string CommExecuteFunctionType::Name

The name parameter, see table.

7.6.2.3 byte CommExecuteFunctionType::Param1

The first parameter, see table.

7.6.2.4 byte CommExecuteFunctionType::Param2

The second parameter, see table.

7.6.2.5 byte CommExecuteFunctionType::Param3

The third parameter, see table.

7.6.2.6 unsigned int CommExecuteFunctionType::Result

The function call result. Possible values include [Loader module error codes](#).

7.6.2.7 unsigned int CommExecuteFunctionType::RetVal

The function call return value. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.7 CommHSCheckStatusType Struct Reference

Parameters for the CommHSCheckStatus system call.

```
#include <NXCDefs.h>
```

Data Fields

- bool [SendingData](#)
- bool [DataAvailable](#)

7.7.1 Detailed Description

Parameters for the CommHSCheckStatus system call. This structure is used when calling the [SysCommHSCheckStatus](#) system call function.

See also:

[SysCommHSCheckStatus\(\)](#)

Examples:

[ex_SysCommHSCheckStatus.nxc](#).

7.7.2 Field Documentation

7.7.2.1 bool CommHSCheckStatusType::DataAvailable

Is data available for reading?

Examples:

[ex_SysCommHSCheckStatus.nxc](#).

7.7.2.2 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](#)

7.8 CommHSControlType Struct Reference

Parameters for the CommHSControl system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)

- byte [Command](#)
- byte [BaudRate](#)
- unsigned int [Mode](#)

7.8.1 Detailed Description

Parameters for the CommHSControl system call. This structure is used when calling the [SysCommHSControl](#) system call function.

See also:

[SysCommHSControl\(\)](#)

Examples:

[ex_SysCommHSControl.nxc](#).

7.8.2 Field Documentation

7.8.2.1 byte CommHSControlType::BaudRate

The hi-speed port baud rate. See [Hi-speed port baud rate constants](#).

7.8.2.2 byte CommHSControlType::Command

The hi-speed port configuration command. See [Hi-speed port SysCommHSControl constants](#).

Examples:

[ex_SysCommHSControl.nxc](#).

7.8.2.3 unsigned int CommHSControlType::Mode

The hi-speed port mode. See [Hi-speed port data bits constants](#), [Hi-speed port stop bits constants](#), [Hi-speed port parity constants](#), and [Hi-speed port combined UART constants](#).

7.8.2.4 char CommHSControlType::Result

The function call result.

Todo

values?

Examples:

[ex_SysCommHSControl.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.9 CommHSReadWriteType Struct Reference

Parameters for the CommHSReadWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Status](#)
- byte [Buffer](#) []

7.9.1 Detailed Description

Parameters for the CommHSReadWrite system call. This structure is used when calling the [SysCommHSRead](#) and [SysCommHSWrite](#) system call functions.

See also:

[SysCommHSRead\(\)](#), [SysCommHSWrite\(\)](#)

Examples:

[ex_SysCommHSRead.nxc](#), and [ex_SysCommHSWrite.nxc](#).

7.9.2 Field Documentation

7.9.2.1 byte CommHSReadWriteType::Buffer[]

The buffer of data to write or to contain the data read from the hi-speed port.

Examples:

[ex_SysCommHSRead.nxc](#), and [ex_SysCommHSWrite.nxc](#).

7.9.2.2 char CommHSReadWriteType::Status

The result of the function call.

Examples:

[ex_SysCommHSRead.nxc](#), and [ex_SysCommHSWrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.10 CommLSCheckStatusType Struct Reference

Parameters for the CommLSCheckStatus system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Port](#)
- byte [BytesReady](#)

7.10.1 Detailed Description

Parameters for the CommLSCheckStatus system call. This structure is used when calling the [SysCommLSCheckStatus](#) system call function.

See also:

[SysCommLSCheckStatus\(\)](#)

Examples:

[ex_syscommlscheckstatus.nxc](#).

7.10.2 Field Documentation

7.10.2.1 byte CommLSCheckStatusType::BytesReady

The number of bytes ready to read from the specified port.

7.10.2.2 byte CommLSCheckStatusType::Port

The port to which the I2C device is connected.

Examples:

[ex_syscommlscheckstatus.nxc](#).

7.10.2.3 char CommLSCheckStatusType::Result

The function call result.

Possible values include [ERR_COMM_BUS_ERR](#), [ERR_COMM_CHAN_INVALID](#), [ERR_COMM_CHAN_NOT_READY](#), [STAT_COMM_PENDING](#), and [NO_ERR](#).

Examples:

[ex_syscommlscheckstatus.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.11 CommLSReadType Struct Reference

Parameters for the CommLSRead system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [BufferLen](#)

7.11.1 Detailed Description

Parameters for the CommLSRead system call. This structure is used when calling the [SysCommLSRead](#) system call function.

See also:

[SysCommLSRead\(\)](#)

Examples:[ex_syscommlsread.nxc](#).**7.11.2 Field Documentation****7.11.2.1 byte CommLSReadType::Buffer[]**

The buffer used to store the bytes read from the I2C device.

Examples:[ex_syscommlsread.nxc](#).**7.11.2.2 byte CommLSReadType::BufferLen**

The size of the output buffer on input. This field is not updated during the function call.

Examples:[ex_syscommlsread.nxc](#).**7.11.2.3 byte CommLSReadType::Port**

The port to which the I2C device is connected.

Examples:[ex_syscommlsread.nxc](#).**7.11.2.4 char CommLSReadType::Result**

The function call result. Possible values include [ERR_COMM_BUS_ERR](#), [ERR_COMM_CHAN_INVALID](#), [ERR_COMM_CHAN_NOT_READY](#), [ERR_INVALID_SIZE](#), [STAT_COMM_PENDING](#), and [NO_ERR](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.12 CommLSSWriteExType Struct Reference

Parameters for the CommLSSWriteEx system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)
- bool [NoRestartOnRead](#)

7.12.1 Detailed Description

Parameters for the CommLSSWriteEx system call. This structure is used when calling the [SysCommLSSWriteEx](#) system call function.

See also:

[SysCommLSSWriteEx\(\)](#)

Examples:

[ex_syscommlsswriteex.nxc](#).

7.12.2 Field Documentation

7.12.2.1 byte CommLSSWriteExType::Buffer[]

The buffer written to the I2C device.

Examples:

[ex_syscommlsswriteex.nxc](#).

7.12.2.2 bool CommLSSWriteExType::NoRestartOnRead

Should a restart occur before reading from the device?

Examples:

[ex_syscommlsswriteex.nxc](#).

7.12.2.3 byte CommLSSWriteExType::Port

The port to which the I2C device is connected.

Examples:

[ex_syscommlsswriteex.nxc](#).

7.12.2.4 char CommLSSWriteExType::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_syscommlsswriteex.nxc](#).

7.12.2.5 byte CommLSSWriteExType::ReturnLen

The number of bytes that you want to read from the I2C device.

Examples:

[ex_syscommlsswriteex.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.13 CommLSSWriteType Struct Reference

Parameters for the CommLSSWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Port](#)
- byte [Buffer](#) []
- byte [ReturnLen](#)

7.13.1 Detailed Description

Parameters for the CommLSWrite system call. This structure is used when calling the [SysCommLSWrite](#) system call function.

See also:

[SysCommLSWrite\(\)](#)

Examples:

[ex_syscommlswrite.nxc](#).

7.13.2 Field Documentation

7.13.2.1 byte CommLSWriteType::Buffer[]

The buffer containing data to be written to the I2C device.

Examples:

[ex_syscommlswrite.nxc](#).

7.13.2.2 byte CommLSWriteType::Port

The port to which the I2C device is connected.

Examples:

[ex_syscommlswrite.nxc](#).

7.13.2.3 char CommLSWriteType::Result

The function call result. Possible values include [ERR_COMM_CHAN_INVALID](#), [ERR_COMM_CHAN_NOT_READY](#), [ERR_INVALID_SIZE](#), and [NO_ERR](#).

7.13.2.4 byte CommLSWriteType::ReturnLen

The number of bytes that you want to read from the I2C device after writing the data. If no read is planned set this to zero.

Examples:

[ex_syscommlswrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.14 ComputeCalibValueType Struct Reference

Parameters for the ComputeCalibValue system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [Result](#)
- string [Name](#)
- unsigned int [RawVal](#)

7.14.1 Detailed Description

Parameters for the ComputeCalibValue system call. This structure is used when calling the [SysComputeCalibValue](#) system call function.

See also:

[SysComputeCalibValue\(\)](#)

Examples:

[ex_SysComputeCalibValue.nxc](#).

7.14.2 Field Documentation

7.14.2.1 string ComputeCalibValueType::Name

The name of the sensor calibration cache.

Todo

?

Examples:

[ex_SysComputeCalibValue.nxc](#).

7.14.2.2 unsigned int ComputeCalibValueType::RawVal

The raw value.

Todo

?.

Examples:

[ex_SysComputeCalibValue.nxc](#).

7.14.2.3 byte ComputeCalibValueType::Result

The function call result.

Todo

?.

Examples:

[ex_SysComputeCalibValue.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.15 DatalogGetTimesType Struct Reference

Parameters for the DatalogGetTimes system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned long [SyncTime](#)
- unsigned long [SyncTick](#)

7.15.1 Detailed Description

Parameters for the DatalogGetTimes system call. This structure is used when calling the [SysDatalogGetTimes](#) system call function.

See also:

[SysDatalogGetTimes\(\)](#)

Examples:

[ex_SysDatalogGetTimes.nxc.](#)

7.15.2 Field Documentation

7.15.2.1 unsigned long DatalogGetTimesType::SyncTick

The datalog synchronized tick.

Examples:

[ex_SysDatalogGetTimes.nxc.](#)

7.15.2.2 unsigned long DatalogGetTimesType::SyncTime

The datalog synchronized time.

Examples:

[ex_SysDatalogGetTimes.nxc.](#)

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.16 DatalogWriteType Struct Reference

Parameters for the DatalogWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Message](#) []

7.16.1 Detailed Description

Parameters for the DatalogWrite system call. This structure is used when calling the [SysDatalogWrite](#) system call function.

See also:

[SysDatalogWrite\(\)](#)

Examples:

[ex_SysDatalogWrite.nxc](#).

7.16.2 Field Documentation

7.16.2.1 byte DatalogWriteType::Message[]

A buffer containing data to write to the datalog.

Examples:

[ex_SysDatalogWrite.nxc](#).

7.16.2.2 char DatalogWriteType::Result

The function call result. [NO_ERR](#) means it succeeded.

Examples:

[ex_SysDatalogWrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.17 DisplayExecuteFunctionType Struct Reference

Parameters for the DisplayExecuteFunction system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [Status](#)

- byte [Cmd](#)
- bool [On](#)
- byte [X1](#)
- byte [Y1](#)
- byte [X2](#)
- byte [Y2](#)

7.17.1 Detailed Description

Parameters for the `DisplayExecuteFunction` system call. This structure is used when calling the [SysDisplayExecuteFunction](#) system call function.

The fields usage depends on the requested command and are documented in the table below. If a field member is shown as 'x' it is ignored by the specified display command.

Cmd	Meaning	Expected parameters
DISPLAY_ERASE_ALL	erase entire screen	()
DISPLAY_PIXEL	set pixel (on/off)	(true/false,X1,Y1,x,x)
DISPLAY_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](#).

7.17.2 Field Documentation

7.17.2.1 byte DisplayExecuteFunctionType::Cmd

The command to execute.

Examples:

[ex_dispfunc.nxc](#), and [ex_sysdisplayexecutefunction.nxc](#).

7.17.2.2 bool DisplayExecuteFunctionType::On

The On parameter, see table.

Examples:

[ex_dispfunc.nxc](#).

7.17.2.3 byte DisplayExecuteFunctionType::Status

The function call result, always [NO_ERR](#).

7.17.2.4 byte DisplayExecuteFunctionType::X1

The X1 parameter, see table.

Examples:

[ex_dispfunc.nxc](#).

7.17.2.5 byte DisplayExecuteFunctionType::X2

The X2 parameter, see table.

Examples:

[ex_dispfunc.nxc](#).

7.17.2.6 byte DisplayExecuteFunctionType::Y1

The Y1 parameter, see table.

Examples:

[ex_dispfunc.nxc](#).

7.17.2.7 byte DisplayExecuteFunctionType::Y2

The Y2 parameter, see table.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.18 div_t Struct Reference

Output type of the div function.

```
#include <NXCDefs.h>
```

Data Fields

- int quot
- int rem

7.18.1 Detailed Description

Output type of the div function. [div_t](#) structure. Structure used to represent the value of an integral division performed by div. It has two members of the same type, defined in either order as: int quot; int rem;.

See also:

[div\(\)](#)

Examples:

[ex_div.nxc](#).

7.18.2 Field Documentation

7.18.2.1 int div_t::quot

Represents the quotient of the integral division operation performed by div, which is the integer of lesser magnitude that is nearest to the algebraic quotient.

Examples:

[ex_div.nxc](#).

7.18.2.2 int div_t::rem

Represents the remainder of the integral division operation performed by div, which is the integer resulting from subtracting quot to the numerator of the operation.

Examples:

[ex_div.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.19 DrawCircleType Struct Reference

Parameters for the DrawCircle system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType Center](#)
- byte [Size](#)
- unsigned long [Options](#)

7.19.1 Detailed Description

Parameters for the DrawCircle system call. This structure is used when calling the [SysDrawCircle](#) system call function. It lets you specify the center of the circle to draw using the [LocationType](#) structure member, the radius, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawCircle\(\)](#)

Examples:

[ex_sysdrawcircle.nxc](#).

7.19.2 Field Documentation

7.19.2.1 LocationType DrawCircleType::Center

The location of the circle center.

Examples:

[ex_sysdrawcircle.nxc](#).

7.19.2.2 unsigned long DrawCircleType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawcircle.nxc](#).

7.19.2.3 char DrawCircleType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.19.2.4 byte DrawCircleType::Size

The circle radius.

Examples:

[ex_sysdrawcircle.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.20 DrawEllipseType Struct Reference

Parameters for the DrawEllipse system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) [Center](#)
- byte [SizeX](#)
- byte [SizeY](#)
- unsigned long [Options](#)

7.20.1 Detailed Description

Parameters for the DrawEllipse system call. This structure is used when calling the [SysDrawEllipse](#) system call function. It lets you specify the center of the ellipse using the [LocationType](#) structure member, the x and y axis radii, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawEllipse\(\)](#)

Examples:

[ex_sysdrawellipse.nxc.](#)

7.20.2 Field Documentation

7.20.2.1 LocationType DrawEllipseType::Center

The location of the ellipse center.

Examples:

[ex_sysdrawellipse.nxc.](#)

7.20.2.2 unsigned long DrawEllipseType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawellipse.nxc.](#)

7.20.2.3 char DrawEllipseType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.20.2.4 byte DrawEllipseType::SizeX

The horizontal ellipse radius.

Examples:

[ex_sysdrawellipse.nxc.](#)

7.20.2.5 byte DrawEllipseType::SizeY

The vertical ellipse radius.

Examples:

[ex_sysdrawellipse.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.21 DrawFontType Struct Reference

Parameters for the DrawFont system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- string [Filename](#)
- string [Text](#)
- unsigned long [Options](#)

7.21.1 Detailed Description

Parameters for the DrawFont system call. This structure is used when calling the [Sys-DrawFont](#) system call function. It lets you specify the text to draw, the LCD line and horizontal position using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawFont\(\)](#)

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

7.21.2 Field Documentation

7.21.2.1 string DrawFontType::Filename

The filename of the RIC-based font file.

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

7.21.2.2 LocationType DrawFontType::Location

The location in X, LCD line number coordinates.

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

7.21.2.3 unsigned long DrawFontType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

7.21.2.4 char DrawFontType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.21.2.5 string DrawFontType::Text

The text to draw on the LCD.

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.22 DrawGraphicArrayType Struct Reference

Parameters for the DrawGraphicArray system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType Location](#)
- byte [Data](#) []
- long [Variables](#) []
- unsigned long [Options](#)

7.22.1 Detailed Description

Parameters for the DrawGraphicArray system call. This structure is used when calling the [SysDrawGraphicArray](#) system call function. It lets you specify the screen location at which to draw the image using the [LocationType](#) structure member, the graphic image data array, the image parameters (if needed), as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawGraphicArray\(\)](#)

Examples:

[ex_dispgout.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

7.22.2 Field Documentation

7.22.2.1 byte DrawGraphicArrayType::Data[]

A byte array containing the RIC opcodes. [RIC Macro Wrappers](#)

Examples:

[ex_dispgout.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

7.22.2.2 LocationType DrawGraphicArrayType::Location

The location on screen.

Examples:

[ex_dispgout.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

7.22.2.3 unsigned long DrawGraphicArrayType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_dispgout.nxc](#).

7.22.2.4 char DrawGraphicArrayType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.22.2.5 long DrawGraphicArrayType::Variables[]

The variables passed as RIC arguments.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.23 DrawGraphicType Struct Reference

Parameters for the DrawGraphic system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- string [Filename](#)
- long [Variables](#) []
- unsigned long [Options](#)

7.23.1 Detailed Description

Parameters for the DrawGraphic system call. This structure is used when calling the [SysDrawGraphic](#) system call function. It lets you specify the screen location at which to draw the image using the [LocationType](#) structure member, the filename of the graphic image, the image parameters (if needed), as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawGraphic\(\)](#)

Examples:

[ex_sysdrawgraphic.nxc](#).

7.23.2 Field Documentation

7.23.2.1 string DrawGraphicType::Filename

The RIC file name.

Examples:[ex_sysdrawgraphic.nxc.](#)**7.23.2.2 LocationType DrawGraphicType::Location**

The location on screen.

Examples:[ex_sysdrawgraphic.nxc.](#)**7.23.2.3 unsigned long DrawGraphicType::Options**

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:[ex_sysdrawgraphic.nxc.](#)**7.23.2.4 char DrawGraphicType::Result**

The function call result. Possible values include [Loader module error codes](#), [ERR_FILE](#), and [NO_ERR](#).

7.23.2.5 long DrawGraphicType::Variables[]

The variables passed as RIC arguments.

Examples:[ex_sysdrawgraphic.nxc.](#)

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.24 DrawLineType Struct Reference

Parameters for the DrawLine system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) StartLoc
- [LocationType](#) EndLoc
- unsigned long [Options](#)

7.24.1 Detailed Description

Parameters for the DrawLine system call. This structure is used when calling the [Sys-DrawLine](#) system call function. It lets you specify the end points of the line to draw using two [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawLine\(\)](#)

Examples:

[ex_sysdrawline.nxc.](#)

7.24.2 Field Documentation**7.24.2.1 LocationType DrawLineType::EndLoc**

The location of the ending point.

Examples:

[ex_sysdrawline.nxc.](#)

7.24.2.2 unsigned long DrawLineType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawline.nxc.](#)

7.24.2.3 char DrawLineType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.24.2.4 LocationType DrawLineType::StartLoc

The location of the starting point.

Examples:

[ex_sysdrawline.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.25 DrawPointType Struct Reference

Parameters for the DrawPoint system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- LocationType [Location](#)
- unsigned long [Options](#)

7.25.1 Detailed Description

Parameters for the DrawPoint system call. This structure is used when calling the [SysDrawPoint](#) system call function. It lets you specify the pixel to draw using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawPoint\(\)](#)

Examples:

[ex_sysdrawpoint.nxc](#).

7.25.2 Field Documentation

7.25.2.1 LocationType DrawPointType::Location

The point location on screen.

Examples:

[ex_sysdrawpoint.nxc](#).

7.25.2.2 unsigned long DrawPointType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawpoint.nxc](#).

7.25.2.3 char DrawPointType::Result

The function call result. [NO_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.26 DrawPolygonType Struct Reference

Parameters for the DrawPolygon system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType Points](#) []
- unsigned long [Options](#)

7.26.1 Detailed Description

Parameters for the DrawPolygon system call. This structure is used when calling the [SysDrawPolygon](#) system call function. It lets you specify the points of the polygon to draw using the [LocationType](#) array structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawPolygon\(\)](#)

Examples:

[ex_sysdrawpolygon.nxc](#).

7.26.2 Field Documentation**7.26.2.1 unsigned long DrawPolygonType::Options**

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawpolygon.nxc](#).

7.26.2.2 LocationType DrawPolygonType::Points[]

An array of [LocationType](#) structures which define the polygon's shape.

Examples:

[ex_sysdrawpolygon.nxc](#).

7.26.2.3 char DrawPolygonType::Result

The function call result. [NO_ERR](#) means it succeeded.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.27 DrawRectType Struct Reference

Parameters for the DrawRect system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- [SizeType](#) [Size](#)
- unsigned long [Options](#)

7.27.1 Detailed Description

Parameters for the DrawRect system call. This structure is used when calling the [SysDrawRect](#) system call function. It lets you specify the corner of the rectangle using the [LocationType](#) structure member, the width and height of the rectangle using the [SizeType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawRect\(\)](#)

Examples:

[ex_sysdrawrect.nxc](#).

7.27.2 Field Documentation

7.27.2.1 LocationType DrawRectType::Location

The top left corner location.

Examples:

[ex_sysdrawrect.nxc](#).

7.27.2.2 unsigned long DrawRectType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawrect.nxc](#).

7.27.2.3 char DrawRectType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.27.2.4 SizeType DrawRectType::Size

The width and height of the rectangle.

Examples:

[ex_sysdrawrect.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.28 DrawTextType Struct Reference

Parameters for the DrawText system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- [LocationType](#) [Location](#)
- string [Text](#)
- unsigned long [Options](#)

7.28.1 Detailed Description

Parameters for the DrawText system call. This structure is used when calling the [Sys-DrawText](#) system call function. It lets you specify the text to draw, the LCD line and horizontal position using the [LocationType](#) structure member, as well as drawing options defined in the [Drawing option constants](#) group.

See also:

[SysDrawText\(\)](#)

Examples:

[ex_syscall.nxc](#), and [ex_sysdrawtext.nxc](#).

7.28.2 Field Documentation

7.28.2.1 LocationType DrawTextType::Location

The location in X, LCD line number coordinates.

Examples:

[ex_syscall.nxc](#), and [ex_sysdrawtext.nxc](#).

7.28.2.2 unsigned long DrawTextType::Options

The options to use when writing to the LCD. [Drawing option constants](#)

Examples:

[ex_sysdrawtext.nxc](#).

7.28.2.3 char DrawTextType::Result

The function call result. [NO_ERR](#) means it succeeded.

7.28.2.4 string DrawTextType::Text

The text to draw on the LCD.

Examples:

[ex_syscall.nxc](#), and [ex_sysdrawtext.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.29 FileCloseType Struct Reference

Parameters for the FileClose system call.

```
#include <NXCDefs.h>
```

Data Fields

- [unsigned int Result](#)
- [byte FileHandle](#)

7.29.1 Detailed Description

Parameters for the FileClose system call. This structure is used when calling the [Sys-FileClose](#) system call function.

See also:

[SysFileClose\(\)](#)

Examples:[ex_sysfileclose.nxc.](#)**7.29.2 Field Documentation****7.29.2.1 byte FileCloseType::FileHandle**

The file handle to close.

Examples:[ex_sysfileclose.nxc.](#)**7.29.2.2 unsigned int FileCloseType::Result**

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.30 FileDeleteType Struct Reference

Parameters for the FileDelete system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int **Result**
- string [Filename](#)

7.30.1 Detailed Description

Parameters for the FileDelete system call. This structure is used when calling the [SysFileDelete](#) system call function.

See also:[SysFileDelete\(\)](#)**Examples:**[ex_sysfiledelete.nxc.](#)

7.30.2 Field Documentation

7.30.2.1 string FileDeleteType::Filename

The name of the file to delete.

Examples:

[ex_sysfiledelete.nxc](#).

7.30.2.2 unsigned int FileDeleteType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.31 FileFindType Struct Reference

Parameters for the FileFind system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int **Result**
- byte **FileHandle**
- string **Filename**
- unsigned long **Length**

7.31.1 Detailed Description

Parameters for the FileFind system call. This structure is used when calling the [SysFileFindFirst](#) and [SysFileFindNext](#) system call functions.

See also:

[SysFileFindFirst\(\)](#) and [SysFileFindNext\(\)](#)

Examples:

[ex_sysfilefindfirst.nxc](#), and [ex_sysfilefindnext.nxc](#).

7.31.2 Field Documentation

7.31.2.1 byte FileFindType::FileHandle

The returned file handle to be used to continue iterations. Close it after usage.

Examples:

[ex_sysfilefindnext.nxc](#).

7.31.2.2 string FileFindType::Filename

The pattern to match file name, then the returned found file name.

Examples:

[ex_sysfilefindfirst.nxc](#), and [ex_sysfilefindnext.nxc](#).

7.31.2.3 unsigned long FileFindType::Length

The found file length.

7.31.2.4 unsigned int FileFindType::Result

The function call result. Possible values include [Loader module error codes](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.32 FileOpenType Struct Reference

Parameters for the FileOpen system call.

```
#include <NXCDefs.h>
```

Data Fields

- [unsigned int Result](#)
- [byte FileHandle](#)
- [string Filename](#)
- [unsigned long Length](#)

7.32.1 Detailed Description

Parameters for the FileOpen system call. This structure is used when calling the [SysFileOpenAppend](#), [SysFileOpenRead](#), [SysFileOpenWrite](#), [SysFileOpenReadLinear](#), [SysFileOpenWriteLinear](#) and [SysFileOpenWriteNonLinear](#) system call functions.

See also:

[SysFileOpenAppend\(\)](#), [SysFileOpenRead\(\)](#), [SysFileOpenWrite\(\)](#), [SysFileOpenReadLinear\(\)](#), [SysFileOpenWriteLinear\(\)](#)

Examples:

[ex_sysfileopenappend.nxc](#), [ex_sysfileopenread.nxc](#), [ex_sysfileopenreadlinear.nxc](#),
[ex_sysfileopenwrite.nxc](#), [ex_sysfileopenwritelinear.nxc](#), and [ex_sysfileopenwritenonlinear.nxc](#).

7.32.2 Field Documentation

7.32.2.1 byte FileOpenType::FileHandle

The returned file handle to use for subsequent file operations.

7.32.2.2 string FileOpenType::Filename

The name of the file to open or create.

Examples:

[ex_sysfileopenappend.nxc](#), [ex_sysfileopenread.nxc](#), [ex_sysfileopenreadlinear.nxc](#),
[ex_sysfileopenwrite.nxc](#), [ex_sysfileopenwritelinear.nxc](#), and [ex_sysfileopenwritenonlinear.nxc](#).

7.32.2.3 unsigned long FileOpenType::Length

For [SysFileOpenWrite\(\)](#), [SysFileOpenWriteLinear\(\)](#) and [SysFileOpenWriteNonLinear\(\)](#): the desired maximum file capacity.

For [SysFileOpenAppend\(\)](#), [SysFileOpenRead\(\)](#) and [SysFileOpenReadLinear\(\)](#): the returned available length in the file.

Examples:

[ex_sysfileopenwrite.nxc](#), [ex_sysfileopenwritelinear.nxc](#), and [ex_sysfileopenwritenonlinear.nxc](#).

7.32.2.4 unsigned int FileOpenType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_sysfileopenappend.nxc](#), [ex_sysfileopenread.nxc](#), [ex_sysfileopenreadlinear.nxc](#),
[ex_sysfileopenwrite.nxc](#), [ex_sysfileopenwritelinear.nxc](#), and [ex_sysfileopenwritenonlinear.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.33 FileReadWriteType Struct Reference

Parameters for the FileReadWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- string [Buffer](#)
- unsigned long [Length](#)

7.33.1 Detailed Description

Parameters for the FileReadWrite system call. This structure is used when calling the [SysFileRead](#) and [SysFileWrite](#) system call functions.

See also:

[SysFileRead\(\)](#) and [SysFileWrite\(\)](#)

Examples:

[ex_sysfileread.nxc](#), and [ex_sysfilewrite.nxc](#).

7.33.2 Field Documentation

7.33.2.1 string FileReadWriteType::Buffer

The buffer to store read bytes or containing bytes to write.

Examples:

[ex_sysfileread.nxc](#), and [ex_sysfilewrite.nxc](#).

7.33.2.2 byte FileReadWriteType::FileHandle

The file handle to access.

Examples:

[ex_sysfileread.nxc](#), and [ex_sysfilewrite.nxc](#).

7.33.2.3 unsigned long FileReadWriteType::Length

The number of bytes to read or the returned number of bytes written.

Examples:

[ex_sysfileread.nxc](#), and [ex_sysfilewrite.nxc](#).

7.33.2.4 unsigned int FileReadWriteType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_sysfileread.nxc](#), and [ex_sysfilewrite.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.34 FileRenameType Struct Reference

Parameters for the FileRename system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int **Result**
- string **OldFilename**
- string **NewFilename**

7.34.1 Detailed Description

Parameters for the FileRename system call. This structure is used when calling the [SysFileRename](#) system call function.

See also:

[SysFileRename\(\)](#)

Examples:

[ex_sysfilerename.nxc](#).

7.34.2 Field Documentation

7.34.2.1 string FileRenameType::NewFilename

The new name to give to the file.

Examples:

[ex_sysfilerename.nxc](#).

7.34.2.2 string FileRenameType::OldFilename

The name of the file to be renamed.

Examples:

[ex_sysfilerename.nxc](#).

7.34.2.3 unsigned int FileRenameType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_sysfilerename.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.35 FileResizeType Struct Reference

Parameters for the FileResize system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int Result
- byte FileHandle
- unsigned int NewSize

7.35.1 Detailed Description

Parameters for the FileResize system call. This structure is used when calling the [SysFileResize](#) system call function.

See also:

[SysFileResize\(\)](#)

Examples:

[ex_sysfileresize.nxc](#).

7.35.2 Field Documentation

7.35.2.1 byte FileResizeType::FileHandle

The handle of the file to resize.

Examples:

[ex_sysfileresize.nxc](#).

7.35.2.2 unsigned int FileResizeType::NewSize

The new file size.

Examples:

[ex_sysfileresize.nxc](#).

7.35.2.3 unsigned int FileResizeType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_sysfileresize.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.36 FileResolveHandleType Struct Reference

Parameters for the FileResolveHandle system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- bool [WriteHandle](#)
- string [Filename](#)

7.36.1 Detailed Description

Parameters for the FileResolveHandle system call. This structure is used when calling the [SysFileResolveHandle](#) system call function.

See also:

[SysFileResolveHandle\(\)](#)

Examples:

[ex_sysfileresolvehandle.nxc](#).

7.36.2 Field Documentation

7.36.2.1 byte FileResolveHandleType::FileHandle

The returned resolved file handle.

7.36.2.2 string FileResolveHandleType::Filename

The name of the file for which to resolve a handle.

Examples:

[ex_sysfileresolvehandle.nxc](#).

7.36.2.3 unsigned int FileResolveHandleType::Result

The function call result. Possible values include [LDR_HANDLEALREADYCLOSED](#) and [LDR_SUCCESS](#).

Examples:

[ex_sysfileresolvehandle.nxc](#).

7.36.2.4 bool FileResolveHandleType::WriteHandle

True if the returned handle is a write handle.

Examples:

[ex_sysfileresolvehandle.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.37 FileSeekType Struct Reference

Parameters for the FileSeek system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int [Result](#)
- byte [FileHandle](#)
- byte [Origin](#)
- long [Length](#)

7.37.1 Detailed Description

Parameters for the FileSeek system call. This structure is used when calling the [Sys-FileSeek](#) system call function.

See also:

[SysFileSeek\(\)](#)

Examples:

[ex_sysfileseek.nxc](#).

7.37.2 Field Documentation

7.37.2.1 byte FileSeekType::FileHandle

The handle of the file to seek in.

Examples:

[ex_sysfileseek.nxc](#).

7.37.2.2 long FileSeekType::Length

The offset from the origin to seek to.

Examples:

[ex_sysfileseek.nxc](#).

7.37.2.3 byte FileSeekType::Origin

The origin of the file seek operation. See [fseek origin constants](#).

Examples:

[ex_sysfileseek.nxc](#).

7.37.2.4 unsigned int FileSeekType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_sysfileseek.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.38 GetStartTickType Struct Reference

Parameters for the GetStartTick system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned long [Result](#)

7.38.1 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](#).

7.38.2 Field Documentation

7.38.2.1 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](#)

7.39 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** []

7.39.1 Detailed Description

Parameters for the IOMapReadByID system call. This structure is used when calling the [SysIOMapReadByID](#) system call function.

See also:

[SysIOMapReadByID\(\)](#)

Examples:

[ex_reladdressof.nxc](#), and [ex_systiomapreadbyid.nxc](#).

7.39.2 Field Documentation

7.39.2.1 byte IOMapReadByIDType::Buffer[]

The buffer used to store read bytes.

Examples:

[ex_reladdressof.nxc](#).

7.39.2.2 unsigned int IOMapReadByIDType::Count

The number of bytes to read.

Examples:

[ex_reladdressof.nxc](#), and [ex_systiomapreadbyid.nxc](#).

7.39.2.3 unsigned long IOMapReadByIDType::ModuleID

The identifier of the module to read from. See the [NXT firmware module IDs](#) group.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapreadbyid.nxc](#).

7.39.2.4 unsigned int IOMapReadByIDType::Offset

The offset in the module IOMap where to start reading.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapreadbyid.nxc](#).

7.39.2.5 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](#)

7.40 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](#) []

7.40.1 Detailed Description

Parameters for the IOMapRead system call. This structure is used when calling the [SysIOMapRead](#) system call function.

See also:

[SysIOMapRead\(\)](#)

Examples:

[ex_systiomapread.nxc](#).

7.40.2 Field Documentation

7.40.2.1 byte IOMapReadType::Buffer[]

The buffer used to store read bytes.

7.40.2.2 unsigned int IOMapReadType::Count

The number of bytes to read.

Examples:

[ex_systiomapread.nxc](#).

7.40.2.3 string IOMapReadType::ModuleName

The name of the module to read from. See the [NXT firmware module names](#) group.

Examples:

[ex_systiomapread.nxc](#).

7.40.2.4 unsigned int IOMapReadType::Offset

The offset in the module IOMap where to start reading.

Examples:

[ex_systiomapread.nxc](#).

7.40.2.5 char IOMapReadType::Result

The function call result. NO_ERR means it succeeded.

Examples:

[ex_systiomapread.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.41 IOMapWriteByIDType Struct Reference

Parameters for the IOMapWriteByID system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- unsigned long [ModuleID](#)
- unsigned int [Offset](#)
- byte [Buffer](#) []

7.41.1 Detailed Description

Parameters for the IOMapWriteByID system call. This structure is used when calling the [SysIOMapWriteByID](#) system call function.

See also:

[SysIOMapWriteByID\(\)](#)

Examples:

[ex_readdressof.nxc](#), and [ex_systiomapwritebyid.nxc](#).

7.41.2 Field Documentation

7.41.2.1 byte IOMapWriteByIDType::Buffer[]

The buffer containing bytes to write.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapwritebyid.nxc](#).

7.41.2.2 unsigned long IOMapWriteByIDType::ModuleID

The identifier of the module to write to. See the [NXT firmware module IDs](#) group.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapwritebyid.nxc](#).

7.41.2.3 unsigned int IOMapWriteByIDType::Offset

The offset in the module IOMap where to start writing.

Examples:

[ex_reladdressof.nxc](#), and [ex_sysiomapwritebyid.nxc](#).

7.41.2.4 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](#)

7.42 IOMapWriteType Struct Reference

Parameters for the IOMapWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- string [ModuleName](#)
- unsigned int [Offset](#)
- byte [Buffer](#) []

7.42.1 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](#).

7.42.2 Field Documentation

7.42.2.1 byte IOMapWriteType::Buffer[]

The buffer containing bytes to write.

Examples:

[ex_sysiomapwrite.nxc](#).

7.42.2.2 string IOMapWriteType::ModuleName

The name of the module to write to. See the [NXT firmware module names](#) group.

Examples:

[ex_sysiomapwrite.nxc](#).

7.42.2.3 unsigned int IOMapWriteType::Offset

The offset in the module IOMap where to start writing.

Examples:

[ex_sysiomapwrite.nxc](#).

7.42.2.4 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](#)

7.43 KeepAliveType Struct Reference

Parameters for the KeepAlive system call.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned long [Result](#)

7.43.1 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](#).

7.43.2 Field Documentation

7.43.2.1 unsigned long KeepAliveType::Result

The current sleep timeout in milliseconds.

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.44 ldiv_t Struct Reference

Output type of the ldiv function.

```
#include <NXCDefs.h>
```

Data Fields

- long [quot](#)
- long [rem](#)

7.44.1 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.](#)

7.44.2 Field Documentation

7.44.2.1 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.](#)

7.44.2.2 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](#)

7.45 ListFileType Struct Reference

Parameters for the ListFiles system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- string [Pattern](#)
- string [FileList](#) []

7.45.1 Detailed Description

Parameters for the ListFiles system call. This structure is used when calling the [Sys-ListFiles](#) system call function.

See also:

[SysListFiles\(\)](#)

Examples:

[ex_syslistfiles.nxc](#).

7.45.2 Field Documentation**7.45.2.1 string ListFileType::FileList[]**

An array of strings containing the list of filenames that matched the file search pattern.

Examples:

[ex_syslistfiles.nxc](#).

7.45.2.2 string ListFileType::Pattern

The file search pattern.

Examples:

[ex_syslistfiles.nxc](#).

7.45.2.3 char ListFileType::Result

The function call result. Possible values include [Loader module error codes](#).

Examples:

[ex_syslistfiles.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.46 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](#)

7.46.1 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_-OPENWITELINEAR	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_sysloaderexecufunction.nxc](#).

7.46.2 Field Documentation

7.46.2.1 byte LoaderExecuteFunctionType::Buffer[]

The Buffer parameter, see table.

7.46.2.2 byte LoaderExecuteFunctionType::Cmd

The command to execute.

Examples:

[ex_sysloaderexecutefunction.nxc](#).

7.46.2.3 string LoaderExecuteFunctionType::Filename

The Filename parameter, see table.

7.46.2.4 unsigned long LoaderExecuteFunctionType::Length

The Length parameter, see table.

7.46.2.5 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](#)

7.47 LocationType Struct Reference

A point on the NXT LCD screen.

```
#include <NXCDefs.h>
```

Data Fields

- int [X](#)
- int [Y](#)

7.47.1 Detailed Description

A point on the NXT LCD screen. This structure is used by other system call structures to specify an X, Y LCD screen coordinate.

See also:

[DrawTextType](#), [DrawPointType](#), [DrawLineType](#), [DrawCircleType](#), [DrawRectType](#), [DrawGraphicType](#), [DrawGraphicArrayType](#), [DrawPolygonType](#), [DrawEllipseType](#), [DrawFontType](#)

Examples:

[ex_PolyOut.nxc](#), and [ex_sysdrawpolygon.nxc](#).

7.47.2 Field Documentation

7.47.2.1 int LocationType::X

The X coordinate. Valid range is from 0 to 99 inclusive.

Examples:

[ex_dispftout.nxc](#), [ex_dispgout.nxc](#), [ex_syscall.nxc](#), [ex_sysdrawcircle.nxc](#), [ex_sysdrawellipse.nxc](#), [ex_sysdrawfont.nxc](#), [ex_sysdrawgraphic.nxc](#), [ex_sysdrawgraphicarray.nxc](#), [ex_sysdrawline.nxc](#), [ex_sysdrawpoint.nxc](#), [ex_sysdrawrect.nxc](#), and [ex_sysdrawtext.nxc](#).

7.47.2.2 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_dispftout.nxc](#), [ex_dispgout.nxc](#), [ex_syscall.nxc](#), [ex_sysdrawcircle.nxc](#), [ex_sysdrawellipse.nxc](#), [ex_sysdrawfont.nxc](#), [ex_sysdrawgraphic.nxc](#), [ex_sysdrawgraphicarray.nxc](#), [ex_sysdrawline.nxc](#), [ex_sysdrawpoint.nxc](#), [ex_sysdrawrect.nxc](#), and [ex_sysdrawtext.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.48 MessageReadType Struct Reference

Parameters for the MessageRead system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [QueueID](#)
- bool [Remove](#)
- string [Message](#)

7.48.1 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](#).

7.48.2 Field Documentation

7.48.2.1 string MessageReadType::Message

The contents of the mailbox/queue.

Examples:

[ex_sysmessageread.nxc](#).

7.48.2.2 byte MessageReadType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

Examples:

[ex_sysmessageread.nxc](#).

7.48.2.3 bool MessageReadType::Remove

If true, remove the read message from the queue.

Examples:

[ex_sysmessageread.nxc](#).

7.48.2.4 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](#)

7.49 MessageWriteType Struct Reference

Parameters for the MessageWrite system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [QueueID](#)
- string [Message](#)

7.49.1 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](#).

7.49.2 Field Documentation

7.49.2.1 string MessageWriteType::Message

The message to write.

Examples:

[ex_sysmessagewrite.nxc](#).

7.49.2.2 byte MessageWriteType::QueueID

The queue identifier. See the [Mailbox constants](#) group.

Examples:

[ex_sysmessagewrite.nxc](#).

7.49.2.3 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](#)

7.50 RandomNumberType Struct Reference

Parameters for the RandomNumber system call.

```
#include <NXCDefs.h>
```

Data Fields

- int [Result](#)

7.50.1 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](#).

7.50.2 Field Documentation**7.50.2.1 int RandomNumberType::Result**

The random number.

Examples:

[ex_sysrandomnumber.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.51 ReadButtonType Struct Reference

Parameters for the ReadButton system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- byte [Index](#)
- bool [Pressed](#)
- byte [Count](#)
- bool [Reset](#)

7.51.1 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](#).

7.51.2 Field Documentation**7.51.2.1 byte ReadButtonType::Count**

The returned button pressed count.

7.51.2.2 byte ReadButtonType::Index

The requested button index. See the [Button name constants](#) group.

Examples:

[ex_sysreadbutton.nxc](#).

7.51.2.3 bool ReadButtonType::Pressed

The returned button state.

Examples:

[ex_sysreadbutton.nxc](#).

7.51.2.4 bool ReadButtonType::Reset

If true, the count is reset after reading.

7.51.2.5 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](#)

7.52 ReadSemDataType Struct Reference

Parameters for the ReadSemData system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [SemData](#)
- bool [Request](#)

7.52.1 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](#).

7.52.2 Field Documentation**7.52.2.1 bool ReadSemDataType::Request**

Which semaphore am I reading from, usage or request?

Examples:

[ex_SysReadSemData.nxc](#).

7.52.2.2 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](#)

7.53 SetScreenModeType Struct Reference

Parameters for the SetScreenMode system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- unsigned long [ScreenMode](#)

7.53.1 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](#).

7.53.2 Field Documentation**7.53.2.1 char SetScreenModeType::Result**

The function call result, always [NO_ERR](#).

7.53.2.2 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](#)

7.54 SetSleepTimeoutType Struct Reference

Parameters for the SetSleepTimeout system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- unsigned long [TheSleepTimeoutMS](#)

7.54.1 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](#).

7.54.2 Field Documentation**7.54.2.1 char SetSleepTimeoutType::Result**

The result of the system call function.

7.54.2.2 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](#)

7.55 SizeType Struct Reference

Width and height dimensions for the DrawRect system call.

```
#include <NXCDefs.h>
```

Data Fields

- int [Width](#)
- int [Height](#)

7.55.1 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](#)

7.55.2 Field Documentation**7.55.2.1 int SizeType::Height**

The rectangle height.

Examples:

[ex_sysdrawrect.nxc](#).

7.55.2.2 int SizeType::Width

The rectangle width.

Examples:

[ex_sysdrawrect.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.56 SoundGetStateType Struct Reference

Parameters for the SoundGetState system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [State](#)
- byte [Flags](#)

7.56.1 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](#).

7.56.2 Field Documentation**7.56.2.1 byte SoundGetType::Flags**

The returned sound flags. See the [SoundFlags constants](#) group.

7.56.2.2 byte SoundGetType::State

The returned sound state. See the [SoundState constants](#) group.

Examples:

[ex_syssoundgetstate.nxc](#).

The documentation for this struct was generated from the following file:

- [NXCDefs.h](#)

7.57 SoundPlayFileType Struct Reference

Parameters for the SoundPlayFile system call.

```
#include <NXCDefs.h>
```

Data Fields

- char [Result](#)
- string [Filename](#)
- bool [Loop](#)
- byte [SoundLevel](#)

7.57.1 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](#).

7.57.2 Field Documentation**7.57.2.1 string SoundPlayFileType::Filename**

The name of the file to play.

Examples:

[ex_syssoundplayfile.nxc](#).

7.57.2.2 bool SoundPlayFileType::Loop

If true, loops at end of file.

Examples:

[ex_syssoundplayfile.nxc](#).

7.57.2.3 char SoundPlayFileType::Result

The function call result, always [NO_ERR](#).

7.57.2.4 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](#)

7.58 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](#)

7.58.1 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](#).

7.58.2 Field Documentation

7.58.2.1 unsigned int SoundPlayToneType::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

Examples:[ex_syssoundplaytone.nxc](#).**7.58.2.2 unsigned int SoundPlayToneType::Frequency**

The tone frequency. See the [Tone constants](#) group.

Examples:[ex_syssoundplaytone.nxc](#).**7.58.2.3 bool SoundPlayToneType::Loop**

If true, loops forever.

Examples:[ex_syssoundplaytone.nxc](#).**7.58.2.4 char SoundPlayToneType::Result**

The function call result, always [NO_ERR](#).

7.58.2.5 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](#)

7.59 SoundSetStateType Struct Reference

Parameters for the SoundSetState system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [Result](#)
- byte [State](#)
- byte [Flags](#)

7.59.1 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](#).

7.59.2 Field Documentation**7.59.2.1 byte SoundSetStateType::Flags**

The new sound flags. See the [SoundFlags constants](#) group.

7.59.2.2 byte SoundSetStateType::Result

The function call result, same as State.

7.59.2.3 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](#)

7.60 Tone Struct Reference

Type used with the PlayTones API function.

```
#include <NXCDefs.h>
```

Data Fields

- unsigned int Frequency
- unsigned int Duration

7.60.1 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](#).

7.60.2 Field Documentation

7.60.2.1 unsigned int Tone::Duration

The tone duration in milliseconds. See the [Time constants](#) group.

7.60.2.2 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](#)

7.61 UpdateCalibCacheInfoType Struct Reference

Parameters for the UpdateCalibCacheInfo system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte **Result**
- string **Name**
- unsigned int **MinVal**
- unsigned int **MaxVal**

7.61.1 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](#).

7.61.2 Field Documentation**7.61.2.1 unsigned int UpdateCalibCacheInfoType::MaxVal**

The maximum calibrated value.

Examples:

[ex_SysUpdateCalibCacheInfo.nxc](#).

7.61.2.2 unsigned int UpdateCalibCacheInfoType::MinVal

The minimum calibrated value.

Examples:

[ex_SysUpdateCalibCacheInfo.nxc](#).

7.61.2.3 string UpdateCalibCacheInfoType::Name

The name of the sensor calibration cache.

Todo

?

Examples:

[ex_SysUpdateCalibCacheInfo.nxc](#).

7.61.2.4 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](#)

7.62 WriteSemDataType Struct Reference

Parameters for the WriteSemData system call.

```
#include <NXCDefs.h>
```

Data Fields

- byte [SemData](#)
- bool [Request](#)
- byte [NewVal](#)
- bool [ClearBits](#)

7.62.1 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](#).

7.62.2 Field Documentation**7.62.2.1 bool WriteSemData::ClearBits**

Should I clear existing bits?

Examples:

[ex_SysWriteSemData.nxc.](#)

7.62.2.2 byte WriteSemData::NewVal

The new semaphore data.

Examples:

[ex_SysWriteSemData.nxc.](#)

7.62.2.3 bool WriteSemData::Request

Which semaphore am I writing to, usage or request?

Examples:

[ex_SysWriteSemData.nxc.](#)

7.62.2.4 byte WriteSemData::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](#)

8 File Documentation

8.1 NBCCommon.h File Reference

Constants and macros common to both NBC and NXC.

Defines

- #define **TRUE** 1
- #define **FALSE** 0
- #define **NA** 0xFFFF
- #define **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_FILENOFOUND` 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_RAMPUP 0x10
- #define OUT_RUNSTATE_RUNNING 0x20
- #define OUT_RUNSTATE_RAMPDOWN 0x40
- #define OUT_RUNSTATE_HOLD 0x60
- #define OUT_REGMODE_IDLE 0
- #define OUT_REGMODE_SPEED 1
- #define OUT_REGMODE_SYNC 2
- #define 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_ADDR_DEFAULT 0x02
- #define I2C_REG_VERSION 0x00
- #define I2C_REG_VENDOR_ID 0x08
- #define I2C_REG_DEVICE_ID 0x10

- #define I2C_REG_CMD 0x41
- #define LEGO_ADDR_US 0x02
- #define LEGO_ADDR_TEMP 0x98
- #define LEGO_ADDR_EMETER 0x04
- #define US_CMD_OFF 0x00
- #define US_CMD_SINGLESHT 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_REG_TEMP 0x00
- #define TEMP_REG_CONFIG 0x01
- #define TEMP_REG_TLOW 0x02
- #define TEMP_REG_THIGH 0x03
- #define EMETER_REG_VIN 0x0a
- #define EMETER_REG_AIN 0x0c
- #define EMETER_REG_VOUT 0x0e
- #define EMETER_REG_AOUT 0x10
- #define EMETER_REG_JOULES 0x12
- #define EMETER_REG_WIN 0x14
- #define EMETER_REG_WOUT 0x16

- #define 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_OFFSET 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_BTTON** 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_RemotePBMessagel** 0x0100
- #define **RCX_RemotePBMessagel2** 0x0200
- #define **RCX_RemotePBMessagel3** 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_RemoteSelPrograml** 0x0002
- #define **RCX_RemoteSelPrograml2** 0x0004
- #define **RCX_RemoteSelPrograml3** 0x0008
- #define **RCX_RemoteSelPrograml4** 0x0010
- #define **RCX_RemoteSelPrograml5** 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_LSSupperThreshOp` 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 HT_ADDR_IRSEEKER 0x02
- #define HT_ADDR_IRSEEKER2 0x10
- #define HT_ADDR_IRRECEIVER 0x02
- #define HT_ADDR_COMPASS 0x02
- #define HT_ADDR_ACCEL 0x02
- #define HT_ADDR_COLOR 0x02
- #define HT_ADDR_COLOR2 0x02
- #define HT_ADDR_IRLINK 0x02
- #define 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 MS_ADDR_RTCLOCK 0xD0
- #define MS_ADDR_DISTNX 0x02
- #define MS_ADDR_NRLINK 0x02
- #define MS_ADDR_ACCLNX 0x02
- #define MS_ADDR_CMPSNX 0x02
- #define MS_ADDR_PSPNX 0x02
- #define MS_ADDR_LINELDR 0x02
- #define MS_ADDR_NXTCAM 0x02
- #define MS_ADDR_NXTHID 0x04
- #define MS_ADDR_NXTSERVO 0xB0
- #define MS_ADDR_PFMATE 0x48
- #define MS_ADDR_MTRMUX 0xB4
- #define MS_ADDR_NXTMMX 0x06
- #define MS_ADDR_IVSENS 0x12
- #define MS_ADDR_RXMUX 0x7E
- #define DIST_CMD_GP2D12 0x31
- #define DIST_CMD_GP2D120 0x32
- #define DIST_CMD_GP2YA21 0x33
- #define DIST_CMD_GP2YA02 0x34
- #define DIST_CMD_CUSTOM 0x35
- #define DIST_REG_DIST 0x42
- #define DIST_REG_VOLT 0x44
- #define DIST_REG_MODULE_TYPE 0x50
- #define DIST_REG_NUM_POINTS 0x51
- #define DIST_REG_DIST_MIN 0x52
- #define DIST_REG_DIST_MAX 0x54
- #define DIST_REG_VOLT1 0x56
- #define DIST_REG_DIST1 0x58
- #define PSP_CMD_DIGITAL 0x41
- #define PSP_CMD_ANALOG 0x73
- #define PSP_REG_BTNSET1 0x42
- #define PSP_REG_BTNSET2 0x43
- #define PSP_REG_XLEFT 0x44
- #define PSP_REG_YLEFT 0x45
- #define PSP_REG_XRIGHT 0x46

- #define PSP_REG_YRIGHT 0x47
- #define PSP_BTNSET1_LEFT 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_RX 0x58
- #define NRLINK_CMD_SET_TRAIN 0x54
- #define NRLINK_CMD_SET_PF 0x50
- #define NRLINK_REG_BYTES 0x40
- #define NRLINK_REG_DATA 0x42
- #define NRLINK_REG_EEPROM 0x50
- #define ACCL_CMD_X_CAL 0x58
- #define ACCL_CMD_Y_CAL 0x59
- #define ACCL_CMD_Z_CAL 0x5a
- #define ACCL_CMD_X_CAL_END 0x78
- #define ACCL_CMD_Y_CAL_END 0x79
- #define ACCL_CMD_Z_CAL_END 0x7a
- #define ACCL_CMD_RESET_CAL 0x52
- #define ACCL_REG_SENS_LVL 0x19
- #define ACCL_REG_X_TILT 0x42
- #define ACCL_REG_Y_TILT 0x43
- #define ACCL_REG_Z_TILT 0x44
- #define ACCL_REG_X_ACCEL 0x45
- #define ACCL_REG_Y_ACCEL 0x47
- #define ACCL_REG_Z_ACCEL 0x49
- #define ACCL_REG_X_OFFSET 0x4b

- #define ACCL_REG_X_RANGE 0x4d
- #define ACCL_REG_Y_OFFSET 0x4f
- #define ACCL_REG_Y_RANGE 0x51
- #define ACCL_REG_Z_OFFSET 0x53
- #define ACCL_REG_Z_RANGE 0x55
- #define ACCL_SENSITIVITY_LEVEL_1 0x31
- #define ACCL_SENSITIVITY_LEVEL_2 0x32
- #define ACCL_SENSITIVITY_LEVEL_3 0x33
- #define ACCL_SENSITIVITY_LEVEL_4 0x34
- #define 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 **RICOOpNumBox**(_CopyOptions, _Point, _Value) 10, 0, 8, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point, (_Value)&0xFF, (_Value)>>8

Output an RIC NumBox opcode.

- #define **RICOOpSprite**(_DataAddr, _Rows, _BytesPerRow, _SpriteData) ((-_Rows*_BytesPerRow)+(_Rows*_BytesPerRow)%2)+8)>>8, ((-_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 **RICOOpVarMap**(_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 **RICOOpPolygon**(_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

8.1.1 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:

2010-06-28

Version:

54

8.1.2 Define Documentation

8.1.2.1 #define ACCL_CMD_RESET_CAL 0x52

Reset to factory calibration

8.1.2.2 #define ACCL_CMD_X_CAL 0x58

Acquire X-axis calibration point

8.1.2.3 #define ACCL_CMD_X_CAL_END 0x78

Acquire X-axis calibration point and end calibration

8.1.2.4 #define ACCL_CMD_Y_CAL 0x59

Acquire Y-axis calibration point

8.1.2.5 #define ACCL_CMD_Y_CAL_END 0x79

Acquire Y-axis calibration point and end calibration

8.1.2.6 #define ACCL_CMD_Z_CAL 0x5a

Acquire Z-axis calibration point

8.1.2.7 #define ACCL_CMD_Z_CAL_END 0x7a

Acquire Z-axis calibration point and end calibration

8.1.2.8 #define ACCL_REG_SENS_LVL 0x19

The current sensitivity

8.1.2.9 #define ACCL_REG_X_ACCEL 0x45

The X-axis acceleration data

8.1.2.10 #define ACCL_REG_X_OFFSET 0x4b

The X-axis offset

8.1.2.11 #define ACCL_REG_X_RANGE 0x4d

The X-axis range

8.1.2.12 #define ACCL_REG_X_TILT 0x42

The X-axis tilt data

8.1.2.13 #define ACCL_REG_Y_ACCEL 0x47

The Y-axis acceleration data

8.1.2.14 #define ACCL_REG_Y_OFFSET 0x4f

The Y-axis offset

8.1.2.15 #define ACCL_REG_Y_RANGE 0x51

The Y-axis range

8.1.2.16 #define ACCL_REG_Y_TILT 0x43

The Y-axis tilt data

8.1.2.17 #define ACCL_REG_Z_ACCEL 0x49

The Z-axis acceleration data

8.1.2.18 #define ACCL_REG_Z_OFFSET 0x53

The Z-axis offset

8.1.2.19 #define ACCL_REG_Z_RANGE 0x55

The Z-axis range

8.1.2.20 #define ACCL_REG_Z_TILT 0x44

The Z-axis tilt data

8.1.2.21 #define ACCL_SENSITIVITY_LEVEL_1 0x31

The ACCL-Nx sensitivity level 1

Examples:

[ex_SetACCLNxSensitivity.nxc](#).

8.1.2.22 #define ACCL_SENSITIVITY_LEVEL_2 0x32

The ACCL-Nx sensitivity level 2

8.1.2.23 #define ACCL_SENSITIVITY_LEVEL_3 0x33

The ACCL-Nx sensitivity level 3

8.1.2.24 #define ACCL_SENSITIVITY_LEVEL_4 0x34

The ACCL-Nx sensitivity level 4

8.1.2.25 #define 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.

8.1.2.26 #define BITMAP_1 0

Bitmap 1

8.1.2.27 #define BITMAP_2 1

Bitmap 2

8.1.2.28 #define BITMAP_3 2

Bitmap 3

8.1.2.29 #define BITMAP_4 3

Bitmap 4

8.1.2.30 #define BITMAPS 4

The number of bitmap bits

8.1.2.31 #define 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.

8.1.2.32 #define BREAKOUT_REQ 3

VM should break out of current thread

8.1.2.33 #define BT_ARM_CMD_MODE 1

BtState constant bluetooth command mode

8.1.2.34 #define BT_ARM_DATA_MODE 2

BtState constant bluetooth data mode

8.1.2.35 #define BT_ARM_OFF 0

BtState constant bluetooth off

8.1.2.36 #define BT_BRICK_PORT_OPEN 0x02

BtStateStatus port open bit

8.1.2.37 #define BT_BRICK_VISIBILITY 0x01

BtStateStatus brick visibility bit

8.1.2.38 #define BT_CMD_BYTE 1

Size of Bluetooth command

8.1.2.39 #define BT_CMD_READY 0x02

A constant representing bluetooth direct command

8.1.2.40 #define BT_CONNECTION_0_ENABLE 0x10

BtStateStatus connection 0 enable/disable bit

8.1.2.41 #define BT_CONNECTION_1_ENABLE 0x20

BtStateStatus connection 1 enable/disable bit

8.1.2.42 #define BT_CONNECTION_2_ENABLE 0x40

BtStateStatus connection 2 enable/disable bit

8.1.2.43 #define BT_CONNECTION_3_ENABLE 0x80

BtStateStatus connection 3 enable/disable bit

8.1.2.44 #define BT_DEFAULT_INQUIRY_MAX 0

Bluetooth default inquiry Max (0 == unlimited)

8.1.2.45 #define BT_DEFAULT_INQUIRY_TIMEOUT_LO 15

Bluetooth inquiry timeout ($15 * 1.28$ sec = 19.2 sec)

8.1.2.46 #define BT_DEVICE_AWAY 0x80

Bluetooth device away

8.1.2.47 #define BT_DEVICE_EMPTY 0x00

Bluetooth device table empty

8.1.2.48 #define BT_DEVICE_KNOWN 0x02

Bluetooth device known

8.1.2.49 #define BT_DEVICE_NAME 0x40

Bluetooth device name

8.1.2.50 #define BT_DEVICE_UNKNOWN 0x01

Bluetooth device unknown

8.1.2.51 #define BT_DISABLE 0x01

BtHwStatus bluetooth disable

8.1.2.52 #define BT_ENABLE 0x00

BtHwStatus bluetooth enable

8.1.2.53 #define BTN1 0

The exit button.

Examples:

[ex_ButtonCount.nxc](#), [ex_ButtonLongPressCount.nxc](#), [ex_ButtonLongReleaseCount.nxc](#), [ex_ButtonPressCount.nxc](#), [ex_ButtonReleaseCount.nxc](#), [ex_ButtonShortReleaseCount.nxc](#), [ex_ButtonState.nxc](#), [ex_ReadButtonEx.nxc](#), [ex_SetButtonLongPressCount.nxc](#), [ex_SetButtonLongReleaseCount.nxc](#), [ex_SetButtonPressCount.nxc](#), [ex_SetButtonReleaseCount.nxc](#), [ex_SetButtonShortReleaseCount.nxc](#), and [ex_SetButtonState.nxc](#).

8.1.2.54 #define BTN2 1

The right button.

8.1.2.55 #define BTN3 2

The left button.

8.1.2.56 #define BTN4 3

The enter button.

8.1.2.57 #define BTNCENTER BTN4

The enter button.

Examples:

[ex_ButtonPressed.nxc](#).

8.1.2.58 #define BTNEXIT BTN1

The exit button.

Examples:

[ex_ButtonPressed.nxc](#).

8.1.2.59 #define BTNLEFT BTN3

The left button.

Examples:

[ex_ButtonPressed.nxc](#).

8.1.2.60 #define BTNRIGHT BTN2

The right button.

Examples:

[ex_ButtonPressed.nxc](#), and [ex_sysreadbutton.nxc](#).

8.1.2.61 #define BTNSTATE_LONG_PRESSED_EV 0x04

Button is in the long pressed state.

Examples:

[ex_SetAbortFlag.nxc](#).

8.1.2.62 #define BTNSTATE_LONG_RELEASED_EV 0x08

Button is in the long released state.

8.1.2.63 #define BTNSTATE_NONE 0x10

The default button state.

8.1.2.64 #define BTNSTATE_PRESSED_EV 0x01

Button is in the pressed state.

Examples:

[ex_SetButtonState.nxc](#).

8.1.2.65 #define BTNSTATE_PRESSED_STATE 0x80

A bitmask for the button pressed state

8.1.2.66 #define BTNSTATE_SHORT_RELEASED_EV 0x02

Button is in the short released state.

8.1.2.67 #define ButtonModuleID 0x00040001

The button module ID

8.1.2.68 #define ButtonModuleName "Button.mod"

The button module name

8.1.2.69 #define ButtonOffsetLongPressCnt(b) (((b)*8)+1)

Offset to the LongPressCnt field. This field stores the long press count.

8.1.2.70 #define ButtonOffsetLongRelCnt(b) (((b)*8)+3)

Offset to the LongRelCnt field. This field stores the long release count.

8.1.2.71 #define ButtonOffsetPressedCnt(b) (((b)*8)+0)

Offset to the PressedCnt field. This field stores the press count.

8.1.2.72 #define ButtonOffsetRelCnt(b) (((b)*8)+4)

Offset to the RelCnt field. This field stores the release count.

8.1.2.73 #define ButtonOffsetShortRelCnt(b) (((b)*8)+2)

Offset to the ShortRelCnt field. This field stores the short release count.

8.1.2.74 #define ButtonOffsetState(b) ((b)+32)

Offset to the State field. This field stores the current button state.

8.1.2.75 #define CHAR_BIT 8

The number of bits in the char type

8.1.2.76 #define CHAR_MAX 127

The maximum value of the char type

8.1.2.77 #define CHAR_MIN -127

The minimum value of the char type

8.1.2.78 #define CLUMP_DONE 1

VM has finished executing thread

8.1.2.79 #define CLUMP_SUSPEND 2

VM should suspend thread

8.1.2.80 #define ColorSensorRead 34

Read data from the NXT 2.0 color sensor

8.1.2.81 #define COM_CHANNEL_FOUR_ACTIVE 0x08

Low speed channel 4 is active

8.1.2.82 #define COM_CHANNEL_NONE_ACTIVE 0x00

None of the low speed channels are active

8.1.2.83 #define COM_CHANNEL_ONE_ACTIVE 0x01

Low speed channel 1 is active

8.1.2.84 #define COM_CHANNEL_THREE_ACTIVE 0x04

Low speed channel 3 is active

8.1.2.85 #define COM_CHANNEL_TWO_ACTIVE 0x02

Low speed channel 2 is active

8.1.2.86 #define CommandModuleID 0x00010001

The command module ID

Examples:

[ex_readdressof.nxc](#), and [ex_systiomapreadbyid.nxc](#).

8.1.2.87 #define CommandModuleName "Command.mod"

The command module name

Examples:

[ex_systiomapread.nxc](#).

8.1.2.88 #define CommandOffsetActivateFlag 30

Offset to the activate flag

8.1.2.89 #define CommandOffsetAwake 29

Offset to the VM's awake state

8.1.2.90 #define CommandOffsetDeactivateFlag 31

Offset to the deactivate flag

8.1.2.91 #define CommandOffsetFileName 32

Offset to the running program's filename

8.1.2.92 #define CommandOffsetFormatString 0

Offset to the format string

8.1.2.93 #define CommandOffsetMemoryPool 52

Offset to the VM's memory pool

Examples:

[ex_reladdressof.nxc](#).

8.1.2.94 #define CommandOffsetOffsetDS 24

Offset to the running program's data space (DS)

8.1.2.95 #define CommandOffsetOffsetDVA 26

Offset to the running program's DOPE vector address (DVA)

8.1.2.96 #define CommandOffsetPRCHandler 16

Offset to the RC Handler function pointer

8.1.2.97 #define CommandOffsetProgStatus 28

Offset to the running program's status

8.1.2.98 #define CommandOffsetSyncTick 32824

Offset to the VM sync tick

8.1.2.99 #define CommandOffsetSyncTime 32820

Offset to the VM sync time

8.1.2.100 #define CommandOffsetTick 20

Offset to the VM's current tick

Examples:

[ex_systomapread.nxc](#), and [ex_systomapreadbyid.nxc](#).

8.1.2.101 #define CommBTCheckStatus 28

Check the bluetooth status

8.1.2.102 #define CommBTConnection 36

Connect or disconnect to a known bluetooth device

8.1.2.103 #define CommBTOnOff 35

Turn the bluetooth radio on or off

8.1.2.104 #define CommBTRead 30

Read from a bluetooth connection

8.1.2.105 #define CommBTWrite 29

Write to a bluetooth connections

8.1.2.106 #define CommExecuteFunction 81

Execute one of the Comm module's internal functions

8.1.2.107 #define CommHSCheckStatus 39

Check the status of the hi-speed port

8.1.2.108 #define CommHSControl 88

Control the hi-speed port

8.1.2.109 #define CommHSRead 38

Read data from the hi-speed port

8.1.2.110 #define CommHSWrite 37

Write data to the hi-speed port

8.1.2.111 #define CommLSCheckStatus 23

Check the status of a lowspeed (aka I2C) device

8.1.2.112 #define CommLSRead 22

Read from a lowspeed (aka I2C) device

8.1.2.113 #define CommLSWrite 21

Write to a lowspeed (aka I2C) device

8.1.2.114 #define CommLSWriteEx 89

Write to a lowspeed (aka I2C) device with optional restart on read

8.1.2.115 #define CommModuleID 0x00050001

The Comm module ID

8.1.2.116 #define CommModuleName "Comm.mod"

The Comm module name

8.1.2.117 #define CommOffsetBrickDataBdAddr 1144

Offset to Bluetooth address (7 bytes)

8.1.2.118 #define CommOffsetBrickDataBluecoreVersion 1142

Offset to Bluecore version (2 bytes)

8.1.2.119 #define CommOffsetBrickDataBtHwStatus 1152

Offset to BtHwStatus (1 byte)

8.1.2.120 #define CommOffsetBrickDataBtStateStatus 1151

Offset to BtStateStatus (1 byte)

8.1.2.121 #define CommOffsetBrickDataName 1126

Offset to brick name (16 bytes)

8.1.2.122 #define CommOffsetBrickDataTimeOutValue 1153

Offset to data timeout value (1 byte)

8.1.2.123 #define CommOffsetBtConnectTableBdAddr(p) (((p)*47)+974)

Offset to Bluetooth connect table address (7 bytes)

8.1.2.124 #define CommOffsetBtConnectTableClassOfDevice(p) (((p)*47)+954)

Offset to Bluetooth connect table device class (4 bytes)

8.1.2.125 #define CommOffsetBtConnectTableHandleNr(p) (((p)*47)+981)

Offset to Bluetooth connect table handle (1 byte)

8.1.2.126 #define CommOffsetBtConnectTableLinkQuality(p) (((p)*47)+983)

Offset to Bluetooth connect table link quality (1 byte)

8.1.2.127 #define CommOffsetBtConnectTableName(p) (((p)*47)+938)

Offset to Bluetooth connect table name (16 bytes)

8.1.2.128 #define CommOffsetBtConnectTablePinCode(p) (((p)*47)+958)

Offset to Bluetooth connect table pin code (16 bytes)

8.1.2.129 #define CommOffsetBtConnectTableStreamStatus(p) (((p)*47)+982)

Offset to Bluetooth connect table stream status (1 byte)

8.1.2.130 #define CommOffsetBtDeviceCnt 1889

Offset to Bluetooth device count (1 byte)

8.1.2.131 #define CommOffsetBtDeviceNameCnt 1890

Offset to Bluetooth device name count (1 byte)

8.1.2.132 #define CommOffsetBtDeviceTableBdAddr(p) (((p)*31)+28)

Offset to Bluetooth device table address (7 bytes)

8.1.2.133 #define CommOffsetBtDeviceTableClassOfDevice(p) (((p)*31)+24)

Offset to Bluetooth device table device class (4 bytes)

8.1.2.134 #define CommOffsetBtDeviceTableDeviceStatus(p) (((p)*31)+35)

Offset to Bluetooth device table status (1 byte)

8.1.2.135 #define CommOffsetBtDeviceTableName(p) (((p)*31)+8)

Offset to BT device table name (16 bytes)

8.1.2.136 #define CommOffsetBtInBufBuf 1157

Offset to Bluetooth input buffer data (128 bytes)

8.1.2.137 #define CommOffsetBtInBufInPtr 1285

Offset to Bluetooth input buffer front pointer (1 byte)

8.1.2.138 #define CommOffsetBtInBufOutPtr 1286

Offset to Bluetooth output buffer back pointer (1 byte)

8.1.2.139 #define CommOffsetBtOutBufBuf 1289

Offset to Bluetooth output buffer offset data (128 bytes)

8.1.2.140 #define CommOffsetBtOutBufInPtr 1417

Offset to Bluetooth output buffer front pointer (1 byte)

8.1.2.141 #define CommOffsetBtOutBufOutPtr 1418

Offset to Bluetooth output buffer back pointer (1 byte)

8.1.2.142 #define CommOffsetHsFlags 1891

Offset to High Speed flags (1 byte)

8.1.2.143 #define CommOffsetHsInBufBuf 1421

Offset to High Speed input buffer data (128 bytes)

8.1.2.144 #define CommOffsetHsInBufInPtr 1549

Offset to High Speed input buffer front pointer (1 byte)

8.1.2.145 #define CommOffsetHsInBufOutPtr 1550

Offset to High Speed input buffer back pointer (1 byte)

8.1.2.146 #define CommOffsetHsMode 1895

Offset to High Speed mode (2 bytes)

8.1.2.147 #define CommOffsetHsOutBufBuf 1553

Offset to High Speed output buffer data (128 bytes)

8.1.2.148 #define CommOffsetHsOutBufInPtr 1681

Offset to High Speed output buffer front pointer (1 byte)

8.1.2.149 #define CommOffsetHsOutBufOutPtr 1682

Offset to High Speed output buffer back pointer (1 byte)

8.1.2.150 #define CommOffsetHsSpeed 1892

Offset to High Speed speed (1 byte)

8.1.2.151 #define CommOffsetHsState 1893

Offset to High Speed state (1 byte)

8.1.2.152 #define CommOffsetPFunc 0

Offset to the Comm module first function pointer (4 bytes)

8.1.2.153 #define CommOffsetPFuncTwo 4

Offset to the Comm module second function pointer (4 bytes)

8.1.2.154 #define CommOffsetUsbInBufBuf 1685

Offset to Usb input buffer data (64 bytes)

8.1.2.155 #define CommOffsetUsbInBufInPtr 1749

Offset to Usb input buffer front pointer (1 byte)

8.1.2.156 #define CommOffsetUsbInBufOutPtr 1750

Offset to Usb input buffer back pointer (1 byte)

8.1.2.157 #define CommOffsetUsbOutBufBuf 1753

Offset to Usb output buffer data (64 bytes)

8.1.2.158 #define CommOffsetUsbOutBufInPtr 1817

Offset to Usb output buffer front pointer (1 byte)

8.1.2.159 #define CommOffsetUsbOutBufOutPtr 1818

Offset to Usb output buffer back pointer (1 byte)

8.1.2.160 #define CommOffsetUsbPollBufBuf 1821

Offset to Usb Poll buffer data (64 bytes)

8.1.2.161 #define CommOffsetUsbPollBufInPtr 1885

Offset to Usb Poll buffer front pointer (1 byte)

8.1.2.162 #define CommOffsetUsbPollBufOutPtr 1886

Offset to Usb Poll buffer back pointer (1 byte)

8.1.2.163 #define CommOffsetUsbState 1894

Offset to Usb State (1 byte)

8.1.2.164 #define ComputeCalibValue 42

Compute a calibration value

8.1.2.165 #define DatalogGetTimes 45

Get datalog timing information

8.1.2.166 #define DatalogWrite 44

Write to the datalog

8.1.2.167 #define DEGREES_PER_RADIAN 180/PI

Used for converting from radians to degrees

8.1.2.168 #define DISPLAY_BUSY 0x80

R - Refresh in progress

8.1.2.169 #define DISPLAY_CHAR 0x04

W - draw char (actual font) (CMD,TRUE,X1,Y1,Char,x)

8.1.2.170 #define DISPLAY_CONTRAST_DEFAULT 0x5A

Default display contrast value

Examples:

[ex_contrast.nxc](#), and [ex_SetDisplayContrast.nxc](#).

8.1.2.171 #define DISPLAY_CONTRAST_MAX 0x7F

Maximum display contrast value

Examples:

[ex_contrast.nxc](#).

8.1.2.172 #define DISPLAY_ERASE_ALL 0x00

W - erase entire screen (CMD,x,x,x,x,x)

Examples:

[ex_sysdisplayexecufunction.nxc](#).

8.1.2.173 #define DISPLAY_ERASE_LINE 0x05

W - erase a single line (CMD,x,LINE,x,x,x)

8.1.2.174 #define DISPLAY_FILL_REGION 0x06

W - fill screen region (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

8.1.2.175 #define DISPLAY_FRAME 0x07

W - draw a frame (on/off) (CMD,TRUE/FALSE,X1,Y1,X2,Y2)

8.1.2.176 #define DISPLAY_HEIGHT 64

The height of the LCD screen in pixels

Examples:

[ex_LineOut.nxc](#).

8.1.2.177 #define DISPLAY_HORIZONTAL_LINE 0x02

W - draw horizontal line (CMD,TRUE/FALSE,X1,Y1,X2,x)

Examples:

[ex_dispfunc.nxc](#).

8.1.2.178 #define DISPLAY_MENUICONS_X_DIFF 31**8.1.2.179 #define DISPLAY_MENUICONS_X_OFFSET 7****8.1.2.180 #define DISPLAY_MENUICONS_Y 40****8.1.2.181 #define DISPLAY_ON 0x01**

W - Display on

8.1.2.182 #define DISPLAY_PIXEL 0x01

W - set pixel (on/off) (CMD,TRUE/FALSE,X,Y,x,x)

8.1.2.183 #define DISPLAY_POPUP 0x08

W - Use popup display memory

Examples:

[ex_dispmiss.nxc.](#)

8.1.2.184 #define DISPLAY_REFRESH 0x02

W - Enable refresh

8.1.2.185 #define DISPLAY_REFRESH_DISABLED 0x40

R - Refresh disabled

8.1.2.186 #define DISPLAY_VERTICAL_LINE 0x03

W - draw vertical line (CMD,TRUE/FALSE,X1,Y1,x,Y2)

8.1.2.187 #define DISPLAY_WIDTH 100

The width of the LCD screen in pixels

Examples:

[ex_LineOut.nxc.](#)

8.1.2.188 #define DisplayExecuteFunction 80

Execute one of the Display module's internal functions

8.1.2.189 #define DisplayModuleID 0x000A0001

The display module ID

8.1.2.190 #define DisplayModuleName "Display.mod"

The display module name

8.1.2.191 #define DisplayOffsetContrast 1719

Adjust the display contrast with this field

8.1.2.192 #define DisplayOffsetDisplay 104

Display content copied to physical display every 17 mS

8.1.2.193 #define DisplayOffsetEraseMask 4

Section erase mask (executed first)

8.1.2.194 #define DisplayOffsetFlags 117

Update flags enumerated above

8.1.2.195 #define DisplayOffsetNormal(l, w) (((l)*100)+(w)+119)

Raw display memory for normal screen

8.1.2.196 #define DisplayOffsetPBitmaps(p) (((p)*4)+68)

Pointer to free bitmap files

8.1.2.197 #define DisplayOffsetPFont 12

Pointer to font file

8.1.2.198 #define DisplayOffsetPFunc 0

Simple draw entry

8.1.2.199 #define DisplayOffsetPMenuIcons(p) (((p)*4)+88)

Pointer to menu icon images (NULL == none)

8.1.2.200 #define DisplayOffsetPMenuText 84

Pointer to menu icon text (NULL == none)

8.1.2.201 #define DisplayOffsetPopup(l, w) (((l)*100)+(w)+919)

Raw display memory for popup screen

8.1.2.202 #define DisplayOffsetPScreens(p) (((p)*4)+56)

Pointer to screen bitmap file

8.1.2.203 #define DisplayOffsetPStatusIcons 52

Pointer to status icon collection file

8.1.2.204 #define DisplayOffsetPStatusText 48

Pointer to status text string

8.1.2.205 #define DisplayOffsetPStepIcons 100

Pointer to step icon collection file

8.1.2.206 #define DisplayOffsetPTextLines(p) (((p)*4)+16)

Pointer to text strings

8.1.2.207 #define DisplayOffsetStatusIcons(p) ((p)+108)

Index in status icon collection file (index = 0 -> none)

8.1.2.208 #define DisplayOffsetStepIcons(p) ((p)+112)

Index in step icon collection file (index = 0 -> none)

8.1.2.209 #define DisplayOffsetTextLinesCenterFlags 118

Mask to center TextLines

8.1.2.210 #define DisplayOffsetUpdateMask 8

Section update mask (executed next)

8.1.2.211 #define DIST_CMD_CUSTOM 0x35

Set the DIST-Nx to a custom mode

8.1.2.212 #define DIST_CMD_GP2D12 0x31

Set the DIST-Nx to GP2D12 mode

8.1.2.213 #define DIST_CMD_GP2D120 0x32

Set the DIST-Nx to GP2D120 mode

8.1.2.214 #define DIST_CMD_GP2YA02 0x34

Set the DIST-Nx to GP2YA02 mode

8.1.2.215 #define DIST_CMD_GP2YA21 0x33

Set the DIST-Nx to GP2YA21 mode

8.1.2.216 #define DIST_REG_DIST 0x42

The DIST-Nx distance register

8.1.2.217 #define DIST_REG_DIST1 0x58

The DIST-Nx distance 1 register

8.1.2.218 #define DIST_REG_DIST_MAX 0x54

The DIST-Nx maximum distance register

8.1.2.219 #define DIST_REG_DIST_MIN 0x52

The DIST-Nx minimum distance register

8.1.2.220 #define DIST_REG_MODULE_TYPE 0x50

The DIST-Nx module type register

8.1.2.221 #define DIST_REG_NUM_POINTS 0x51

The DIST-Nx number of data points in Custom curve register

8.1.2.222 #define DIST_REG_VOLT 0x44

The DIST-Nx voltage register

8.1.2.223 #define DIST_REG_VOLT1 0x56

The DIST-Nx voltage 1 register

8.1.2.224 #define DRAW_OPT_CLEAR (0x0004)

Clear pixels while drawing (aka draw in white)

**8.1.2.225 #define DRAW_OPT_CLEAR_EXCEPT_STATUS_-
SCREEN (0x0002)**

Clear the screen except for the status line before drawing

8.1.2.226 #define DRAW_OPT_CLEAR_PIXELS (0x0004)

Clear pixels while drawing (aka draw in white)

8.1.2.227 #define DRAW_OPT_CLEAR_SCREEN_MODES (0x0003)

Bit mask for the clear screen modes

8.1.2.228 #define DRAW_OPT_CLEAR_WHOLE_SCREEN (0x0001)

Clear the entire screen before drawing

Examples:

[ex_dispgoutex.nxc](#).

8.1.2.229 #define DRAW_OPT_FILL_SHAPE (0x0020)

Fill the shape while drawing (rectangle, circle, ellipses, and polygon)

Examples:

[ex_CircleOut.nxc](#), [ex_EllipseOut.nxc](#), [ex_PolyOut.nxc](#), [ex_sysdrawellipse.nxc](#),
and [ex_sysdrawpolygon.nxc](#).

8.1.2.230 #define DRAW_OPT_FONT_DIR_B2TL (0x0100)

Font bottom to top left align

8.1.2.231 #define DRAW_OPT_FONT_DIR_B2TR (0x0140)

Font bottom to top right align

8.1.2.232 #define DRAW_OPT_FONT_DIR_L2RB (0x0000)

Font left to right bottom align

Examples:

[ex_dispftout.nxc](#).

8.1.2.233 #define DRAW_OPT_FONT_DIR_L2RT (0x0040)

Font left to right top align

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

8.1.2.234 #define DRAW_OPT_FONT_DIR_R2LB (0x0080)

Font right to left bottom align

8.1.2.235 #define DRAW_OPT_FONT_DIR_R2LT (0x00C0)

Font right to left top align

8.1.2.236 #define DRAW_OPT_FONT_DIR_T2BL (0x0180)

Font top to bottom left align

Examples:

[ex_dispftout.nxc.](#)

8.1.2.237 #define DRAW_OPT_FONT_DIR_T2BR (0x01C0)

Font top to bottom right align

8.1.2.238 #define DRAW_OPT_FONT_DIRECTIONS (0x01C0)

Bit mask for the font direction bits

8.1.2.239 #define DRAW_OPT_FONT_WRAP (0x0200)

Option to have text wrap in [FontNumOut](#) and [FontTextOut](#) calls

Examples:

[ex_dispftout.nxc.](#)

8.1.2.240 #define DRAW_OPT_INVERT (0x0004)

Invert text or graphics

Examples:

[ex_dispftout.nxc.](#)

8.1.2.241 #define DRAW_OPT_LOGICAL_AND (0x0008)

Draw pixels using a logical AND operation

Examples:

[ex_dispftout.nxc.](#)

8.1.2.242 #define DRAW_OPT_LOGICAL_COPY (0x0000)

Draw pixels using a logical copy operation

8.1.2.243 #define DRAW_OPT_LOGICAL_OPERATIONS (0x0018)

Bit mask for the logical drawing operations

8.1.2.244 #define DRAW_OPT_LOGICAL_OR (0x0010)

Draw pixels using a logical OR operation

Examples:

[ex_dispftout.nxc](#).

8.1.2.245 #define DRAW_OPT_LOGICAL_XOR (0x0018)

Draw pixels using a logical XOR operation

Examples:

[ex_CircleOut.nxc](#), [ex_EllipseOut.nxc](#), [ex_LineOut.nxc](#), [ex_PolyOut.nxc](#), [ex_sysdrawellipse.nxc](#), and [ex_sysdrawpolygon.nxc](#).

8.1.2.246 #define DRAW_OPT_NORMAL (0x0000)

Normal drawing

Examples:

[ex_CircleOut.nxc](#), [ex_dispftout.nxc](#), [ex_disfunc.nxc](#), and [ex_sysdrawfont.nxc](#).

8.1.2.247 #define DRAW_OPT_POLYGON_POLYLINE (0x0400)

When drawing polygons, do not close (i.e., draw a polyline instead)

8.1.2.248 #define DrawCircle 16

Draw a circle on the LCD screen

8.1.2.249 #define DrawEllipse 94

Draw an ellipse on the LCD screen

8.1.2.250 #define DrawFont 95

Draw text using a custom RIC-based font to the LCD screen

8.1.2.251 #define DrawGraphic 18

Draw a graphic image on the LCD screen

8.1.2.252 #define DrawGraphicArray 92

Draw a graphic image from a byte array to the LCD screen

Examples:

[ex_dispgout.nxc](#).

8.1.2.253 #define DrawLine 15

Draw a line on the LCD screen

8.1.2.254 #define DrawPoint 14

Draw a single pixel on the LCD screen

8.1.2.255 #define DrawPolygon 93

Draw a polygon on the LCD screen

8.1.2.256 #define DrawRect 17

Draw a rectangle on the LCD screen

8.1.2.257 #define DrawText 13

Draw text to one of 8 LCD lines

Examples:

[ex_syscall.nxc](#).

8.1.2.258 #define EMETER_REG_AIN 0x0c

The register address for amps in

8.1.2.259 #define EMETER_REG_AOUT 0x10

The register address for amps out

8.1.2.260 #define EMETER_REG_JOULES 0x12

The register address for joules

8.1.2.261 #define EMETER_REG_VIN 0x0a

The register address for voltage in

8.1.2.262 #define EMETER_REG_VOUT 0x0e

The register address for voltage out

8.1.2.263 #define EMETER_REG_WIN 0x14

The register address for watts in

8.1.2.264 #define EMETER_REG_WOUT 0x16

The register address for watts out

8.1.2.265 #define EOF -1

A constant representing end of file

8.1.2.266 #define ERR_ARG -1

0xFF Bad arguments

8.1.2.267 #define ERR_BAD_POOL_SIZE -10

0xF6 VarsCmd.PoolSize > POOL_MAX_SIZE

8.1.2.268 #define ERR_BAD_PTR -6

0xFA Someone passed us a bad pointer!

8.1.2.269 #define ERR_CLUMP_COUNT -7

0xF9 (FileClumpCount == 0 || FileClumpCount >= NOT_A_CLUMP)

8.1.2.270 #define ERR_COMM_BUFFER_FULL -34

0xDE No room in comm buffer

8.1.2.271 #define ERR_COMM_BUS_ERR -35

0xDD Something went wrong on the communications bus

8.1.2.272 #define ERR_COMM_CHAN_INVALID -33

0xDF Specified channel/connection is not valid

8.1.2.273 #define ERR_COMM_CHAN_NOT_READY -32

0xE0 Specified channel/connection not configured or busy

8.1.2.274 #define ERR_DEFAULT_OFFSETS -14

0xF2 (DefaultsOffset != FileOffsets.DynamicDefaults) || (DefaultsOffset + FileOffsets.DynamicDefaultsSize != FileOffsets.DSDefaultsSize)

8.1.2.275 #define ERR_FILE -3

0xFD Malformed file contents

8.1.2.276 #define ERR_INSANE_OFFSET -9

0xF7 CurrOffset != (DataSize - VarsCmd.CodespaceCount * 2)

8.1.2.277 #define ERR_INSTR -2

0xFE Illegal bytecode instruction

8.1.2.278 #define ERR_INVALID_FIELD -17

0xEF Attempted to access invalid field of a structure

8.1.2.279 #define ERR_INVALID_PORT -16

0xF0 Bad input or output port specified

8.1.2.280 #define ERR_INVALID_QUEUE -18

0xEE Illegal queue ID specified

8.1.2.281 #define ERR_INVALID_SIZE -19

0xED Illegal size specified

8.1.2.282 #define ERR_LOADER_ERR -11

0xF5 LOADER_ERR(LStatus) != SUCCESS || pData == NULL || DataSize == 0

8.1.2.283 #define ERR_MEM -5

0xFB Insufficient memory available

8.1.2.284 #define ERR_MEMMGR_FAIL -15

0xF1 (UBYTE *)VarsCmd.MemMgr.pDopeVectorArray != VarsCmd.pDataspace + DV_ARRAY[0].Offset

8.1.2.285 #define ERR_NO_ACTIVE_CLUMP -13

0xF3 VarsCmd.RunQ.Head == NOT_A_CLUMP

8.1.2.286 #define ERR_NO_CODE -8

0xF8 VarsCmd.CodespaceCount == 0

8.1.2.287 #define ERR_NO_PROG -20

0xEC No active program

8.1.2.288 #define ERR_NON_FATAL -16

Fatal errors are greater than this value

8.1.2.289 #define ERR_RC_BAD_PACKET -65

0xBF Clearly insane packet

8.1.2.290 #define ERR_RC_FAILED -67

0xBD Request failed (i.e. specified file not found)

8.1.2.291 #define ERR_RC_ILLEGAL_VAL -64

0xC0 Data contains out-of-range values

8.1.2.292 #define ERR_RC_UNKNOWN_CMD -66

0xBE Unknown command opcode

8.1.2.293 #define ERR_SPOTCHECK_FAIL -12

0xF4 ((UBYTE*)(VarsCmd.pCodespace) < pData) (c_cmd.c 1893)

8.1.2.294 #define ERR_VER -4

0xFC Version mismatch between firmware and compiler

8.1.2.295 #define FALSE 0

A false value

8.1.2.296 #define FileClose 5

Close the specified file

8.1.2.297 #define FileDelete 8

Delete a file

8.1.2.298 #define FileFindFirst 83

Start a search for a file using a filename pattern

8.1.2.299 #define FileFindNext 84

Continue searching for a file

8.1.2.300 #define FileOpenAppend 2

Open a file for appending to the end of the file

8.1.2.301 #define FileOpenRead 0

Open a file for reading

8.1.2.302 #define FileOpenReadLinear 87

Open a linear file for reading

8.1.2.303 #define FileOpenWrite 1

Open a file for writing (creates a new file)

8.1.2.304 #define FileOpenWriteLinear 85

Open a linear file for writing

8.1.2.305 #define FileOpenWriteNonLinear 86

Open a non-linear file for writing

8.1.2.306 #define FileRead 3

Read from the specified file

8.1.2.307 #define FileRename 7

Rename a file

8.1.2.308 #define FileResize 91

Resize a file (not yet implemented)

8.1.2.309 #define FileResolveHandle 6

Get a file handle for the specified filename if it is already open

8.1.2.310 #define FileSeek 90

Seek to a specific position in an open file

8.1.2.311 #define FileWrite 4

Write to the specified file

8.1.2.312 #define FRAME_SELECT 0

Center icon select frame

8.1.2.313 #define FREQUENCY_MAX 14080

Maximum frequency [Hz]

8.1.2.314 #define FREQUENCY_MIN 220

Minimum frequency [Hz]

8.1.2.315 #define GetStartTick 25

Get the current system tick count

8.1.2.316 #define GL_CAMERA_DEPTH 3

Set the camera depth.

8.1.2.317 #define GL_CIRCLE 4

Use circle mode.

Examples:

[glCircleDemo.nxc](#).

8.1.2.318 #define GL_CIRCLE_SIZE 1

Set the circle size.

8.1.2.319 #define GL_CULL_BACK 2

Cull lines in back.

8.1.2.320 #define GL_CULL_FRONT 3

Cull lines in front.

8.1.2.321 #define GL_CULL_MODE 2

Set the cull mode.

Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.1.2.322 #define GL_CULL_NONE 4

Do not cull any lines.

Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.1.2.323 #define GL_LINE 2

Use line mode.

8.1.2.324 #define GL_POINT 3

Use point mode.

8.1.2.325 #define GL_POLYGON 1

Use polygon mode.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.1.2.326 #define GL_ROTATE_X 4

Rotate around the X axis.

Examples:

[glRotateDemo.nxc](#).

8.1.2.327 #define GL_ROTATE_Y 5

Rotate around the Y axis.

Examples:

[glRotateDemo.nxc](#).

8.1.2.328 #define GL_ROTATE_Z 6

Rotate around the Z axis.

8.1.2.329 #define GL_SCALE_X 7

Scale along the X axis.

Examples:

[glScaleDemo.nxc](#).

8.1.2.330 #define GL_SCALE_Y 8

Scale along the Y axis.

8.1.2.331 #define GL_SCALE_Z 9

Scale along the Z axis.

8.1.2.332 #define GL_TRANSLATE_X 1

Translate along the X axis.

Examples:

[glBoxDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.1.2.333 #define GL_TRANSLATE_Y 2

Translate along the Y axis.

Examples:

[glTranslateDemo.nxc](#).

8.1.2.334 #define GL_TRANSLATE_Z 3

Translate along the Z axis.

Examples:

[glTranslateDemo.nxc](#).

8.1.2.335 #define GL_ZOOM_FACTOR 4

Set the zoom factor.

8.1.2.336 #define HS_BAUD_115200 12

HsMode 115200 Baud

Examples:

[ex_RS485Uart.nxc](#).

8.1.2.337 #define HS_BAUD_1200 0

HsMode 1200 Baud

8.1.2.338 #define HS_BAUD_14400 6

HsMode 14400 Baud

8.1.2.339 #define HS_BAUD_19200 7

HsMode 19200 Baud

8.1.2.340 #define HS_BAUD_230400 13

HsMode 230400 Baud

8.1.2.341 #define HS_BAUD_2400 1

HsMode 2400 Baud

8.1.2.342 #define HS_BAUD_28800 8

HsMode 28800 Baud

8.1.2.343 #define HS_BAUD_3600 2

HsMode 3600 Baud

8.1.2.344 #define HS_BAUD_38400 9

HsMode 38400 Baud

8.1.2.345 #define HS_BAUD_460800 14

HsMode 460800 Baud

8.1.2.346 #define HS_BAUD_4800 3

HsMode 4800 Baud

8.1.2.347 #define HS_BAUD_57600 10

HsMode 57600 Baud

8.1.2.348 #define HS_BAUD_7200 4

HsMode 7200 Baud

8.1.2.349 #define HS_BAUD_76800 11

HsMode 76800 Baud

8.1.2.350 #define HS_BAUD_921600 15

HsMode 921600 Baud

8.1.2.351 #define HS_BAUD_9600 5

HsMode 9600 Baud

Examples:

[ex_RS485Control.nxc.](#)

8.1.2.352 #define HS_CMD_READY 0x04

A constant representing high speed direct command

8.1.2.353 #define HS_CTRL_EXIT 2

Ddisable the high speed port

8.1.2.354 #define HS_CTRL_INIT 0

Enable the high speed port

Examples:

[ex_SysCommHSControl.nxc.](#)

8.1.2.355 #define HS_CTRL_UART 1

Setup the high speed port UART configuration

Examples:

[ex_RS485Control.nxc](#).

8.1.2.356 #define HS_DISABLE 4

HsState disable

8.1.2.357 #define HS_ENABLE 5

HsState enable

8.1.2.358 #define HS_INIT_RECEIVER 2

HsState initialize receiver

8.1.2.359 #define HS_INITIALISE 1

HsState initialize

8.1.2.360 #define HS_MODE_10_STOP 0x0000

HsMode 1 stop bit

8.1.2.361 #define HS_MODE_15_STOP 0x1000

HsMode 1.5 stop bits

8.1.2.362 #define HS_MODE_20_STOP 0x2000

HsMode 2 stop bits

8.1.2.363 #define HS_MODE_5_DATA 0x0000

HsMode 5 data bits

8.1.2.364 #define HS_MODE_6_DATA 0x0040

HsMode 6 data bits

8.1.2.365 #define HS_MODE_7_DATA 0x0080

HsMode 7 data bits

**8.1.2.366 #define HS_MODE_7E1 (HS_MODE_7_DATA|HS_MODE_E_-
PARITY|HS_MODE_10_STOP)**

HsMode 7 data bits, even parity, 1 stop bit

8.1.2.367 #define HS_MODE_8_DATA 0x00C0

HsMode 8 data bits

**8.1.2.368 #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](#).

8.1.2.369 #define HS_MODE_E_PARITY 0x0000

HsMode Even parity

8.1.2.370 #define HS_MODE_M_PARITY 0x0600

HsMode Mark parity

8.1.2.371 #define HS_MODE_N_PARITY 0x0800

HsMode No parity

8.1.2.372 #define HS_MODE_O_PARITY 0x0200

HsMode Odd parity

8.1.2.373 #define HS_MODE_S_PARITY 0x0400

HsMode Space parity

8.1.2.374 #define HS_SEND_DATA 3

HsState send data

8.1.2.375 #define HS_UPDATE 1

HsFlags high speed update required

8.1.2.376 #define HT_ADDR_ACCEL 0x02

HiTechnic Accel I2C address

8.1.2.377 #define HT_ADDR_COLOR 0x02

HiTechnic Color I2C address

8.1.2.378 #define HT_ADDR_COLOR2 0x02

HiTechnic Color2 I2C address

8.1.2.379 #define HT_ADDR_COMPASS 0x02

HiTechnic Compass I2C address

8.1.2.380 #define HT_ADDR_IRLINK 0x02

HiTechnic IRLink I2C address

8.1.2.381 #define HT_ADDR_IRRECEIVER 0x02

HiTechnic IRReceiver I2C address

8.1.2.382 #define HT_ADDR_IRSEEKER 0x02

HiTechnic IRSeeker I2C address

8.1.2.383 #define HT_ADDR_IRSEEKER2 0x10

HiTechnic IRSeeker2 I2C address

8.1.2.384 #define HT_CH1_A 0

Use IRReceiver channel 1 output A

Examples:

[ex_ReadSensorHTIRReceiverEx.nxc.](#)

8.1.2.385 #define HT_CH1_B 1

Use IRReceiver channel 1 output B

8.1.2.386 #define HT_CH2_A 2

Use IRReceiver channel 2 output A

8.1.2.387 #define HT_CH2_B 3

Use IRReceiver channel 2 output B

8.1.2.388 #define HT_CH3_A 4

Use IRReceiver channel 3 output A

8.1.2.389 #define HT_CH3_B 5

Use IRReceiver channel 3 output B

8.1.2.390 #define HT_CH4_A 6

Use IRReceiver channel 4 output A

8.1.2.391 #define HT_CH4_B 7

Use IRReceiver channel 4 output B

8.1.2.392 #define HT_CMD_COLOR2_50HZ 0x35

Set the Color2 sensor to 50Hz mode

8.1.2.393 #define HT_CMD_COLOR2_60HZ 0x36

Set the Color2 sensor to 60Hz mode

8.1.2.394 #define HT_CMD_COLOR2_ACTIVE 0x00

Set the Color2 sensor to active mode

Examples:

[ex_I2CSendCommand.nxc](#), and [ex_SetHTColor2Mode.nxc](#).

8.1.2.395 #define HT_CMD_COLOR2_BLCAL 0x42

Set the Color2 sensor to black level calibration mode

8.1.2.396 #define HT_CMD_COLOR2_FAR 0x46

Set the Color2 sensor to far mode

8.1.2.397 #define HT_CMD_COLOR2_LED_HI 0x48

Set the Color2 sensor to LED high mode

8.1.2.398 #define HT_CMD_COLOR2_LED_LOW 0x4C

Set the Color2 sensor to LED low mode

8.1.2.399 #define HT_CMD_COLOR2_NEAR 0x4E

Set the Color2 sensor to near mode

8.1.2.400 #define HT_CMD_COLOR2_PASSIVE 0x01

Set the Color2 sensor to passive mode

8.1.2.401 #define HT_CMD_COLOR2_RAW 0x03

Set the Color2 sensor to raw mode

8.1.2.402 #define HT_CMD_COLOR2_WBCAL 0x43

Set the Color2 sensor to white level calibration mode

8.1.2.403 #define HTIR2_MODE_1200 0

Set IRSeeker2 to 1200 mode

Examples:

[ex_SetHTIRSeeker2Mode.nxc](#), and [ex_SetSensorBoolean.nxc](#).

8.1.2.404 #define HTIR2_MODE_600 1

Set IRSeeker2 to 600 mode

8.1.2.405 #define HTIR2_REG_AC01 0x4A

IRSeeker2 AC 01 register

8.1.2.406 #define HTIR2_REG_AC02 0x4B

IRSeeker2 AC 02 register

8.1.2.407 #define HTIR2_REG_AC03 0x4C

IRSeeker2 AC 03 register

8.1.2.408 #define HTIR2_REG_AC04 0x4D

IRSeeker2 AC 04 register

8.1.2.409 #define HTIR2_REG_AC05 0x4E

IRSeeker2 AC 05 register

8.1.2.410 #define HTIR2_REG_ACDIR 0x49

IRSeeker2 AC direction register

8.1.2.411 #define HTIR2_REG_DC01 0x43

IRSeeker2 DC 01 register

8.1.2.412 #define HTIR2_REG_DC02 0x44

IRSeeker2 DC 02 register

8.1.2.413 #define HTIR2_REG_DC03 0x45

IRSeeker2 DC 03 register

8.1.2.414 #define HTIR2_REG_DC04 0x46

IRSeeker2 DC 04 register

8.1.2.415 #define HTIR2_REG_DC05 0x47

IRSeeker2 DC 05 register

8.1.2.416 #define HTIR2_REG_DCAVG 0x48

IRSeeker2 DC average register

Examples:

[ex_SensorHTIRSeeker2Addr.nxc](#).

8.1.2.417 #define HTIR2_REG_DCDIR 0x42

IRSeeker2 DC direction register

8.1.2.418 #define HTIR2_REG_MODE 0x41

IRSeeker2 mode register

8.1.2.419 #define I2C_ADDR_DEFAULT 0x02

Standard NXT I2C device address

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CSendCommand.nxc](#),
[ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_MSDeenergize.nxc](#), [ex_MSEnergize.nxc](#),
[ex_MSIRTrain.nxc](#), [ex_MSPFComboDirect.nxc](#), [ex_MSPFComboPWM.nxc](#),
[ex_MSPFRawOutput.nxc](#), [ex_MSPFRepeat.nxc](#),
[ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), [ex_MSPFSinglePin.nxc](#),
[ex_MSPFTrain.nxc](#), [ex_MSReadValue.nxc](#), [ex_readi2register.nxc](#), and [ex_writei2register.nxc](#).

8.1.2.420 #define I2C_REG_CMD 0x41

Standard NXT I2C device command register

Examples:

[ex_MSReadValue.nxc](#), [ex_readi2register.nxc](#), and [ex_writei2register.nxc](#).

8.1.2.421 #define I2C_REG_DEVICE_ID 0x10

Standard NXT I2C device ID register

Examples:

[ex_I2CDeviceInfo.nxc](#).

8.1.2.422 #define I2C_REG_VENDOR_ID 0x08

Standard NXT I2C vendor ID register

Examples:

[ex_I2CDeviceInfo.nxc](#).

8.1.2.423 #define I2C_REG_VERSION 0x00

Standard NXT I2C version register

Examples:

[ex_I2CDeviceInfo.nxc](#).

8.1.2.424 #define IN_1 0x00

Input port 1

8.1.2.425 #define IN_2 0x01

Input port 2

8.1.2.426 #define IN_3 0x02

Input port 3

8.1.2.427 #define IN_4 0x03

Input port 4

8.1.2.428 #define IN_MODE_ANGLESTEP 0xE0

RCX rotation sensor (16 ticks per revolution)

8.1.2.429 #define IN_MODE_BOOLEAN 0x20

Boolean value (0 or 1)

8.1.2.430 #define IN_MODE_CELSIUS 0xA0

RCX temperature sensor value in degrees celcius

8.1.2.431 #define IN_MODE_FAHRENHEIT 0xC0

RCX temperature sensor value in degrees fahrenheit

8.1.2.432 #define IN_MODE_MODEMASK 0xE0

Mask for the mode without any slope value

8.1.2.433 #define IN_MODE_PCTFULLSCALE 0x80

Scaled value from 0 to 100

8.1.2.434 #define IN_MODE_PERIODCOUNTER 0x60

Counts the number of boolean periods

8.1.2.435 #define IN_MODE_RAW 0x00

Raw value from 0 to 1023

8.1.2.436 #define IN_MODE_SLOPEMASK 0x1F

Mask for slope parameter added to mode

8.1.2.437 #define IN_MODE_TRANSITIONCNT 0x40

Counts the number of boolean transitions

8.1.2.438 #define IN_TYPE_ANGLE 0x04

RCX rotation sensor

8.1.2.439 #define IN_TYPE_COLORBLUE 0x10

NXT 2.0 color sensor with blue light

8.1.2.440 #define IN_TYPE_COLOREXIT 0x12

NXT 2.0 color sensor internal state

8.1.2.441 #define IN_TYPE_COLORFULL 0x0D

NXT 2.0 color sensor in full color mode

8.1.2.442 #define IN_TYPE_COLORGREEN 0x0F

NXT 2.0 color sensor with green light

8.1.2.443 #define IN_TYPE_COLORNONE 0x11

NXT 2.0 color sensor with no light

8.1.2.444 #define IN_TYPE_COLORRED 0x0E

NXT 2.0 color sensor with red light

8.1.2.445 #define IN_TYPE_CUSTOM 0x09

NXT custom sensor

8.1.2.446 #define IN_TYPE_HISPEED 0x0C

NXT Hi-speed port (only S4)

8.1.2.447 #define IN_TYPE_LIGHT_ACTIVE 0x05

NXT light sensor with light

8.1.2.448 #define IN_TYPE_LIGHT_INACTIVE 0x06

NXT light sensor without light

8.1.2.449 #define IN_TYPE_LOWSPEED 0x0A

NXT I2C digital sensor

8.1.2.450 #define IN_TYPE_LOWSPEED_9V 0x0B

NXT I2C digital sensor with 9V power

8.1.2.451 #define IN_TYPE_NO_SENSOR 0x00

No sensor configured

8.1.2.452 #define IN_TYPE_REFLECTION 0x03

RCX light sensor

8.1.2.453 #define IN_TYPE_SOUND_DB 0x07

NXT sound sensor with dB scaling

8.1.2.454 #define IN_TYPE_SOUND_DBIA 0x08

NXT sound sensor with dBA scaling

8.1.2.455 #define IN_TYPE_SWITCH 0x01

NXT or RCX touch sensor

8.1.2.456 #define IN_TYPE_TEMPERATURE 0x02

RCX temperature sensor

8.1.2.457 #define INPUT_BLACKCOLOR 1

The color value is black

8.1.2.458 #define INPUT_BLANK 3

Access the blank value from color sensor value arrays

8.1.2.459 #define INPUT_BLUE 2

Access the blue value from color sensor value arrays

8.1.2.460 #define INPUT_BLUECOLOR 2

The color value is blue

8.1.2.461 #define INPUT_CAL_POINT_0 0

Calibration point 0

Examples:

[ex_ColorCalibration.nxc](#), and [ex_ColorCalLimits.nxc](#).

8.1.2.462 #define INPUT_CAL_POINT_1 1

Calibration point 1

8.1.2.463 #define INPUT_CAL_POINT_2 2

Calibration point 2

8.1.2.464 #define INPUT_CUSTOM9V 0x01

Custom sensor 9V

8.1.2.465 #define INPUT_CUSTOMACTIVE 0x02

Custom sensor active

8.1.2.466 #define INPUT_CUSTOMINACTIVE 0x00

Custom sensor inactive

8.1.2.467 #define INPUT_DIGI0 1

Digital pin 0

8.1.2.468 #define INPUT_DIGI1 2

Digital pin 1

8.1.2.469 #define INPUT_GREEN 1

Access the green value from color sensor value arrays

8.1.2.470 #define INPUT_GREENCOLOR 3

The color value is green

8.1.2.471 #define INPUT_INVALID_DATA 0x01

Invalid data flag

8.1.2.472 #define INPUT_NO_OF_COLORS 4

The number of entries in the color sensor value arrays

8.1.2.473 #define INPUT_NO_OF_POINTS 3

The number of calibration points

8.1.2.474 #define INPUT_RED 0

Access the red value from color sensor value arrays

Examples:

[ex_ColorADRaw.nxc](#), [ex_ColorBoolean.nxc](#), [ex_ColorCalibration.nxc](#), [ex_ColorSensorRaw.nxc](#), and [ex_ColorSensorValue.nxc](#).

8.1.2.475 #define INPUT_REDCOLOR 5

The color value is red

8.1.2.476 #define INPUT_RESETCAL 0x80

Unused calibration state constant

8.1.2.477 #define INPUT_RUNNINGCAL 0x20

Unused calibration state constant

8.1.2.478 #define INPUT_SENSORCAL 0x01

The state returned while the color sensor is calibrating

8.1.2.479 #define INPUT_SENSOROFF 0x02

The state returned once calibration has completed

8.1.2.480 #define INPUT_STARTCAL 0x40

Unused calibration state constant

8.1.2.481 #define INPUT_WHITECOLOR 6

The color value is white

8.1.2.482 #define INPUT_YELLOWCOLOR 4

The color value is yellow

8.1.2.483 #define InputMode 1

Input mode field. Contains one of the sensor mode constants. Read/write.

8.1.2.484 #define InputModuleID 0x00030001

The input module ID

8.1.2.485 #define InputModuleName "Input.mod"

The input module name.

8.1.2.486 #define InputOffsetADRaw(p) (((p)*20)+2)

Read the AD raw sensor value (2 bytes) uword

8.1.2.487 #define InputOffsetColorADRaw(p, nc) (80+((p)*84)+52+((nc)*2))

Read AD raw color sensor values

8.1.2.488 #define InputOffsetColorCalibration(p, np, nc) (80+((p)*84)+0+((np)*16)+((nc)*4))

Read/write color calibration point values

8.1.2.489 #define InputOffsetColorCalibrationState(p) (80+((p)*84)+80)

Read color sensor calibration state

8.1.2.490 #define InputOffsetColorCalLimits(p, np) (80+((p)*84)+48+((np)*2))

Read/write color calibration limits

**8.1.2.491 #define InputOffsetColorSensorBoolean(p,
nc) (80+((p)*84)+76+((nc)*2))**

Read color sensor boolean values

8.1.2.492 #define InputOffsetColorSensorRaw(p, nc) (80+((p)*84)+60+((nc)*2))

Read raw color sensor values

**8.1.2.493 #define InputOffsetColorSensorValue(p,
nc) (80+((p)*84)+68+((nc)*2))**

Read scaled color sensor values

8.1.2.494 #define InputOffsetCustomActiveStatus(p) (((p)*20)+15)

Read/write the active or inactive state of the custom sensor

8.1.2.495 #define InputOffsetCustomPctFullScale(p) (((p)*20)+14)

Read/write the Pct full scale of the custom sensor

8.1.2.496 #define InputOffsetCustomZeroOffset(p) (((p)*20)+0)

Read/write the zero offset of a custom sensor (2 bytes) uword

8.1.2.497 #define InputOffsetDigiPinsDir(p) (((p)*20)+11)

Read/write the direction of the Digital pins (1 is output, 0 is input)

8.1.2.498 #define InputOffsetDigiPinsIn(p) (((p)*20)+12)

Read/write the status of the digital pins

8.1.2.499 #define InputOffsetDigiPinsOut(p) (((p)*20)+13)

Read/write the output level of the digital pins

8.1.2.500 #define InputOffsetInvalidData(p) (((p)*20)+16)

Indicates whether data is invalid (1) or valid (0)

8.1.2.501 #define InputOffsetSensorBoolean(p) (((p)*20)+10)

Read the sensor boolean value

8.1.2.502 #define InputOffsetSensorMode(p) (((p)*20)+9)

Read/write the sensor mode

8.1.2.503 #define InputOffsetSensorRaw(p) (((p)*20)+4)

Read the raw sensor value (2 bytes) uword

8.1.2.504 #define InputOffsetSensorType(p) (((p)*20)+8)

Read/write the sensor type

8.1.2.505 #define InputOffsetSensorValue(p) (((p)*20)+6)

Read/write the scaled sensor value (2 bytes) sword

8.1.2.506 #define INT_MAX 32767

The maximum value of the int type

8.1.2.507 #define INT_MIN -32767

The minimum value of the int type

8.1.2.508 #define INTF_BTOFF 13

Turn off the bluetooth radio

Examples:

[ex_syscommexecufunction.nxc](#).

8.1.2.509 #define INTF_BTTON 12

Turn on the bluetooth radio

8.1.2.510 #define INTF_CONNECT 3

Connect to one of the known devices

8.1.2.511 #define INTF_CONNECTBYNAME 18

Connect to a bluetooth device by name

8.1.2.512 #define INTF_CONNECTREQ 17

Connection request from another device

8.1.2.513 #define INTF_DISCONNECT 4

Disconnect from one of the connected devices

8.1.2.514 #define INTF_DISCONNECTALL 5

Disconnect all devices

8.1.2.515 #define INTF_EXTREAD 15

External read request

8.1.2.516 #define INTF_FACTORYRESET 11

Reset bluetooth settings to factory values

8.1.2.517 #define INTF_OPENSTREAM 9

Open a bluetooth stream

8.1.2.518 #define INTF_PINREQ 16

Bluetooth PIN request

8.1.2.519 #define INTF_REMOVEDEVICE 6

Remove a device from the known devices table

8.1.2.520 #define INTF_SEARCH 1

Search for bluetooth devices

8.1.2.521 #define INTF_SENDDATA 10

Send data over a bluetooth connection

8.1.2.522 #define INTF_SENDFILE 0

Send a file via bluetooth to another device

8.1.2.523 #define INTF_SETBTNAME 14

Set the bluetooth name

8.1.2.524 #define INTF_SETCMDMODE 8

Set bluetooth into command mode

8.1.2.525 #define INTF_STOPSEARCH 2

Stop searching for bluetooth devices

8.1.2.526 #define INTF_VISIBILITY 7

Set the bluetooth visibility on or off

8.1.2.527 #define InvalidData 5

Invalid data field. Contains a boolean value indicating whether the sensor data is valid or not. Read/write.

8.1.2.528 #define IOCTRL_BOOT 0xA55A

Reboot the NXT into SAMBA mode

8.1.2.529 #define IOCTRL_POWERDOWN 0x5A00

Power down the NXT

8.1.2.530 #define IOCtlModuleID 0x00060001

The IOCtl module ID

8.1.2.531 #define IOCtlModuleName "IOCtl.mod"

The IOCtl module name

8.1.2.532 #define IOCtlOffsetPowerOn 0

Offset to power on field

8.1.2.533 #define IOMapRead 32

Read data from one of the firmware module's IOMap structures using the module's name

8.1.2.534 #define IOMapReadByID 78

Read data from one of the firmware module's IOMap structures using the module's ID

8.1.2.535 #define IOMapWrite 33

Write data to one of the firmware module's IOMap structures using the module's name

8.1.2.536 #define IOMapWriteByID 79

Write data to one of the firmware module's IOMap structures using the module's ID

8.1.2.537 #define KeepAlive 31

Reset the NXT sleep timer

8.1.2.538 #define LCD_LINE1 56

The 1st line of the LCD screen

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_atof.nxc](#), [ex_atoi.nxc](#), [ex_atol.nxc](#), [ex_ButtonPressed.nxc](#), [ex_ClearLine.nxc](#), [ex_contrast.nxc](#), [ex_Copy.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_dispgaout.nxc](#), [ex_dispgout.nxc](#), [ex_DisplayFont.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_file_system.nxc](#), [ex_FlattenVar.nxc](#), [ex_GetBrickDataAddress.nxc](#), [ex_getchar.nxc](#), [ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_isNAN.nxc](#), [ex_labs.nxc](#), [ex_ldiv.nxc](#), [ex_leftstr.nxc](#), [ex_memcmp.nxc](#), [ex_midstr.nxc](#), [ex_NumOut.nxc](#), [ex_ReadSensorHTTouchMultiplexer.nxc](#), [ex_readdressof.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_sysfwrite.nxc](#), [ex_sysmessageread.nxc](#), [ex_SysReadSemData.nxc](#), [ex_SysUpdateCalibCacheInfo.nxc](#), [ex_SysWriteSemData.nxc](#), and [ex_UnflattenVar.nxc](#).

8.1.2.539 #define LCD_LINE2 48

The 2nd line of the LCD screen

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_ButtonPressed.nxc](#), [ex_ctype.nxc](#), [ex_DisplayFont.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_file_system.nxc](#), [ex_FlattenVar.nxc](#), [ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_isNAN.nxc](#), [ex_labs.nxc](#), [ex_ldiv.nxc](#), [ex_memcmp.nxc](#), [ex_ReadSensorHTTouchMultiplexer.nxc](#), [ex_readdressof.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](#), [ex_UnflattenVar.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

8.1.2.540 #define LCD_LINE3 40

The 3rd line of the LCD screen

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_ArraySort.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_ButtonPressed.nxc](#), [ex_ctype.nxc](#), [ex_dispmisc.nxc](#), [ex_FlattenVar.nxc](#), [ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_memcmp.nxc](#), [ex_ReadSensorHTTouchMultiplexer.nxc](#), [ex_relassressof.nxc](#), [ex_StrCatOld.nxc](#), [ex_string.nxc](#), [ex strtod.nxc](#), [ex strtol.nxc](#), [ex strtoul.nxc](#), [ex_SysCommBTConnection.nxc](#), [ex_TextOut.nxc](#), and [ex_UnflattenVar.nxc](#).

8.1.2.541 #define LCD_LINE4 32

The 4th line of the LCD screen

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_arraybuild.nxc](#), [ex_ArraySort.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_ButtonPressed.nxc](#), [ex_ctype.nxc](#), [ex_DisplayFont.nxc](#), [ex_Dispmisc.nxc](#), [ex_FlattenVar.nxc](#), [ex_ReadSensorHTTouchMultiplexer.nxc](#), [ex_relassressof.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_string.nxc](#), [ex_StrReplace.nxc](#), [ex_SysDatalogGetTimes.nxc](#), and [ex_UnflattenVar.nxc](#).

8.1.2.542 #define LCD_LINE5 24

The 5th line of the LCD screen

Examples:

[ex_arraybuild.nxc](#), [ex_ArraySort.nxc](#), [ex_atan.nxc](#), [ex_atand.nxc](#), [ex_ctype.nxc](#), [ex_Dispmisc.nxc](#), [ex_StrIndex.nxc](#), [ex_string.nxc](#), and [ex_SysDatalogGetTimes.nxc](#).

8.1.2.543 #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](#).

8.1.2.544 #define LCD_LINE7 8

The 7th line of the LCD screen

Examples:

[ex_ArraySort.nxc](#), [ex_ctype.nxc](#), and [ex_string.nxc](#).

8.1.2.545 #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](#).

8.1.2.546 #define LDR_APPENDNOTPOSSIBLE 0x8D00

Only datafiles can be appended to.

8.1.2.547 #define LDR_BTBUSY 0x9400

The bluetooth system is busy.

8.1.2.548 #define LDR_BTCONNECTFAIL 0x9500

Bluetooth connection attempt failed.

8.1.2.549 #define LDR_BTTIMEOUT 0x9600

A timeout in the bluetooth system has occurred.

8.1.2.550 #define LDR_CMD_BOOTCMD 0x97

Reboot the NXT into SAMBA mode

8.1.2.551 #define LDR_CMD_BTFACTORYRESET 0xA4

Reset bluetooth configuration to factory defaults

8.1.2.552 #define LDR_CMD_BTGETADR 0x9A

Get the NXT's bluetooth brick address

8.1.2.553 #define LDR_CMD_CLOSE 0x84

Close a file handle

8.1.2.554 #define LDR_CMD_CLOSEMODHANDLE 0x92

Close a module handle

8.1.2.555 #define LDR_CMD_CROPDATAFILE 0x8D

Crop a data file to its used space

8.1.2.556 #define LDR_CMD_DELETE 0x85

Delete a file

8.1.2.557 #define LDR_CMD_DELETEUSERFLASH 0xA0

Delete all files from user flash memory

8.1.2.558 #define LDR_CMD_DEVICEINFO 0x9B

Read device information

8.1.2.559 #define LDR_CMD_FINDFIRST 0x86

Find the first file matching the specified pattern

8.1.2.560 #define LDR_CMD_FINDFIRSTMODULE 0x90

Find the first module matching the specified pattern

8.1.2.561 #define LDR_CMD_FINDNEXT 0x87

Find the next file matching the specified pattern

8.1.2.562 #define LDR_CMD_FINDNEXTMODULE 0x91

Find the next module matching the specified pattern

8.1.2.563 #define LDR_CMD_IOMAPREAD 0x94

Read data from a module IOMAP

8.1.2.564 #define LDR_CMD_IOMAPWRITE 0x95

Write data to a module IOMAP

8.1.2.565 #define LDR_CMD_OPENAPPENDDATA 0x8C

Open a data file for appending

8.1.2.566 #define LDR_CMD_OPENREAD 0x80

Open a file for reading

8.1.2.567 #define LDR_CMD_OPENREADLINEAR 0x8A

Open a linear file for reading

8.1.2.568 #define LDR_CMD_OPENWRITE 0x81

Open a file for writing

8.1.2.569 #define LDR_CMD_OPENWRITEDATA 0x8B

Open a data file for writing

8.1.2.570 #define LDR_CMD_OPENWRITELINEAR 0x89

Open a linear file for writing

8.1.2.571 #define LDR_CMD_POLLCMD 0xA2

Poll command

8.1.2.572 #define LDR_CMD_POLLCMDLEN 0xA1

Read poll command length

8.1.2.573 #define LDR_CMD_READ 0x82

Read from a file

8.1.2.574 #define LDR_CMD_RENAMEFILE 0xA3

Rename a file

8.1.2.575 #define LDR_CMD_RESIZEDATAFILE 0xD0

Resize a data file

8.1.2.576 #define LDR_CMD_SEEKFROMCURRENT 0xD2

Seek from the current position

8.1.2.577 #define LDR_CMD_SEEKFROMEND 0xD3

Seek from the end of the file

8.1.2.578 #define LDR_CMD_SEEKFROMSTART 0xD1

Seek from the start of the file

8.1.2.579 #define LDR_CMD_SETBRICKNAME 0x98

Set the NXT's brick name

8.1.2.580 #define LDR_CMD_VERSIONS 0x88

Read firmware version information

8.1.2.581 #define LDR_CMD_WRITE 0x83

Write to a file

8.1.2.582 #define LDR_ENDOFFILE 0x8500

The end of the file has been reached.

Examples:

[ex_file_system.nxc](#).

8.1.2.583 #define LDR_EOFEXPECTED 0x8400

EOF expected.

Examples:

[ex_file_system.nxc](#).

8.1.2.584 #define LDR_FILEEXISTS 0x8F00

A file with the same name already exists.

Examples:

[ex_file_system.nxc](#).

8.1.2.585 #define LDR_FILEISBUSY 0x8B00

The file is already being used.

8.1.2.586 #define LDR_FILEISFULL 0x8E00

The allocated file size has been filled.

Examples:

[ex_file_system.nxc](#).

8.1.2.587 #define LDR_FILENOFOUND 0x8700

No files matched the search criteria.

8.1.2.588 #define LDR_FILETX_CLOSEERROR 0x9B00

Error transmitting file: attempt to close file failed.

8.1.2.589 #define LDR_FILETX_DSTEXISTS 0x9800

Error transmitting file: destination file exists.

8.1.2.590 #define LDR_FILETX_SRCMISSING 0x9900

Error transmitting file: source file is missing.

8.1.2.591 #define LDR_FILETX_STREAMERROR 0x9A00

Error transmitting file: a stream error occurred.

8.1.2.592 #define LDR_FILETX_TIMEOUT 0x9700

Error transmitting file: a timeout occurred.

8.1.2.593 #define LDR_HANDLEALREADYCLOSED 0x8800

The file handle has already been closed.

8.1.2.594 #define LDR_ILLEGALFILENAME 0x9200

Filename length to long or attempted open a system file (*.rxe, *.rtm, or *.sys) for writing as a datafile.

8.1.2.595 #define LDR_ILLEGALHANDLE 0x9300

Invalid file handle.

8.1.2.596 #define LDR_INPROGRESS 0x0001

The function is executing but has not yet completed.

8.1.2.597 #define LDR_INVALIDSEEK 0x9C00

Invalid file seek operation.

8.1.2.598 #define LDR_MODULENOTFOUND 0x9000

No modules matched the specified search criteria.

8.1.2.599 #define LDR_NOLINEARSPACE 0x8900

Not enough linear flash memory is available.

8.1.2.600 #define LDR_NOMOREFILES 0x8300

The maximum number of files has been reached.

8.1.2.601 #define LDR_NOMOREHANDLES 0x8100

All available file handles are in use.

8.1.2.602 #define LDR_NOSPACE 0x8200

Not enough free flash memory for the specified file size.

8.1.2.603 #define LDR_NOTLINEARFILE 0x8600

The specified file is not linear.

8.1.2.604 #define LDR_NOWRITEBUFFERS 0x8C00

No more write buffers are available.

8.1.2.605 #define LDR_OUTOFBOUNDARY 0x9100

Specified IOMap offset is outside the bounds of the IOMap.

8.1.2.606 #define LDR_REQPIN 0x0002

A PIN exchange request is in progress.

8.1.2.607 #define LDR_SUCCESS 0x0000

The function completed successfully.

Examples:

`ex_file_system.nxc`, `ex_syscommbtcheckstatus.nxc`, `ex_-
SysCommBTConnection.nxc`, `ex_sysfilerename.nxc`, and `ex_-
sysfileresolvehandle.nxc`.

8.1.2.608 #define LDR_UNDEFINEDERROR 0x8A00

An undefined error has occurred.

8.1.2.609 #define LEGO_ADDR_EMETER 0x04

The LEGO e-meter sensor's I2C address

8.1.2.610 #define LEGO_ADDR_TEMP 0x98

The LEGO temperature sensor's I2C address

8.1.2.611 #define LEGO_ADDR_US 0x02

The LEGO ultrasonic sensor's I2C address

8.1.2.612 #define ListFiles 47

List files that match the specified filename pattern

8.1.2.613 #define LoaderExecuteFunction 82

Execute one of the Loader module's internal functions

8.1.2.614 #define LoaderModuleID 0x00090001

The Loader module ID

8.1.2.615 #define LoaderModuleName "Loader.mod"

The Loader module name

8.1.2.616 #define LoaderOffsetFreeUserFlash 4

Offset to the amount of free user flash

8.1.2.617 #define LoaderOffsetPFunc 0

Offset to the Loader module function pointer

8.1.2.618 #define LONG_MAX 2147483647

The maximum value of the long type

8.1.2.619 #define LONG_MIN -2147483647

The minimum value of the long type

8.1.2.620 #define LOWSPEED_CH_NOT_READY 1

Lowspeed port is not ready

8.1.2.621 #define LOWSPEED_COMMUNICATING 3

Channel is actively communicating

8.1.2.622 #define LOWSPEED_DATA RECEIVED 3

Lowspeed port is in data received mode

8.1.2.623 #define LOWSPEED_DONE 5

Channel is done communicating

8.1.2.624 #define LOWSPEED_ERROR 4

Channel is in an error state

8.1.2.625 #define LOWSPEED_IDLE 0

Channel is idle

Examples:

[ex_syscommlscheckstatus.nxc](#).

8.1.2.626 #define LOWSPEED_INIT 1

Channel is being initialized

8.1.2.627 #define LOWSPEED_LOAD_BUFFER 2

Channel buffer is loading

8.1.2.628 #define LOWSPEED_NO_ERROR 0

Lowspeed port has no error

8.1.2.629 #define LOWSPEED RECEIVING 2

Lowspeed port is in receiving mode

8.1.2.630 #define LOWSPEED_RX_ERROR 3

Lowspeed port encountered an error while receiving data

8.1.2.631 #define LOWSPEED_TRANSMITTING 1

Lowspeed port is in transmitting mode

8.1.2.632 #define LOWSPEED_TX_ERROR 2

Lowspeed port encountered an error while transmitting data

8.1.2.633 #define LowSpeedModuleID 0x000B0001

The low speed module ID

8.1.2.634 #define LowSpeedModuleName "Low Speed.mod"

The low speed module name

8.1.2.635 #define LowSpeedOffsetChannelState(p) ((p)+156)

R - Lowspeed channel state (1 byte)

8.1.2.636 #define LowSpeedOffsetErrorType(p) ((p)+160)

R - Lowspeed port error type (1 byte)

8.1.2.637 #define LowSpeedOffsetInBufBuf(p) (((p)*19)+0)

RW - Input buffer data buffer field offset (16 bytes)

8.1.2.638 #define LowSpeedOffsetInBufBytesToRx(p) (((p)*19)+18)

RW - Input buffer bytes to receive field offset (1 byte)

8.1.2.639 #define LowSpeedOffsetInBufInPtr(p) (((p)*19)+16)

RW - Input buffer in pointer field offset (1 byte)

8.1.2.640 #define LowSpeedOffsetInBufOutPtr(p) (((p)*19)+17)

RW - Input buffer out pointer field offset (1 byte)

8.1.2.641 #define LowSpeedOffsetMode(p) ((p)+152)

R - Lowspeed port mode (1 byte)

8.1.2.642 #define LowSpeedOffsetNoRestartOnRead 166

RW - Lowspeed option for no restart on read (all channels) (NBC/NXC)

8.1.2.643 #define LowSpeedOffsetOutBufBuf(p) (((p)*19)+76)

RW - Output buffer data buffer field offset (16 bytes)

8.1.2.644 #define LowSpeedOffsetOutBufBytesToRx(p) (((p)*19)+94)

RW - Output buffer bytes to receive field offset (1 byte)

8.1.2.645 #define LowSpeedOffsetOutBufInPtr(p) (((p)*19)+92)

RW - Output buffer in pointer field offset (1 byte)

8.1.2.646 #define LowSpeedOffsetOutBufOutPtr(p) (((p)*19)+93)

RW - Output buffer out pointer field offset (1 byte)

8.1.2.647 #define LowSpeedOffsetSpeed 165

R - Lowspeed speed (unused)

8.1.2.648 #define LowSpeedOffsetState 164

R - Lowspeed state (all channels)

8.1.2.649 #define LR_COULD_NOT_SAVE 0x51

Bluetooth list result could not save

8.1.2.650 #define LR_ENTRY_REMOVED 0x53

Bluetooth list result entry removed

8.1.2.651 #define LR_STORE_IS_FULL 0x52

Bluetooth list result store is full

8.1.2.652 #define LR_SUCCESS 0x50

Bluetooth list result success

8.1.2.653 #define LR_UNKNOWN_ADDR 0x54

Bluetooth list result unknown address

8.1.2.654 #define LSREAD_NO_RESTART_1 0x01

No restart on read for channel 1

8.1.2.655 #define LSREAD_NO_RESTART_2 0x02

No restart on read for channel 2

8.1.2.656 #define LSREAD_NO_RESTART_3 0x04

No restart on read for channel 3

8.1.2.657 #define LSREAD_NO_RESTART_4 0x08

No restart on read for channel 4

8.1.2.658 #define LSREAD_NO_RESTART_MASK 0x10

No restart mask

8.1.2.659 #define LSREAD_RESTART_ALL 0x00

Restart on read for all channels (default)

8.1.2.660 #define LSREAD_RESTART_NONE 0x0F

No restart on read for all channels

8.1.2.661 #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](#).

8.1.2.662 #define MAILBOX10 9

Mailbox number 10

8.1.2.663 #define MAILBOX2 1

Mailbox number 2

8.1.2.664 #define MAILBOX3 2

Mailbox number 3

8.1.2.665 #define MAILBOX4 3

Mailbox number 4

8.1.2.666 #define MAILBOX5 4

Mailbox number 5

8.1.2.667 #define MAILBOX6 5

Mailbox number 6

8.1.2.668 #define MAILBOX7 6

Mailbox number 7

8.1.2.669 #define MAILBOX8 7

Mailbox number 8

8.1.2.670 #define MAILBOX9 8

Mailbox number 9

8.1.2.671 #define MAX_BT_MSG_SIZE 60000

Max Bluetooth Message Size

8.1.2.672 #define MENUICON_CENTER 1

Center icon

8.1.2.673 #define MENUICON_LEFT 0

Left icon

8.1.2.674 #define MENUICON_RIGHT 2

Right icon

8.1.2.675 #define MENUICONS 3

The number of menu icons

8.1.2.676 #define MENUTEXT 2

Center icon text

8.1.2.677 #define MessageRead 27

Read a message from a mailbox

8.1.2.678 #define MessageWrite 26

Write a message to a mailbox

8.1.2.679 #define MIN_1 60000

1 minute

Examples:

[ex_SysSetSleepTimeout.nxc](#).

8.1.2.680 #define MS_1 1

1 millisecond

8.1.2.681 #define MS_10 10

10 milliseconds

8.1.2.682 #define MS_100 100

100 milliseconds

Examples:[ex_PolyOut.nxc](#), and [ex_sysdrawpolygon.nxc](#).**8.1.2.683 #define MS_150 150**

150 milliseconds

8.1.2.684 #define MS_2 2

2 milliseconds

8.1.2.685 #define MS_20 20

20 milliseconds

Examples:[ex_dispgaout.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [glBoxDemo.nxc](#), and [glScaleDemo.nxc](#).**8.1.2.686 #define MS_200 200**

200 milliseconds

Examples:[ex_dispgoutex.nxc](#), and [ex_playtones.nxc](#).**8.1.2.687 #define MS_250 250**

250 milliseconds

8.1.2.688 #define MS_3 3

3 milliseconds

8.1.2.689 #define MS_30 30

30 milliseconds

8.1.2.690 #define MS_300 300

300 milliseconds

8.1.2.691 #define MS_350 350

350 milliseconds

8.1.2.692 #define MS_4 4

4 milliseconds

8.1.2.693 #define MS_40 40

40 milliseconds

8.1.2.694 #define MS_400 400

400 milliseconds

8.1.2.695 #define MS_450 450

450 milliseconds

8.1.2.696 #define MS_5 5

5 milliseconds

Examples:

[ex_getchar.nxc](#).

8.1.2.697 #define MS_50 50

50 milliseconds

Examples:

[ex_CircleOut.nxc](#), and [ex_playtones.nxc](#).

8.1.2.698 #define MS_500 500

500 milliseconds

Examples:

[alternating_tasks.nxc](#), [ex_dispgout.nxc](#), [ex_playsound.nxc](#), [ex_yield.nxc](#), and [util_rpm.nxc](#).

8.1.2.699 #define MS_6 6

6 milliseconds

8.1.2.700 #define MS_60 60

60 milliseconds

8.1.2.701 #define MS_600 600

600 milliseconds

8.1.2.702 #define MS_7 7

7 milliseconds

8.1.2.703 #define MS_70 70

70 milliseconds

8.1.2.704 #define MS_700 700

700 milliseconds

8.1.2.705 #define MS_8 8

8 milliseconds

8.1.2.706 #define MS_80 80

80 milliseconds

8.1.2.707 #define MS_800 800

800 milliseconds

8.1.2.708 #define MS_9 9

9 milliseconds

8.1.2.709 #define MS_90 90

90 milliseconds

8.1.2.710 #define MS_900 900

900 milliseconds

8.1.2.711 #define MS_ADDR_ACCLNX 0x02

MindSensors ACCL-Nx I2C address

Examples:

[ex_ACCLNxCalibrateX.nxc](#), [ex_ACCLNxCalibrateXEnd.nxc](#), [ex_ACCLNxCalibrateY.nxc](#), [ex_ACCLNxCalibrateYEnd.nxc](#), [ex_ACCLNxCalibrateZ.nxc](#), [ex_ACCLNxCalibrateZEnd.nxc](#),
[ACCLNxResetCalibration.nxc](#), [ex_ACCLNxSensitivity.nxc](#), [ex_ACCLNxXOffset.nxc](#), [ex_ACCLNxXRange.nxc](#), [ex_ACCLNxYOffset.nxc](#),
[ex_ACCLNxYRange.nxc](#), [ex_ACCLNxZOffset.nxc](#), [ex_ACCLNxZRange.nxc](#),
[ex_ReadSensorMSAccel.nxc](#), [ex_ReadSensorMSTilt.nxc](#), and [ex_SetACCLNxSensitivity.nxc](#).

8.1.2.712 #define MS_ADDR_CMPSNX 0x02

MindSensors CMPS-Nx I2C address

Examples:

[ex_SensorMSCompass.nxc](#).

8.1.2.713 #define MS_ADDR_DISTNX 0x02

MindSensors DIST-Nx I2C address

Examples:

[ex_DISTNxDistance.nxc](#), [ex_DISTNxGP2D12.nxc](#), [ex_DISTNxGP2D120.nxc](#),
[ex_DISTNxGP2YA02.nxc](#), [ex_DISTNxGP2YA21.nxc](#), [ex_DISTNxMaxDistance.nxc](#), [ex_DISTNxMinDistance.nxc](#),
[ex_DISTNxModuleType.nxc](#), [ex_DISTNxNumPoints.nxc](#), [ex_DISTNxVoltage.nxc](#),
[ex_MSADPAOff.nxc](#), and [ex_MSADPAOn.nxc](#).

8.1.2.714 #define MS_ADDR_IVSENS 0x12

MindSensors IVSens I2C address

8.1.2.715 #define MS_ADDR_LINELDR 0x02

MindSensors LineLdr I2C address

8.1.2.716 #define MS_ADDR_MTRMUX 0xB4

MindSensors MTRMux I2C address

8.1.2.717 #define MS_ADDR_NRLINK 0x02

MindSensors NRLink I2C address

Examples:

[ex_MSRCXSetNRLinkPort.nxc](#), [ex_NRLink2400.nxc](#), [ex_NRLink4800.nxc](#),
[ex_NRLinkFlush.nxc](#), [ex_NRLinkIRLong.nxc](#), [ex_NRLinkIRShort.nxc](#),
[ex_NRLinkSetPF.nxc](#), [ex_NRLinkSetRCX.nxc](#), [ex_NRLinkSetTrain.nxc](#),
[ex_NRLinkStatus.nxc](#), [ex_NRLinkTxRaw.nxc](#), [ex_ReadNRLinkBytes.nxc](#),
[ex_RunNRLinkMacro.nxc](#), and [ex_WriteNRLinkBytes.nxc](#).

8.1.2.718 #define MS_ADDR_NXTCAM 0x02

MindSensors NXTCam I2C address

8.1.2.719 #define MS_ADDR_NXTHID 0x04

MindSensors NXTHID I2C address

8.1.2.720 #define MS_ADDR_NXTMMX 0x06

MindSensors NXTMMX I2C address

8.1.2.721 #define MS_ADDR_NXTSERVO 0xB0

MindSensors NXTServo I2C address

8.1.2.722 #define MS_ADDR_PFMATE 0x48

MindSensors PFMate I2C address

8.1.2.723 #define MS_ADDR_PSPNX 0x02

MindSensors PSP-Nx I2C address

Examples:

[ex_PSPNxAnalog.nxc](#), [ex_PSPNxDigital.nxc](#), and [ex_-ReadSensorMSPlayStation.nxc](#).

8.1.2.724 #define MS_ADDR_RTCLOCK 0xD0

MindSensors RTClock I2C address

8.1.2.725 #define MS_ADDR_RXMUX 0x7E

MindSensors RXMux I2C address

8.1.2.726 #define MS_CMD_ADPA_OFF 0x4F

Turn MindSensors ADPA mode off

8.1.2.727 #define MS_CMD_ADPA_ON 0x4E

Turn MindSensors ADPA mode on

8.1.2.728 #define MS_CMD_DEENERGIZED 0x44

De-energize the MindSensors device

8.1.2.729 #define MS_CMD_ENERGIZED 0x45

Energize the MindSensors device

8.1.2.730 #define NA 0xFFFF

The specified argument does not apply (aka unwired)

Examples:

[ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), and [ex_ArraySumSqr.nxc](#).

8.1.2.731 #define NO_ERR 0

Successful execution of the specified command

Examples:

[ex_SysColorSensorRead.nxc](#), [ex_SysCommBTConnection.nxc](#), [ex_SysCommBTOFF.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_SysCommHSWrite.nxc](#), [ex_syscommlswriteex.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_systiomapread.nxc](#), [ex_systiomapreadyid.nxc](#), [ex_syslistfiles.nxc](#), and [ex_sysmessageread.nxc](#).

8.1.2.732 #define NO_OF_BTNS 4

The number of NXT buttons.

8.1.2.733 #define NormalizedValue 3

Normalized value field. Contains the current normalized analog sensor value. Read only.

8.1.2.734 #define NRLINK_CMD_2400 0x44

Set NRLink to 2400 baud

8.1.2.735 #define NRLINK_CMD_4800 0x48

Set NRLink to 4800 baud

8.1.2.736 #define NRLINK_CMD_FLUSH 0x46

Flush the NRLink

8.1.2.737 #define NRLINK_CMD_IR_LONG 0x4C

Set the NRLink to long range IR

8.1.2.738 #define NRLINK_CMD_IR_SHORT 0x53

Set the NRLink to short range IR

8.1.2.739 #define NRLINK_CMD_RUN_MACRO 0x52

Run an NRLink macro

8.1.2.740 #define NRLINK_CMD_SET_PF 0x50

Set the NRLink to Power Function mode

8.1.2.741 #define NRLINK_CMD_SET_RCX 0x58

Set the NRLink to RCX mode

8.1.2.742 #define NRLINK_CMD_SET_TRAIN 0x54

Set the NRLink to IR Train mode

8.1.2.743 #define NRLINK_CMD_TX_RAW 0x55

Set the NRLink to transmit raw bytes

8.1.2.744 #define NRLINK_REG_BYTES 0x40

The NRLink bytes register

8.1.2.745 #define NRLINK_REG_DATA 0x42

The NRLink data register

8.1.2.746 #define NRLINK_REG_EEPROM 0x50

The NRLink eeprom register

8.1.2.747 #define NULL 0

A constant representing NULL

8.1.2.748 #define OPARR_MAX 0x05

Calculate the maximum value of the elements in the numeric input array

Examples:

[ex_ArrayOp.nxc](#).

8.1.2.749 #define OPARR_MEAN 0x01

Calculate the mean value for the elements in the numeric input array

8.1.2.750 #define OPARR_MIN 0x04

Calculate the minimum value of the elements in the numeric input array

8.1.2.751 #define OPARR_SORT 0x06

Sort the elements in the numeric input array

8.1.2.752 #define OPARR_STD 0x03

Calculate the standard deviation of the elements in the numeric input array

8.1.2.753 #define OPARR_SUM 0x00

Calculate the sum of the elements in the numeric input array

8.1.2.754 #define OPARR_SUMSQR 0x02

Calculate the sum of the squares of the elements in the numeric input array

8.1.2.755 #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_motorregvalue.nxc](#), [ex_motorregpvalue.nxc](#), [ex_motorregulation.nxc](#), [ex_motorrotationcount.nxc](#), [ex_motorrunstate.nxc](#), [ex_motortachocount.nxc](#), [ex_motortacholimit.nxc](#), [ex_motorturnratio.nxc](#), [ex_off.nxc](#), [ex_offex.nxc](#), [ex_onfwd.nxc](#), [ex_onfwdxex.nxc](#), [ex_onfwdreg.nxc](#), [ex_onfwdrexex.nxc](#), [ex_onfwdregpid.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.pid.nxc](#), and [ex_yield.nxc](#).

8.1.2.756 #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_resettachocount.nxc](#), [ex_rotatemotorex.nxc](#), [ex_rotatemotorexpид.nxc](#), and [ex_setoutput.nxc](#).

8.1.2.757 #define OUT_ABC 0x06

Output ports A, B, and C

8.1.2.758 #define OUT_AC 0x04

Output ports A and C

8.1.2.759 #define OUT_B 0x01

Output port B

8.1.2.760 #define OUT_BC 0x05

Output ports B and C

8.1.2.761 #define OUT_C 0x02

Output port C

8.1.2.762 #define OUT_MODE BRAKE 0x02

Uses electronic braking to outputs

8.1.2.763 #define OUT_MODE COAST 0x00

No power and no braking so motors rotate freely.

8.1.2.764 #define OUT_MODE MOTORON 0x01

Enables PWM power to the outputs given the Power setting

Examples:

[ex_RemoteSetOutputState.nxc](#).

8.1.2.765 #define OUT_MODE REGMETHOD 0xF0

Mask for unimplemented regulation mode

8.1.2.766 #define OUT_MODE REGULATED 0x04

Enables active power regulation using the RegMode value

8.1.2.767 #define OUT_OPTION_HOLDATLIMIT 0x10

Option to have the firmware hold the motor when it reaches the tachometer limit

8.1.2.768 #define OUT_OPTION_RAMPDOWNTOLIMIT 0x20

Option to have the firmware rampdown the motor power as it approaches the tachometer limit (not implemented yet)

8.1.2.769 #define OUT_REGMODE_IDLE 0

No motor regulation.

Examples:

[ex_RemoteSetOutputState.nxc](#).

8.1.2.770 #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](#).

8.1.2.771 #define OUT_REGMODE_SYNC 2

Synchronize the rotation of two motors.

8.1.2.772 #define OUT_RUNSTATE_HOLD 0x60

Set motor run state to hold at the current position.

8.1.2.773 #define OUT_RUNSTATE_IDLE 0x00

Disable all power to motors.

8.1.2.774 #define OUT_RUNSTATE_RAMPDOWN 0x40

Enable ramping down from a current Power to a new (lower) Power over a specified TachoLimit goal.

8.1.2.775 #define OUT_RUNSTATE_RAMPUP 0x10

Enable ramping up from a current Power to a new (higher) Power over a specified TachoLimit goal.

8.1.2.776 #define OUT_RUNSTATE_RUNNING 0x20

Enable power to motors at the specified Power level.

Examples:

[ex_RemoteSetOutputState.nxc](#).

8.1.2.777 #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.

8.1.2.778 #define OutputModuleID 0x00020001

The output module ID

8.1.2.779 #define OutputModuleName "Output.mod"

The output module name

8.1.2.780 #define OutputOffsetActualSpeed(p) (((p)*32)+21)

R - Holds the current motor speed (1 byte) sbyte

8.1.2.781 #define OutputOffsetBlockTachoCount(p) (((p)*32)+4)

R - Holds current number of counts for the current output block (4 bytes) slong

8.1.2.782 #define OutputOffsetFlags(p) (((p)*32)+18)

RW - Holds flags for which data should be updated (1 byte) ubyte

8.1.2.783 #define OutputOffsetMode(p) (((p)*32)+19)

RW - Holds motor mode: Run, Break, regulated, ... (1 byte) ubyte

8.1.2.784 #define OutputOffsetMotorRPM(p) (((p)*32)+16)

Not updated, will be removed later !! (2 bytes) sword

8.1.2.785 #define OutputOffsetOptions(p) (((p)*32)+29)

RW - holds extra motor options related to the tachometer limit (1 byte) ubyte (NBC/NXC)

8.1.2.786 #define OutputOffsetOverloaded(p) (((p)*32)+27)

R - True if the motor has been overloaded within speed control regulation (1 byte) ubyte

8.1.2.787 #define OutputOffsetPwnFreq 96

use for frequency of checking regulation mode (1 byte) ubyte (NBC/NXC)

8.1.2.788 #define OutputOffsetRegDParameter(p) (((p)*32)+24)

RW - Holds the D-constant used in the regulation (1 byte) ubyte

8.1.2.789 #define OutputOffsetRegIParameter(p) (((p)*32)+23)

RW - Holds the I-constant used in the regulation (1 byte) ubyte

8.1.2.790 #define OutputOffsetRegMode(p) (((p)*32)+26)

RW - Tells which regulation mode should be used (1 byte) ubyte

8.1.2.791 #define OutputOffsetRegPParameter(p) (((p)*32)+22)

RW - Holds the P-constant used in the regulation (1 byte) ubyte

8.1.2.792 #define OutputOffsetRotationCount(p) (((p)*32)+8)

R - Holds current number of counts for the rotation counter to the output (4 bytes)
ulong

8.1.2.793 #define OutputOffsetRunState(p) (((p)*32)+25)

RW - Holds the current RunState in the output module (1 byte) ubyte

8.1.2.794 #define OutputOffsetSpeed(p) (((p)*32)+20)

RW - Holds the wanted speed (1 byte) sbyte

8.1.2.795 #define OutputOffsetSyncTurnParameter(p) (((p)*32)+28)

RW - Holds the turning parameter need within MoveBlock (1 byte) sbyte

8.1.2.796 #define OutputOffsetTachoCount(p) (((p)*32)+0)

R - Holds current number of counts, since last reset, updated every 1 mS (4 bytes)
ulong

8.1.2.797 #define OutputOffsetTachoLimit(p) (((p)*32)+12)

RW - Holds number of counts to travel, 0 => Run forever (4 bytes) ulong

8.1.2.798 #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).

8.1.2.799 #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.

8.1.2.800 #define PF_CHANNEL_1 0

Power function channel 1

Examples:

[ex_HTPFComboDirect.nxc](#), [ex_HTPFComboPWM.nxc](#), [ex_HTPFSingleOutputCST.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#), [ex_HTPFSinglePin.nxc](#), [ex_HTPFTrain.nxc](#), [ex_MSPFComboDirect.nxc](#), [ex_MSPFComboPWM.nxc](#), [ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), [ex_MSPFSinglePin.nxc](#), and [ex_MSPFTrain.nxc](#).

8.1.2.801 #define PF_CHANNEL_2 1

Power function channel 2

8.1.2.802 #define PF_CHANNEL_3 2

Power function channel 3

8.1.2.803 #define PF_CHANNEL_4 3

Power function channel 4

8.1.2.804 #define PF_CMD_BRAKE 3

Power function command brake

8.1.2.805 #define PF_CMD_FWD 1

Power function command forward

Examples:

[ex_HTPFComboDirect.nxc](#), and [ex_MSPFComboDirect.nxc](#).

8.1.2.806 #define PF_CMD_REV 2

Power function command reverse

8.1.2.807 #define PF_CMD_STOP 0

Power function command stop

Examples:

[ex_HTPFComboDirect.nxc](#), and [ex_MSPFComboDirect.nxc](#).

8.1.2.808 #define PF_CST_CLEAR1_CLEAR2 0

Power function CST clear 1 and clear 2

8.1.2.809 #define PF_CST_CLEAR1_SET2 2

Power function CST clear 1 and set 2

8.1.2.810 #define PF_CST_DECREMENT_PWM 5

Power function CST decrement PWM

8.1.2.811 #define PF_CST_FULL_FWD 6

Power function CST full forward

8.1.2.812 #define PF_CST_FULL_REV 7

Power function CST full reverse

8.1.2.813 #define PF_CST_INCREMENT_PWM 4

Power function CST increment PWM

8.1.2.814 #define PF_CST_SET1_CLEAR2 1

Power function CST set 1 and clear 2

8.1.2.815 #define PF_CST_SET1_SET2 3

Power function CST set 1 and set 2

Examples:

[ex_HTPFSingleOutputCST.nxc](#), and [ex_MSPFSingleOutputCST.nxc](#).

8.1.2.816 #define PF_CST_TOGGLE_DIR 8

Power function CST toggle direction

8.1.2.817 #define PF_FUNC_CLEAR 1

Power function single pin - clear

8.1.2.818 #define PF_FUNC_NOCHANGE 0

Power function single pin - no change

8.1.2.819 #define PF_FUNC_SET 2

Power function single pin - set

Examples:

[ex_HTPFSinglePin.nxc](#), and [ex_MSPFSinglePin.nxc](#).

8.1.2.820 #define PF_FUNC_TOGGLE 3

Power function single pin - toggle

8.1.2.821 #define PF_MODE_COMBO_DIRECT 1

Power function mode combo direct

8.1.2.822 #define PF_MODE_COMBO_PWM 4

Power function mode combo pulse width modulation (PWM)

8.1.2.823 #define PF_MODE_SINGLE_OUTPUT_CST 6

Power function mode single output clear, set, toggle (CST)

8.1.2.824 #define PF_MODE_SINGLE_OUTPUT_PWM 4

Power function mode single output pulse width modulation (PWM)

8.1.2.825 #define PF_MODE_SINGLE_PIN_CONT 2

Power function mode single pin continuous

8.1.2.826 #define PF_MODE_SINGLE_PIN_TIME 3

Power function mode single pin timed

8.1.2.827 #define PF_MODE_TRAIN 0

Power function mode IR Train

8.1.2.828 #define PF_OUT_A 0

Power function output A

Examples:

[ex_HTPFSingleOutputCST.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#),
[ex_HTPFSinglePin.nxc](#), [ex_MSPFSingleOutputCST.nxc](#), [ex_MSPFSingleOutputPWM.nxc](#), and [ex_MSPFSinglePin.nxc](#).

8.1.2.829 #define PF_OUT_B 1

Power function output B

8.1.2.830 #define PF_PIN_C1 0

Power function pin C1

Examples:

[ex_HTPFSinglePin.nxc](#), and [ex_MSPFSinglePin.nxc](#).

8.1.2.831 #define PF_PIN_C2 1

Power function pin C2

8.1.2.832 #define PF_PWM_BRAKE 8

Power function PWM brake

8.1.2.833 #define PF_PWM_FLOAT 0

Power function PWM float

8.1.2.834 #define PF_PWM_FWD1 1

Power function PWM foward level 1

8.1.2.835 #define PF_PWM_FWD2 2

Power function PWM foward level 2

8.1.2.836 #define PF_PWM_FWD3 3

Power function PWM foward level 3

8.1.2.837 #define PF_PWM_FWD4 4

Power function PWM foward level 4

8.1.2.838 #define PF_PWM_FWD5 5

Power function PWM foward level 5

Examples:

[ex_HTPFComboPWM.nxc](#), [ex_HTPFSingleOutputPWM.nxc](#), [ex_MSPFComboPWM.nxc](#), and [ex_MSPFSingleOutputPWM.nxc](#).

8.1.2.839 #define PF_PWM_FWD6 6

Power function PWM foward level 6

8.1.2.840 #define PF_PWM_FWD7 7

Power function PWM foward level 7

8.1.2.841 #define PF_PWM_REV1 15

Power function PWM reverse level 1

8.1.2.842 #define PF_PWM_REV2 14

Power function PWM reverse level 2

8.1.2.843 #define PF_PWM_REV3 13

Power function PWM reverse level 3

8.1.2.844 #define PF_PWM_REV4 12

Power function PWM reverse level 4

Examples:

[ex_HTPFComboPWM.nxc](#), and [ex_MSPFComboPWM.nxc](#).

8.1.2.845 #define PF_PWM_REV5 11

Power function PWM reverse level 5

8.1.2.846 #define PF_PWM_REV6 10

Power function PWM reverse level 6

8.1.2.847 #define PF_PWM_REV7 9

Power function PWM reverse level 7

8.1.2.848 #define PI 3.141593

A constant for PI

Examples:[ex_dispfnout.nxc](#), and [ex_string.nxc](#).**8.1.2.849 #define PID_0 0**

PID zero

8.1.2.850 #define PID_1 32

PID one

8.1.2.851 #define PID_2 64

PID two

8.1.2.852 #define PID_3 96

PID three

8.1.2.853 #define PID_4 128

PID four

8.1.2.854 #define PID_5 160

PID five

8.1.2.855 #define PID_6 192

PID six

8.1.2.856 #define PID_7 224

PID seven

8.1.2.857 #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.

8.1.2.858 #define PROG_ABORT 4

Program has been aborted

8.1.2.859 #define PROG_ERROR 3

A program error has occurred

8.1.2.860 #define PROG_IDLE 0

Program state is idle

8.1.2.861 #define PROG_OK 1

Program state is okay

8.1.2.862 #define PROG_RESET 5

Program has been reset

8.1.2.863 #define PROG_RUNNING 2

Program is running

8.1.2.864 #define PSP_BTNSET1_DOWN 0x02

The PSP-Nx button set 1 down arrow

8.1.2.865 #define PSP_BTNSET1_L3 0x40

The PSP-Nx button set 1 L3

8.1.2.866 #define PSP_BTNSET1_LEFT 0x01

The PSP-Nx button set 1 left arrow

8.1.2.867 #define PSP_BTNSET1_R3 0x20

The PSP-Nx button set 1 R3

8.1.2.868 #define PSP_BTNSET1_RIGHT 0x04

The PSP-Nx button set 1 right arrow

8.1.2.869 #define PSP_BTNSET1_UP 0x08

The PSP-Nx button set 1 up arrow

8.1.2.870 #define PSP_BTNSET2_CIRCLE 0x04

The PSP-Nx button set 2 circle

8.1.2.871 #define PSP_BTNSET2_CROSS 0x02

The PSP-Nx button set 2 cross

8.1.2.872 #define PSP_BTNSET2_L1 0x20

The PSP-Nx button set 2 L1

8.1.2.873 #define PSP_BTNSET2_L2 0x80

The PSP-Nx button set 2 L2

8.1.2.874 #define PSP_BTNSET2_R1 0x10

The PSP-Nx button set 2 R1

8.1.2.875 #define PSP_BTNSET2_R2 0x40

The PSP-Nx button set 2 R2

8.1.2.876 #define PSP_BTNSET2_SQUARE 0x01

The PSP-Nx button set 2 square

8.1.2.877 #define PSP_BTNSET2_TRIANGLE 0x08

The PSP-Nx button set 2 triangle

8.1.2.878 #define PSP_CMD_ANALOG 0x73

Set the PSP-Nx to analog mode

8.1.2.879 #define PSP_CMD_DIGITAL 0x41

Set the PSP-Nx to digital mode

8.1.2.880 #define PSP_REG_BTNSET1 0x42

The PSP-Nx button set 1 register

8.1.2.881 #define PSP_REG_BTNSET2 0x43

The PSP-Nx button set 2 register

8.1.2.882 #define PSP_REG_XLEFT 0x44

The PSP-Nx X left register

8.1.2.883 #define PSP_REG_XRIGHT 0x46

The PSP-Nx X right register

8.1.2.884 #define PSP_REG_YLEFT 0x45

The PSP-Nx Y left register

8.1.2.885 #define PSP_REG_YRIGHT 0x47

The PSP-Nx Y right register

8.1.2.886 #define RADIANS_PER_DEGREE PI/180

Used for converting from degrees to radians

Examples:

[ex_sin_cos.nxc](#).

8.1.2.887 #define RAND_MAX 32768

The maximum unsigned int random number returned by rand

8.1.2.888 #define RandomNumber 24

Generate a random number

8.1.2.889 #define RawValue 2

Raw value field. Contains the current raw analog sensor value. Read only.

8.1.2.890 #define RCX_AbsVarOp 0x74

Absolute value function

8.1.2.891 #define RCX_AndVarOp 0x84

AND function

8.1.2.892 #define RCX_AutoOffOp 0xb1

Set auto off timer

8.1.2.893 #define RCX_BatteryLevelOp 0x30

Read the battery level

8.1.2.894 #define RCX_BatteryLevelSrc 34

The RCX battery level source

8.1.2.895 #define RCX_BootModeOp 0x65

Set into book mode

8.1.2.896 #define RCX_CalibrateEventOp 0x04

Calibrate event

8.1.2.897 #define RCX_ClearAllEventsOp 0x06

Clear all events

8.1.2.898 #define RCX_ClearCounterOp 0xb7

Clear a counter

8.1.2.899 #define RCX_ClearMsgOp 0x90

Clear message

8.1.2.900 #define RCX_ClearSensorOp 0xd1

Clear a sensor

8.1.2.901 #define RCX_ClearSoundOp 0x80

Clear sound

8.1.2.902 #define RCX_ClearTimerOp 0xa1

Clear a timer

8.1.2.903 #define RCX_ClickCounterSrc 27

The RCX event click counter source

8.1.2.904 #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.`

8.1.2.905 #define RCX_CounterSrc 21

The RCX counter source

8.1.2.906 #define RCX_DatalogOp 0x62

Datalog the specified source/value

8.1.2.907 #define RCX_DatalogRawDirectSrc 42

The RCX direct datalog raw source

8.1.2.908 #define RCX_DatalogRawIndirectSrc 41

The RCX indirect datalog raw source

8.1.2.909 #define RCX_DatalogSrcDirectSrc 38

The RCX direct datalog source source

8.1.2.910 #define RCX_DatalogSrcIndirectSrc 37

The RCX indirect datalog source source

8.1.2.911 #define RCX_DatalogValueDirectSrc 40

The RCX direct datalog value source

8.1.2.912 #define RCX_DatalogValueIndirectSrc 39

The RCX indirect datalog value source

8.1.2.913 #define RCX_DecCounterOp 0xa7

Decrement a counter

8.1.2.914 #define RCX_DeleteSubOp 0xc1

Delete a subroutine

8.1.2.915 #define RCX_DeleteSubsOp 0x70

Delete subroutines

8.1.2.916 #define RCX_DeleteTaskOp 0x61

Delete a task

8.1.2.917 #define RCX_DeleteTasksOp 0x40

Delete tasks

8.1.2.918 #define RCX_DirectEventOp 0x03

Fire an event

8.1.2.919 #define RCX_DisplayOp 0x33

Set LCD display value

8.1.2.920 #define RCX_DivVarOp 0x44

Divide function

8.1.2.921 #define RCX_DurationSrc 31

The RCX event duration source

8.1.2.922 #define RCX_EventStateSrc 25

The RCX event static source

8.1.2.923 #define RCX_FirmwareVersionSrc 35

The RCX firmware version source

8.1.2.924 #define RCX_GlobalMotorStatusSrc 17

The RCX global motor status source

8.1.2.925 #define RCX_GOutputDirOp 0x77

Set global motor direction

8.1.2.926 #define RCX_GOutputModeOp 0x67

Set global motor mode

8.1.2.927 #define RCX_GOutputPowerOp 0xa3

Set global motor power levels

8.1.2.928 #define RCX_HysteresisSrc 30

The RCX event hysteresis source

8.1.2.929 #define RCX_IncCounterOp 0x97

Increment a counter

8.1.2.930 #define RCX_IndirectVarSrc 36

The RCX indirect variable source

8.1.2.931 #define RCX_InputBooleanSrc 13

The RCX input boolean source

8.1.2.932 #define RCX_InputModeOp 0x42

Set the input mode

8.1.2.933 #define RCX_InputModeSrc 11

The RCX input mode source

8.1.2.934 #define RCX_InputRawSrc 12

The RCX input raw source

8.1.2.935 #define RCX_InputTypeOp 0x32

Set the input type

8.1.2.936 #define RCX_InputTypeSrc 10

The RCX input type source

8.1.2.937 #define RCX_InputValueSrc 9

The RCX input value source

Examples:

[ex_HTRCXAddToDatalog.nxc](#), [ex_MSRCXAddToDatalog.nxc](#), and [ex_MSRCXSumVar.nxc](#).

8.1.2.938 #define RCX_IRModeOp 0x31

Set the IR transmit mode

8.1.2.939 #define RCX_LightOp 0x87

Light opcode

8.1.2.940 #define RCX_LowerThresholdSrc 29

The RCX event lower threshold source

8.1.2.941 #define RCX_LSBlinkTimeOp 0xe3

Set the light sensor blink time

8.1.2.942 #define RCX_LSCalibrateOp 0xc0

Calibrate the light sensor

8.1.2.943 #define RCX_LSHysteresisOp 0xd3

Set the light sensor hysteresis

8.1.2.944 #define RCX_LSLowerThreshOp 0xc3

Set the light sensor lower threshold

8.1.2.945 #define RCX_LSUppThreshOp 0xb3

Set the light sensor upper threshold

8.1.2.946 #define RCX_MessageOp 0xf7

Set message

8.1.2.947 #define RCX_MessageSrc 15

The RCX message source

8.1.2.948 #define RCX_MulVarOp 0x54

Multiply function

8.1.2.949 #define RCX_MuteSoundOp 0xd0

Mute sound

8.1.2.950 #define RCX_OnOffFloatOp 0x21

Control motor state - on, off, float

8.1.2.951 #define RCX_OrVarOp 0x94

OR function

8.1.2.952 #define RCX_OUT_A 0x01

RCX Output A

Examples:

[ex_HTRCXDisableOutput.nxc](#), [ex_HTRCXEnableOutput.nxc](#), [ex_HTRCXBloat.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_MSRCXBloat.nxc](#), [ex_MSRCXFwd.nxc](#), [ex_MSRCXIvertOutput.nxc](#), [ex_MSRCXObvertOutput.nxc](#), [ex_MSRCXOff.nxc](#), [ex_MSRCXOn.nxc](#), [ex_MSRCXOnFor.nxc](#), [ex_MSRCXOnFwd.nxc](#), [ex_MSRCXOnRev.nxc](#), [ex_MSRCXRev.nxc](#), [ex_MSRCXSetDirection.nxc](#), [ex_MSRCXSetGlobalDirection.nxc](#), [ex_MSRCXSetGlobalOutput.nxc](#), [ex_MSRCXSetMaxPower.nxc](#), [ex_MSRCXSetOutput.nxc](#), [ex_MSRCXSetPower.nxc](#), and [ex_MSRCXToggle.nxc](#).

8.1.2.953 #define RCX_OUT_AB 0x03

RCX Outputs A and B

8.1.2.954 #define RCX_OUT_ABC 0x07

RCX Outputs A, B, and C

8.1.2.955 #define RCX_OUT_AC 0x05

RCX Outputs A and C

8.1.2.956 #define RCX_OUT_B 0x02

RCX Output B

8.1.2.957 #define RCX_OUT_BC 0x06

RCX Outputs B and C

8.1.2.958 #define RCX_OUT_C 0x04

RCX Output C

8.1.2.959 #define RCX_OUT_FLOAT 0

Set RCX output to float

8.1.2.960 #define RCX_OUT_FULL 7

Set RCX output power level to full

Examples:

[ex_HTRCXSetPower.nxc](#), and [ex_MSRCXSetPower.nxc](#).

8.1.2.961 #define RCX_OUT_FWD 0x80

Set RCX output direction to forward

Examples:

[ex_HTRCXSetDirection.nxc](#), [ex_HTRCXSetGlobalDirection.nxc](#), [ex_MSRCXSetDirection.nxc](#), and [ex_MSRCXSetGlobalDirection.nxc](#).

8.1.2.962 #define RCX_OUT_HALF 3

Set RCX output power level to half

8.1.2.963 #define RCX_OUT_LOW 0

Set RCX output power level to low

8.1.2.964 #define RCX_OUT_OFF 0x40

Set RCX output to off

8.1.2.965 #define RCX_OUT_ON 0x80

Set RCX output to on

Examples:

[ex_HTRCXSetGlobalOutput.nxc](#), [ex_HTRCXSetOutput.nxc](#), [ex_MSRCXSetGlobalOutput.nxc](#), and [ex_MSRCXSetOutput.nxc](#).

8.1.2.966 #define RCX_OUT_REV 0

Set RCX output direction to reverse

8.1.2.967 #define RCX_OUT_TOGGLE 0x40

Set RCX output direction to toggle

8.1.2.968 #define RCX_OutputDirOp 0xe1

Set the motor direction

8.1.2.969 #define RCX_OutputPowerOp 0x13

Set the motor power level

8.1.2.970 #define RCX_OutputStatusSrc 3

The RCX output status source

8.1.2.971 #define RCX_PBTurnOffOp 0x60

Turn off the brick

8.1.2.972 #define RCX_PingOp 0x10

Ping the brick

8.1.2.973 #define RCX_PlaySoundOp 0x51

Play a sound

8.1.2.974 #define RCX_PlayToneOp 0x23

Play a tone

8.1.2.975 #define RCX_PlayToneVarOp 0x02

Play a tone using a variable

8.1.2.976 #define RCX_PollMemoryOp 0x63

Poll a memory location

8.1.2.977 #define RCX_PollOp 0x12

Poll a source/value combination

8.1.2.978 #define RCX_ProgramSlotSrc 8

The RCX program slot source

8.1.2.979 #define RCX_RandomSrc 4

The RCX random number source

Examples:

[ex_MSRCXSet.nxc](#), and [ex_MSRCXSubVar.nxc](#).

8.1.2.980 #define RCX_RemoteKeysReleased 0x0000

All remote keys have been released

8.1.2.981 #define RCX_RemoteOp 0xd2

Execute simulated remote control buttons

8.1.2.982 #define RCX_RemoteOutABackward 0x4000

Set output A backward

8.1.2.983 #define RCX_RemoteOutAForward 0x0800

Set output A forward

8.1.2.984 #define RCX_RemoteOutBBackward 0x8000

Set output B backward

8.1.2.985 #define RCX_RemoteOutBForward 0x1000

Set output B forward

8.1.2.986 #define RCX_RemoteOutCBackward 0x0001

Set output C backward

8.1.2.987 #define RCX_RemoteOutCForward 0x2000

Set output C forward

8.1.2.988 #define RCX_RemotePBMessage1 0x0100

Send PB message 1

8.1.2.989 #define RCX_RemotePBMessage2 0x0200

Send PB message 2

8.1.2.990 #define RCX_RemotePBMessage3 0x0400

Send PB message 3

8.1.2.991 #define RCX_RemotePlayASound 0x0080

Play a sound

Examples:

[ex_HTRCXRemote.nxc](#), and [ex_MSRCXRemote.nxc](#).

8.1.2.992 #define RCX_RemoteSelProgram1 0x0002

Select program 1

8.1.2.993 #define RCX_RemoteSelProgram2 0x0004

Select program 2

8.1.2.994 #define RCX_RemoteSelProgram3 0x0008

Select program 3

8.1.2.995 #define RCX_RemoteSelProgram4 0x0010

Select program 4

8.1.2.996 #define RCX_RemoteSelProgram5 0x0020

Select program 5

8.1.2.997 #define RCX_RemoteStopOutOff 0x0040

Stop and turn off outputs

8.1.2.998 #define RCX_ScoutCounterLimitSrc 22

The Scout counter limit source

8.1.2.999 #define RCX_ScoutEventFBSrc 24

The Scout event feedback source

8.1.2.1000 #define RCX_ScoutLightParamsSrc 19

The Scout light parameters source

8.1.2.1001 #define RCX_ScoutOp 0x47

Scout opcode

8.1.2.1002 #define RCX_ScoutRulesOp 0xd5

Set Scout rules

8.1.2.1003 #define RCX_ScoutRulesSrc 18

The Scout rules source

8.1.2.1004 #define RCX_ScoutTimerLimitSrc 20

The Scout timer limit source

8.1.2.1005 #define RCX_SelectProgramOp 0x91

Select a program slot

8.1.2.1006 #define RCX_SendUARTDataOp 0xc2

Send data via IR using UART settings

8.1.2.1007 #define RCX_SetCounterOp 0xd4

Set counter value

8.1.2.1008 #define RCX_SetDatalogOp 0x52

Set the datalog size

8.1.2.1009 #define RCX_SetEventOp 0x93

Set an event

8.1.2.1010 #define RCX_SetFeedbackOp 0x83

Set Scout feedback

8.1.2.1011 #define RCX_SetPriorityOp 0xd7

Set task priority

8.1.2.1012 #define RCX_SetSourceValueOp 0x05

Set a source/value

8.1.2.1013 #define RCX_SetTimerLimitOp 0xc4

Set timer limit

8.1.2.1014 #define RCX_SetVarOp 0x14

Set function

8.1.2.1015 #define RCX_SetWatchOp 0x22

Set the watch source/value

8.1.2.1016 #define RCX_SgnVarOp 0x64

Sign function

8.1.2.1017 #define RCX_SoundOp 0x57

Sound opcode

8.1.2.1018 #define RCX_StartTaskOp 0x71

Start a task

8.1.2.1019 #define RCX_StopAllTasksOp 0x50

Stop all tasks

8.1.2.1020 #define RCX_StopTaskOp 0x81

Stop a task

8.1.2.1021 #define RCX_SubVarOp 0x34

Subtract function

8.1.2.1022 #define RCX_SumVarOp 0x24

Sum function

8.1.2.1023 #define RCX_TaskEventsSrc 23

The RCX task events source

8.1.2.1024 #define RCX_TenMSTimerSrc 26

The RCX 10ms timer source

8.1.2.1025 #define RCX_TimerSrc 1

The RCX timer source

8.1.2.1026 #define RCX_UARTSetupSrc 33

The RCX UART setup source

8.1.2.1027 #define RCX_UnlockFirmOp 0xa5

Unlock the firmware

8.1.2.1028 #define RCX_UnlockOp 0x15

Unlock the brick

8.1.2.1029 #define RCX_UnmuteSoundOp 0xe0

Unmute sound

8.1.2.1030 #define RCX_UploadDatalogOp 0xa4

Upload datalog contents

8.1.2.1031 #define RCX_UpperThresholdSrc 28

The RCX event upper threshold source

8.1.2.1032 #define RCX_VariableSrc 0

The RCX variable source

Examples:

[ex_HTRCXPoll.nxc](#), [ex_HTRCXSelectDisplay.nxc](#), [ex_HTScoutSetSensorLowerLimit.nxc](#), [ex_HTScoutSetSensorUpperLimit.nxc](#), [ex_MSRCXAbsVar.nxc](#), [ex_MSRCXMulVar.nxc](#), [ex_MSRCXPoll.nxc](#), [ex_MSRCXSelectDisplay.nxc](#), [ex_MSRCXSet.nxc](#), [ex_MSRCXSetUserDisplay.nxc](#), [ex_MSRCXSetVar.nxc](#), [ex_MSRCXSgnVar.nxc](#), [ex_MSScoutSetSensorLowerLimit.nxc](#), and [ex_MSScoutSetSensorUpperLimit.nxc](#).

8.1.2.1033 #define RCX_ViewSourceValOp 0xe5

View a source/value

8.1.2.1034 #define RCX_VLLOp 0xe2

Send visual light link (VLL) data

8.1.2.1035 #define RCX_WatchSrc 14

The RCX watch source

8.1.2.1036 #define ReadButton 20

Read the current button state

8.1.2.1037 #define ReadSemData 40

Read motor semaphore data

8.1.2.1038 #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.

8.1.2.1039 #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.

8.1.2.1040 #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 [Actual-Speed](#) 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 sync 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.

8.1.2.1041 #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.

8.1.2.1042 #define RESET_ALL 0x68

Reset all three tachometer counters

8.1.2.1043 #define RESET_BLOCK_COUNT 0x20

Reset the NXT-G block tachometer counter

8.1.2.1044 #define RESET_BLOCKANDTACHO 0x28

Reset both the internal counter and the NXT-G block counter

8.1.2.1045 #define RESET_COUNT 0x08

Reset the internal tachometer counter

8.1.2.1046 #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`.

8.1.2.1047 #define RESET_ROTATION_COUNT 0x40

Reset the rotation counter

8.1.2.1048 #define RICArg(_arg) ((_arg)|0x1000)

Output an RIC parameterized argument.

Parameters:

`_arg` The argument that you want to parameterize.

Examples:

[ex_dispgaoutex.nxc](#).

8.1.2.1049 #define RICImgPoint(_X, _Y) (_X)&0xFF, (_X)>>8, (_Y)&0xFF, (_Y)>>8

Output an RIC ImgPoint structure.

Parameters:

_X The X coordinate.

_Y The Y coordinate.

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

8.1.2.1050 #define RICImgRect(_Pt, _W, _H) _Pt, (_W)&0xFF, (_W)>>8, (_H)&0xFF, (_H)>>8

Output an RIC ImgRect structure.

Parameters:

_Pt An ImgPoint. See [RICImgPoint](#).

_W The rectangle width.

_H The rectangle height.

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

8.1.2.1051 #define RICMapArg(_mapidx, _arg) (((_arg)|0x1000|(((-_mapidx)&0xF)<<8))

Output an RIC parameterized and mapped argument.

Parameters:

- _mapidx* The varmap data address.
- _arg* The parameterized argument you want to pass through a varmap.

**8.1.2.1052 #define RICMapElement(_Domain, _Range) (_Domain)&0xFF,
 (_Domain)>>8, (_Range)&0xFF, (_Range)>>8**

Output an RIC map element.

Parameters:

- _Domain* The map element domain.
- _Range* The map element range.

**8.1.2.1053 #define RICMapFunction(_MapElement, ...) _MapElement,
 __VA_ARGS__**

Output an RIC VarMap function.

Parameters:

- _MapElement* An entry in the varmap function. At least 2 elements are required.
See [RICMapElement](#).

**8.1.2.1054 #define RICOpCircle(_CopyOptions, _Point, _Radius) 10,
 0, 7, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,
 (_Radius)&0xFF, (_Radius)>>8**

Output an RIC Circle opcode.

Parameters:

- _CopyOptions* Circle copy options. See [Drawing option constants](#).
- _Point* The circle's center point. See [RICImgPoint](#).
- _Radius* The circle's radius.

```
8.1.2.1055 #define RICOpCopyBits(_CopyOptions, _DataAddr, _SrcRect,  
_DstPoint) 18, 0, 3, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8,  
(_DataAddr)&0xFF, (_DataAddr)>>8, _SrcRect, _DstPoint
```

Output an RIC CopyBits opcode.

Parameters:

_CopyOptions CopyBits copy options. See [Drawing option constants](#).
_DataAddr The address of the sprite from which to copy data.
_SrcRect The rectangular portion of the sprite to copy. See [RICImgRect](#).
_DstPoint The LCD coordinate to which to copy the data. See [RICImgPoint](#).

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

```
8.1.2.1056 #define RICOpDescription(_Options, _Width, _Height) 8, 0, 0, 0,  
(_Options)&0xFF, (_Options)>>8, (_Width)&0xFF, (_Width)>>8,  
(_Height)&0xFF, (_Height)>>8
```

Output an RIC Description opcode.

Parameters:

_Options RIC options.
_Width The total RIC width.
_Height The total RIC height.

Examples:

[ex_dispgaoutex.nxc](#).

```
8.1.2.1057 #define RICOpEllipse(_CopyOptions, _Point, _RadiusX,  
_RadiusY) 12, 0, 9, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8,  
_Point, (_RadiusX)&0xFF, (_RadiusX)>>8, (_RadiusY)&0xFF,  
(_RadiusY)>>8
```

Output an RIC Ellipse opcode.

Parameters:

_CopyOptions Ellipse copy options. See [Drawing option constants](#).

_Point The center of the ellipse. See [RICImgPoint](#).

_RadiusX The x-axis radius of the ellipse.

_RadiusY The y-axis radius of the ellipse.

8.1.2.1058 `#define RICOpLine(_CopyOptions, _Point1, _Point2) 12, 0, 5, 0,
(_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point1, _Point2`

Output an RIC Line opcode.

Parameters:

_CopyOptions Line copy options. See [Drawing option constants](#).

_Point1 The starting point of the line. See [RICImgPoint](#).

_Point2 The ending point of the line. See [RICImgPoint](#).

8.1.2.1059 `#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.

8.1.2.1060 `#define RICOpPixel(_CopyOptions, _Point, _Value) 10, 0,
4, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,
(_Value)&0xFF, (_Value)>>8`

Output an RIC Pixel opcode.

Parameters:

_CopyOptions Pixel copy options. See [Drawing option constants](#).

_Point The pixel coordinate. See [RICImgPoint](#).

_Value The pixel value (unused).

```
8.1.2.1061 #define RICOpPolygon(_CopyOptions, _Count,
    _ThePoints) ((_Count*4)+6)&0xFF, ((_Count*4)+6)>>8, 10, 0,
    (_CopyOptions)&0xFF, (_CopyOptions)>>8, (_Count)&0xFF,
    (_Count)>>8, _ThePoints
```

Output an RIC Polygon opcode.

Parameters:

_CopyOptions Polygon copy options. See [Drawing option constants](#).

_Count The number of points in the polygon.

_ThePoints The list of polygon points. See [RICPolygonPoints](#).

```
8.1.2.1062 #define RICOpRect(_CopyOptions, _Point, _Width, _Height) 12,
    0, 6, 0, (_CopyOptions)&0xFF, (_CopyOptions)>>8, _Point,
    (_Width)&0xFF, (_Width)>>8, (_Height)&0xFF, (_Height)>>8
```

Output an RIC Rect opcode.

Parameters:

_CopyOptions Rect copy options. See [Drawing option constants](#).

_Point The rectangle's top left corner. See [RICImgPoint](#).

_Width The rectangle's width.

_Height The rectangle's height.

```
8.1.2.1063 #define RICOpSprite(_DataAddr, _Rows,
    _BytesPerRow, _SpriteData) ((_Rows*-
    BytesPerRow)+(((_Rows*_BytesPerRow)%2)+8)&0xFF,
    ((_Rows*_BytesPerRow)+(((_Rows*_BytesPerRow)%2)+8))>>8,
    1, 0, (_DataAddr)&0xFF, (_DataAddr)>>8, (_Rows)&0xFF,
    (_Rows)>>8, (_BytesPerRow)&0xFF, (_BytesPerRow)>>8,
    _SpriteData
```

Output an RIC Sprite opcode.

Parameters:

_DataAddr The address of the sprite.
_Rows The number of rows of data.
_BytesPerRow The number of bytes per row.
_SpriteData The actual sprite data. See [RICSpriteData](#).

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

**8.1.2.1064 #define RICOpVarMap(_DataAddr, _MapCount,
_MapFunction) ((_MapCount*4)+6)&0xFF,
((_MapCount*4)+6)>>8, 2, 0, (_DataAddr)&0xFF,
(_DataAddr)>>8, (_MapCount)&0xFF, (_MapCount)>>8,
_MapFunction**

Output an RIC VarMap opcode.

Parameters:

_DataAddr The address of the varmap.
_MapCount The number of points in the function.
_MapFunction The definition of the varmap function. See [RICMapFunction](#).

**8.1.2.1065 #define RICPolygonPoints(_pPoint1, _pPoint2, ...) _pPoint1,
_pPoint2, __VA_ARGS__**

Output RIC polygon points.

Parameters:

_pPoint1 The first polygon point. See [RICImgPoint](#).
_pPoint2 The second polygon point (at least 3 points are required). See [RICImgPoint](#).

8.1.2.1066 #define RICSpriteData(...) __VA_ARGS__

Output RIC sprite data.

Examples:

[ex_dispgaout.nxc](#), [ex_dispgaoutex.nxc](#), and [ex_sysdrawgraphicarray.nxc](#).

8.1.2.1067 #define ROTATE_QUEUE 5

VM should rotate queue

8.1.2.1068 #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.

8.1.2.1069 #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](#).

8.1.2.1070 #define SAMPLERATE_DEFAULT 8000

Default sample rate [sps]

8.1.2.1071 #define SAMPLERATE_MAX 16000

Max sample rate [sps]

8.1.2.1072 #define SAMPLERATE_MIN 2000

Min sample rate [sps]

8.1.2.1073 #define ScaledValue 4

Scaled value field. Contains the current scaled analog sensor value. Read/write.

8.1.2.1074 #define SCHAR_MAX 127

The maximum value of the signed char type

8.1.2.1075 #define SCHAR_MIN -127

The minimum value of the signed char type

8.1.2.1076 #define SCOUT_FXR_ALARM 2

Alarm special effects

8.1.2.1077 #define SCOUT_FXR_BUG 1

Bug special effects

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.1.2.1078 #define SCOUT_FXR_NONE 0

No special effects

8.1.2.1079 #define SCOUT_FXR_RANDOM 3

Random special effects

8.1.2.1080 #define SCOUT_FXR_SCIENCE 4

Science special effects

8.1.2.1081 #define SCOUT_LIGHT_OFF 0

Turn off the scout light

8.1.2.1082 #define SCOUT_LIGHT_ON 0x80

Turn on the scout light

Examples:

[ex_HTScoutSetLight.nxc](#).

8.1.2.1083 #define SCOUT_LR_AVOID 3

Light rule avoid

8.1.2.1084 #define SCOUT_LR_IGNORE 0

Light rule ignore

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.1.2.1085 #define SCOUT_LR_OFF_WHEN 5

Light rule off when

8.1.2.1086 #define SCOUT_LR_SEEK_DARK 2

Light rule seek dark

8.1.2.1087 #define SCOUT_LR_SEEK_LIGHT 1

Light rule seek light

8.1.2.1088 #define SCOUT_LR_WAIT_FOR 4

Light rule wait for

8.1.2.1089 #define SCOUT_MODE_POWER 1

Enter power mode

Examples:

[ex_HTScoutSetScoutMode.nxc](#), and [ex_MSScoutSetScoutMode.nxc](#).

8.1.2.1090 #define SCOUT_MODE_STANDALONE 0

Enter stand alone mode

8.1.2.1091 #define SCOUT_MR_CIRCLE_LEFT 4

Motion rule circle left

8.1.2.1092 #define SCOUT_MR_CIRCLE_RIGHT 3

Motion rule circle right

8.1.2.1093 #define SCOUT_MR_FORWARD 1

Motion rule forward

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.1.2.1094 #define SCOUT_MR_LOOP_A 5

Motion rule loop A

8.1.2.1095 #define SCOUT_MR_LOOP_AB 7

Motion rule loop A then B

8.1.2.1096 #define SCOUT_MR_LOOP_B 6

Motion rule loop B

8.1.2.1097 #define SCOUT_MR_NO_MOTION 0

Motion rule none

8.1.2.1098 #define SCOUT_MR_ZIGZAG 2

Motion rule zigzag

8.1.2.1099 #define SCOUT SNDSET_ALARM 3

Set sound set to alarm

8.1.2.1100 #define SCOUT SNDSET_BASIC 1

Set sound set to basic

8.1.2.1101 #define SCOUT SNDSET_BUG 2

Set sound set to bug

8.1.2.1102 #define SCOUT SNDSET_NONE 0

Set sound set to none

8.1.2.1103 #define SCOUT SNDSET_RANDOM 4

Set sound set to random

8.1.2.1104 #define SCOUT SNDSET_SCIENCE 5

Set sound set to science

8.1.2.1105 #define SCOUT_SOUND_1_BLINK 17

Play the Scout 1 blink sound

8.1.2.1106 #define SCOUT_SOUND_2_BLINK 18

Play the Scout 2 blink sound

8.1.2.1107 #define SCOUT_SOUND_COUNTER1 19

Play the Scout counter 1 sound

8.1.2.1108 #define SCOUT_SOUND_COUNTER2 20

Play the Scout counter 2 sound

8.1.2.1109 #define SCOUT_SOUND_ENTER_BRIGHT 14

Play the Scout enter bright sound

8.1.2.1110 #define SCOUT_SOUND_ENTER_DARK 16

Play the Scout enter dark sound

8.1.2.1111 #define SCOUT_SOUND_ENTER_NORMAL 15

Play the Scout enter normal sound

8.1.2.1112 #define SCOUT_SOUND_ENTERSA 7

Play the Scout enter standalone sound

8.1.2.1113 #define SCOUT_SOUND_KEYERROR 8

Play the Scout key error sound

8.1.2.1114 #define SCOUT_SOUND_MAIL_RECEIVED 24

Play the Scout mail received sound

8.1.2.1115 #define SCOUT_SOUND_NONE 9

Play the Scout none sound

8.1.2.1116 #define SCOUT_SOUND_REMOTE 6

Play the Scout remote sound

8.1.2.1117 #define SCOUT_SOUND_SPECIAL1 25

Play the Scout special 1 sound

8.1.2.1118 #define SCOUT_SOUND_SPECIAL2 26

Play the Scout special 2 sound

8.1.2.1119 #define SCOUT_SOUND_SPECIAL3 27

Play the Scout special 3 sound

8.1.2.1120 #define SCOUT_SOUND_TIMER1 21

Play the Scout timer 1 sound

8.1.2.1121 #define SCOUT_SOUND_TIMER2 22

Play the Scout timer 2 sound

8.1.2.1122 #define SCOUT_SOUND_TIMER3 23

Play the Scout timer 3 sound

8.1.2.1123 #define SCOUT_SOUND_TOUCH1_PRES 10

Play the Scout touch 1 pressed sound

8.1.2.1124 #define SCOUT_SOUND_TOUCH1_REL 11

Play the Scout touch 1 released sound

8.1.2.1125 #define SCOUT_SOUND_TOUCH2_PRES 12

Play the Scout touch 2 pressed sound

8.1.2.1126 #define SCOUT_SOUND_TOUCH2_REL 13

Play the Scout touch 2 released sound

8.1.2.1127 #define SCOUT_TGS_LONG 2

Transmit level long

8.1.2.1128 #define SCOUT_TGS_MEDIUM 1

Transmit level medium

8.1.2.1129 #define SCOUT_TGS_SHORT 0

Transmit level short

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.1.2.1130 #define SCOUT_TR_AVOID 2

Touch rule avoid

8.1.2.1131 #define SCOUT_TR_IGNORE 0

Touch rule ignore

8.1.2.1132 #define SCOUT_TR_OFF_WHEN 4

Touch rule off when

8.1.2.1133 #define SCOUT_TR_REVERSE 1

Touch rule reverse

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.1.2.1134 #define SCOUT_TR_WAIT_FOR 3

Touch rule wait for

8.1.2.1135 #define SCREEN_BACKGROUND 0

Entire screen

8.1.2.1136 #define SCREEN_LARGE 1

Entire screen except status line

8.1.2.1137 #define SCREEN_MODE_CLEAR 0x01

Clear the screen

See also:

[SetScreenMode\(\)](#)

8.1.2.1138 #define SCREEN_MODE_RESTORE 0x00

Restore the screen

See also:

[SetScreenMode\(\)](#)

8.1.2.1139 #define SCREEN_SMALL 2

Screen between menu icons and status line

8.1.2.1140 #define SCREENS 3

The number of screen bits

8.1.2.1141 #define SEC_1 1000

1 second

Examples:

[alternating_tasks.nxc](#), [ex_dispmisc.nxc](#), [ex_file_system.nxc](#), [ex_playsound.nxc](#), [ex_playtones.nxc](#), [ex_PolyOut.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_sysdrawpolygon.nxc](#), [ex_wait.nxc](#), and [ex_yield.nxc](#).

8.1.2.1142 #define SEC_10 10000

10 seconds

Examples:

[ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ClearScreen.nxc](#), [ex_DisplayFont.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_reladdressof.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_string.nxc](#), [ex_SysCommBTConnection.nxc](#), and [ex_SysCommHSControl.nxc](#).

8.1.2.1143 #define SEC_15 15000

15 seconds

Examples:

[ex_dispfunc.nxc](#), and [ex_memcmp.nxc](#).

8.1.2.1144 #define SEC_2 2000

2 seconds

Examples:

[ex_CircleOut.nxc](#), [ex_dispmisc.nxc](#), [ex_file_system.nxc](#), [ex_LineOut.nxc](#), [ex_PolyOut.nxc](#), and [ex_sysdrawpolygon.nxc](#).

8.1.2.1145 #define SEC_20 20000

20 seconds

8.1.2.1146 #define SEC_3 3000

3 seconds

Examples:

[ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_div.nxc](#), and [ex_ldiv.nxc](#).

8.1.2.1147 #define SEC_30 30000

30 seconds

8.1.2.1148 #define SEC_4 4000

4 seconds

Examples:

[ex_Copy.nxc](#), [ex_dispftout.nxc](#), [ex_dispmisc.nxc](#), [ex_leftstr.nxc](#), [ex_midstr.nxc](#), [ex_rightstr.nxc](#), [ex_sysdrawfont.nxc](#), [ex_syslistfiles.nxc](#), [util_battery_1.nxc](#), and [util_battery_2.nxc](#).

8.1.2.1149 #define SEC_5 5000

5 seconds

Examples:

[ex_atof.nxc](#), [ex_atoi.nxc](#), [ex_atol.nxc](#), [ex_ClearLine.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_dispftout.nxc](#), [ex_dispgout.nxc](#), [ex_FlattenVar.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](#), [ex_UnflattenVar.nxc](#), and [ex_wait.nxc](#).

8.1.2.1150 #define SEC_6 6000

6 seconds

Examples:

[ex strtod.nxc](#), [ex strtol.nxc](#), and [ex strtoul.nxc](#).

8.1.2.1151 #define SEC_7 7000

7 seconds

8.1.2.1152 #define SEC_8 8000

8 seconds

Examples:

[ex_file_system.nxc.](#)

8.1.2.1153 #define SEC_9 9000

9 seconds

Examples:

[ex_SensorHTGyro.nxc.](#)

8.1.2.1154 #define SetScreenMode 19

Set the screen mode

8.1.2.1155 #define SetSleepTimeoutVal 46

Set the NXT sleep timeout value

8.1.2.1156 #define SHRT_MAX 32767

The maximum value of the short type

8.1.2.1157 #define SHRT_MIN -32767

The minimum value of the short type

8.1.2.1158 #define SIZE_OF_BDADDR 7

Size of Bluetooth Address

8.1.2.1159 #define SIZE_OF_BRICK_NAME 8

Size of NXT Brick name

8.1.2.1160 #define SIZE_OF_BT_CONNECT_TABLE 4

Size of Bluetooth connection table -- Index 0 is always incoming connection

8.1.2.1161 #define SIZE_OF_BT_DEVICE_TABLE 30

Size of Bluetooth device table

8.1.2.1162 #define SIZE_OF_BT_NAME 16

Size of Bluetooth name

8.1.2.1163 #define SIZE_OF_BT_PINCODE 16

Size of Bluetooth PIN

8.1.2.1164 #define SIZE_OF_BTBUF 128

Size of Bluetooth buffer

8.1.2.1165 #define SIZE_OF_CLASS_OF_DEVICE 4

Size of class of device

8.1.2.1166 #define SIZE_OF_HSBUF 128

Size of High Speed Port 4 buffer

8.1.2.1167 #define SIZE_OF_USBBUF 64

Size of USB Buffer in bytes

8.1.2.1168 #define SIZE_OF_USBDATA 62

Size of USB Buffer available for data

8.1.2.1169 #define SOUND_CLICK 0

Play the standard key click sound

8.1.2.1170 #define SOUND_DOUBLE_BEEP 1

Play the standard double beep sound

8.1.2.1171 #define SOUND_DOWN 2

Play the standard sweep down sound

Examples:

[ex_playsound.nxc.](#)

8.1.2.1172 #define SOUND_FAST_UP 5

Play the standard fast up sound

Examples:

[ex_playsound.nxc.](#)

8.1.2.1173 #define SOUND_FLAGS_IDLE 0x00

R - Sound is idle

8.1.2.1174 #define SOUND_FLAGS_RUNNING 0x02

R - Currently processing a tone or file

8.1.2.1175 #define SOUND_FLAGS_UPDATE 0x01

W - Make changes take effect

Examples:

[ex_SetSoundFlags.nxc.](#)

8.1.2.1176 #define SOUND_LOW_BEEP 4

Play the standard low beep sound

Examples:

[ex_playsound.nxc.](#)

8.1.2.1177 #define SOUND_MODE_LOOP 0x01

W - Play file until writing SOUND_STATE_STOP into SoundState

8.1.2.1178 #define SOUND_MODE_ONCE 0x00

W - Only play file once

Examples:

[ex_SetSoundMode.nxc](#).

8.1.2.1179 #define SOUND_MODE_TONE 0x02

W - Play tone specified in Frequency for Duration ms

8.1.2.1180 #define SOUND_STATE_FILE 0x02

R - Processing a file of sound/melody data

8.1.2.1181 #define SOUND_STATE_IDLE 0x00

R - Idle, ready for start sound (SOUND_UPDATE)

Examples:

[ex_syssoundgetstate.nxc](#).

8.1.2.1182 #define SOUND_STATE_STOP 0x04

W - Stop sound immediately and close hardware

Examples:

[ex_SetSoundModuleState.nxc](#), and [ex_syssoundsetstate.nxc](#).

8.1.2.1183 #define SOUND_STATE_TONE 0x03

R - Processing a play tone request

8.1.2.1184 #define SOUND_UP 3

Play the standard sweep up sound

Examples:

[ex_playsound.nxc](#).

8.1.2.1185 #define SoundGetState 11

Get the current sound module state

8.1.2.1186 #define SoundModuleID 0x00080001

The sound module ID

Examples:

[ex_sysiomapwritebyid.nxc](#).

8.1.2.1187 #define SoundModuleName "Sound.mod"

The sound module name

Examples:

[ex_sysiomapwrite.nxc](#).

8.1.2.1188 #define SoundOffsetDuration 2

RW - **Tone** duration [mS] (2 bytes)

8.1.2.1189 #define SoundOffsetFlags 26

RW - Play flag - described above (1 byte) [SoundFlags constants](#)

8.1.2.1190 #define SoundOffsetFreq 0

RW - **Tone** frequency [Hz] (2 bytes)

8.1.2.1191 #define SoundOffsetMode 28

RW - Play mode - described above (1 byte) [SoundMode constants](#)

8.1.2.1192 #define SoundOffsetSampleRate 4

RW - Sound file sample rate [2000..16000] (2 bytes)

Examples:

[ex_sysiomapwrite.nxc](#), and [ex_sysiomapwritebyid.nxc](#).

8.1.2.1193 #define SoundOffsetSoundFilename 6

RW - Sound/melody filename (20 bytes)

8.1.2.1194 #define SoundOffsetState 27

RW - Play state - described above (1 byte) [SoundState constants](#)

8.1.2.1195 #define SoundOffsetVolume 29

RW - Sound/melody volume [0..4] 0 = off (1 byte)

8.1.2.1196 #define SoundPlayFile 9

Play a sound or melody file

8.1.2.1197 #define SoundPlayTone 10

Play a simple tone with the specified frequency and duration

8.1.2.1198 #define SoundSetState 12

Set the sound module state

8.1.2.1199 #define SPECIALS 5

The number of special bit values

8.1.2.1200 #define STAT_COMM_PENDING 32

Pending setup operation in progress

8.1.2.1201 #define STAT_MSG_EMPTY_MAILBOX 64

Specified mailbox contains no new messages

8.1.2.1202 #define STATUSICON_BATTERY 3

Battery status icon collection

8.1.2.1203 #define STATUSICON_BLUETOOTH 0

BlueTooth status icon collection

8.1.2.1204 #define STATUSICON_USB 1

USB status icon collection

8.1.2.1205 #define STATUSICON_VM 2

VM status icon collection

8.1.2.1206 #define STATUSICONS 4

The number of status icons

8.1.2.1207 #define STATUSTEXT 1

Status text (BT name)

8.1.2.1208 #define STEPICON_1 0

Left most step icon

8.1.2.1209 #define STEPICON_2 1

8.1.2.1210 #define STEPICON_3 2

8.1.2.1211 #define STEPICON_4 3

8.1.2.1212 #define STEPICON_5 4

Right most step icon

8.1.2.1213 #define STEPICONS 5

8.1.2.1214 #define STEPLINE 3

Step collection lines

8.1.2.1215 #define STOP_REQ 4

VM should stop executing program

8.1.2.1216 #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.

8.1.2.1217 #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](#).

8.1.2.1218 #define TEMP_FQ_1 0x00

Set fault queue 1

8.1.2.1219 #define TEMP_FQ_2 0x08

Set fault queue 2

8.1.2.1220 #define TEMP_FQ_4 0x10

Set fault queue 4

8.1.2.1221 #define TEMP_FQ_6 0x18

Set fault queue 6

8.1.2.1222 #define TEMP_OS_ONESHOT 0x80

Set the sensor into oneshot mode

8.1.2.1223 #define TEMP_POL_HIGH 0x04

Set polarity to high

8.1.2.1224 #define TEMP_POL_LOW 0x00

Set polarity to low

8.1.2.1225 #define TEMP_REG_CONFIG 0x01

The register for reading/writing sensor configuration values

8.1.2.1226 #define TEMP_REG_TEMP 0x00

The register where temperature values can be read

8.1.2.1227 #define TEMP_REG_THIGH 0x03

The register where temperature high values can be read

8.1.2.1228 #define TEMP_REG_TLOW 0x02

The register where temperature low values can be read

8.1.2.1229 #define TEMP_RES_10BIT 0x20

Set the resolution to 10 bit

8.1.2.1230 #define TEMP_RES_11BIT 0x40

Set the resolution to 11 bit

8.1.2.1231 #define TEMP_RES_12BIT 0x60

Set the resolution to 12 bit

8.1.2.1232 #define TEMP_RES_9BIT 0x00

Set the resolution to 9 bit

8.1.2.1233 #define TEMP_SD_CONTINUOUS 0x00

Set the mode to continuous

8.1.2.1234 #define TEMP_SD_SHUTDOWN 0x01

Set the mode to shutdown

8.1.2.1235 #define TEMP_TM_COMPARATOR 0x00

Set the thermostat mode to comparator

8.1.2.1236 #define TEMP_TM_INTERRUPT 0x02

Set the thermostat mode to interrupt

8.1.2.1237 #define TEXTLINE_1 0

Text line 1

Examples:

[ex_GetDisplayNormal.nxc](#), [ex_GetDisplayPopup.nxc](#), [ex_SetDisplayNormal.nxc](#),
and [ex_SetDisplayPopup.nxc](#).

8.1.2.1238 #define TEXTLINE_2 1

Text line 2

8.1.2.1239 #define TEXTLINE_3 2

Text line 3

8.1.2.1240 #define TEXTLINE_4 3

Text line 4

8.1.2.1241 #define TEXTLINE_5 4

Text line 5

8.1.2.1242 #define TEXTLINE_6 5

Text line 6

8.1.2.1243 #define TEXTLINE_7 6

Text line 7

8.1.2.1244 #define TEXTLINE_8 7

Text line 8

8.1.2.1245 #define TEXTLINES 8

The number of text lines on the LCD

8.1.2.1246 #define TIMES_UP 6

VM time is up

8.1.2.1247 #define TONE_A3 220

Third octave A

8.1.2.1248 #define TONE_A4 440

Fourth octave A

Examples:

[ex_yield.nxc.](#)

8.1.2.1249 #define TONE_A5 880

Fifth octave A

8.1.2.1250 #define TONE_A6 1760

Sixth octave A

8.1.2.1251 #define TONE_A7 3520

Seventh octave A

8.1.2.1252 #define TONE_AS3 233

Third octave A sharp

8.1.2.1253 #define TONE_AS4 466

Fourth octave A sharp

8.1.2.1254 #define TONE_AS5 932

Fifth octave A sharp

8.1.2.1255 #define TONE_AS6 1865

Sixth octave A sharp

8.1.2.1256 #define TONE_AS7 3729

Seventh octave A sharp

8.1.2.1257 #define TONE_B3 247

Third octave B

8.1.2.1258 #define TONE_B4 494

Fourth octave B

8.1.2.1259 #define TONE_B5 988

Fifth octave B

8.1.2.1260 #define TONE_B6 1976

Sixth octave B

8.1.2.1261 #define TONE_B7 3951

Seventh octave B

8.1.2.1262 #define TONE_C4 262

Fourth octave C

Examples:

[alternating_tasks.nxc](#), and [ex_playtones.nxc](#).

8.1.2.1263 #define TONE_C5 523

Fifth octave C

Examples:[ex_file_system.nxc](#), and [ex_playtones.nxc](#).**8.1.2.1264 #define TONE_C6 1047**

Sixth octave C

Examples:[alternating_tasks.nxc](#), and [ex_playtones.nxc](#).**8.1.2.1265 #define TONE_C7 2093**

Seventh octave C

8.1.2.1266 #define TONE_CS4 277

Fourth octave C sharp

8.1.2.1267 #define TONE_CS5 554

Fifth octave C sharp

8.1.2.1268 #define TONE_CS6 1109

Sixth octave C sharp

8.1.2.1269 #define TONE_CS7 2217

Seventh octave C sharp

8.1.2.1270 #define TONE_D4 294

Fourth octave D

8.1.2.1271 #define TONE_D5 587

Fifth octave D

8.1.2.1272 #define TONE_D6 1175

Sixth octave D

8.1.2.1273 #define TONE_D7 2349

Seventh octave D

8.1.2.1274 #define TONE_DS4 311

Fourth octave D sharp

8.1.2.1275 #define TONE_DS5 622

Fifth octave D sharp

8.1.2.1276 #define TONE_DS6 1245

Sixth octave D sharp

8.1.2.1277 #define TONE_DS7 2489

Seventh octave D sharp

8.1.2.1278 #define TONE_E4 330

Fourth octave E

Examples:

[ex_playtones.nxc](#).

8.1.2.1279 #define TONE_E5 659

Fifth octave E

Examples:

[ex_playtones.nxc](#).

8.1.2.1280 #define TONE_E6 1319

Sixth octave E

8.1.2.1281 #define TONE_E7 2637

Seventh octave E

8.1.2.1282 #define TONE_F4 349

Fourth octave F

8.1.2.1283 #define TONE_F5 698

Fifth octave F

8.1.2.1284 #define TONE_F6 1397

Sixth octave F

8.1.2.1285 #define TONE_F7 2794

Seventh octave F

8.1.2.1286 #define TONE_FS4 370

Fourth octave F sharp

8.1.2.1287 #define TONE_FS5 740

Fifth octave F sharp

8.1.2.1288 #define TONE_FS6 1480

Sixth octave F sharp

8.1.2.1289 #define TONE_FS7 2960

Seventh octave F sharp

8.1.2.1290 #define TONE_G4 392

Fourth octave G

Examples:

[ex_playtones.nxc.](#)

8.1.2.1291 #define TONE_G5 784

Fifth octave G

Examples:

[ex_playtones.nxc.](#)

8.1.2.1292 #define TONE_G6 1568

Sixth octave G

8.1.2.1293 #define TONE_G7 3136

Seventh octave G

8.1.2.1294 #define TONE_GS4 415

Fourth octave G sharp

8.1.2.1295 #define TONE_GS5 831

Fifth octave G sharp

8.1.2.1296 #define TONE_GS6 1661

Sixth octave G sharp

8.1.2.1297 #define TONE_GS7 3322

Seventh octave G sharp

8.1.2.1298 #define TOPLINE 4

Top status underline

8.1.2.1299 #define TRAIN_CHANNEL_1 0

IR Train channel 1

Examples:

[ex_HTIRTrain.nxc](#), and [ex_MSIRTrain.nxc](#).

8.1.2.1300 #define TRAIN_CHANNEL_2 1

IR Train channel 2

8.1.2.1301 #define TRAIN_CHANNEL_3 2

IR Train channel 3

8.1.2.1302 #define TRAIN_CHANNEL_ALL 3

IR Train channel all

8.1.2.1303 #define TRAIN_FUNC_DECR_SPEED 2

PF/IR Train function decrement speed

8.1.2.1304 #define TRAIN_FUNC_INCR_SPEED 1

PF/IR Train function increment speed

Examples:

[ex_HTIRTrain.nxc](#), [ex_HTPFTrain.nxc](#), [ex_MSIRTrain.nxc](#), and [ex_MSPFTrain.nxc](#).

8.1.2.1305 #define TRAIN_FUNC_STOP 0

PF/IR Train function stop

8.1.2.1306 #define TRAIN_FUNC_TOGGLE_LIGHT 4

PF/IR Train function toggle light

8.1.2.1307 #define TRUE 1

A true value

Examples:

[ex_SysCommBTConnection.nxc](#).

8.1.2.1308 #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.

8.1.2.1309 #define Type 0

Type field. Contains one of the sensor type constants. Read/write.

Examples:

[ex_GetInput.nxc](#), and [ex_SetInput.nxc](#).

8.1.2.1310 #define UCHAR_MAX 255

The maximum value of the unsigned char type

8.1.2.1311 #define UF_PENDING_UPDATES 0x80

Are there any pending motor updates?

8.1.2.1312 #define UF_UPDATE_MODE 0x01

Commits changes to the OutputMode output property

8.1.2.1313 #define UF_UPDATE_PID_VALUES 0x10

Commits changes to the PID motor regulation properties

8.1.2.1314 #define UF_UPDATE_RESET_BLOCK_COUNT 0x20

Resets the NXT-G block-relative rotation counter

8.1.2.1315 #define UF_UPDATE_RESET_COUNT 0x08

Resets all rotation counters, cancels the current goal, and resets the rotation error-correction system

8.1.2.1316 #define UF_UPDATE_RESET_ROTATION_COUNT 0x40

Resets the program-relative (user) rotation counter

8.1.2.1317 #define UF_UPDATE_SPEED 0x02

Commits changes to the Power output property

8.1.2.1318 #define UF_UPDATE_TACHO_LIMIT 0x04

Commits changes to the TachoLimit output property

8.1.2.1319 #define UI_BT_CONNECT_REQUEST 0x40

RW - BT get connect accept in progress

8.1.2.1320 #define UI_BT_ERROR_ATTENTION 0x08

W - BT error attention

8.1.2.1321 #define UI_BT_PIN_REQUEST 0x80

RW - BT get pin code

8.1.2.1322 #define UI_BT_STATE_CONNECTED 0x02

RW - BT connected to something

8.1.2.1323 #define UI_BT_STATE_OFF 0x04

RW - BT power off

Examples:

[ex_SetBluetoothState.nxc](#).

8.1.2.1324 #define UI_BT_STATE_VISIBLE 0x01

RW - BT visible

8.1.2.1325 #define UI_BUTTON_ENTER 2

W - Insert enter button

Examples:

[ex_SetUIButton.nxc](#).

8.1.2.1326 #define UI_BUTTON_EXIT 4

W - Insert exit button

8.1.2.1327 #define UI_BUTTON_LEFT 1

W - Insert left arrow button

8.1.2.1328 #define UI_BUTTON_NONE 0

R - Button inserted are executed

8.1.2.1329 #define UI_BUTTON_RIGHT 3

W - Insert right arrow button

8.1.2.1330 #define UI_FLAGS_BUSY 0x40

R - UI busy running or datalogging (popup disabled)

8.1.2.1331 #define UI_FLAGS_DISABLE_EXIT 0x04

RW - Disable exit button

8.1.2.1332 #define UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER 0x02

RW - Disable left, right and enter button

8.1.2.1333 #define UI_FLAGS_ENABLE_STATUS_UPDATE 0x80

W - Enable status line to be updated

8.1.2.1334 #define UI_FLAGS_EXECUTE_LMS_FILE 0x20

W - Execute LMS file in "LMSfilename" (Try It)

8.1.2.1335 #define UI_FLAGS_REDRAW_STATUS 0x08

W - Redraw entire status line

Examples:

[ex_SetCommandFlags.nxc](#).

8.1.2.1336 #define UI_FLAGS_RESET_SLEEP_TIMER 0x10

W - Reset sleep timeout timer

8.1.2.1337 #define UI_FLAGS_UPDATE 0x01

W - Make changes take effect

8.1.2.1338 #define UI_STATE_BT_ERROR 16

R - BT error

8.1.2.1339 #define UI_STATE_CONNECT_REQUEST 12

RW - Request for connection accept

8.1.2.1340 #define UI_STATE_DRAW_MENU 6

RW - Execute function and draw menu icons

8.1.2.1341 #define UI_STATE_ENTER_PRESSED 10

RW - Load selected function and next menu id

8.1.2.1342 #define UI_STATE_EXECUTE_FILE 13

RW - Execute file in "LMSfilename"

8.1.2.1343 #define UI_STATE_EXECUTING_FILE 14

R - Executing file in "LMSfilename"

8.1.2.1344 #define UI_STATE_EXIT_PRESSED 11

RW - Load selected function and next menu id

8.1.2.1345 #define UI_STATE_INIT_DISPLAY 0

RW - Init display and load font, menu etc.

8.1.2.1346 #define UI_STATE_INIT_INTRO 2

R - Display intro

8.1.2.1347 #define UI_STATE_INIT_LOW_BATTERY 1

R - Low battery voltage at power on

8.1.2.1348 #define UI_STATE_INIT_MENU 4

RW - Init menu system

8.1.2.1349 #define UI_STATE_INIT_WAIT 3

RW - Wait for initialization end

8.1.2.1350 #define UI_STATE_LEFT_PRESSED 8

RW - Load selected function and next menu id

8.1.2.1351 #define UI_STATE_LOW_BATTERY 15

R - Low battery at runtime

Examples:

[ex_SetUIState.nxc](#).

8.1.2.1352 #define UI_STATE_NEXT_MENU 5

RW - Next menu icons ready for drawing

8.1.2.1353 #define UI_STATE_RIGHT_PRESSED 9

RW - Load selected function and next menu id

8.1.2.1354 #define UI_STATE_TEST_BUTTONS 7

RW - Wait for buttons to be pressed

8.1.2.1355 #define UI_VM_IDLE 0

VM_IDLE: Just sitting around. Request to run program will lead to ONE of the VM_RUN* states.

8.1.2.1356 #define UI_VM_RESET1 4

VM_RESET2: Final clean up and return to IDLE

8.1.2.1357 #define UI_VM_RESET2 5

VM_RESET1: Initialize state variables and some I/O devices -- executed when programs end

8.1.2.1358 #define UI_VM_RUN_FREE 1

VM_RUN_FREE: Attempt to run as many instructions as possible within our timeslice

8.1.2.1359 #define UI_VM_RUN_PAUSE 3

VM_RUN_PAUSE: Program still "active", but someone has asked us to pause

8.1.2.1360 #define UI_VM_RUN_SINGLE 2

VM_RUN_SINGLE: Run exactly one instruction per timeslice

8.1.2.1361 #define UIModuleID 0x000C0001

The Ui module ID

8.1.2.1362 #define UIModuleName "Ui.mod"

The Ui module name

8.1.2.1363 #define UINT_MAX 65535

The maximum value of the unsigned int type

8.1.2.1364 #define UIOffsetAbortFlag 40

RW - Long Abort (true == use long press to abort) (1 byte)

8.1.2.1365 #define UIOffsetBatteryState 30

W - Battery state (0..4 capacity) (1 byte)

8.1.2.1366 #define UIOffsetBatteryVoltage 4

R - Battery voltage in millivolts (2 bytes)

8.1.2.1367 #define UIOffsetBluetoothState 31

W - Bluetooth state (0=on, 1=visible, 2=conn, 3=conn.visible, 4=off, 5=dfu) (1 byte)

8.1.2.1368 #define UIOffsetButton 28

RW - Insert button (buttons enumerated above) (1 byte)

8.1.2.1369 #define UIOffsetError 37

W - Error code (1 byte)

8.1.2.1370 #define UIOffsetFlags 26

RW - Update command flags (flags enumerated above) (1 byte)

8.1.2.1371 #define UIOffsetForceOff 39

W - Force off (> 0 = off) (1 byte)

8.1.2.1372 #define UIOffsetLMSfilename 6

W - LMS filename to execute (Try It) (20 bytes)

8.1.2.1373 #define UIOffsetOBPPointer 38

W - Actual OBP step (0 - 4) (1 byte)

8.1.2.1374 #define UIOffsetPMenu 0

W - Pointer to menu file (4 bytes)

8.1.2.1375 #define UIOffsetRechargeable 35

R - Rechargeable battery (0 = no, 1 = yes) (1 byte)

8.1.2.1376 #define UIOffsetRunState 29

W - VM Run state (0 = stopped, 1 = running) (1 byte)

8.1.2.1377 #define UIOffsetSleepTimeout 33

RW - Sleep timeout time (min) (1 byte)

8.1.2.1378 #define UIOffsetSleepTimer 34

RW - Sleep timer (min) (1 byte)

8.1.2.1379 #define UIOffsetState 27

RW - UI state (states enumerated above) (1 byte)

8.1.2.1380 #define UIOffsetUsbState 32

W - Usb state (0=disconnected, 1=connected, 2=working) (1 byte)

8.1.2.1381 #define UIOffsetVolume 36

RW - Volume used in UI (0 - 4) (1 byte)

8.1.2.1382 #define ULONG_MAX 4294967295

The maximum value of the unsigned long type

8.1.2.1383 #define UpdateCalibCacheInfo 43

Update sensor calibration cache information

8.1.2.1384 #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.

8.1.2.1385 #define US_CMD_CONTINUOUS 0x02

Command to put the ultrasonic sensor into continuous polling mode (default)

8.1.2.1386 #define US_CMD_EVENTCAPTURE 0x03

Command to put the ultrasonic sensor into event capture mode

8.1.2.1387 #define US_CMD_OFF 0x00

Command to turn off the ultrasonic sensor

Examples:

[ex_writei2cregister.nxc](#).

8.1.2.1388 #define US_CMD_SINGLESHTOT 0x01

Command to put the ultrasonic sensor into single shot mode

8.1.2.1389 #define US_CMD_WARMRESET 0x04

Command to warm reset the ultrasonic sensor

8.1.2.1390 #define US_REG_ACTUAL_ZERO 0x50

The register address used to store the actual zero value

8.1.2.1391 #define US_REG_CM_INTERVAL 0x40

The register address used to store the CM interval

8.1.2.1392 #define US_REG_FACTORY_ACTUAL_ZERO 0x11

The register address containing the factory setting for the actual zero value

8.1.2.1393 #define US_REG_FACTORY_SCALE_DIVISOR 0x13

The register address containing the factory setting for the scale divisor value

8.1.2.1394 #define US_REG_FACTORY_SCALE_FACTOR 0x12

The register address containing the factory setting for the scale factor value

8.1.2.1395 #define US_REG_MEASUREMENT_UNITS 0x14

The register address containing the measurement units (degrees C or F)

8.1.2.1396 #define US_REG_SCALE_DIVISOR 0x52

The register address used to store the scale divisor value

8.1.2.1397 #define US_REG_SCALE_FACTOR 0x51

The register address used to store the scale factor value

8.1.2.1398 #define USB_CMD_READY 0x01

A constant representing usb direct command

8.1.2.1399 #define USB_PROTOCOL_OVERHEAD 2

Size of USB Overhead in bytes -- Command type byte + Command

8.1.2.1400 #define USHRT_MAX 65535

The maximum value of the unsigned short type

8.1.2.1401 #define WriteSemData 41

Write motor semaphore data

8.2 NXCAPIDocs.h File Reference

Additional documentation for the NXC API. #include "NXCDefs.h"

8.2.1 Detailed Description

Additional documentation for the NXC API. [NXCAPIDocs.h](#) contains additional documentation for the NXC API

License:

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of this code is John Hansen. Portions created by John Hansen are Copyright (C) 2009-2010 John Hansen. All Rights Reserved.

Author:

John Hansen (briexcc_at_comcast.net)

Date:

2010-06-22

Version:

5

8.3 NXCDefs.h File Reference

Constants, macros, and API functions for NXC. #include "NBCCCommon.h"

Data Structures

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

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 __RETFVAL__, _X }
Compute integral part.
- #define **Frac**(_X) asm { frac __FLTRETVAL__, _X }
Compute fractional part.
- #define **MulDiv32**(_A, _B, _C) asm { muldiv __RETFVAL__, _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 _X __FLTRETVAL__ }
Compute hyperbolic sine (degrees).
- #define **Atan2D**(_Y, _X) asm { atan2d __FLTRETVAL__, _Y, _X }
Compute arc tangent with two parameters (degrees).
- #define **getc**(_handle) fgetc(_handle)
Get character from file.
- #define **putc**(_ch, _handle) fputc(_ch, _handle)
Write character to file.
- #define **SEEK_SET** 0
- #define **SEEK_CUR** 1
- #define **SEEK_END** 2
- #define **RICSetValue**(_data, _idx, _newval) _data[(_idx)] = (_newval)&0xFF;
 _data[(_idx)+1] = (_newval)>>8
Set the value of an element in an RIC data array.

Functions

- void **SetSensorType** (const byte &port, byte type)
Set sensor type.
- void **SetSensorMode** (const byte &port, byte mode)
Set sensor mode.
- void **ClearSensor** (const byte &port)
Clear a sensor value.
- void **ResetSensor** (const byte &port)
Reset the sensor port.
- void **SetSensor** (const byte &port, const unsigned int config)
Set sensor configuration.
- void **SetSensorTouch** (const byte &port)

Configure a touch sensor.

- void **SetSensorLight** (const byte &port, bool bActive=true)

Configure a light sensor.

- void **SetSensorSound** (const byte &port, bool bdBScaling=true)

Configure a sound sensor.

- void **SetSensorLowspeed** (const byte &port, bool bIsPowered=true)

Configure an I2C sensor.

- void **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 &pct-FullScale)

Set percent full scale.

- void [SetCustomSensorZeroOffset](#) (const byte port, const int &zeroOffset)

Set custom 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 &output-Level)

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 PWN 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 fontaddr)
Set the display font memory address.

- void **SetDisplayDisplay** (unsigned long dispaddr)
Set the display memory address.

- void **SetDisplayEraseMask** (unsigned long eraseMask)
Set the display erase mask.

- void **SetDisplayFlags** (byte flags)
Set the display flags.

- void **SetDisplayTextLinesCenterFlags** (byte ctrFlags)
Set the display text lines center flags.

- void **SetDisplayUpdateMask** (unsigned long updateMask)
Set the display update mask.

- void **SetDisplayContrast** (byte contrast)
Set the display contrast.

- char **PlayFile** (string filename)
Play a file.

- char **PlayFileEx** (string filename, byte volume, bool loop)
Play a file with extra options.

- char **PlayTone** (unsigned int frequency, unsigned int duration)

Play a tone.

- char [PlayToneEx](#) (unsigned int frequency, unsigned int duration, byte volume, bool loop)

Play a tone with extra options.

- byte [SoundState](#) ()

Get sound module state.

- byte [SoundFlags](#) ()

Get sound module flags.

- byte [StopSound](#) ()

Stop sound.

- unsigned int [SoundFrequency](#) ()

Get sound frequency.

- unsigned int [SoundDuration](#) ()

Get sound duration.

- unsigned int [SoundSampleRate](#) ()

Get sample rate.

- byte [SoundMode](#) ()

Get sound mode.

- byte [SoundVolume](#) ()

Get volume.

- void [SetSoundDuration](#) (unsigned int duration)

Set sound duration.

- void [SetSoundFlags](#) (byte flags)

Set sound module flags.

- void [SetSoundFrequency](#) (unsigned int frequency)

Set sound frequency.

- void [SetSoundMode](#) (byte mode)

Set sound mode.

- void [SetSoundModuleState](#) (byte state)

Set sound module state.

- void [SetSoundSampleRate](#) (unsigned int sampleRate)
Set sample rate.
- void [SetSoundVolume](#) (byte volume)
Set sound volume.
- void [SysSoundPlayFile](#) (SoundPlayFileType &args)
Play sound file.
- void [SysSoundPlayTone](#) (SoundPlayToneType &args)
Play tone.
- void [SysSoundGetState](#) (SoundGetStateType &args)
Get sound state.
- void [SysSoundSetState](#) (SoundSetStateType &args)
Set sound state.
- void [PlaySound](#) (const int &aCode)
Play a system sound.
- void [PlayTones](#) (Tone tones[])
Play multiple tones.
- byte [SensorUS](#) (const byte port)
Read ultrasonic sensor value.
- char [ReadSensorUSEx](#) (const byte port, byte &values[])
Read multiple ultrasonic sensor values.
- char [ReadSensorEMeter](#) (const byte port, float &vIn, float &aIn, float &vOut, float &aOut, int &joules, float &wIn, float &wOut)
Read the LEGO EMeter values.
- long [LowspeedStatus](#) (const byte port, byte &bytesready)
Get lowspeed status.
- long [LowspeedCheckStatus](#) (const byte port)
Check lowspeed status.
- byte [LowspeedBytesReady](#) (const byte port)

Get lowspeed bytes ready.

- long **LowspeedWrite** (const byte port, byte retlen, byte buffer[])
Write lowspeed data.

- long **LowspeedRead** (const byte port, byte buflen, byte &buffer[])
Read lowspeed data.

- long **I2CStatus** (const byte port, byte &bytesready)
Get I2C status.

- long **I2CCheckStatus** (const byte port)
Check I2C status.

- byte **I2CBytesReady** (const byte port)
Get I2C bytes ready.

- long **I2CWrite** (const byte port, byte retlen, byte buffer[])
Write I2C data.

- long **I2CRead** (const byte port, byte buflen, byte &buffer[])
Read I2C data.

- long **I2CBytes** (const byte port, byte inbuf[], byte &count, byte &outbuf[])
Perform an I2C write/read transaction.

- char **ReadI2CRegister** (byte port, byte i2caddr, byte reg, byte &out)
Read I2C register.

- char **WriteI2CRegister** (byte port, byte i2caddr, byte reg, byte val)
Write I2C register.

- string **I2CDeviceInfo** (byte port, byte i2caddr, byte info)
Read I2C device information.

- string **I2CVersion** (byte port, byte i2caddr)
Read I2C device version.

- string **I2CVendorId** (byte port, byte i2caddr)
Read I2C device vendor.

- string **I2CDeviceId** (byte port, byte i2caddr)

Read I2C device identifier.

- long **I2CSendCommand** (byte port, byte i2caddr, byte cmd)
Send an I2C command.
- void **GetLSInputBuffer** (const byte port, const byte offset, byte cnt, byte &data[])
Get I2C input buffer data.
- void **GetLSOutputBuffer** (const byte port, const byte offset, byte cnt, byte &data[])
Get I2C output buffer data.
- byte **LSInputBufferInPtr** (const byte port)
Get I2C input buffer in-pointer.
- byte **LSInputBufferOutPtr** (const byte port)
Get I2C input buffer out-pointer.
- byte **LSInputBufferBytesToRx** (const byte port)
Get I2C input buffer bytes to rx.
- byte **LSOutputBufferInPtr** (const byte port)
Get I2C output buffer in-pointer.
- byte **LSOutputBufferOutPtr** (const byte port)
Get I2C output buffer out-pointer.
- byte **LSOutputBufferBytesToRx** (const byte port)
Get I2C output buffer bytes to rx.
- byte **LSMode** (const byte port)
Get I2C mode.
- byte **LSChannelState** (const byte port)
Get I2C channel state.
- byte **LSErrorType** (const byte port)
Get I2C error type.
- byte **LSState** ()
Get I2C state.

- byte **LSSpeed** ()
Get I2C speed.
- byte **LSNoRestartOnRead** ()
Get I2C no restart on read setting.
- void **SysCommLSSWrite** (**CommLSSWriteType** &args)
Write to a Lowspeed sensor.
- void **SysCommLSRead** (**CommLSReadType** &args)
Read from a Lowspeed sensor.
- void **SysCommLSCheckStatus** (**CommLSCheckStatusType** &args)
Check Lowspeed sensor status.
- void **SysCommLSSWriteEx** (**CommLSSWriteExType** &args)
Write to a Lowspeed sensor (extra).
- unsigned long **CurrentTick** ()
Read the current system tick.
- unsigned long **FirstTick** ()
Get the first tick.
- long **ResetSleepTimer** ()
Reset the sleep timer.
- void **SysCall** (byte funcID, variant &args)
Call any system function.
- void **SysGetStartTick** (**GetStartTickType** &args)
Get start tick.
- void **SysKeepAlive** (**KeepAliveType** &args)
Keep alive.
- void **SysIOMapRead** (**IOMapReadType** &args)
Read from IOMap by name.
- void **SysIOMapWrite** (**IOMapWriteType** &args)
Write to IOMap by name.

- void **SysIOMapReadByID** (**IOMapReadByIDType** &args)
Read from IOMap by identifier.
- void **SysIOMapWriteByID** (**IOMapWriteByIDType** &args)
Write to IOMap by identifier.
- void **SysDatalogWrite** (**DatalogWriteType** &args)
Write to the datalog.
- void **SysDatalogGetTimes** (**DatalogGetTimesType** &args)
Get datalog times.
- void **SysReadSemData** (**ReadSemDataType** &args)
Read semaphore data.
- void **SysWriteSemData** (**WriteSemDataType** &args)
Write semaphore data.
- void **SysUpdateCalibCacheInfo** (**UpdateCalibCacheInfoType** &args)
Update calibration cache information.
- void **SysComputeCalibValue** (**ComputeCalibValueType** &args)
Compute calibration values.
- 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)
Release 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.
- void **SetIOMapBytes** (string moduleName, unsigned int offset, unsigned int count, byte data[])
Set IOMap bytes by name.
- void **SetIOMapValue** (string moduleName, unsigned int offset, variant value)
Set IOMap value by name.
- void **GetIOMapBytes** (string moduleName, unsigned int offset, unsigned int count, byte &data[])
Get IOMap bytes by name.
- void **GetIOMapValue** (string moduleName, unsigned int offset, variant &value)
Get IOMap value by name.
- void **GetLowSpeedModuleBytes** (unsigned int offset, unsigned int count, byte &data[])
Get Lowspeed module IOMap bytes.
- void **GetDisplayModuleBytes** (unsigned int offset, unsigned int count, byte &data[])
Get Display module IOMap bytes.
- void **GetCommModuleBytes** (unsigned int offset, unsigned int count, byte &data[])
Get Comm module IOMap bytes.
- void **GetCommandModuleBytes** (unsigned int offset, unsigned int count, byte &data[])
Get Command module IOMap bytes.

- void [SetCommandModuleBytes](#) (unsigned int offset, unsigned int count, byte data[])
Set Command module IOMap bytes.
- void [SetLowSpeedModuleBytes](#) (unsigned int offset, unsigned int count, byte data[])
Set Lowspeed module IOMap bytes.
- void [SetDisplayModuleBytes](#) (unsigned int offset, unsigned int count, byte data[])
Set Display module IOMap bytes.
- void [SetCommModuleBytes](#) (unsigned int offset, unsigned int count, byte data[])
Set Comm module IOMap bytes.
- void [SetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte data[])
Set IOMap bytes by ID.
- void [SetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant value)
Set IOMap value by ID.
- void [GetIOMapBytesByID](#) (unsigned long moduleId, unsigned int offset, unsigned int count, byte &data[])
Get IOMap bytes by ID.
- void [GetIOMapValueByID](#) (unsigned long moduleId, unsigned int offset, variant &value)
Get IOMap value by ID.
- void [SetCommandModuleValue](#) (unsigned int offset, variant value)
Set Command module IOMap value.
- void [SetIOCctrlModuleValue](#) (unsigned int offset, variant value)
Set IOCctrl module IOMap value.
- void [SetLoaderModuleValue](#) (unsigned int offset, variant value)
Set Loader module IOMap value.
- void [SetUIModuleValue](#) (unsigned int offset, variant value)
Set Ui module IOMap value.

- void **SetSoundModuleValue** (unsigned int offset, variant value)
Set Sound module IOMap value.
- void **SetButtonModuleValue** (unsigned int offset, variant value)
Set Button module IOMap value.
- void **SetInputModuleValue** (unsigned int offset, variant value)
Set Input module IOMap value.
- void **SetOutputModuleValue** (unsigned int offset, variant value)
Set Output module IOMap value.
- void **SetLowSpeedModuleValue** (unsigned int offset, variant value)
Set Lowspeed module IOMap value.
- void **SetDisplayModuleValue** (unsigned int offset, variant value)
Set Display module IOMap value.
- void **SetCommModuleValue** (unsigned int offset, variant value)
Set Comm module IOMap value.
- void **GetCommandModuleValue** (unsigned int offset, variant &value)
Get Command module IOMap value.
- void **GetLoaderModuleValue** (unsigned int offset, variant &value)
Get Loader module IOMap value.
- void **GetSoundModuleValue** (unsigned int offset, variant &value)
Get Sound module IOMap value.
- void **GetButtonModuleValue** (unsigned int offset, variant &value)
Get Button module IOMap value.
- void **GetUIModuleValue** (unsigned int offset, variant &value)
Get Ui module IOMap value.
- void **GetInputModuleValue** (unsigned int offset, variant &value)
Get Input module IOMap value.
- void **GetOutputModuleValue** (unsigned int offset, variant &value)
Get Output module IOMap value.

- void [GetLowSpeedModuleValue](#) (unsigned int offset, variant &value)
Get LowSpeed module IOMap value.
- void [GetDisplayModuleValue](#) (unsigned int offset, variant &value)
Get Display module IOMap value.
- void [GetCommModuleValue](#) (unsigned int offset, variant &value)
Get Comm module IOMap value.
- void [PowerDown](#) ()
Power down the NXT.
- void [SleepNow](#) ()
Put the brick to sleep immediately.
- void [RebootInFirmwareMode](#) ()
Reboot the NXT in firmware download mode.
- char [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 **HMode** (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 **SetBTOOutputBuffer** (const byte offset, byte cnt, byte data[])

Set bluetooth output buffer data.

- void **SetBTOOutputBufferInPtr** (byte n)

Set bluetooth output buffer in-pointer.

- void **SetBTOOutputBufferOutPtr** (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 [SetHSMMode](#) (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 **VRunState** (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 reg)
Read HiTechnic IRSeeker2 register.

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

- bool [ReadSensorHTIRReceiver](#) (const byte port, char &pfdata[])
Read HiTechnic IRReceiver Power Function bytes.
- bool [ReadSensorHTIRReceiverEx](#) (const byte port, const byte offset, char &pfchar)
Read HiTechnic IRReceiver Power Function value.
- bool [ReadSensorHTIRSeeker2AC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9)
Read HiTechnic IRSeeker2 AC values.
- bool [ReadSensorHTIRSeeker2DC](#) (const byte port, byte &dir, byte &s1, byte &s3, byte &s5, byte &s7, byte &s9, byte &avg)
Read HiTechnic IRSeeker2 DC values.
- void [ReadSensorHTITouchMultiplexer](#) (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 [HTPFCComboDirect](#) (const byte port, const byte channel, const byte outa, const byte outb)
HTPFCComboDirect function.
- char [HTPFCComboPWM](#) (const byte port, const byte channel, const byte outa, const byte outb)
HTPFCComboPWM 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 **HTPFTTrain** (const byte port, const byte channel, const byte func)

HTPFTTrain 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 pwrvval)
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 pwrvval)
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, const byte i2caddr)
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, const byte i2caddr, int &x, int &y, int &z)
Read mindsensors acceleration values.
- bool **ReadSensorMSPlayStation** (const byte port, const byte i2caddr, byte &btnset1, byte &btnset2, byte &xleft, byte &yleft, byte &xright, byte &yright)
Read mindsensors playstation controller values.

- bool **ReadSensorMSRTClock** (const byte port, byte &sec, byte &min, byte &hrs, byte &dow, byte &date, byte &month, byte &year)
Read mindsensors RTClock values.
- bool **ReadSensorMSTilt** (const byte port, const byte i2caddr, byte &x, byte &y, byte &z)
Read mindsensors tilt values.
- int **MSReadValue** (const byte port, const byte i2caddr, const byte reg, const byte numbytes)
Read a mindsensors device value.
- char **MSEnergize** (const byte port, const byte i2caddr)
Turn on power to device.
- char **MSDeenergize** (const byte port, const byte i2caddr)
Turn off power to device.
- char **MSADPAOn** (const byte port, const byte i2caddr)
Turn on mindsensors ADPA mode.
- char **MSADPAOff** (const byte port, const byte i2caddr)
Turn off mindsensors ADPA mode.
- char **DISTNxGP2D12** (const byte port, const byte i2caddr)
Configure DISTNx as GP2D12.
- char **DISTNxGP2D120** (const byte port, const byte i2caddr)
Configure DISTNx as GP2D120.
- char **DISTNxGP2YA02** (const byte port, const byte i2caddr)
Configure DISTNx as GP2YA02.
- char **DISTNxGP2YA21** (const byte port, const byte i2caddr)
Configure DISTNx as GP2YA21.
- int **DISTNxDistance** (const byte port, const byte i2caddr)
Read DISTNx distance value.
- int **DISTNxMaxDistance** (const byte port, const byte i2caddr)
Read DISTNx maximum distance value.
- int **DISTNxMinDistance** (const byte port, const byte i2caddr)

Read DISTNx minimum distance value.

- byte **DISTNxModuleType** (const byte port, const byte i2caddr)
Read DISTNx module type value.

- byte **DISTNxNumPoints** (const byte port, const byte i2caddr)
Read DISTNx num points value.

- int **DISTNxVoltage** (const byte port, const byte i2caddr)
Read DISTNx voltage value.

- char **ACCLNxCalibrateX** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx X-axis.

- char **ACCLNxCalibrateXEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx X-axis.

- char **ACCLNxCalibrateY** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx Y-axis.

- char **ACCLNxCalibrateYEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx Y-axis.

- char **ACCLNxCalibrateZ** (const byte port, const byte i2caddr)
Calibrate ACCL-Nx Z-axis.

- char **ACCLNxCalibrateZEnd** (const byte port, const byte i2caddr)
Stop calibrating ACCL-Nx Z-axis.

- char **ACCLNxResetCalibration** (const byte port, const byte i2caddr)
Reset ACCL-Nx calibration.

- char **SetACCLNxSensitivity** (const byte port, const byte i2caddr, byte slevel)
Set ACCL-Nx sensitivity.

- byte **ACCLNxSensitivity** (const byte port, const byte i2caddr)
Read ACCL-Nx sensitivity value.

- int **ACCLNxXOffset** (const byte port, const byte i2caddr)
Read ACCL-Nx X offset value.

- int **ACCLNxXRange** (const byte port, const byte i2caddr)

Read ACCL-Nx X range value.

- int **ACCLNxYOffset** (const byte port, const byte i2caddr)

Read ACCL-Nx Y offset value.

- int **ACCLNxYRange** (const byte port, const byte i2caddr)

Read ACCL-Nx Y range value.

- int **ACCLNxZOffset** (const byte port, const byte i2caddr)

Read ACCL-Nx Z offset value.

- int **ACCLNxZRange** (const byte port, const byte i2caddr)

Read ACCL-Nx Z range value.

- char **PSPNxDigital** (const byte &port, const byte &i2caddr)

Configure PSPNx in digital mode.

- char **PSPNxAnalog** (const byte &port, const byte &i2caddr)

Configure PSPNx in analog mode.

- char **NRLLink2400** (const byte port, const byte i2caddr)

Configure NRLLink in 2400 baud mode.

- char **NRLLink4800** (const byte port, const byte i2caddr)

Configure NRLLink in 4800 baud mode.

- char **NRLLinkFlush** (const byte port, const byte i2caddr)

Flush NRLLink buffers.

- char **NRLLinkIRLong** (const byte port, const byte i2caddr)

Configure NRLLink in IR long mode.

- char **NRLLinkIRShort** (const byte port, const byte i2caddr)

Configure NRLLink in IR short mode.

- char **NRLLinkSetPF** (const byte port, const byte i2caddr)

Configure NRLLink in power function mode.

- char **NRLLinkSetRCX** (const byte port, const byte i2caddr)

Configure NRLLink in RCX mode.

- char **NRLLinkSetTrain** (const byte port, const byte i2caddr)

Configure NRLink in IR train mode.

- char **NRLinkTxRaw** (const byte port, const byte i2caddr)

Configure NRLink in raw IR transmit mode.

- byte **NRLinkStatus** (const byte port, const byte i2caddr)

Read NRLink status.

- char **RunNRLinkMacro** (const byte port, const byte i2caddr, const byte macro)

Run NRLink macro.

- char **WriteNRLinkBytes** (const byte port, const byte i2caddr, const byte data[])

Write data to NRLink.

- bool **ReadNRLinkBytes** (const byte port, const byte i2caddr, byte &data[])

Read data from NRLink.

- char **MSIRTrain** (const byte port, const byte i2caddr, const byte channel, const byte func)

MSIRTrain function.

- char **MSPFComboDirect** (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)

MSPFComboDirect function.

- char **MSPFComboPWM** (const byte port, const byte i2caddr, const byte channel, const byte outa, const byte outb)

MSPFComboPWM function.

- char **MSPFRawOutput** (const byte port, const byte i2caddr, const byte nibble0, const byte nibble1, const byte nibble2)

MSPFRawOutput function.

- char **MSPFRepeat** (const byte port, const byte i2caddr, const byte count, const unsigned int delay)

MSPFRepeat function.

- char **MSPFSingleOutputCST** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)

MSPFSingleOutputCST function.

- char **MSPFSingleOutputPWM** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte func)

MSPFSingleOutputPWM function.

- char **MSPFSinglePin** (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont)

MSPFSinglePin function.

- char **MSPFTrain** (const byte port, const byte i2caddr, const byte channel, const byte func)

MSPFTrain function.

- void **MSRCXSetNRLLinkPort** (const byte port, const byte i2caddr)

MSRCXSetIRLinkPort function.

- int **MSRCXBatteryLevel** (void)

MSRCXBatteryLevel function.

- int **MSRCXPoll** (const byte src, const byte value)

MSRCXPoll function.

- int **MSRCXPollMemory** (const unsigned int address)

MSRCXPollMemory function.

- void **MSRCXAbsVar** (const byte varnum, const byte 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 **MSRCXFFloat** (const byte outputs)
MSRCXFFloat function.
- void **MSRCXFwd** (const byte outputs)
MSRCXFwd function.
- void **MSRCXIncCounter** (const byte counter)
MSRCXIncCounter function.
- void **MSRCXIvertOutput** (const byte outputs)
MSRCXIvertOutput function.
- void **MSRCXMulVar** (const byte varnum, const byte src, unsigned int value)
MSRCXMulVar function.
- void **MSRCXMutetSound** (void)
MSRCXMutetSound 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 pwrvval)
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 pwrvval)
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 **multdiv32** (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 **streat** (string &dest, const string &src)
Concatenate strings.
- string **strncat** (string &dest, const string &src, unsigned int num)
Append characters from string.

- string **strcpy** (string &dest, const string &src)
Copy string.
- string **strncpy** (string &dest, const string &src, unsigned int num)
Copy characters from string.
- int **strcmp** (const string &str1, const string &str2)
Compare two strings.
- int **strncmp** (const string &str1, const string &str2, unsigned int num)
Compare characters of two strings.
- void **memcpy** (variant dest, variant src, byte num)
Copy memory.
- void **memmove** (variant dest, variant src, byte num)
Move memory.
- char **memcmp** (variant ptr1, variant ptr2, byte num)
Compare two blocks of memory.
- unsigned long **addressOf** (variant data)
Get the absolute address of a variable.
- unsigned long **reladdressOf** (variant data)
Get the relative address of a variable.
- unsigned long **addressOfEx** (variant data, bool relative)
Get the absolute or relative address of a variable.
- int **isupper** (int c)
Check if character is uppercase letter.
- int **islower** (int c)
Check if character is lowercase letter.
- int **isalpha** (int c)
Check if character is alphabetic.
- int **isdigit** (int c)
Check if character is decimal digit.

- int **isalnum** (int c)
Check if character is alphanumeric.
- int **isspace** (int c)
Check if character is a white-space.
- int **iscntrl** (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.

8.3.1 Detailed Description

Constants, macros, and API functions for NXC. `NXCDefs.h` contains declarations for the NXC NXT API resources

License:

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Initial Developer of this code is John Hansen. Portions created by John Hansen are Copyright (C) 2009-2010 John Hansen. All Rights Reserved.

Author:

John Hansen (bricxcc_at_comcast.net)

Date:

2010-06-28

Version:

79

8.3.2 Define Documentation

8.3.2.1 `#define _SENSOR_CFG(_type, _mode) (((_type)<<8)+(_mode))`

Macro for defining `SetSensor` combined type and mode constants

8.3.2.2 #define Acos(_X) asm { acos __FLTRETVAL__, _X }

Compute arc cosine. Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

Deprecated

Use [acos\(\)](#) instead.

Parameters:

`_X` Floating point value.

Returns:

Arc cosine of `_X`.

8.3.2.3 #define AcosD(_X) asm {acosd __FLTRETVAL__, _X }

Compute arc cosine (degrees). Computes the arc cosine of `_X`. Only constants or variables allowed (no expressions).

Deprecated

Use [acosd\(\)](#) instead.

Parameters:

`_X` Floating point value.

Returns:

Arc cosine of `_X`.

8.3.2.4 #define Asin(_X) asm { asin __FLTRETVAL__, _X }

Compute arc sine. Computes the arc sine of `_X`. Only constants or variables allowed (no expressions).

Deprecated

Use [asin\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Arc sine of $_X$.

8.3.2.5 #define AsinD(_X) asm { asind __FLTRETVAL__, _X }

Compute arch sine (degrees). Computes the arc sine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [asind\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Arc sine of $_X$.

8.3.2.6 #define Atan(_X) asm { atan __FLTRETVAL__, _X }

Compute arc tangent. Computes the arc tangent of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [atan\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Arc tangent of $_X$.

8.3.2.7 #define Atan2(_Y, _X) asm { atan2 __FLTRETVAL__, _Y, _X }

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of $_Y/_X$, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant. Only constants or variables allowed (no expressions).

Deprecated

Use [atan2\(\)](#) instead.

Parameters:

$_Y$ Floating point value representing a y coordinate.

$_X$ Floating point value representing an x coordinate.

Returns:

Arc tangent of $_Y/_X$, in the interval [-pi,+pi] radians.

8.3.2.8 #define Atan2D(_Y, _X) asm { atan2d __FLTRETVAL__, _Y, _X }

Compute arc tangent with two parameters (degrees). Computes the arc tangent of $_Y/_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [atan2d\(\)](#) instead.

Parameters:

$_Y$ Floating point value.

$_X$ Floating point value.

Returns:

Arc tangent of $_Y/_X$.

8.3.2.9 #define AtanD(_X) asm { atand __FLTRETVAL__, _X }

Compute arc tangent (degrees). Computes the arc tangent of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [atand\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Arc tangent of $_X$.

8.3.2.10 #define Ceil(_X) asm { ceil __FLTRETVAL__, _X }

Round up value. Computes the smallest integral value that is not less than $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [ceil\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

The smallest integral value not less than $_X$.

8.3.2.11 #define Cos(_X) asm { cos __FLTRETVAL__, _X }

Compute cosine. Computes the cosine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [cos\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Cosine of $_X$.

8.3.2.12 #define CosD(_X) asm { cosd __FLTRETVAL__, _X }

Compute cosine (degrees). Computes the cosine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [cosd\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Cosine of $_X$.

8.3.2.13 #define Cosh(_X) asm { cosh __FLTRETVAL__, _X }

Compute hyperbolic cosine. Computes the hyperbolic cosine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [cosh\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Hyperbolic cosine of $_X$.

8.3.2.14 #define CoshD(_X) asm { coshd __FLTRETVAL__, _X }

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [coshd\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Hyperbolic cosine of $_X$.

8.3.2.15 #define Exp(_X) asm { exp __FLTRETVAL__, _X }

Compute exponential function . Computes the base-e exponential function of $_X$, which is the e number raised to the power $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [exp\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Exponential value of $_X$.

8.3.2.16 #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 $_X$.

8.3.2.17 #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`.

8.3.2.18 #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](#).

8.3.2.19 #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 $_X$.

8.3.2.20 #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 $_X$.

8.3.2.21 #define MulDiv32(_A, _B, _C) asm { muldiv __RETVL__, _A, _B, _C }

Multiply and divide. Multiplies two 32-bit values and then divides the 64-bit result by a third 32-bit value. Only constants or variables allowed (no expressions).

Deprecated

Use [muldiv32\(\)](#) instead.

Parameters:

$_A$ 32-bit long value.

_B 32-bit long value.

_C 32-bit long value.

Returns:

The result of multiplying *_A* times *_B* and dividing by *_C*.

8.3.2.22 #define Pow(_Base, _Exponent) asm { pow __FLTRETRVAL__, _Base, _Exponent }

Raise to power. Computes *_Base* raised to the power *_Exponent*. Only constants or variables allowed (no expressions).

Deprecated

Use [pow\(\)](#) instead.

Parameters:

_Base Floating point value.

_Exponent Floating point value.

Returns:

The result of raising *_Base* to the power *_Exponent*.

8.3.2.23 #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](#).

```
8.3.2.24 #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

8.3.2.25 #define S1 0

Input port 1

Examples:

ex_ACCLNxCalibrateX.nxc ,	ex_ACCLNxCalibrateXEnd.nxc ,	ex_-
ACCLNxCalibrateY.nxc ,	ex_ACCLNxCalibrateYEnd.nxc ,	ex_-
ACCLNxCalibrateZ.nxc ,	ex_ACCLNxCalibrateZEnd.nxc ,	ex_-
ACCLNxResetCalibration.nxc ,	ex_ACCLNxSensitivity.nxc ,	ex_-
ACCLNxXOffset.nxc ,	ex_ACCLNxXRange.nxc ,	ex_ACCLNxYOffset.nxc ,
ex_ACCLNxYRange.nxc ,	ex_ACCLNxZOffset.nxc ,	ex_ACCLNxZRange.nxc ,
ex_ClearSensor.nxc ,	ex_ColorADRaw.nxc ,	ex_ColorBoolean.nxc ,
ex_ColorCalibration.nxc ,	ex_ColorCalibrationState.nxc ,	ex_-
ColorCalLimits.nxc ,	ex_ColorSensorRaw.nxc ,	ex_ColorSensorValue.nxc ,
ex_CustomSensorActiveStatus.nxc ,	ex_CustomSensorPercentFullScale.nxc ,	
ex_CustomSensorZeroOffset.nxc ,	ex_DISTNxDistance.nxc ,	ex_-
DISTNxGP2D12.nxc ,	ex_DISTNxGP2D120.nxc ,	ex_DISTNxGP2YA02.nxc ,
ex_DISTNxGP2YA21.nxc ,	ex_DISTNxMaxDistance.nxc ,	ex_-
DISTNxMinDistance.nxc ,	ex_DISTNxModuleType.nxc ,	ex_-
DISTNxNumPoints.nxc ,	ex_DISTNxVoltage.nxc ,	ex_GetInput.nxc ,
ex_GetLSInputBuffer.nxc ,	ex_GetLSSOutputBuffer.nxc ,	ex_HTIIRTrain.nxc ,
ex_HTPFComboDirect.nxc ,	ex_HTPFComboPWM.nxc ,	ex_-
HTPFRawOutput.nxc ,	ex_HTPFRepeat.nxc ,	ex_HTPFSingleOutputCST.nxc ,
ex_HTPFSingleOutputPWM.nxc ,	ex_HTPFSinglePin.nxc ,	ex_HTPFTrain.nxc ,
ex_HTRCXAddToDatalog.nxc ,	ex_HTRCXClearSensor.nxc ,	ex_-
HTRCXSetIRLinkPort.nxc ,	ex_HTRCXSetSensorMode.nxc ,	ex_-
HTRCXSetSensorType.nxc ,	ex_I2CBytesReady.nxc ,	ex_I2CCheckStatus.nxc ,
ex_I2CDeviceId.nxc ,	ex_I2CDeviceInfo.nxc ,	ex_I2CRead.nxc ,
I2CSendCommand.nxc ,	ex_I2CStatus.nxc ,	ex_I2CVendorId.nxc ,
I2CVersion.nxc ,	ex_i2cwrite.nxc ,	ex_LowspeedBytesReady.nxc ,
ex_LowspeedCheckStatus.nxc ,	ex_LowspeedRead.nxc ,	ex_-
lowspeedstatus.nxc ,	ex_LowspeedWrite.nxc ,	ex_LSChannelState.nxc ,

```
ex_LSErrorType.nxc,          ex_LSIInputBufferBytesToRx.nxc,          ex_-
LSIInputBufferInPtr.nxc,      ex_LSIInputBufferOutPtr.nxc,          ex_LSMODE.nxc,
ex_LSOOutputBufferBytesToRx.nxc,  ex_LSOOutputBufferInPtr.nxc,          ex_-
LSOOutputBufferOutPtr.nxc,    ex_MSADPAOff.nxc,          ex_MSADPAOn.nxc,
ex_MSDeenergize.nxc,         ex_MSEnergize.nxc,          ex_MSIRTrain.nxc,
ex_MSPFComboDirect.nxc,      ex_MSPFComboPWM.nxc,          ex_-
MSPFRawOutput.nxc,          ex_MSPFRepeat.nxc,          ex_MSPFSingleOutputCST.nxc,
ex_MSPFSingleOutputPWM.nxc,  ex_MSPFSinglePin.nxc,          ex_-
MSPFTrain.nxc,              ex_MSRCXAddToDatalog.nxc,          ex_MSRCXClearSensor.nxc,
ex_MSRCXSetNRLLinkPort.nxc,  ex_MSRCXSetSensorMode.nxc,
ex_MSRCXSetSensorType.nxc,   ex_MSRCXSumVar.nxc,          ex_-
MSReadValue.nxc,             ex_NRLLink2400.nxc,          ex_NRLLink4800.nxc,
ex_NRLLinkFlush.nxc,         ex_NRLLinkIRLong.nxc,          ex_NRLLinkIRShort.nxc,
ex_NRLLinkSetPF.nxc,         ex_NRLLinkSetRCX.nxc,          ex_NRLLinkSetTrain.nxc,
ex_NRLLinkStatus.nxc,        ex_NRLLinkTxRaw.nxc,          ex_PSPNxAnalog.nxc,
ex_PSPNxDigital.nxc,         ex_readi2cregister.nxc,          ex_ReadNRLLinkBytes.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_ReadSensorMSPlayStation.nxc,
ex_ReadSensorMSRTClock.nxc, ex_ReadSensorMSTilt.nxc,          ex_-
readsensorusex.nxc,         ex_RemoteResetScaledValue.nxc,          ex_-
RemoteSetInputMode.nxc,     ex_RunNRLLinkMacro.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_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_SetACCLNxSensitivity.nxc, ex_SetCustomSensorActiveStatus.nxc,          ex_-
SetCustomSensorPercentFullScale.nxc, ex_SetCustomSensorZeroOffset.nxc,
ex_SetHTColor2Mode.nxc,     ex_SetHTIRSeeker2Mode.nxc,          ex_SetInput.nxc,
ex_SetSensor.nxc,            ex_SetSensorBoolean.nxc,          ex_SetSensorColorBlue.nxc,
ex_SetSensorColorFull.nxc,   ex_SetSensorColorGreen.nxc,          ex_-
SetSensorColorNone.nxc,     ex_SetSensorColorRed.nxc,          ex_-
SetSensorDigiPinsDirection.nxc, ex_SetSensorDigiPinsOutputLevel.nxc,
ex_SetSensorDigiPinsStatus.nxc, ex_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](#), and [ex_WriteNRLinkBytes.nxc](#).

8.3.2.26 #define s16 int

Signed 16 bit type

8.3.2.27 #define S2 1

Input port 2

8.3.2.28 #define S3 2

Input port 3

8.3.2.29 #define s32 long

Signed 32 bit type

8.3.2.30 #define S4 3

Input port 4

Examples:

[ex_I2CBytes.nxc](#), and [ex_sensorus.nxc](#).

8.3.2.31 #define s8 char

Signed 8 bit type

8.3.2.32 #define SEEK_CUR 1

Seek from the current file position

Examples:

[ex_fseek.nxc](#).

8.3.2.33 #define SEEK_END 2

Seek from the end of the file

8.3.2.34 #define SEEK_SET 0

Seek from the beginning of the file

Examples:

[ex_sysfileseek.nxc](#).

8.3.2.35 #define SENSOR_1 Sensor(S1)

Read the value of the analog sensor on port S1

8.3.2.36 #define SENSOR_2 Sensor(S2)

Read the value of the analog sensor on port S2

8.3.2.37 #define SENSOR_3 Sensor(S3)

Read the value of the analog sensor on port S3

8.3.2.38 #define SENSOR_4 Sensor(S4)

Read the value of the analog sensor on port S4

**8.3.2.39 #define SENSOR_CELSIUS _SENSOR_CFG(SENSOR_TYPE_-
TEMPERATURE, SENSOR_MODE_CELSIUS)**

RCX temperature sensor in celcius mode

**8.3.2.40 #define SENSOR_COLORBLUE _SENSOR_CFG(SENSOR_TYPE_-
COLORBLUE, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (blue) in percent mode

**8.3.2.41 #define SENSOR_COLORFULL_SENSOR_CFG(SENSOR_TYPE_-
COLORFULL, SENSOR_MODE_RAW)**

NXT 2.0 color sensor (full) in raw mode

**8.3.2.42 #define SENSOR_COLORGREEN_SENSOR_CFG(SENSOR_TYPE_-
COLORGREEN, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (green) in percent mode

**8.3.2.43 #define SENSOR_COLORNONE_SENSOR_CFG(SENSOR_TYPE_-
COLORNONE, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (none) in percent mode

**8.3.2.44 #define SENSOR_COLORRED_SENSOR_CFG(SENSOR_TYPE_-
COLORRED, SENSOR_MODE_PERCENT)**

NXT 2.0 color sensor (red) in percent mode

**8.3.2.45 #define SENSOR_EDGE_SENSOR_CFG(SENSOR_TYPE_TOUCH,
SENSOR_MODE_EDGE)**

Touch sensor in edge mode

**8.3.2.46 #define SENSOR_FAHRENHEIT_SENSOR_CFG(SENSOR_TYPE_TEMPERATURE,
SENSOR_MODE_FAHRENHEIT)**

RCX temperature sensor in fahrenheit mode

**8.3.2.47 #define SENSOR_LIGHT_SENSOR_CFG(SENSOR_TYPE_LIGHT,
SENSOR_MODE_PERCENT)**

Light sensor in percent mode

**8.3.2.48 #define SENSOR_LOWSPEED_SENSOR_CFG(SENSOR_TYPE_-
LOWSPEED, SENSOR_MODE_RAW)**

NXT I2C sensor without 9V power in raw mode

8.3.2.49 #define SENSOR_LOWSPEED_9V _SENSOR_CFG(SENSOR_TYPE_LOWSPEED_9V, SENSOR_MODE_RAW)

NXT I2C sensor with 9V power in raw mode

8.3.2.50 #define SENSOR_MODE_BOOL IN_MODE_BOOLEAN

Boolean value (0 or 1)

Examples:

[ex_HTRCXSetSensorMode.nxc](#), and [ex_MSRCXSetSensorMode.nxc](#).

8.3.2.51 #define SENSOR_MODE_CELSIUS IN_MODE_CELSIUS

RCX temperature sensor value in degrees celcius

8.3.2.52 #define SENSOR_MODE_EDGE IN_MODE_TRANSITIONCNT

Counts the number of boolean transitions

8.3.2.53 #define SENSOR_MODE_FAHRENHEIT IN_MODE_FAHRENHEIT

RCX temperature sensor value in degrees fahrenheit

8.3.2.54 #define SENSOR_MODE_PERCENT IN_MODE_PCTFULLSCALE

Scaled value from 0 to 100

8.3.2.55 #define SENSOR_MODE_PULSE IN_MODE_PERIODCOUNTER

Counts the number of boolean periods

8.3.2.56 #define SENSOR_MODE_RAW IN_MODE_RAW

Raw value from 0 to 1023

Examples:

[ex_RemoteSetInputMode.nxc](#), and [ex_SetSensorMode.nxc](#).

8.3.2.57 #define SENSOR_MODE_ROTATION IN_MODE_ANGLESTEP

RCX rotation sensor (16 ticks per revolution)

**8.3.2.58 #define SENSOR_NXTLIGHT_SENSOR_CFG(SENSOR_TYPE_-
LIGHT_ACTIVE, SENSOR_MODE_PERCENT)**

NXT light sensor in active mode

**8.3.2.59 #define SENSOR_PULSE_SENSOR_CFG(SENSOR_TYPE_TOUCH,
SENSOR_MODE_PULSE)**

Touch sensor in pulse mode

**8.3.2.60 #define SENSOR_ROTATION_SENSOR_CFG(SENSOR_TYPE_-
ROTATION, SENSOR_MODE_ROTATION)**

RCX rotation sensor in rotation mode

**8.3.2.61 #define SENSOR_SOUND_SENSOR_CFG(SENSOR_TYPE_-
SOUND_DB, SENSOR_MODE_PERCENT)**

NXT sound sensor (dB) in percent mode

**8.3.2.62 #define SENSOR_TOUCH_SENSOR_CFG(SENSOR_TYPE_-
TOUCH, SENSOR_MODE_BOOL)**

Touch sensor in boolean mode

Examples:

[ex_SetSensor.nxc](#).

8.3.2.63 #define SENSOR_TYPE_COLORBLUE IN_TYPE_COLORBLUE

NXT 2.0 color sensor with blue light

8.3.2.64 #define SENSOR_TYPE_COLORFULL IN_TYPE_COLORFULL

NXT 2.0 color sensor in full color mode

**8.3.2.65 #define SENSOR_TYPE_COLORGREEN IN_TYPE_-
COLORGREEN**

NXT 2.0 color sensor with green light

8.3.2.66 #define SENSOR_TYPE_COLORNONE IN_TYPE_COLORNONE

NXT 2.0 color sensor with no light

8.3.2.67 #define SENSOR_TYPE_COLORRED IN_TYPE_COLORRED

NXT 2.0 color sensor with red light

8.3.2.68 #define SENSOR_TYPE_CUSTOM IN_TYPE_CUSTOM

NXT custom sensor

8.3.2.69 #define SENSOR_TYPE_HIGHSPEED IN_TYPE_HISPEED

NXT Hi-speed port (only S4)

8.3.2.70 #define SENSOR_TYPE_LIGHT IN_TYPE_REFLECTION

RCX light sensor

**8.3.2.71 #define SENSOR_TYPE_LIGHT_ACTIVE IN_TYPE_LIGHT_-
ACTIVE**

NXT light sensor with light

**8.3.2.72 #define SENSOR_TYPE_LIGHT_INACTIVE IN_TYPE_LIGHT_-
INACTIVE**

NXT light sensor without light

8.3.2.73 #define SENSOR_TYPE_LOWSPEED IN_TYPE_LOWSPEED

NXT I2C digital sensor

Examples:

[ex_RemoteSetInputMode.nxc](#).

8.3.2.74 #define SENSOR_TYPE_LOWSPEED_9V IN_TYPE_LOWSPEED_-9V

NXT I2C digital sensor with 9V power

8.3.2.75 #define SENSOR_TYPE_NONE IN_TYPE_NO_SENSOR

No sensor configured

8.3.2.76 #define SENSOR_TYPE_ROTATION IN_TYPE_ANGLE

RCX rotation sensor

8.3.2.77 #define SENSOR_TYPE_SOUND_DB IN_TYPE_SOUND_DB

NXT sound sensor with dB scaling

Examples:

[ex_SetInput.nxc](#).

8.3.2.78 #define SENSOR_TYPE_SOUND_DBA IN_TYPE_SOUND_DBA

NXT sound sensor with dBA scaling

8.3.2.79 #define SENSOR_TYPE_TEMPERATURE IN_TYPE_TEMPERATURE

RCX temperature sensor

8.3.2.80 #define SENSOR_TYPE_TOUCH IN_TYPE_SWITCH

NXT or RCX touch sensor

Examples:

[ex_HTRCXSetSensorType.nxc](#), [ex_MSRCXSetSensorType.nxc](#), and [ex_SetSensorType.nxc](#).

8.3.2.81 #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 $_X$.

8.3.2.82 #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 $_X$.

8.3.2.83 #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 $_X$.

8.3.2.84 #define SinhD(_X) asm { sinhd __FLTRETVAL__, _X }

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [sinhd\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Hyperbolic sine of $_X$.

8.3.2.85 #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 $_X$.

8.3.2.86 #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 $_X$.

8.3.2.87 #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 $_X$.

8.3.2.88 #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 $_X$.

8.3.2.89 #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 $_X$.

8.3.2.90 #define Trunc(_X) asm { trunc __RETNAL__, _X }

Compute integral part. Computes the integral part of $_X$. Only constants or variables allowed (no expressions).

Deprecated

Use [trunc\(\)](#) instead.

Parameters:

$_X$ Floating point value.

Returns:

Integral part of $_X$.

8.3.2.91 #define u16 unsigned int

Unsigned 16 bit type

8.3.2.92 #define u32 unsigned long

Unsigned 32 bit type

8.3.2.93 #define u8 unsigned char

Unsigned 8 bit type

8.3.3 Function Documentation**8.3.3.1 void abort() [inline]**

Abort current process. Aborts the process with an abnormal program termination. The function never returns to its caller.

Examples:

[ex_abort.nxc](#).

8.3.3.2 byte AbortFlag(void) [inline]

Read abort flag. Return the enhanced NBC/NXC firmware's abort flag.

Returns:

The current abort flag value. See [ButtonState constants](#).

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_AbortFlag.nxc](#).

8.3.3.3 variant abs (variant *num*) [inline]

Absolute value. Return the absolute value of the value argument. Any scalar type can be passed into this function.

Parameters:

num The numeric value.

Returns:

The absolute value of num. The return type matches the input type.

Examples:

[ex_abs.nxc](#).

**8.3.3.4 char ACCLNxCalibrateX (const byte *port*, const byte *i2caddr*)
[inline]**

Calibrate ACCL-Nx X-axis. Calibrate the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateX.nxc](#).

**8.3.3.5 char ACCLNxCalibrateXEnd (const byte *port*, const byte *i2caddr*)
[inline]**

Stop calibrating ACCL-Nx X-axis. Stop calibrating the mindsensors ACCL-Nx sensor X-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateXEnd.nxc](#).

8.3.3.6 char ACCLNxCalibrateY (const byte *port*, const byte *i2caddr*) [inline]

Calibrate ACCL-Nx Y-axis. Calibrate the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateY.nxc](#).

8.3.3.7 char ACCLNxCalibrateYEnd (const byte *port*, const byte *i2caddr*) [inline]

Stop calibrating ACCL-Nx Y-axis. Stop calibrating the mindsensors ACCL-Nx sensor Y-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateYEnd.nxc](#).

8.3.3.8 char ACCLNxCalibrateZ (const byte *port*, const byte *i2caddr*) [inline]

Calibrate ACCL-Nx Z-axis. Calibrate the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateZ.nxc](#).

8.3.3.9 char ACCLNxCalibrateZEnd (const byte *port*, const byte *i2caddr*) [inline]

Stop calibrating ACCL-Nx Z-axis. Stop calibrating the mindsensors ACCL-Nx sensor Z-axis. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxCalibrateZEnd.nxc](#).

**8.3.3.10 char ACCLNxResetCalibration (const byte *port*, const byte *i2caddr*)
[inline]**

Reset ACCL-Nx calibration. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_ACCLNxResetCalibration.nxc](#).

**8.3.3.11 byte ACCLNxSensitivity (const byte *port*, const byte *i2caddr*)
[inline]**

Read ACCL-Nx sensitivity value. Read the mindsensors ACCL-Nx sensitivity value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The sensitivity value.

Examples:

[ex_ACCLNxSensitivity.nxc](#).

8.3.3.12 int ACCLNxXOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx X offset value. Read the mindsensors ACCL-Nx sensor's X offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The X offset value.

Examples:

[ex_ACCLNxXOffset.nxc](#).

8.3.3.13 int ACCLNxXRange (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx X range value. Read the mindsensors ACCL-Nx sensor's X range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The X range value.

Examples:

[ex_ACCLNxXRange.nxc](#).

8.3.3.14 int ACCLNxYOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Y offset value. Read the mindsensors ACCL-Nx sensor's Y offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Y offset value.

Examples:

[ex_ACCLNxYOffset.nxc](#).

8.3.3.15 int ACCLNxYRange (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Y range value. Read the mindsensors ACCL-Nx sensor's Y range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Y range value.

Examples:

[ex_ACCLNxYRange.nxc](#).

8.3.3.16 int ACCLNxZOffset (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Z offset value. Read the mindsensors ACCL-Nx sensor's Z offset value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Z offset value.

Examples:

[ex_ACCLNxZOffset.nxc](#).

8.3.3.17 int ACCLNxZRange (const byte *port*, const byte *i2caddr*) [inline]

Read ACCL-Nx Z range value. Read the mindsensors ACCL-Nx sensor's Z range value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The Z range value.

Examples:

[ex_ACCLNxZRange.nxc](#).

8.3.3.18 float acos (float *x*) [inline]

Compute arc cosine. Computes the principal value of the arc cosine of *x*, expressed in radians. In trigonometrics, arc cosine is the inverse operation of cosine.

Parameters:

x Floating point value in the interval [-1,+1].

Returns:

Arc cosine of *x*, in the interval [0,pi] radians.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_acos.nxc](#).

8.3.3.19 float acosd (float *x*) [inline]

Compute arc cosine (degrees). Computes the principal value of the arc cosine of *x*, expressed in degrees. In trigonometrics, arc cosine is the inverse operation of cosine.

Parameters:

x Floating point value in the interval [-1,+1].

Returns:

Arc cosine of *x*, in the interval [0,180] degrees.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_acosd.nxc](#).

8.3.3.20 void Acquire (mutex *m*) [inline]

Acquire a mutex. Acquire the specified mutex variable. If another task already has acquired the mutex then the current task will be suspended until the mutex is released by the other task. This function is used to ensure that the current task has exclusive access to a shared resource, such as the display or a motor. After the current task has finished using the shared resource the program should call Release to allow other tasks to acquire the mutex.

Parameters:

m The mutex to acquire.

Examples:

[ex_Acquire.nxc](#), and [ex_Release.nxc](#).

8.3.3.21 unsigned long addressOf (variant *data*) [inline]

Get the absolute address of a variable. Get the absolute address of a variable and return it to the calling routine as an unsigned long value.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

Returns:

The absolute address of the variable.

Examples:

[ex_addressof.nxc](#).

8.3.3.22 unsigned long addressOfEx (variant *data*, bool *relative*) [inline]

Get the absolute or relative address of a variable. Get the absolute or relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

relative A boolean flag indicating whether you want to get the relative or absolute address.

Returns:

The absolute or relative address of the variable.

Examples:

[ex_addressofex.nxc](#).

8.3.3.23 void ArrayBuild (variant & *aout*[], variant *src1*, variant *src2*, ..., variant *srcN*) [inline]

Build an array. Build a new array from the specified source(s). The sources can be of any type so long as the number of dimensions is equal to or one less than the number of dimensions in the output array and the type is compatible with the type of the output array. If a source is an array with the same number of dimensions as the output array then all of its elements are added to the output array.

Parameters:

aout The output array to build.
src1 The first source to build into the output array.
src2 The second source to build into the output array.
srcN The first source to build into the output array.

Examples:

[ex_arraybuild.nxc](#), [ex_SysCommHSWrite.nxc](#), and [ex_SysDatalogWrite.nxc](#).

**8.3.3.24 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](#).

8.3.3.25 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](#), [ex_atan2.nxc](#), [ex_atan2d.nxc](#), [ex_syslistfiles.nxc](#), [ex_tan.nxc](#),
and [ex_tand.nxc](#).

8.3.3.26 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](#).

8.3.3.27 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](#).

8.3.3.28 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](#).

8.3.3.29 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](#).

8.3.3.30 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](#).

8.3.3.31 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](#).

8.3.3.32 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](#).

8.3.3.33 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](#).

8.3.3.34 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](#).

8.3.3.35 float asin (float x) [inline]

Compute arc sine. Computes the principal value of the arc sine of x, expressed in radians. In trigonometrics, arc sine is the inverse operation of sine.

Parameters:

x Floating point value in the interval [-1,+1].

Returns:

Arc sine of *x*, in the interval [-pi/2,+pi/2] radians.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_asin.nxc](#).

8.3.3.36 float asind (float *x*) [inline]

Compute arc sine (degrees). Computes the principal value of the arc sine of *x*, expressed in degrees. In trigonometrics, arc sine is the inverse operation of sine.

Parameters:

x Floating point value in the interval [-1,+1].

Returns:

Arc sine of *x*, in the interval [-90,+90] degrees.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_asind.nxc](#).

8.3.3.37 float atan (float *x*) [inline]

Compute arc tangent. Computes the principal value of the arc tangent of *x*, expressed in radians. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use [atan2\(\)](#) if you need to determine the quadrant.

See also:[atan2\(\)](#)**Parameters:**

x Floating point value.

Returns:

Arc tangent of *x*, in the interval [-pi/2,+pi/2] radians.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:[ex_atan.nxc.](#)**8.3.3.38 float atan2 (float *y*, float *x*) [inline]**

Compute arc tangent with 2 parameters. Computes the principal value of the arc tangent of *y/x*, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant.

See also:[atan\(\)](#)**Parameters:**

y Floating point value representing a *y* coordinate.

x Floating point value representing an *x* coordinate.

Returns:

Arc tangent of *y/x*, in the interval [-pi,+pi] radians.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:[ex_atan2.nxc.](#)

8.3.3.39 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](#).

8.3.3.40 float atand (float x) [inline]

Compute arc tangent (degrees). Computes the principal value of the arc tangent of x, expressed in degrees. In trigonometrics, arc tangent is the inverse operation of tangent. Notice that because of the sign ambiguity, a function cannot determine with certainty in which quadrant the angle falls only by its tangent value. You can use atan2d if you need to determine the quadrant.

Parameters:

- x* Floating point value.

Returns:

Arc tangent of x, in the interval [-90,+90] degrees.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_atand.nxc](#).

8.3.3.41 float atof (const string & str) [inline]

Convert string to float. Parses the string str interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. The rest of the string after the last valid character is ignored and has no effect on the behavior of this function.

A valid floating point number for atof is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in str does not form a valid floating-point number as just defined, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String beginning with the representation of a floating-point number.

Returns:

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

Examples:

[ex_atof.nxc](#).

8.3.3.42 int atoi (const string & str) [inline]

Convert string to integer. Parses the string str interpreting its content as an integral number, which is returned as an int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str does not form a valid integral number, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String beginning with the representation of an integral number.

Returns:

On success, the function returns the converted integral number as an int value. If no valid conversion could be performed a zero value is returned.

Examples:

[ex_atoi.nxc](#).

8.3.3.43 long atol (const string & str) [inline]

Convert string to long integer. Parses the string str interpreting its content as an integral number, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str does not form a valid integral number, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String beginning with the representation of an integral number.

Returns:

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

Examples:

[ex_atol.nxc](#).

8.3.3.44 unsigned int BatteryLevel (void) [inline]

Get battery Level. Return the battery level in millivolts.

Returns:

The battery level

Examples:

[util_battery_1.nxc](#), and [util_battery_2.nxc](#).

8.3.3.45 byte BatteryState (void) [inline]

Get battery state. Return battery state information (0..4).

Returns:

The battery state (0..4)

Examples:

[ex_BatteryState.nxc](#).

8.3.3.46 byte bcd2dec (byte *bcd*) [inline]

Convert from BCD to decimal Return the decimal equivalent of the binary coded decimal value provided.

Parameters:

bcd The value you want to convert from bcd to decimal.

Returns:

The decimal equivalent of the binary coded decimal byte.

Examples:

[ex_bcd2dec.nxc](#).

8.3.3.47 byte BluetoothState (void) [inline]

Get bluetooth state. Return the bluetooth state.

Returns:

The bluetooth state. See [BluetoothState constants](#).

Examples:

[ex_BluetoothState.nxc](#).

8.3.3.48 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](#).

8.3.3.49 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](#).

8.3.3.50 int BrickDataBluecoreVersion (void) [inline]

Get NXT bluecore version. This method returns the bluecore version of the NXT.

Returns:

The NXT's bluecore version number.

Examples:

[ex_BrickDataBluecoreVersion.nxc](#).

8.3.3.51 byte BrickDataBtHardwareStatus (void) [inline]

Get NXT bluetooth hardware status. This method returns the Bluetooth hardware status of the NXT.

Returns:

The NXT's bluetooth hardware status.

Examples:

[ex_BrickDataBtHardwareStatus.nxc](#).

8.3.3.52 byte BrickDataBtStateStatus (void) [inline]

Get NXT bluetooth state status. This method returns the Bluetooth state status of the NXT.

Returns:

The NXT's bluetooth state status.

Examples:

[ex_BrickDataBtStateStatus.nxc](#).

8.3.3.53 string BrickDataName (void) [inline]

Get NXT name. This method returns the name of the NXT.

Returns:

The NXT's bluetooth name.

Examples:

[ex_BrickDataName.nxc](#).

8.3.3.54 byte BrickDataTimeoutValue (void) [inline]

Get NXT bluetooth timeout value. This method returns the Bluetooth timeout value of the NXT.

Returns:

The NXT's bluetooth timeout value.

Examples:

[ex_BrickDataTimeoutValue.nxc](#).

8.3.3.55 long BTConnectionClass (const byte *conn*) [inline]

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The class of the bluetooth device at the specified connection slot.

Examples:

[ex_BTConnectionClass.nxc](#).

8.3.3.56 byte BTConnectionHandleNum (const byte *conn*) [inline]

Get bluetooth device handle number. This method returns the handle number of the device at the specified index within the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The handle number of the bluetooth device at the specified connection slot.

Examples:

[ex_BTConnectionHandleNum.nxc](#).

8.3.3.57 byte BTConnectionLinkQuality (const byte *conn*) [inline]

Get bluetooth device link quality. This method returns the link quality of the device at the specified index within the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The link quality of the specified connection slot (unimplemented).

Warning:

This function is not implemented at the firmware level.

Examples:

[ex_BTConnectionLinkQuality.nxc](#).

8.3.3.58 string BTConnectionName (const byte *conn*) [inline]

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The name of the bluetooth device at the specified connection slot.

Examples:

[ex_BTConnectionName.nxc](#).

8.3.3.59 string BTConnectionPinCode (const byte *conn*) [inline]

Get bluetooth device pin code. This method returns the pin code of the device at the specified index in the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The pin code for the bluetooth device at the specified connection slot.

Examples:

[ex_BTConnectionPinCode.nxc](#).

8.3.3.60 byte BTConnectionStreamStatus (const byte *conn*) [inline]

Get bluetooth device stream status. This method returns the stream status of the device at the specified index within the Bluetooth connection table.

Parameters:

conn The connection slot (0..3).

Returns:

The stream status of the bluetooth device at the specified connection slot.

Examples:

[ex_BTConnectionStreamStatus.nxc](#).

8.3.3.61 long BTDeviceClass (const byte *devidx*) [inline]

Get bluetooth device class. This method returns the class of the device at the specified index within the Bluetooth device table.

Parameters:

devidx The device table index.

Returns:

The device class of the specified bluetooth device.

Examples:

[ex_BTDeviceClass.nxc](#).

8.3.3.62 byte BTDeviceCount (void) [inline]

Get bluetooth device count. This method returns the number of devices defined within the Bluetooth device table.

Returns:

The count of known bluetooth devices.

Examples:

[ex_BTDeviceCount.nxc](#).

8.3.3.63 string BTDeviceName (const byte *devidx*) [inline]

Get bluetooth device name. This method returns the name of the device at the specified index in the Bluetooth device table.

Parameters:

devidx The device table index.

Returns:

The device name of the specified bluetooth device.

Examples:

[ex_BTDeviceName.nxc](#).

8.3.3.64 byte BTDeviceNameCount (void) [inline]

Get bluetooth device name count. This method returns the number of device names defined within the Bluetooth device table. This usually has the same value as BTDeviceCount but it can differ in some instances.

Returns:

The count of known bluetooth device names.

Examples:

[ex_BTDeviceNameCount.nxc](#).

8.3.3.65 byte BTDeviceStatus (const byte devidx) [inline]

Get bluetooth device status. This method returns the status of the device at the specified index within the Bluetooth device table.

Parameters:

devidx The device table index.

Returns:

The status of the specified bluetooth device.

Examples:

[ex_BTDeviceStatus.nxc](#).

8.3.3.66 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](#).

8.3.3.67 byte BTInputBufferOutPtr (void) [inline]

Get bluetooth input buffer out-pointer. This method returns the value of the output pointer of the Bluetooth input buffer.

Returns:

The bluetooth input buffer's out-pointer value.

Examples:

[ex_BTInputBufferOutPtr.nxc](#).

8.3.3.68 byte BTOutputBufferInPtr (void) [inline]

Get bluetooth output buffer in-pointer. This method returns the value of the input pointer of the Bluetooth output buffer.

Returns:

The bluetooth output buffer's in-pointer value.

Examples:

[ex_BTOutputBufferInPtr.nxc](#).

8.3.3.69 byte BTOutputBufferOutPtr (void) [inline]

Get bluetooth output buffer out-pointer. This method returns the value of the output pointer of the Bluetooth output buffer.

Returns:

The bluetooth output buffer's out-pointer value.

Examples:

[ex_BTOutputBufferOutPtr.nxc](#).

8.3.3.70 byte ButtonCount (const byte *btn*, bool *resetCount*) [inline]

Get button press count. Return the number of times the specified button has been pressed since the last time the button press count was reset. Optionally clear the count after reading it.

Parameters:

btn The button to check. See [Button name constants](#).

resetCount Whether or not to reset the press counter.

Returns:

The button press count.

Examples:

[ex_ButtonCount.nxc](#).

8.3.3.71 byte ButtonLongPressCount (const byte *btn*) [inline]

Get button long press count. Return the long press count of the specified button.

Parameters:

btn The button to check. See [Button name constants](#).

Returns:

The button long press count.

Examples:

[ex_ButtonLongPressCount.nxc](#).

8.3.3.72 byte ButtonLongReleaseCount (const byte *btn*) [inline]

Get button long release count. Return the long release count of the specified button.

Parameters:

btn The button to check. See [Button name constants](#).

Returns:

The button long release count.

Examples:

[ex_ButtonLongReleaseCount.nxc](#).

8.3.3.73 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](#).

8.3.3.74 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](#).

8.3.3.75 byte ButtonReleaseCount (const byte *btn*) [inline]

Get button release count. Return the release count of the specified button.

Parameters:

btn The button to check. See [Button name constants](#).

Returns:

The button release count.

Examples:

[ex_ButtonReleaseCount.nxc](#).

8.3.3.76 byte ButtonShortReleaseCount (const byte *btn*) [inline]

Get button short release count. Return the short release count of the specified button.

Parameters:

btn The button to check. See [Button name constants](#).

Returns:

The button short release count.

Examples:

[ex_ButtonShortReleaseCount.nxc](#).

8.3.3.77 byte ButtonState (const byte *btn*) [inline]

Get button state. Return the state of the specified button. See [ButtonState constants](#).

Parameters:

btn The button to check. See [Button name constants](#).

Returns:

The button state.

Examples:

[ex_ButtonState.nxc](#).

8.3.3.78 string ByteArrayToStr (byte *data*[]) [inline]

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

See also:

[StrToByteArray](#), [ByteArrayToStrEx](#)

Parameters:

data A byte array.

Returns:

A string containing data and a null terminator byte.

Examples:

[ex_ByteArrayToStr.nxc](#), and [ex_string.nxc](#).

8.3.3.79 void ByteArrayToStrEx (byte *data*[], string & *str*) [inline]

Convert a byte array to a string. Convert the specified array to a string by appending a null terminator to the end of the array elements. The array must be a one-dimensional array of byte.

See also:

[StrToByteArray](#), [ByteArrayToStr](#)

Parameters:

data A byte array.

str A string variable reference which, on output, will contain data and a null terminator byte.

Examples:

[ex_ByteArrayToStrEx.nxc](#), and [ex_string.nxc](#).

8.3.3.80 float ceil (float *x*) [inline]

Round up value. Computes the smallest integral value that is not less than x.

Parameters:

x Floating point value.

Returns:

The smallest integral value not less than x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex.ceil.nxc](#).

8.3.3.81 char CircleOut (int *x*, int *y*, byte *radius*, unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw a circle. This function lets you draw a circle on the screen with its center at the specified x and y location, using the specified radius. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawCircle](#), [DrawCircleType](#)

Parameters:

- x* The x value for the center of the circle.
- y* The y value for the center of the circle.
- radius* The radius of the circle.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_CircleOut.nxc](#), and [ex_file_system.nxc](#).

8.3.3.82 void ClearLine (byte *line*) [inline]

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

Parameters:

- line* The line you want to clear. See [Line number constants](#).

Examples:

[ex_ClearLine.nxc](#).

8.3.3.83 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](#).

8.3.3.84 void ClearSensor (const byte & port) [inline]

Clear a sensor value. Clear the value of a sensor - only affects sensors that are configured to measure a cumulative quantity such as rotation or a pulse count.

Parameters:

port The port to clear. See [Input port constants](#).

Examples:

[ex_ClearSensor.nxc](#).

8.3.3.85 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](#), and [ex_file_system.nxc](#).

8.3.3.86 void Coast (byte *outputs*) [inline]

Coast motors. Turn off the specified outputs, making them coast to a stop.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

Examples:

[ex_coast.nxc](#).

8.3.3.87 void CoastEx (byte *outputs*, const byte *reset*) [inline]

Coast motors and reset counters. Turn off the specified outputs, making them coast to a stop.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_coastex.nxc](#).

8.3.3.88 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](#).

8.3.3.89 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](#).

8.3.3.90 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](#).

8.3.3.91 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](#).

8.3.3.92 `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](#).

8.3.3.93 `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](#).

**8.3.3.94 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](#).

8.3.3.95 byte CommandFlags (void) [inline]

Get command flags. Return the command flags.

Returns:

Command flags. See [CommandFlags constants](#)

Examples:

[ex_CommandFlags.nxc](#).

8.3.3.96 string Copy (string *str*, unsigned int *idx*, unsigned int *len*) [inline]

Copy a portion of a string. Returns a substring of a string.

Parameters:

str A string

idx The starting index of the substring.

len The length of the substring.

Returns:

The specified substring.

Examples:

[ex_Copy.nxc](#).

8.3.3.97 float cos (float *x*) [inline]

Compute cosine. Computes the cosine of an angle of *x* radians.

Parameters:

x Floating point value representing an angle expressed in radians.

Returns:

Cosine of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sin_cos.nxc](#).

8.3.3.98 float cosd (float *x*) [inline]

Compute cosine (degrees). Computes the cosine of an angle of *x* degrees.

Parameters:

x Floating point value representing an angle expressed in degrees.

Returns:

Cosine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sind_cosd.nxc](#).

8.3.3.99 float cosh (float x) [inline]

Compute hyperbolic cosine. Computes the hyperbolic cosine of x, expressed in radians.

Parameters:

x Floating point value.

Returns:

Hyperbolic cosine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_cosh.nxc](#).

8.3.3.100 float coshd (float x) [inline]

Compute hyperbolic cosine (degrees). Computes the hyperbolic cosine of x, expressed in degrees.

Parameters:

x Floating point value.

Returns:

Hyperbolic cosine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

8.3.3.101 unsigned int CreateFile (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]

Create a file. Create a new file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

Parameters:

fname The name of the file to create.

fsize The size of the file.

handle The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_CreateFile.nxc](#), and [ex_file_system.nxc](#).

8.3.3.102 unsigned int CreateFileLinear (string *fname*, unsigned int *fsize*, byte & *handle*) [inline]

Create a linear file. Create a new linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

Parameters:

fname The name of the file to create.

fsize The size of the file.

handle The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_CreateFileLinear.nxc](#).

8.3.3.103 `unsigned int CreateFileNonLinear (string fname, unsigned int fsize, byte & handle) [inline]`

Create a non-linear file. Create a new non-linear file with the specified filename and size and open it for writing. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename and size parameters must be constants, constant expressions, or variables. A file created with a size of zero bytes cannot be written to since the NXC file writing functions do not grow the file if its capacity is exceeded during a write attempt.

Parameters:

fname The name of the file to create.

fsize The size of the file.

handle The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_CreateFileNonLinear.nxc](#).

8.3.3.104 unsigned long CurrentTick () [inline]

Read the current system tick. This function lets you current system tick count.

Returns:

The current system tick count.

Examples:

[ex_CurrentTick.nxc](#), [ex_dispgout.nxc](#), and [util_rpm.nxc](#).

8.3.3.105 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](#).

8.3.3.106 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](#).

8.3.3.107 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](#).

8.3.3.108 unsigned int DeleteFile (string *fname*) [inline]

Delete a file. Delete the specified file. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

Parameters:

fname The name of the file to delete.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_delete_data_file.nxc](#), and [ex_DeleteFile.nxc](#).

8.3.3.109 byte DisplayContrast () [inline]

Read the display contrast setting. This function lets you read the current display contrast setting.

Returns:

The current display contrast (byte).

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_contrast.nxc](#).

8.3.3.110 unsigned long DisplayDisplay () [inline]

Read the display memory address. This function lets you read the current display memory address.

Returns:

The current display memory address.

Examples:

[ex_DisplayDisplay.nxc](#), and [ex_dispmisc.nxc](#).

8.3.3.111 unsigned long DisplayEraseMask () [inline]

Read the display erase mask value. This function lets you read the current display erase mask value.

Returns:

The current display erase mask value.

Examples:

[ex_DisplayEraseMask.nxc](#), and [ex_dispmisc.nxc](#).

8.3.3.112 byte DisplayFlags () [inline]

Read the display flags. This function lets you read the current display flags. Valid flag values are listed in the [Display flags](#) group.

Returns:

The current display flags.

Examples:

[ex_DisplayFlags.nxc](#), and [ex_dispmisc.nxc](#).

8.3.3.113 unsigned long DisplayFont () [inline]

Read the display font memory address. This function lets you read the current display font memory address.

Returns:

The current display font memory address.

Examples:

[ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_DisplayFont.nxc](#), and [ex_SetDisplayFont.nxc](#).

8.3.3.114 byte DisplayTextLinesCenterFlags () [inline]

Read the display text lines center flags. This function lets you read the current display text lines center flags.

Returns:

The current display text lines center flags.

Examples:

[ex_DisplayTextLinesCenterFlags.nxc](#), and [ex_dispmisc.nxc](#).

8.3.3.115 unsigned long DisplayUpdateMask () [inline]

Read the display update mask value. This function lets you read the current display update mask value.

Returns:

The current display update mask.

Examples:

[ex_DisplayUpdateMask.nxc](#), and [ex_dispmisc.nxc](#).

8.3.3.116 int DISTNxDistance (const byte *port*, const byte *i2caddr*) [inline]

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The distance value.

Examples:

[ex_DISTNxDistance.nxc](#).

8.3.3.117 char DISTNxGP2D12 (const byte *port*, const byte *i2caddr*) [inline]

Configure DISTNx as GP2D12. Configure the mindsensors DISTNx sensor as GP2D12. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2D12.nxc](#).

**8.3.3.118 char DISTNxGP2D120 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2D120. Configure the mindsensors DISTNx sensor as GP2D120. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2D120.nxc](#).

**8.3.3.119 char DISTNxGP2YA02 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2YA02. Configure the mindsensors DISTNx sensor as GP2YA02. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2YA02.nxc](#).

**8.3.3.120 char DISTNxGP2YA21 (const byte *port*, const byte *i2caddr*)
[inline]**

Configure DISTNx as GP2YA21. Configure the mindsensors DISTNx sensor as GP2YA21. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_DISTNxGP2YA21.nxc](#).

**8.3.3.121 int DISTNxMaxDistance (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The maximum distance value.

Examples:

[ex_DISTNxMaxDistance.nxc](#).

**8.3.3.122 int DISTNxMinDistance (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The distance value.

Examples:

[ex_DISTNxMinDistance.nxc](#).

**8.3.3.123 byte DISTNxModuleType (const byte *port*, const byte *i2caddr*)
[inline]**

Read DISTNx module type value. Read the mindsensors DISTNx sensor's module type value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The module type value.

Examples:

[ex_DISTNxModuleType.nxc](#).

**8.3.3.124 byte DISTNxNumPoints (const byte *port*, const byte *i2caddr*)
[inline]**

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

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The num points value.

Examples:

[ex_DISTNxNumPoints.nxc](#).

8.3.3.125 int DISTNxVoltage (const byte *port*, const byte *i2caddr*) [inline]

Read DISTNx voltage value. Read the mindsensors DISTNx sensor's voltage value. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The voltage value.

Examples:

[ex_DISTNxVoltage.nxc](#).

8.3.3.126 div_t div (int *numer*, int *denom*) [inline]

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type [div_t](#), which has two members: quot and rem.

Parameters:

numer Numerator.

denom Denominator.

Returns:

The result is returned by value in a structure defined in cstdlib, which has two members. For [div_t](#), these are, in either order: int quot; int rem.

Examples:

[ex_div.nxc](#).

8.3.3.127 `char EllipseOut (int x, int y, byte radiusX, byte radiusY, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw an ellipse. This function lets you draw an ellipse on the screen with its center at the specified x and y location, using the specified radii. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawEllipse](#), [DrawEllipseType](#)

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

- x* The x value for the center of the ellipse.
- y* The y value for the center of the ellipse.
- radiusX* The x axis radius.
- radiusY* The y axis radius.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_EllipseOut.nxc](#).

8.3.3.128 void ExitTo (task *newTask*) [inline]

Exit to another task. Immediately exit the current task and start executing the specified task.

Parameters:

newTask The task to start executing after exiting the current task.

Examples:

[alternating_tasks.nxc](#).

8.3.3.129 float exp (float *x*) [inline]

Compute exponential function. Computes the base-e exponential function of *x*, which is the e number raised to the power *x*.

Parameters:

x Floating point value.

Returns:

Exponential value of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_exp.nxc](#).

8.3.3.130 int fclose (byte *handle*) [inline]

Close file. Close the file associated with the specified file handle. The loader result code is returned as the value of the function call.

Parameters:

handle The handle of the file to be closed.

Returns:

The loader result code.

Examples:

[ex_fclose.nxc](#).

8.3.3.131 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_eof.nxc](#).

8.3.3.132 int fflush (byte handle) [inline]

Flush file. Writes any buffered data to the file. A zero value indicates success.

Parameters:

handle The handle of the file to be flushed.

Returns:

Currently always returns 0.

Examples:

[ex_fflush.nxc](#).

8.3.3.133 char fgetc (byte handle) [inline]

Get character from file. Returns the character currently pointed to by the internal file position indicator of the file specified by the handle. The internal file position indicator is then advanced by one character to point to the next character. The functions fgetc and getc are equivalent.

Parameters:

handle The handle of the file from which the character is read.

Returns:

The character read from the file.

Examples:

[ex_fgetc.nxc](#).

8.3.3.134 string fgets (string & str, int num, byte handle) [inline]

Get string from file. Reads characters from a file and stores them as a string into str until (num-1) characters have been read or either a newline or a the End-of-File is reached, whichever comes first. A newline character makes fgets stop reading, but it is considered a valid character and therefore it is included in the string copied to str. A null character is automatically appended in str after the characters read to signal the end of the string. Returns the string parameter.

Parameters:

str The string where the characters are stored.

num The maximum number of characters to be read.

handle The handle of the file from which the characters are read.

Returns:

The string read from the file.

Examples:

[ex_fgets.nxc](#).

8.3.3.135 `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](#).

8.3.3.136 `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](#).

8.3.3.137 unsigned long FirstTick () [inline]

Get the first tick. Return an unsigned 32-bit value, which is the system timing value (called a "tick") in milliseconds at the time that the program began running.

Returns:

The tick count at the start of program execution.

Examples:

[ex_FirstTick.nxc](#).

8.3.3.138 string Flatten (variant *num*) [inline]

Flatten a number to a string. Return a string containing the byte representation of the specified value.

Parameters:

num A number.

Returns:

A string containing the byte representation of the parameter num.

Examples:

[ex_Flatten.nxc](#), and [ex_string.nxc](#).

8.3.3.139 string FlattenVar (variant *x*) [inline]

Flatten any data to a string. Return a string containing the byte representation of the specified value.

See also:

[UnflattenVar](#)

Parameters:

x Any NXC datatype.

Returns:

A string containing the byte representation of the parameter x.

Examples:

[ex_FlattenVar.nxc](#), [ex_string.nxc](#), and [ex_UnflattenVar.nxc](#).

8.3.3.140 void Float (byte *outputs*) [inline]

Float motors. Make outputs float. Float is an alias for Coast.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

Examples:

[ex_float.nxc](#).

8.3.3.141 float floor (float x) [inline]

Round down value. Computes the largest integral value that is not greater than x.

Parameters:

x Floating point value.

Returns:

The largest integral value not greater than x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_floor.nxc](#).

8.3.3.142 void Follows (task *task1*, task *task2*, ..., task *taskN*) [inline]

Declare tasks that this task follows. Schedule this task to follow the specified tasks so that it will execute once any of the specified tasks has completed executing. This statement should occur once within a task - preferably at the start of the task definition. If multiple tasks declare that they follow the same task then they will all execute simultaneously unless other dependencies prevent them from doing so. Any number of tasks may be listed in the Follows statement.

Parameters:

task1 The first task that this task follows.

task2 The second task that this task follows.

taskN The last task that this task follows.

Examples:

[ex_Follows.nxc](#).

**8.3.3.143 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_dispnout.nxc](#).

8.3.3.144 `char FontTextOut (int x, int y, string filename, string str, unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw text with font. Draw a text value on the screen at the specified x and y location using a custom RIC font. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group. See the [Font drawing option constants](#) for options specific to the font drawing functions.

See also:

[FontNumOut](#), [SysDrawFont](#), [DrawFontType](#)

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

- x* The x value for the start of the text output.
- y* The y value for the start of the text output.
- filename* The filename of the RIC font.
- str* The text to output to the LCD screen.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_dispftout.nxc](#).

8.3.3.145 byte fopen (string *filename*, const string *mode*)

Open file. Opens the file whose name is specified in the parameter *filename* and associates it with a file handle that can be identified in future operations by the handle that is returned. The operations that are allowed on the stream and how these are performed are defined by the *mode* parameter.

Parameters:

filename The name of the file to be opened.

mode The file access mode. Valid values are "r" - opens an existing file for reading, "w" - creates a new file and opens it for writing, and "a" - opens an existing file for appending to the end of the file.

Returns:

The handle to the opened file.

Examples:

[ex_fopen.nxc](#).

8.3.3.146 void ForceOff (byte *num*) [inline]

Turn off NXT. Force the NXT to turn off if the specified value is greater than zero.

Parameters:

num If greater than zero the NXT will turn off.

Examples:

[ex_ForceOff.nxc](#).

8.3.3.147 string FormatNum (string *fmt*, variant *num*) [inline]

Format a number. Return the formatted string using the format and value. Use a standard numeric sprintf format specifier within the format string. The input string parameter may be a variable, constant, or expression.

Parameters:

fmt The string format containing a sprintf numeric format specifier.

num A number.

Returns:

A string containing the formatted numeric value.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressoffex.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atan2.nxc](#), [ex_atan2d.nxc](#), [ex_atand.nxc](#), [ex_delete_data_file.nxc](#), [ex_DisplayFont.nxc](#), [ex_file_system.nxc](#), [ex_FormatNum.nxc](#), [ex_GetBrickDataAddress.nxc](#), [ex_reladdressof.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_string.nxc](#), [ex_tan.nxc](#), [ex_tand.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

8.3.3.148 void fprintf (byte handle, string format, variant value) [inline]

Write formatted data to file. Writes a sequence of data formatted as the format argument specifies to a file. After the format parameter, the function expects one value argument.

Parameters:

handle The handle of the file to write to.

format A string specifying the desired format.

value A value to be formatted for writing to the file.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_fprintf.nxc](#).

8.3.3.149 char fputc (char *ch*, byte *handle*) [inline]

Write character to file. Writes a character to the file and advances the position indicator. The character is written at the current position of the file as indicated by the internal position indicator, which is then advanced one character. If there are no errors, the same character that has been written is returned. If an error occurs, EOF is returned.

Parameters:

ch The character to be written.

handle The handle of the file where the character is to be written.

Returns:

The character written to the file.

Examples:

[ex_fputc.nxc](#).

8.3.3.150 int fputs (string *str*, byte *handle*) [inline]

Write string to file. Writes the string to the file specified by the handle. The null terminating character at the end of the string is not written to the file. If there are no errors, a non-negative value is returned. If an error occurs, EOF is returned.

Parameters:

str The string of characters to be written.

handle The handle of the file where the string is to be written.

Returns:

The number of characters written to the file.

Examples:

[ex_fputs.nxc](#).

8.3.3.151 float frac (float *x*) [inline]

Compute fractional part. Computes the fractional part of *x*.

Parameters:

x Floating point value.

Returns:

Fractional part of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_frac.nxc](#).

8.3.3.152 unsigned int FreeMemory (void) [inline]

Get free flash memory. Get the number of bytes of flash memory that are available for use.

Returns:

The number of bytes of unused flash memory.

Examples:

[ex_FreeMemory.nxc](#).

8.3.3.153 int fseek (byte *handle*, long *offset*, int *origin*) [inline]

Reposition file position indicator. Sets the position indicator associated with the file to a new position defined by adding offset to a reference position specified by origin.

Parameters:

handle The handle of the file.

offset The number of bytes to offset from origin.

origin Position from where offset is added. It is specified by one of the following constants: SEEK_SET - beginning of file, SEEK_CUR - current position of the file pointer, or SEEK_END - end of file. [fseek origin constants](#)

Returns:

A value of zero if successful or non-zero otherwise. See [Loader module error codes](#).

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_fseek.nxc](#).

8.3.3.154 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](#).

8.3.3.155 void GetBrickDataAddress (byte & data[]) [inline]

Get NXT address. This method reads the address of the NXT and stores it in the data buffer provided.

Parameters:

data The byte array reference that will contain the device address.

Examples:

[ex_GetBrickDataAddress.nxc](#).

8.3.3.156 void GetBTConnectionAddress (const byte *conn*, byte & *data*[]) [inline]

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth connection table and stores it in the data buffer provided.

Parameters:

conn The connection slot (0..3).

data The byte array reference that will contain the device address.

Examples:

[ex_GetBTConnectionAddress.nxc](#).

8.3.3.157 void GetBTDeviceAddress (const byte *devidx*, byte & *data*[]) [inline]

Get bluetooth device address. This method reads the address of the device at the specified index within the Bluetooth device table and stores it in the data buffer provided.

Parameters:

devidx The device table index.

data The byte array reference that will contain the device address.

Examples:

[ex_GetBTDeviceAddress.nxc](#).

8.3.3.158 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_BTInputBuffer.nxc](#).

8.3.3.159 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_BTOutputBuffer.nxc](#).

8.3.3.160 void GetButtonModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Button module IOMap value. Read a value from the Button module IOMap structure. You provide the offset into the Button module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Button module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

8.3.3.161 int getch () [inline]

Get character from stdin. Returns the next character from the standard input (stdin). It is equivalent to getc with stdin as its argument. On the NXT this means wait for a button press and return the value of the button pressed.

Returns:

The pressed button. See [Button name constants](#).

Examples:

[ex_getchar.nxc](#).

8.3.3.162 void GetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Command module IOMap bytes. Read one or more bytes of data from Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the data should be read. See [Command module IOMAP offsets](#).

count The number of bytes to read from the specified Command module IOMap offset.

data A byte array that will contain the data read from the Command module IOMap.

8.3.3.163 void GetCommandModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Command module IOMap value. Read a value from the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Command module IOMAP offsets](#).
value A variable that will contain the value read from the IOMap.

8.3.3.164 void GetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Comm module IOMap bytes. Read one or more bytes of data from Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the data should be read. See [Comm module IOMAP offsets](#).
count The number of bytes to read from the specified Comm module IOMap offset.
data A byte array that will contain the data read from the Comm module IOMap.

8.3.3.165 void GetCommModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Comm module IOMap value. Read a value from the Comm module IOMap structure. You provide the offset into the Comm module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Comm module IOMAP offsets](#).
value A variable that will contain the value read from the IOMap.

8.3.3.166 void GetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Display module IOMap bytes. Read one or more bytes of data from Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the data should be read. See [Display module IOMAP offsets](#).

count The number of bytes to read from the specified Display module IOMap offset.

data A byte array that will contain the data read from the Display module IOMap.

**8.3.3.167 void GetDisplayModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Display module IOMap value. Read a value from the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Display module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

8.3.3.168 void GetDisplayNormal (const byte *x*, const byte *line*, unsigned int *cnt*, byte & *data*[]) [inline]

Read pixel data from the normal display buffer. Read "cnt" bytes from the normal display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE_1 through TEXTLINE_8 for the "line" parameter.

Parameters:

x The desired x position from which to read pixel data.

line The desired line from which to read pixel data.

cnt The number of bytes of pixel data to read.

data The array of bytes into which pixel data is read.

Examples:

[ex_GetDisplayNormal.nxc](#).

8.3.3.169 void GetDisplayPopup (const byte *x*, const byte *line*, unsigned int *cnt*, byte & *data*[]) [inline]

Read pixel data from the popup display buffer. Read "cnt" bytes from the popup display memory into the data array. Start reading from the specified x, line coordinate. Each byte of data read from screen memory is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE_1 through TEXTLINE_8 for the "line" parameter.

Parameters:

x The desired x position from which to read pixel data.

line The desired line from which to read pixel data.

cnt The number of bytes of pixel data to read.

data The array of bytes into which pixel data is read.

Examples:

[ex_GetDisplayPopup.nxc](#).

8.3.3.170 void GetHSInputBuffer (const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get hi-speed port input buffer data. This method reads count bytes of data from the hi-speed port input buffer and writes it to the buffer provided.

Parameters:

offset A constant offset into the hi-speed port input buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the hi-speed port input buffer.

Examples:

[ex_GetHSInputBuffer.nxc](#).

8.3.3.171 void GetHSOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get hi-speed port output buffer data. This method reads count bytes of data from the hi-speed port output buffer and writes it to the buffer provided.

Parameters:

offset A constant offset into the hi-speed port output buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the hi-speed port output buffer.

Examples:

[ex_GetHSOutputBuffer.nxc](#).

8.3.3.172 variant GetInput (const byte & *port*, const byte *field*) [inline]

Get an input field value. Return the value of the specified field of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

field An input field constant. See [Input field constants](#).

Returns:

The input field value.

Examples:

[ex_GetInput.nxc](#).

**8.3.3.173 void GetInputModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Input module IOMap value. Read a value from the Input module IOMap structure. You provide the offset into the Input module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Input module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**8.3.3.174 void GetIOMapBytes (string *moduleName*, unsigned int *offset*,
unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by name. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

moduleName The module name of the IOMap. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be read

count The number of bytes to read from the specified IOMap offset.

data A byte array that will contain the data read from the IOMap

**8.3.3.175 void GetIOMapBytesByID (unsigned long *moduleId*, unsigned int
offset, unsigned int *count*, byte & *data*[]) [inline]**

Get IOMap bytes by ID. Read one or more bytes of data from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

moduleId The module ID of the IOMap. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be read.

count The number of bytes to read from the specified IOMap offset.

data A byte array that will contain the data read from the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

8.3.3.176 void GetIOMapValue (string *moduleName*, unsigned int *offset*, variant & *value*) [inline]

Get IOMap value by name. Read a value from an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

Parameters:

moduleName The module name of the IOMap. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the value should be read

value A variable that will contain the value read from the IOMap

8.3.3.177 void GetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant & *value*) [inline]

Get IOMap value by ID. Read a value from an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to read the value along with a variable that will contain the IOMap value.

Parameters:

moduleId The module ID of the IOMap. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the value should be read.

value A variable that will contain the value read from the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

**8.3.3.178 void GetLoaderModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Loader module IOMap value. Read a value from the Loader module IOMap structure. You provide the offset into the Loader module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Loader module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

8.3.3.179 void GetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte & *data*[]) [inline]

Get Lowspeed module IOMap bytes. Read one or more bytes of data from Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start reading, the number of bytes to read from that location, and a byte array where the data will be stored.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be read. See [Low speed module IOMAP offsets](#).

count The number of bytes to read from the specified Lowspeed module IOMap offset.

data A byte array that will contain the data read from the Lowspeed module IOMap.

**8.3.3.180 void GetLowSpeedModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get LowSpeed module IOMap value. Read a value from the LowSpeed module IOMap structure. You provide the offset into the Command module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Low speed module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**8.3.3.181 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_GLSInputBuffer.nxc](#).

**8.3.3.182 void GetLSSOutputBuffer (const byte *port*, const byte *offset*, byte *cnt*,
byte & *data*[]) [inline]**

Get I2C output buffer data. This method reads cnt bytes of data from the I2C output buffer for the specified port and writes it to the buffer provided.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

offset A constant offset into the I2C output buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the I2C output buffer.

Examples:

[ex_GetLSOutputBuffer.nxc](#).

8.3.3.183 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](#).

8.3.3.184 void GetOutputModuleValue (unsigned int *offset*, variant & *value*) [inline]

Get Output module IOMap value. Read a value from the Output module IOMap structure. You provide the offset into the Output module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Output module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**8.3.3.185 void GetSoundModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Sound module IOMap value. Read a value from the Sound module IOMap structure. You provide the offset into the Sound module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Sound module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**8.3.3.186 void GetUiModuleValue (unsigned int *offset*, variant & *value*)
[inline]**

Get Ui module IOMap value. Read a value from the Ui module IOMap structure. You provide the offset into the Ui module IOMap structure where you want to read the value from along with a variable that will store the value. The type of the variable determines how many bytes are read from the IOMap.

Parameters:

offset The number of bytes offset from the start of the IOMap structure where the value should be read. See [Ui module IOMAP offsets](#).

value A variable that will contain the value read from the IOMap.

**8.3.3.187 void GetUSBInputBuffer (const byte *offset*, byte *cnt*, byte & *data*[])
[inline]**

Get usb input buffer data. This method reads count bytes of data from the usb input buffer and writes it to the buffer provided.

Parameters:

offset A constant offset into the usb input buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the usb input buffer.

Examples:

[ex_GetUSBInputBuffer.nxc](#).

8.3.3.188 void GetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get usb output buffer data. This method reads count bytes of data from the usb output buffer and writes it to the buffer provided.

Parameters:

offset A constant offset into the usb output buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the usb output buffer.

Examples:

[ex_GetUSBOutputBuffer.nxc](#).

8.3.3.189 void GetUSBPollBuffer (const byte *offset*, byte *cnt*, byte & *data*[]) [inline]

Get usb poll buffer data. This method reads count bytes of data from the usb poll buffer and writes it to the buffer provided.

Parameters:

offset A constant offset into the usb poll buffer.

cnt The number of bytes to read.

data The byte array reference which will contain the data read from the usb poll buffer.

Examples:

[ex_GetUSBPollBuffer.nxc](#).

8.3.3.190 void glAddToAngleX (int *glValue*) [inline]

Add to the X axis angle. Add the specified value to the existing X axis angle.

Parameters:

glValue The value to add to the X axis angle.

Examples:

[glBoxDemo.nxc](#), and [glCircleDemo.nxc](#).

8.3.3.191 void glAddToAngleY (int *glValue*) [inline]

Add to the Y axis angle. Add the specified value to the existing Y axis angle.

Parameters:

glValue The value to add to the Y axis angle.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.192 void glAddToAngleZ (int *glValue*) [inline]

Add to the Z axis angle. Add the specified value to the existing Z axis angle.

Parameters:

glValue The value to add to the Z axis angle.

8.3.3.193 void glAddVertex (int *glX*, int *glY*, int *glZ*) [inline]

Add a vertex to an object. Add a vertex to an object currently being defined. This function should only be used between [glBegin](#) and [glEnd](#) which are themselves nested within a [glBeginObject](#) and [glEndObject](#) pair.

Parameters:

glX The X axis coordinate.

glY The Y axis coordinate.

glZ The Z axis coordinate.

8.3.3.194 void glBegin (int *glBeginMode*) [inline]

Begin a new polygon for the current object. Start defining a polygon surface for the current graphics object using the specified begin mode.

Parameters:

glBeginMode The desired mode. See [Graphics library begin modes](#).

8.3.3.195 int glBeginObject () [inline]

Begin defining an object. Start the process of defining a graphics library object using low level functions such as [glBegin](#), [glAddVertex](#), and [glEnd](#).

Returns:

The object index of the new object being created.

8.3.3.196 void glBeginRender () [inline]

Begin a new render. Start the process of rendering the existing graphic objects.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.197 int glBox (int *glMode*, int *glSizeX*, int *glSizeY*, int *glSizeZ*) [inline]

Create a 3D box. Define a 3D box using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with width, height, and depth specified via the `glSizeX`, `glSizeY`, and `glSizeZ` parameters.

Parameters:

- `glMode` The begin mode for each surface. See [Graphics library begin modes](#).
- `glSizeX` The X axis size (width).
- `glSizeY` The Y axis size (height).
- `glSizeZ` The Z axis size (depth).

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.198 void `glCallObject` (int `glObjectId`) [inline]

Call a graphic object. Tell the graphics library that you want it to include the specified object in the render.

Parameters:

- `glObjectId` The desired object id.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.199 int `glCos32768` (int `glAngle`) [inline]

Table-based cosine scaled by 32768. Return the cosine of the specified angle in degrees. The result is scaled by 32768.

Parameters:

- `glAngle` The angle in degrees.

Returns:

The cosine value scaled by 32768.

8.3.3.200 int glCube (int *glMode*, int *glSize*) [inline]

Create a 3D cube. Define a 3D cube using the specified begin mode for all faces. The center of the box is at the origin of the XYZ axis with equal width, height, and depth specified via the *glSize* parameter.

Parameters:

glMode The begin mode for each surface. See [Graphics library begin modes](#).

glSize The cube's width, height, and depth.

Examples:

[glBoxDemo.nxc](#).

8.3.3.201 void glEnd () [inline]

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

8.3.3.202 void glEndObject () [inline]

Stop defining an object. Finish the process of defining a graphics library object. Call this function after you have completed the object definition.

8.3.3.203 void glFinishRender () [inline]

Finish the current render. Rotate the vertex list, clear the screen, and draw the rendered objects to the LCD.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.204 void glInit () [inline]

Initialize graphics library. Setup all the necessary data for the graphics library to function. Call this function before any other graphics library routine.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.205 void glObjectAction (int *glObjectId*, int *glAction*, int *glValue*) [inline]

Perform an object action. Execute the specified action on the specified object.

Parameters:

glObjectId The object id.

glAction The action to perform on the object. See [Graphics library actions](#).

glValue The setting value.

Examples:

[glBoxDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.206 int glPyramid (int *glMode*, int *glSizeX*, int *glSizeY*, int *glSizeZ*) [inline]

Create a 3D pyramid. Define a 3D pyramid using the specified begin mode for all faces. The center of the pyramid is at the origin of the XYZ axis with width, height, and depth specified via the *glSizeX*, *glSizeY*, and *glSizeZ* parameters.

Parameters:

glMode The begin mode for each surface. See [Graphics library begin modes](#).

glSizeX The X axis size (width).

glSizeY The Y axis size (height).

glSizeZ The Z axis size (depth).

8.3.3.207 void glSet (int *glType*, int *glValue*) [inline]

Set graphics library options. Adjust graphic library settings for circle size and cull mode.

Parameters:

glType The setting type. See [Graphics library settings](#).

glValue The setting value. For culling modes see [Graphics library cull mode](#).

Examples:

[glCircleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.208 void glSetAngleX (int *glValue*) [inline]

Set the X axis angle. Set the X axis angle to the specified value.

Parameters:

glValue The new X axis angle.

Examples:

[glBoxDemo.nxc](#), [glCircleDemo.nxc](#), [glRotateDemo.nxc](#), [glScaleDemo.nxc](#), and [glTranslateDemo.nxc](#).

8.3.3.209 void glSetAngleY (int *glValue*) [inline]

Set the Y axis angle. Set the Y axis angle to the specified value.

Parameters:

glValue The new Y axis angle.

8.3.3.210 void glSetAngleZ (int *glValue*) [inline]

Set the Z axis angle. Set the Z axis angle to the specified value.

Parameters:

glValue The new Z axis angle.

8.3.3.211 int glSin32768 (int *glAngle*) [inline]

Table-based sine scaled by 32768. Return the sine of the specified angle in degrees. The result is scaled by 32768.

Parameters:

glAngle The angle in degrees.

Returns:

The sine value scaled by 32768.

8.3.3.212 char GraphicArrayOut (int *x*, int *y*, byte *data*[], unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw a graphic image from byte array. Draw a graphic image byte array on the screen at the specified x and y location. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

See also:

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

Parameters:

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_dispgaout.nxc](#).

**8.3.3.213 char GraphicArrayOutEx (int *x*, int *y*, byte *data*[], byte *vars*[],
unsigned long *options* = DRAW_OPT_NORMAL) [inline]**

Draw a graphic image from byte array with parameters. Draw a graphic image byte array on the screen at the specified *x* and *y* location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to **DRAW_OPT_NORMAL**. Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

See also:

[SysDrawGraphicArray](#), [DrawGraphicArrayType](#)

Parameters:

- x* The *x* value for the position of the graphic image.
- y* The *y* value for the position of the graphic image.
- data* The byte array of the RIC graphic image.
- vars* The byte array of parameters.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_dispgaoutex.nxc](#).

**8.3.3.214 char GraphicOut (int *x*, int *y*, string *filename*, unsigned long *options*
= DRAW_OPT_NORMAL) [inline]**

Draw a graphic image. Draw a graphic image file on the screen at the specified *x* and *y* location. Optionally specify drawing options. If this argument is not specified it defaults to **DRAW_OPT_NORMAL**. Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

See also:

[SysDrawGraphic](#), [DrawGraphicType](#)

Parameters:

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- filename* The filename of the RIC graphic image.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_dispgout.nxc](#), and [ex_GraphicOut.nxc](#).

8.3.3.215 `char GraphicOutEx (int x, int y, string filename, byte vars[], unsigned long options = DRAW_OPT_NORMAL) [inline]`

Draw a graphic image with parameters. Draw a graphic image file on the screen at the specified x and y location using an array of parameters. Optionally specify drawing options. If this argument is not specified it defaults to `DRAW_OPT_NORMAL`. Valid display option constants are listed in the [Drawing option constants](#) group. If the file cannot be found then nothing will be drawn and no errors will be reported.

See also:

[SysDrawGraphic](#), [DrawGraphicType](#)

Parameters:

- x* The x value for the position of the graphic image.
- y* The y value for the position of the graphic image.
- filename* The filename of the RIC graphic image.
- vars* The byte array of parameters.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_dispgoutex.nxc](#), and [ex_GraphicOutEx.nxc](#).

8.3.3.216 byte HSFlags (void) [inline]

Get hi-speed port flags. This method returns the value of the hi-speed port flags.

Returns:

The hi-speed port flags. See [Hi-speed port flags constants](#).

Examples:

[ex_HSFlags.nxc](#).

8.3.3.217 byte HSInputBufferInPtr (void) [inline]

Get hi-speed port input buffer in-pointer. This method returns the value of the input pointer of the hi-speed port input buffer.

Returns:

The hi-speed port input buffer's in-pointer value.

Examples:

[ex_HSInputBufferInPtr.nxc](#).

8.3.3.218 byte HSInputBufferOutPtr (void) [inline]

Get hi-speed port input buffer out-pointer. This method returns the value of the output pointer of the hi-speed port input buffer.

Returns:

The hi-speed port input buffer's out-pointer value.

Examples:

[ex_HSInputBufferOutPtr.nxc](#).

8.3.3.219 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](#).

8.3.3.220 byte HSOutputBufferInPtr (void) [inline]

Get hi-speed port output buffer in-pointer. This method returns the value of the input pointer of the hi-speed port output buffer.

Returns:

The hi-speed port output buffer's in-pointer value.

Examples:

[ex_HSOutputBufferInPtr.nxc](#).

8.3.3.221 byte HSOutputBufferOutPtr (void) [inline]

Get hi-speed port output buffer out-pointer. This method returns the value of the output pointer of the hi-speed port output buffer.

Returns:

The hi-speed port output buffer's out-pointer value.

Examples:

[ex_HSOutputBufferOutPtr.nxc](#).

8.3.3.222 byte HSSpeed (void) [inline]

Get hi-speed port speed. This method returns the value of the hi-speed port speed (baud rate).

Returns:

The hi-speed port speed (baud rate). See [Hi-speed port baud rate constants](#).

Examples:

[ex_HSSpeed.nxc](#).

8.3.3.223 byte HSState (void) [inline]

Get hi-speed port state. This method returns the value of the hi-speed port state.

Returns:

The hi-speed port state. See [Hi-speed port state constants](#).

Examples:

[ex_HSState.nxc](#).

8.3.3.224 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_HTIIRTrain.nxc](#).

8.3.3.225 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 iRLLink device. Commands for outa and outb are [PF_CMD_STOP](#), [PF_CMD_REV](#), [PF_CMD_FWD](#), and [PF_CMD_BRAKE](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

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

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

Returns:

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

Examples:

[ex_HTPFComboDirect.nxc](#).

8.3.3.226 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 iRLLink 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](#).

8.3.3.227 `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 iRLLink 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](#).

8.3.3.228 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](#).

8.3.3.229 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](#).

8.3.3.230 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 iRLLink 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](#).

8.3.3.231 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 iRLLink device. Select the desired output using

`PF_OUT_A` or `PF_OUT_B`. Select the desired pin using `PF_PIN_C1` or `PF_PIN_C2`. Valid functions are `PF_FUNC_NOCHANGE`, `PF_FUNC_CLEAR`, `PF_FUNC_SET`, and `PF_FUNC_TOGGLE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. Specify whether the mode by passing true (continuous) or false (timeout) as the final parameter. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

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

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

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

cont Control whether the mode is continuous or timeout.

Returns:

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

Examples:

[ex_HTPFSinglePin.nxc](#).

8.3.3.232 `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 iRLLink device as if it were an IR Train receiver. Valid function values are `TRAIN_FUNC_STOP`, `TRAIN_FUNC_INCR_SPEED`, `TRAIN_FUNC_DECR_SPEED`, and `TRAIN_FUNC_TOGGLE_LIGHT`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

Parameters:

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

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

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

Returns:

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

Examples:

[ex_HTPFTrain.nxc](#).

**8.3.3.233 void HTRECXAddToDatalog (const byte *src*, const unsigned int *value*)
[inline]**

HTRECXAddToDatalog function. Send the AddToDatalog command to an RCX.

Parameters:

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

value The RCX value.

Examples:

[ex_HTRCXAddToDatalog.nxc](#).

8.3.3.234 int HTRECXBatteryLevel (void) [inline]

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

Returns:

The RCX battery level.

Examples:

[ex_HTRCXBatteryLevel.nxc](#).

8.3.3.235 void HTRECXClearAllEvents (void) [inline]

HTRECXClearAllEvents function. Send the ClearAllEvents command to an RCX.

Examples:

[ex_HTRCXClearAllEvents.nxc](#).

8.3.3.236 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](#).

8.3.3.237 void HTRCXClearMsg (void) [inline]

HTRCXClearMsg function. Send the ClearMsg command to an RCX.

Examples:

[ex-HTRCXClearMsg.nxc](#).

8.3.3.238 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](#).

8.3.3.239 void HTRCXClearSound (void) [inline]

HTRCXClearSound function. Send the ClearSound command to an RCX.

Examples:

[ex-HTRCXClearSound.nxc](#).

8.3.3.240 void HTRECXClearTimer (const byte *timer*) [inline]

HTRECXClearTimer function. Send the ClearTimer command to an RCX.

Parameters:

timer The timer to clear.

Examples:

[ex_HTRCXClearTimer.nxc](#).

8.3.3.241 void HTRECXCreateDatalog (const unsigned int *size*) [inline]

HTRECXCreateDatalog function. Send the CreateDatalog command to an RCX.

Parameters:

size The new datalog size.

Examples:

[ex_HTRCXCreateDatalog.nxc](#).

8.3.3.242 void HTRECXDecCounter (const byte *counter*) [inline]

HTRECXDecCounter function. Send the DecCounter command to an RCX.

Parameters:

counter The counter to decrement.

Examples:

[ex_HTRCXDecCounter.nxc](#).

8.3.3.243 void HTRECXDeleteSub (const byte *s*) [inline]

HTRECXDeleteSub function. Send the DeleteSub command to an RCX.

Parameters:

s The subroutine number to delete.

Examples:

[ex_HTRCXDeleteSub.nxc](#).

8.3.3.244 void HTRCXDeleteSubs (void) [inline]

HTRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

Examples:

[ex_HTRCXDeleteSubs.nxc](#).

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

HTRCXDeleteTask function. Send the DeleteTask command to an RCX.

Parameters:

t The task number to delete.

Examples:

[ex_HTRCXDeleteTask.nxc](#).

8.3.3.246 void HTRCXDeleteTasks (void) [inline]

HTRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

Examples:

[ex_HTRCXDeleteTasks.nxc](#).

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

HTRCXDisableOutput function. Send the DisableOutput command to an RCX.

Parameters:

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

Examples:

[ex-HTRCXDisableOutput.nxc](#).

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

HTRCXEnableOutput function. Send the EnableOutput command to an RCX.

Parameters:

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

Examples:

[ex-HTRCXEnableOutput.nxc](#).

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

HTRCXEvent function. Send the Event command to an RCX.

Parameters:

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

value The RCX value.

Examples:

[ex-HTRCXEvent.nxc](#).

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

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

Parameters:

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

Examples:

[ex-HTRCXFloat.nxc](#).

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

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

Parameters:

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

Examples:

[ex-HTRCXFwd.nxc](#).

8.3.3.252 void HTRCXIIncCounter (const byte *counter*) [inline]

HTRCXIIncCounter function. Send the IncCounter command to an RCX.

Parameters:

counter The counter to increment.

Examples:

[ex-HTRCXIIncCounter.nxc](#).

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

HTRCXInvertOutput function. Send the InvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXInvertOutput.nxc](#).

8.3.3.254 void HTRCXMuteSound (void) [inline]

HTRCXMuteSound function. Send the MuteSound command to an RCX.

Examples:

[ex_HTRCXMuteSound.nxc](#).

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

HTRCXObvertOutput function. Send the ObvertOutput command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXObvertOutput.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOff.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOn.nxc](#).

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

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

Parameters:

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

ms The number of milliseconds to leave the outputs on

Examples:

[ex_HTRCXOnFor.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOnFwd.nxc](#).

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

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

Parameters:

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

Examples:

[ex_HTRCXOnRev.nxc](#).

8.3.3.261 void HTRCXPBTurnOff (void) [inline]

HTRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

Examples:

[ex_HTRCXPBTurnOff.nxc](#).

8.3.3.262 void HTRCXPing (void) [inline]

HTRCXPing function. Send the Ping command to an RCX.

Examples:

[ex_HTRCXPing.nxc](#).

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

HTRCXPlaySound function. Send the PlaySound command to an RCX.

Parameters:

snd The sound number to play.

Examples:

[ex_HTRCXPlaySound.nxc](#).

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

HTRCXPlayTone function. Send the PlayTone command to an RCX.

Parameters:

freq The frequency of the tone to play.

duration The duration of the tone to play.

Examples:

[ex_HTRCXPlayTone.nxc](#).

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

HTRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

Parameters:

varnum The variable containing the tone frequency to play.

duration The duration of the tone to play.

Examples:

[ex_HTRCXPlayToneVar.nxc](#).

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

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

Parameters:

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

value The RCX value.

Returns:

The value read from the specified port and value.

Examples:

[ex_HTRCXPoll.nxc](#).

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

HTRCXPollMemory function. Send the PollMemory command to an RCX.

Parameters:

address The RCX memory address.

Returns:

The value read from the specified address.

Examples:

[ex_HTRCXPollMemory.nxc](#).

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

HTRCXRemote function. Send the Remote command to an RCX.

Parameters:

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

Examples:

[ex_HTRCXRemote.nxc](#).

8.3.3.269 void HTRECXRev (const byte *outputs*) [inline]

HTRECXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

Parameters:

outputs The RCX output(s) to reverse direction. See [RCX output constants](#).

Examples:

[ex_HTRCXRev.nxc](#).

8.3.3.270 void HTRECXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]

HTRECXSelectDisplay function. Send the SelectDisplay command to an RCX.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_HTRCXSelectDisplay.nxc](#).

8.3.3.271 void HTRECXSelectProgram (const byte *prog*) [inline]

HTRECXSelectProgram function. Send the SelectProgram command to an RCX.

Parameters:

prog The program number to select.

Examples:

[ex_HTRCXSelectProgram.nxc](#).

**8.3.3.272 void HTRCXSendSerial (const byte *first*, const byte *count*)
[inline]**

HTRCXSendSerial function. Send the SendSerial command to an RCX.

Parameters:

first The first byte address.
count The number of bytes to send.

Examples:

[ex_HTRCXSendSerial.nxc](#).

**8.3.3.273 void HTRCXSetDirection (const byte *outputs*, const byte *dir*)
[inline]**

HTRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

Parameters:

outputs The RCX output(s) to set direction. See [RCX output constants](#).
dir The RCX output direction. See [RCX output direction constants](#).

Examples:

[ex_HTRCXSetDirection.nxc](#).

**8.3.3.274 void HTRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)
[inline]**

HTRCXSetEvent function. Send the SetEvent command to an RCX.

Parameters:

evt The event number to set.
src The RCX source. See [RCX and Scout source constants](#).
type The event type.

Examples:

[ex_HTRCXSetEvent.nxc](#).

**8.3.3.275 void HTRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)
[inline]**

HTRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

Parameters:

outputs The RCX output(s) to set global direction. See [RCX output constants](#).

dir The RCX output direction. See [RCX output direction constants](#).

Examples:

[ex_HTRCXSetGlobalDirection.nxc](#).

**8.3.3.276 void HTRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)
[inline]**

HTRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

Parameters:

outputs The RCX output(s) to set global mode. See [RCX output constants](#).

mode The RCX output mode. See [RCX output mode constants](#).

Examples:

[ex_HTRCXSetGlobalOutput.nxc](#).

8.3.3.277 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 iRLLink device. The port must be configured as a Lowspeed port before using any of the HiTechnic RCX and Scout iRLLink functions.

Parameters:

port The sensor port. See [Input port constants](#).

8.3.3.278 void HTRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrvval*) [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](#).

pwrvval The RCX value.

Examples:

[ex-HTRCXSetMaxPower.nxc](#).

8.3.3.279 void HTRCXSetMessage (const byte *msg*) [inline]

HTRCXSetMessage function. Send the SetMessage command to an RCX.

Parameters:

msg The numeric message to send.

Examples:

[ex-HTRCXSetMessage.nxc](#).

8.3.3.280 void HTRCXSetOutput (const byte *outputs*, const byte *mode*) [inline]

HTRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

Parameters:

outputs The RCX output(s) to set mode. See [RCX output constants](#).

mode The RCX output mode. See [RCX output mode constants](#).

Examples:

[ex_HTRCXSetOutput.nxc](#).

8.3.3.281 void HTRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]

HTRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

Parameters:

outputs The RCX output(s) to set power. See [RCX output constants](#).

pwrsrc The RCX source. See [RCX and Scout source constants](#).

pwrval The RCX value.

Examples:

[ex_HTRCXSetPower.nxc](#).

8.3.3.282 void HTRCXSetPriority (const byte *p*) [inline]

HTRCXSetPriority function. Send the SetPriority command to an RCX.

Parameters:

p The new task priority.

Examples:

[ex_HTRCXSetPriority.nxc](#).

8.3.3.283 void HTRCXSetSensorMode (const byte *port*, const byte *mode*) [inline]

HTRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

Parameters:

port The RCX sensor port.
mode The RCX sensor mode.

Examples:

[ex_HTRCXSetSensorMode.nxc](#).

8.3.3.284 void HTRCXSetSensorType (const byte *port*, const byte *type*) [inline]

HTRCXSetSensorType function. Send the SetSensorType command to an RCX.

Parameters:

port The RCX sensor port.
type The RCX sensor type.

Examples:

[ex_HTRCXSetSensorType.nxc](#).

8.3.3.285 void HTRCXSetSleepTime (const byte *t*) [inline]

HTRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

Parameters:

t The new sleep time value.

Examples:

[ex_HTRCXSetSleepTime.nxc](#).

8.3.3.286 void HTRCXSetTxPower (const byte *pwr*) [inline]

HTRCXSetTxPower function. Send the SetTxPower command to an RCX.

Parameters:

pwr The IR transmit power level.

Examples:

[ex_HTRCXSetTxPower.nxc](#).

**8.3.3.287 void HTRCXSetWatch (const byte *hours*, const byte *minutes*)
[inline]**

HTRCXSetWatch function. Send the SetWatch command to an RCX.

Parameters:

hours The new watch time hours value.

minutes The new watch time minutes value.

Examples:

[ex_HTRCXSetWatch.nxc](#).

8.3.3.288 void HTRCXStartTask (const byte *t*) [inline]

HTRCXStartTask function. Send the StartTask command to an RCX.

Parameters:

t The task number to start.

Examples:

[ex_HTRCXStartTask.nxc](#).

8.3.3.289 void HTRCXStopAllTasks (void) [inline]

HTRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

Examples:

[ex_HTRCXStopAllTasks.nxc](#).

8.3.3.290 void HTRCXStopTask (const byte *t*) [inline]

HTRCXStopTask function. Send the StopTask command to an RCX.

Parameters:

t The task number to stop.

Examples:

[ex_HTRCXStopTask.nxc](#).

8.3.3.291 void HTRCXToggle (const byte *outputs*) [inline]

HTRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

Parameters:

outputs The RCX output(s) to toggle. See [RCX output constants](#).

Examples:

[ex_HTRCXToggle.nxc](#).

8.3.3.292 void HTRCXUnmuteSound (void) [inline]

HTRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

Examples:

[ex_HTRCXUnmuteSound.nxc](#).

8.3.3.293 void HTScoutCalibrateSensor (void) [inline]

HTScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

Examples:

[ex_HTScoutCalibrateSensor.nxc](#).

8.3.3.294 void HTScoutMuteSound (void) [inline]

HTScoutMuteSound function. Send the MuteSound command to a Scout.

Examples:

[ex_HTScoutMuteSound.nxc](#).

8.3.3.295 void HTScoutSelectSounds (const byte *grp*) [inline]

HTScoutSelectSounds function. Send the SelectSounds command to a Scout.

Parameters:

grp The Scout sound group to select.

Examples:

[ex_HTScoutSelectSounds.nxc](#).

8.3.3.296 void HTScoutSendVLL (const byte *src*, const unsigned int *value*) [inline]

HTScoutSendVLL function. Send the SendVLL command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_HTScoutSendVLL.nxc](#).

8.3.3.297 void HTScoutSetEventFeedback (const byte *src*, const unsigned int *value*) [inline]

HTScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_HTScoutSetEventFeedback.nxc](#).

8.3.3.298 void HTScoutSetLight (const byte *x*) [inline]

HTScoutSetLight function. Send the SetLight command to a Scout.

Parameters:

x Set the light on or off using this value. See [Scout light constants](#).

Examples:

[ex_HTScoutSetLight.nxc](#).

8.3.3.299 void HTScoutSetScoutMode (const byte *mode*) [inline]

HTScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

Parameters:

mode Set the scout mode. See [Scout mode constants](#).

Examples:

[ex_HTScoutSetScoutMode.nxc](#).

8.3.3.300 void HTScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [inline]

HTScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_HTScoutSetSensorClickTime.nxc](#).

8.3.3.301 void HTScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [inline]

HTScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_HTScoutSetSensorHysteresis.nxc](#).

8.3.3.302 void HTScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]

HTScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_HTScoutSetSensorLowerLimit.nxc](#).

8.3.3.303 void HTScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]

HTScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).
value The Scout value.

Examples:

[ex_HTScoutSetSensorUpperLimit.nxc](#).

8.3.3.304 void HTScoutUnmuteSound (void) [inline]

HTScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

Examples:

[ex_HTScoutUnmuteSound.nxc](#).

8.3.3.305 long I2CBytes (const byte *port*, byte *inbuf*[], byte & *count*, byte & *outbuf*[]) [inline]

Perform an I2C write/read transaction. This method writes the bytes contained in the input buffer (inbuf) to the I2C device on the specified port, checks for the specified number of bytes to be ready for reading, and then tries to read the specified number (count) of bytes from the I2C device into the output buffer (outbuf).

This is a higher-level wrapper around the three main I2C functions. It also maintains a "last good read" buffer and returns values from that buffer if the I2C communication transaction fails.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

inbuf A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

count The number of bytes that should be returned by the I2C device. On output count is set to the number of bytes in outbuf.

outbuf A byte array that contains the data read from the internal I2C buffer.

Returns:

Returns true or false indicating whether the I2C transaction succeeded or failed.

See also:

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [I2CRead](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_I2CBytes.nxc](#).

8.3.3.306 byte I2CBytesReady (const byte *port*) [inline]

Get I2C bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [LowspeedBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

Examples:[ex_I2CBytesReady.nxc.](#)**8.3.3.307 long I2CCheckStatus (const byte *port*) [inline]**

Check I2C status. This method checks the status of the I2C communication on the specified port.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while this function returns [STAT_COMM_PENDING](#).

See also:

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [LowspeedStatus](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

Examples:[ex_I2CCheckStatus.nxc.](#)**8.3.3.308 string I2CDeviceId (byte *port*, byte *i2caddr*) [inline]**

Read I2C device identifier. Read standard I2C device identifier. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device identifier.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

8.3.3.309 string I2CDeviceInfo (byte *port*, byte *i2caddr*, byte *info*) [inline]

Read I2C device information. Read standard I2C device information: version, vendor, and device ID. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

info A value indicating the type of device information you are requesting. See [Standard I2C constants](#).

Returns:

A string containing the requested device information.

Examples:

[ex_I2CDeviceInfo.nxc](#).

8.3.3.310 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 [CommLSSReadType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_I2CRead.nxc](#).

8.3.3.311 long I2CSendCommand (byte *port*, byte *i2caddr*, byte *cmd*) [inline]

Send an I2C command. Send a command to an I2C device at the standard command register: [I2C_REG_CMD](#). The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

cmd The command to send to the I2C device.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

Examples:

[ex_I2CSendCommand.nxc](#).

8.3.3.312 long I2CStatus (const byte *port*, byte & *bytesready*) [inline]

Get I2C status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful I2CWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

bytesready The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible return values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [I2CRead](#) or [I2CWrite](#) while I2CStatus returns [STAT_COMM_PENDING](#).

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [LowspeedStatus](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

Examples:

[ex_I2CStatus.nxc](#).

8.3.3.313 string I2CVendorId (byte *port*, byte *i2caddr*) [inline]

Read I2C device vendor. Read standard I2C device vendor. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device vendor.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

8.3.3.314 string I2CVersion (byte *port*, byte *i2caddr*) [inline]

Read I2C device version. Read standard I2C device version. The I2C device uses the specified address.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

i2caddr The I2C device address.

Returns:

A string containing the device version.

Examples:

[ex_I2CDeviceId.nxc](#), [ex_I2CVendorId.nxc](#), and [ex_I2CVersion.nxc](#).

8.3.3.315 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 [CommLSSWriteType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_i2cwrite.nxc](#).

8.3.3.316 int isalnum (int c) [inline]

Check if character is alphanumeric. Checks if parameter c is either a decimal digit or an uppercase or lowercase letter. The result is true if either isalpha or isdigit would also return true.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is either a digit or a letter, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isalnum.nxc](#).

8.3.3.317 int isalpha (int *c*) [inline]

Check if character is alphabetic. Checks if parameter c is either an uppercase or lowercase letter.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is an alphabetic letter, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isalpha.nxc](#).

8.3.3.318 int iscntrl (int *c*) [inline]

Check if character is a control character. Checks if parameter c is a control character.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is a control character, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_iscntrl.nxc](#).

8.3.3.319 int isdigit (int *c*) [inline]

Check if character is decimal digit. Checks if parameter c is a decimal digit character.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is a decimal digit, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isdigit.nxc](#).

8.3.3.320 int isgraph (int c) [inline]

Check if character has graphical representation. Checks if parameter c is a character with a graphical representation.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c has a graphical representation, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isgraph.nxc](#).

8.3.3.321 int islower (int c) [inline]

Check if character is lowercase letter. Checks if parameter c is an lowercase alphabetic letter.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is an lowercase alphabetic letter, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_islower.nxc](#).

8.3.3.322 bool isNaN (float *value*) [inline]

Is the value NaN. Returns true if the floating point value is NaN (not a number).

Parameters:

value A floating point variable.

Returns:

Whether the value is NaN.

Examples:

[ex_isNAN.nxc](#), and [ex_labs.nxc](#).

8.3.3.323 int isprint (int *c*) [inline]

Check if character is printable. Checks if parameter c is a printable character (i.e., not a control character).

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is a printable character, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isprint.nxc](#).

8.3.3.324 int ispunct (int *c*) [inline]

Check if character is a punctuation. Checks if parameter c is a punctuation character.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is a punctuation character, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_ispunct.nxc](#).

8.3.3.325 int isspace (int c) [inline]

Check if character is a white-space. Checks if parameter c is a white-space character.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is a white-space character, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isspace.nxc](#).

8.3.3.326 int isupper (int c) [inline]

Check if character is uppercase letter. Checks if parameter c is an uppercase alphabetic letter.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if c is an uppercase alphabetic letter, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isupper.nxc](#).

8.3.3.327 int isxdigit (int *c*) [inline]

Check if character is hexadecimal digit. Checks if parameter *c* is a hexadecimal digit character.

Parameters:

c Character to be checked.

Returns:

Returns a non-zero value (true) if *c* is a hexadecimal digit character, otherwise it returns 0 (false).

Examples:

[ex_ctype.nxc](#), and [ex_isxdigit.nxc](#).

8.3.3.328 long labs (long *n*) [inline]

Absolute value. Return the absolute value of parameter *n*.

Parameters:

n Integral value.

Returns:

The absolute value of *n*.

8.3.3.329 ldiv_t ldiv (long *numer*, long *denom*) [inline]

Integral division. Returns the integral quotient and remainder of the division of numerator by denominator as a structure of type [ldiv_t](#), which has two members: *quot* and *rem*.

Parameters:

numer Numerator.

denom Denominator.

Returns:

The result is returned by value in a structure defined in cstdlib, which has two members. For [ldiv_t](#), these are, in either order: long quot; long rem.

Examples:

[ex_ldiv.nxc](#).

8.3.3.330 string LeftStr (string str, unsigned int size) [inline]

Copy a portion from the start of a string. Returns the substring of a specified length that appears at the start of a string.

Parameters:

str A string

size The size or length of the substring.

Returns:

The substring of a specified length that appears at the start of a string.

Examples:

[ex_leftstr.nxc](#).

8.3.3.331 char LineOut (int x1, int y1, int x2, int y2, unsigned long options = DRAW_OPT_NORMAL) [inline]

Draw a line. This function lets you draw a line on the screen from x1, y1 to x2, y2. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawLine](#), [DrawLineType](#)

Parameters:

x1 The x value for the start of the line.

y1 The y value for the start of the line.
x2 The x value for the end of the line.
y2 The y value for the end of the line.
options The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_LineOut.nxc](#).

8.3.3.332 float log (float x) [inline]

Compute natural logarithm. Computes the natural logarithm of x. The natural logarithm is the base-e logarithm, the inverse of the natural exponential function (exp). For base-10 logarithms, a specific function [log10\(\)](#) exists.

See also:

[log10\(\)](#), [exp\(\)](#)

Parameters:

x Floating point value.

Returns:

Natural logarithm of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_log.nxc](#).

8.3.3.333 float log10 (float x) [inline]

Compute common logarithm. Computes the common logarithm of x. The common logarithm is the base-10 logarithm. For base-e logarithms, a specific function [log\(\)](#) exists.

See also:

[log\(\)](#), [exp\(\)](#)

Parameters:

x Floating point value.

Returns:

Common logarithm of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_log10.nxc](#).

8.3.3.334 byte LongAbort (void) [inline]

Read long abort setting. Return the enhanced NBC/NXC firmware's long abort setting.

See also:

[AbortFlag](#)

Returns:

The current abort flag value. See [ButtonState constants](#).

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_LongAbort.nxc](#).

8.3.3.335 byte LowspeedBytesReady (const byte port) [inline]

Get lowspeed bytes ready. This method checks the number of bytes that are ready to be read on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then the return value will be the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedBytesReady.nxc](#).

8.3.3.336 long LowspeedCheckStatus (const byte *port*) [inline]

Check lowspeed status. This method checks the status of the I2C communication on the specified port.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while [LowspeedCheckStatus](#) returns [STAT_COMM_PENDING](#).

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedCheckStatus.nxc](#).

8.3.3.337 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](#), [I2CWwrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedWrite](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedRead.nxc](#).

8.3.3.338 long LowspeedStatus (const byte *port*, byte & *bytesready*) [inline]

Get lowspeed status. This method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful LowspeedWrite call that requested response data from the device then bytesready will be set to the number of bytes in the internal read buffer.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

bytesready The number of bytes available to be read from the internal I2C buffer. The maximum number of bytes that can be read is 16.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors. Avoid calls to [LowspeedRead](#) or [LowspeedWrite](#) while LowspeedStatus returns [STAT_COMM_PENDING](#).

See also:

[I2CStatus](#), [I2CRead](#), [I2CWrite](#), [I2CCheckStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedWrite](#), and [LowspeedCheckStatus](#)

Examples:

[ex_lowspeedstatus.nxc](#).

8.3.3.339 long LowspeedWrite (const byte *port*, byte *retlen*, byte *buffer*[]) [inline]

Write lowspeed data. This method starts a transaction to write the bytes contained in the array *buffer* to the I2C device on the specified port. It also tells the I2C device the number of bytes that should be included in the response. The maximum number of bytes that can be written or read is 16.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable. Constants should be used where possible to avoid blocking access to I2C devices on other ports by code running on other threads.

retlen The number of bytes that should be returned by the I2C device.

buffer A byte array containing the address of the I2C device, the I2C device register at which to write data, and up to 14 bytes of data to be written at the specified register.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSWriteType](#) for possible result values. If the return value is [NO_ERR](#) then the last operation did not cause any errors.

See also:

[I2CCheckStatus](#), [I2CRead](#), [I2CWrite](#), [I2CStatus](#), [I2CBytesReady](#), [LowspeedRead](#), [LowspeedCheckStatus](#), [LowspeedBytesReady](#), and [LowspeedStatus](#)

Examples:

[ex_LowspeedWrite.nxc](#).

8.3.3.340 byte LSChannelState (const byte *port*) [inline]

Get I2C channel state. This method returns the value of the I2C channel state for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C port channel state. See [LSChannelState constants](#).

Examples:

[ex_LSChannelState.nxc](#).

8.3.3.341 byte LSErrorType (const byte *port*) [inline]

Get I2C error type. This method returns the value of the I2C error type for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C port error type. See [LSErrorType constants](#).

Examples:[ex_LSErrorType.nxc.](#)**8.3.3.342 byte LSInputBufferBytesToRx (const byte *port*) [inline]**

Get I2C input buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C input buffer for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C input buffer's bytes to rx value.

Examples:[ex_LSInputBufferBytesToRx.nxc.](#)**8.3.3.343 byte LSInputBufferInPtr (const byte *port*) [inline]**

Get I2C input buffer in-pointer. This method returns the value of the input pointer of the I2C input buffer for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C input buffer's in-pointer value.

Examples:[ex_LSInputBufferInPtr.nxc.](#)**8.3.3.344 byte LSInputBufferOutPtr (const byte *port*) [inline]**

Get I2C input buffer out-pointer. This method returns the value of the output pointer of the I2C input buffer for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C input buffer's out-pointer value.

Examples:

[ex_LSInputBufferOutPtr.nxc](#).

8.3.3.345 byte LSMode (const byte *port*) [inline]

Get I2C mode. This method returns the value of the I2C mode for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C port mode. See [LSMode constants](#).

Examples:

[ex_LSMODE.nxc](#).

8.3.3.346 byte LSNoRestartOnRead () [inline]

Get I2C no restart on read setting. This method returns the value of the I2C no restart on read field.

Returns:

The I2C no restart on read field. See [LSNoRestartOnRead constants](#).

Examples:

[ex_LSNORESTARTONREAD.nxc](#).

8.3.3.347 byte LSOutputBufferBytesToRx (const byte *port*) [inline]

Get I2C output buffer bytes to rx. This method returns the value of the bytes to rx field of the I2C output buffer for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C output buffer's bytes to rx value.

Examples:

[ex_LSOutputBufferBytesToRx.nxc](#).

8.3.3.348 byte LSOutputBufferInPtr (const byte *port*) [inline]

Get I2C output buffer in-pointer. This method returns the value of the input pointer of the I2C output buffer for the specified port.

Parameters:

port A constant port number (S1..S4). See [Input port constants](#).

Returns:

The I2C output buffer's in-pointer value.

Examples:

[ex_LSOutputBufferInPtr.nxc](#).

8.3.3.349 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_LSOOutputBufferOutPtr.nxc](#).

8.3.3.350 byte LSSpeed () [inline]

Get I2C speed. This method returns the value of the I2C speed.

Returns:

The I2C speed.

Warning:

This function is unimplemented within the firmware.

Examples:

[ex_LSSpeed.nxc](#).

8.3.3.351 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](#).

8.3.3.352 char memcmp (variant *ptr1*, variant *ptr2*, byte *num*) [inline]

Compare two blocks of memory. Compares the variant *ptr1* to the variant *ptr2*. Returns an integral value indicating the relationship between the variables. The *num* argument is ignored.

Parameters:

ptr1 A variable to be compared.
ptr2 A variable to be compared.
num The number of bytes to compare (ignored).

Examples:

[ex_memcmp.nxc](#).

8.3.3.353 void memcpy (variant *dest*, variant *src*, byte *num*) [inline]

Copy memory. Copies memory contents from the source to the destination. The num argument is ignored.

Parameters:

dest The destination variable.
src The source variable.
num The number of bytes to copy (ignored).

Examples:

[ex_memcpy.nxc](#).

8.3.3.354 void memmove (variant *dest*, variant *src*, byte *num*) [inline]

Move memory. Moves memory contents from the source to the destination. The num argument is ignored.

Parameters:

dest The destination variable.
src The source variable.
num The number of bytes to copy (ignored).

Examples:

[ex_memmove.nxc](#).

**8.3.3.355 string MidStr (string *str*, unsigned int *idx*, unsigned int *len*)
[inline]**

Copy a portion from the middle of a string. Returns the substring of a specified length that appears at a specified position in a string.

Parameters:

str A string
idx The starting index of the substring.
len The length of the substring.

Returns:

The substring of a specified length that appears at a specified position in a string.

Examples:

[ex_midstr.nxc](#).

8.3.3.356 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](#).

8.3.3.357 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](#).

8.3.3.358 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](#).

8.3.3.359 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](#).

8.3.3.360 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](#).

8.3.3.361 byte MotorPwnFreq () [inline]

Get motor PWN frequency. Get the current motor pulse width modulation frequency.

Returns:

The motor pwn frequency

Examples:

[ex_motorpwnfreq.nxc](#).

8.3.3.362 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](#).

8.3.3.363 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](#).

8.3.3.364 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](#).

8.3.3.365 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](#).

8.3.3.366 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](#).

8.3.3.367 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](#).

8.3.3.368 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](#).

8.3.3.369 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](#).

8.3.3.370 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](#).

8.3.3.371 char MSADPAOff (const byte *port*, const byte *i2caddr*) [inline]

Turn off mindsensors ADPA mode. Turn ADPA mode off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSADPAOff.nxc](#).

8.3.3.372 char MSADPAOn (const byte *port*, const byte *i2caddr*) [inline]

Turn on mindsensors ADPA mode. Turn ADPA mode on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSADPAOn.nxc](#).

8.3.3.373 char MSDeenergize (const byte *port*, const byte *i2caddr*) [inline]

Turn off power to device. Turn power off for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSDeenergize.nxc](#).

8.3.3.374 char MSEnergize (const byte *port*, const byte *i2caddr*) [inline]

Turn on power to device. Turn the power on for the mindsensors device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_MSEnergize.nxc](#).

8.3.3.375 char MSIRTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [inline]

MSIRTrain function. Control an IR Train receiver set to the specified channel using the mindsensors NRLink device. Valid function values are [TRAIN_FUNC_STOP](#), [TRAIN_FUNC_INCR_SPEED](#), [TRAIN_FUNC_DECR_SPEED](#), and [TRAIN_FUNC_TOGGLE_LIGHT](#). Valid channels are [TRAIN_CHANNEL_1](#) through [TRAIN_CHANNEL_3](#) and [TRAIN_CHANNEL_ALL](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The IR Train channel. See [IR Train channel constants](#).

func The IR Train function. See [PF/IR Train function constants](#)

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSIRTrain.nxc](#).

8.3.3.376 char MSPFComboDirect (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

MSPFComboDirect function. Execute a pair of Power Function motor commands on the specified channel using the mindsensors NRLink device. Commands for outa and outb are [PF_CMD_STOP](#), [PF_CMD_REV](#), [PF_CMD_FWD](#), and [PF_CMD_BRAKE](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

outa The Power Function command for output A. See [Power Function command constants](#).

outb The Power Function command for output B. See [Power Function command constants](#).

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSPFComboDirect.nxc](#).

8.3.3.377 char MSPFComboPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *outa*, const byte *outb*) [inline]

MSPFComboPWM function. Control the speed of both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Valid output values are [PF_PWM_FLOAT](#), [PF_PWM_FWD1](#), [PF_PWM_FWD2](#), [PF_PWM_FWD3](#), [PF_PWM_FWD4](#), [PF_PWM_FWD5](#), [PF_PWM_FWD6](#), [PF_PWM_FWD7](#), [PF_PWM_BRAKE](#), [PF_PWM_REV7](#), [PF_PWM_REV6](#), [PF_PWM_REV5](#), [PF_PWM_REV4](#), [PF_PWM_REV3](#), [PF_PWM_REV2](#), and [PF_PWM_REV1](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

outa The Power Function PWM command for output A. See [Power Function PWM option constants](#).

outb The Power Function PWM command for output B. See [Power Function PWM option constants](#).

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSPFComboPWM.nxc](#).

8.3.3.378 char MSPFRawOutput (const byte *port*, const byte *i2caddr*, const byte *nibble0*, const byte *nibble1*, const byte *nibble2*) [inline]

MSPFRawOutput function. Control a Power Function receiver set to the specified channel using the mindsensors NRLink device. Build the raw data stream using the 3 nibbles (4 bit values). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

nibble0 The first raw data nibble.

nibble1 The second raw data nibble.

nibble2 The third raw data nibble.

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSPFRawOutput.nxc](#).

8.3.3.379 char MSPFRepeat (const byte *port*, const byte *i2caddr*, const byte *count*, const unsigned int *delay*) [inline]

MSPFRepeat function. Repeat sending the last Power Function command using the mindsensors NRLink device. Specify the number of times to repeat the command and the number of milliseconds of delay between each repetition. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

count The number of times to repeat the command.

delay The number of milliseconds to delay between each repetition.

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:[ex_MSPFRepeat.nxc](#).**8.3.3.380 char MSPFSingleOutputCST (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

MSPFSingleOutputCST function. Control a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Valid functions are [PF_CST_CLEAR1_CLEAR2](#), [PF_CST_SET1_CLEAR2](#), [PF_CST_CLEAR1_SET2](#), [PF_CST_SET1_SET2](#), [PF_CST_INCREMENT_PWM](#), [PF_CST_DECREMENT_PWM](#), [PF_CST_FULL_FWD](#), [PF_CST_FULL_REV](#), and [PF_CST_TOGGLE_DIR](#). Valid channels are [PF_CHANNEL_1](#) through [PF_CHANNEL_4](#). The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

out The Power Function output. See [Power Function output constants](#).

func The Power Function CST function. See [Power Function CST options constants](#).

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:[ex_MSPFSingleOutputCST.nxc](#).**8.3.3.381 char MSPFSingleOutputPWM (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *out*, const byte *func*) [inline]**

MSPFSingleOutputPWM function. Control the speed of a single output on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using [PF_OUT_A](#) or [PF_OUT_B](#). Valid functions

are `PF_PWM_FLOAT`, `PF_PWM_FWD1`, `PF_PWM_FWD2`, `PF_PWM_FWD3`, `PF_PWM_FWD4`, `PF_PWM_FWD5`, `PF_PWM_FWD6`, `PF_PWM_FWD7`, `PF_PWM_BRAKE`, `PF_PWM_REV7`, `PF_PWM_REV6`, `PF_PWM_REV5`, `PF_PWM_REV4`, `PF_PWM_REV3`, `PF_PWM_REV2`, and `PF_PWM_REV1`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

out The Power Function output. See [Power Function output constants](#).

func The Power Function PWM function. See [Power Function PWM option constants](#).

Returns:

The function call result. `NO_ERR` or [Communications specific errors](#).

Examples:

[ex_MSPFSingleOutputPWM.nxc](#).

8.3.3.382 `char MSPFSinglePin (const byte port, const byte i2caddr, const byte channel, const byte out, const byte pin, const byte func, bool cont)` [inline]

MSPFSinglePin function. Control a single pin on a Power Function receiver set to the specified channel using the mindsensors NRLink device. Select the desired output using `PF_OUT_A` or `PF_OUT_B`. Select the desired pin using `PF_PIN_C1` or `PF_PIN_C2`. Valid functions are `PF_FUNC_NOCHANGE`, `PF_FUNC_CLEAR`, `PF_FUNC_SET`, and `PF_FUNC_TOGGLE`. Valid channels are `PF_CHANNEL_1` through `PF_CHANNEL_4`. Specify whether the mode by passing true (continuous) or false (time-out) as the final parameter. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

out The Power Function output. See [Power Function output constants](#).

pin The Power Function pin. See [Power Function pin constants](#).

func The Power Function single pin function. See [Power Function single pin function constants](#).

cont Control whether the mode is continuous or timeout.

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSPFSinglePin.nxc](#).

8.3.3.383 char MSPFTrain (const byte *port*, const byte *i2caddr*, const byte *channel*, const byte *func*) [[inline](#)]

MSPFTrain function. Control both outputs on a Power Function receiver set to the specified channel using the mindsensors NRLink device as if it were an IR Train receiver. Valid function values are [TRAIN_FUNC_STOP](#), [TRAIN_FUNC_INCR_SPEED](#), [TRAIN_FUNC_DECR_SPEED](#), and [TRAIN_FUNC_TOGGLE_LIGHT](#). Valid channels are PF_CHANNEL_1 through PF_CHANNEL_4. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

channel The Power Function channel. See [Power Function channel constants](#).

func The Power Function train function. See [PF/IR Train function constants](#).

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_MSPFTrain.nxc](#).

8.3.3.384 void MSRCXAbsVar (const byte *varnum*, const byte byte *src*, const unsigned int *value*) [inline]

MSRCXAbsVar function. Send the AbsVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXAbsVar.nxc](#).

8.3.3.385 void MSRCXAddToDatalog (const byte *src*, const unsigned int *value*) [inline]

MSRCXAddToDatalog function. Send the AddToDatalog command to an RCX.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXAddToDatalog.nxc](#).

8.3.3.386 void MSRCXAndVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXAndVar function. Send the AndVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXAndVar.nxc](#).

8.3.3.387 int MSRCXBatteryLevel (void) [inline]

MSRCXBatteryLevel function. Send the BatteryLevel command to an RCX to read the current battery level.

Returns:

The RCX battery level.

Examples:

[ex_MSRCXBatteryLevel.nxc](#).

8.3.3.388 void MSRCXBoot (void) [inline]

MSRCXBoot function. Send the Boot command to an RCX.

Examples:

[ex_MSRCXBoot.nxc](#).

8.3.3.389 void MSRCXCalibrateEvent (const byte *evt*, const byte *low*, const byte *hi*, const byte *hyst*) [inline]

MSRCXCalibrateEvent function. Send the CalibrateEvent command to an RCX.

Parameters:

evt The event number.

low The low threshold.

hi The high threshold.

hyst The hysteresis value.

Examples:

[ex_MSRCXCalibrateEvent.nxc](#).

8.3.3.390 void MSRCXClearAllEvents (void) [inline]

MSRCXClearAllEvents function. Send the ClearAllEvents command to an RCX.

Examples:

[ex_MSRCXClearAllEvents.nxc](#).

8.3.3.391 void MSRCXClearCounter (const byte *counter*) [inline]

MSRCXClearCounter function. Send the ClearCounter command to an RCX.

Parameters:

counter The counter to clear.

Examples:

[ex_MSRCXClearCounter.nxc](#).

8.3.3.392 void MSRCXClearMsg (void) [inline]

MSRCXClearMsg function. Send the ClearMsg command to an RCX.

Examples:

[ex_MSRCXClearMsg.nxc](#).

8.3.3.393 void MSRCXClearSensor (const byte *port*) [inline]

MSRCXClearSensor function. Send the ClearSensor command to an RCX.

Parameters:

port The RCX port number.

Examples:

[ex_MSRCXClearSensor.nxc](#).

8.3.3.394 void MSRCXClearSound (void) [inline]

MSRCXClearSound function. Send the ClearSound command to an RCX.

Examples:

[ex_MSRCXClearSound.nxc](#).

8.3.3.395 void MSRCXClearTimer (const byte *timer*) [inline]

MSRCXClearTimer function. Send the ClearTimer command to an RCX.

Parameters:

timer The timer to clear.

Examples:

[ex_MSRCXClearTimer.nxc](#).

8.3.3.396 void MSRCXCreateDatalog (const unsigned int *size*) [inline]

MSRCXCreateDatalog function. Send the CreateDatalog command to an RCX.

Parameters:

size The new datalog size.

Examples:

[ex_MSRCXCreateDatalog.nxc](#).

8.3.3.397 void MSRCXDecCounter (const byte *counter*) [inline]

MSRCXDecCounter function. Send the DecCounter command to an RCX.

Parameters:

counter The counter to decrement.

Examples:

[ex_MSRCXDecCounter.nxc](#).

8.3.3.398 void MSRCXDeleteSub (const byte *s*) [inline]

MSRCXDeleteSub function. Send the DeleteSub command to an RCX.

Parameters:

s The subroutine number to delete.

Examples:

[ex_MSRCXDeleteSub.nxc](#).

8.3.3.399 void MSRCXDeleteSubs (void) [inline]

MSRCXDeleteSubs function. Send the DeleteSubs command to an RCX.

Examples:

[ex_MSRCXDeleteSubs.nxc](#).

8.3.3.400 void MSRCXDeleteTask (const byte *t*) [inline]

MSRCXDeleteTask function. Send the DeleteTask command to an RCX.

Parameters:

t The task number to delete.

Examples:

[ex_MSRCXDeleteTask.nxc](#).

8.3.3.401 void MSRCXDeleteTasks (void) [inline]

MSRCXDeleteTasks function. Send the DeleteTasks command to an RCX.

Examples:

[ex_MSRCXDeleteTasks.nxc](#).

8.3.3.402 void MSRCXDisableOutput (const byte *outputs*) [inline]

MSRCXDisableOutput function. Send the DisableOutput command to an RCX.

Parameters:

outputs The RCX output(s) to disable. See [RCX output constants](#).

Examples:

[ex_MSRCXDisableOutput.nxc](#).

8.3.3.403 void MSRCXDivVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXDivVar function. Send the DivVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXDivVar.nxc](#).

8.3.3.404 void MSRCXEnableOutput (const byte *outputs*) [inline]

MSRCXEnableOutput function. Send the EnableOutput command to an RCX.

Parameters:

outputs The RCX output(s) to enable. See [RCX output constants](#).

Examples:

[ex_MSRCXEnableOutput.nxc](#).

**8.3.3.405 void MSRCXEvent (const byte *src*, const unsigned int *value*)
[inline]**

MSRCXEvent function. Send the Event command to an RCX.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXEvent.nxc](#).

8.3.3.406 void MSRCXFloat (const byte *outputs*) [inline]

MSRCXFloat function. Send commands to an RCX to float the specified outputs.

Parameters:

outputs The RCX output(s) to float. See [RCX output constants](#).

Examples:

[ex_MSRCXFloat.nxc](#).

8.3.3.407 void MSRCXFwd (const byte *outputs*) [inline]

MSRCXFwd function. Send commands to an RCX to set the specified outputs to the forward direction.

Parameters:

outputs The RCX output(s) to set forward. See [RCX output constants](#).

Examples:

[ex_MSRCXFwd.nxc](#).

8.3.3.408 void MSRCXIncCounter (const byte *counter*) [inline]

MSRCXIncCounter function. Send the IncCounter command to an RCX.

Parameters:

counter The counter to increment.

Examples:

[ex_MSRCXIncCounter.nxc](#).

8.3.3.409 void MSRCXInvertOutput (const byte *outputs*) [inline]

MSRCXInvertOutput function. Send the InvertOutput command to an RCX.

Parameters:

outputs The RCX output(s) to invert. See [RCX output constants](#).

Examples:

[ex_MSRCXInvertOutput.nxc](#).

8.3.3.410 void MSRCXMulVar (const byte *varnum*, const byte *src*, unsigned int *value*) [inline]

MSRCXMulVar function. Send the MulVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXMulVar.nxc](#).

8.3.3.411 void MSRCXMutableSound (void) [inline]

MSRCXMutableSound function. Send the MuteSound command to an RCX.

Examples:

[ex_MSRCXMuteSound.nxc](#).

8.3.3.412 void MSRCXObvertOutput (const byte outputs) [inline]

MSRCXObvertOutput function. Send the ObvertOutput command to an RCX.

Parameters:

outputs The RCX output(s) to obvert. See [RCX output constants](#).

Examples:

[ex_MSRCXObvertOutput.nxc](#).

8.3.3.413 void MSRCXOff (const byte outputs) [inline]

MSRCXOff function. Send commands to an RCX to turn off the specified outputs.

Parameters:

outputs The RCX output(s) to turn off. See [RCX output constants](#).

Examples:

[ex_MSRCXOff.nxc](#).

8.3.3.414 void MSRCXOn (const byte *outputs*) [inline]

MSRCXOn function. Send commands to an RCX to turn on the specified outputs.

Parameters:

outputs The RCX output(s) to turn on. See [RCX output constants](#).

Examples:

[ex_MSRCXOn.nxc](#).

8.3.3.415 void MSRCXOnFor (const byte *outputs*, const unsigned int *ms*) [inline]

MSRCXOnFor function. Send commands to an RCX to turn on the specified outputs in the forward direction for the specified duration.

Parameters:

outputs The RCX output(s) to turn on. See [RCX output constants](#).

ms The number of milliseconds to leave the outputs on

Examples:

[ex_MSRCXOnFor.nxc](#).

8.3.3.416 void MSRCXOnFwd (const byte *outputs*) [inline]

MSRCXOnFwd function. Send commands to an RCX to turn on the specified outputs in the forward direction.

Parameters:

outputs The RCX output(s) to turn on in the forward direction. See [RCX output constants](#).

Examples:

[ex_MSRCXOnFwd.nxc](#).

8.3.3.417 void MSRCXOnRev (const byte *outputs*) [inline]

MSRCXOnRev function. Send commands to an RCX to turn on the specified outputs in the reverse direction.

Parameters:

outputs The RCX output(s) to turn on in the reverse direction. See [RCX output constants](#).

Examples:

[ex_MSRCXOnRev.nxc](#).

8.3.3.418 void MSRCXOrVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXOrVar function. Send the OrVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXOrVar.nxc](#).

8.3.3.419 void MSRCXPBTurnOff (void) [inline]

MSRCXPBTurnOff function. Send the PBTurnOff command to an RCX.

Examples:

[ex_MSRCXPBTurnOff.nxc](#).

8.3.3.420 void MSRCXPing (void) [inline]

MSRCXPing function. Send the Ping command to an RCX.

Examples:

[ex_MSRCXPing.nxc](#).

8.3.3.421 void MSRCXPlaySound (const byte *snd*) [inline]

MSRCXPlaySound function. Send the PlaySound command to an RCX.

Parameters:

snd The sound number to play.

Examples:

[ex_MSRCXPlaySound.nxc](#).

8.3.3.422 void MSRCXPlayTone (const unsigned int *freq*, const byte *duration*) [inline]

MSRCXPlayTone function. Send the PlayTone command to an RCX.

Parameters:

freq The frequency of the tone to play.

duration The duration of the tone to play.

Examples:

[ex_MSRCXPlayTone.nxc](#).

8.3.3.423 void MSRCXPlayToneVar (const byte *varnum*, const byte *duration*) [inline]

MSRCXPlayToneVar function. Send the PlayToneVar command to an RCX.

Parameters:

varnum The variable containing the tone frequency to play.

duration The duration of the tone to play.

Examples:

[ex_MSRCXPlayToneVar.nxc](#).

8.3.3.424 int MSRCXPoll (const byte *src*, const byte *value*) [inline]

MSRCXPoll function. Send the Poll command to an RCX to read a signed 2-byte value at the specified source and value combination.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Returns:

The value read from the specified port and value.

Examples:

[ex_MSRCXPoll.nxc](#).

8.3.3.425 int MSRCXPollMemory (const unsigned int *address*) [inline]

MSRCXPollMemory function. Send the PollMemory command to an RCX.

Parameters:

address The RCX memory address.

Returns:

The value read from the specified address.

Examples:

[ex_MSRCXPollMemory.nxc](#).

8.3.3.426 void MSRCXRemote (unsigned int *cmd*) [inline]

MSRCXRemote function. Send the Remote command to an RCX.

Parameters:

cmd The RCX IR remote command to send. See [RCX IR remote constants](#).

Examples:

[ex_MSRCXRemote.nxc](#).

8.3.3.427 void MSRCXReset (void) [inline]

MSRCXReset function. Send the Reset command to an RCX.

Examples:

[ex_MSRCXReset.nxc](#).

8.3.3.428 void MSRCXRev (const byte *outputs*) [inline]

MSRCXRev function. Send commands to an RCX to set the specified outputs to the reverse direction.

Parameters:

outputs The RCX output(s) to reverse direction. See [RCX output constants](#).

Examples:

[ex_MSRCXRev.nxc](#).

8.3.3.429 void MSRCXSelectDisplay (const byte *src*, const unsigned int *value*) [inline]

MSRCXSelectDisplay function. Send the SelectDisplay command to an RCX.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXSelectDisplay.nxc](#).

8.3.3.430 void MSRCXSelectProgram (const byte *prog*) [inline]

MSRCXSelectProgram function. Send the SelectProgram command to an RCX.

Parameters:

prog The program number to select.

Examples:

[ex_MSRCXSelectProgram.nxc](#).

8.3.3.431 void MSRCXSendSerial (const byte *first*, const byte *count*) [inline]

MSRCXSendSerial function. Send the SendSerial command to an RCX.

Parameters:

first The first byte address.

count The number of bytes to send.

Examples:

[ex_MSRCXSendSerial.nxc](#).

8.3.3.432 void MSRCXSet (const byte *dtsrc*, const byte *dstval*, const byte *src*, unsigned int *value*) [inline]

MSRCXSet function. Send the Set command to an RCX.

Parameters:

dstsrc The RCX destination source. See [RCX and Scout source constants](#).
dstval The RCX destination value.
src The RCX source. See [RCX and Scout source constants](#).
value The RCX value.

Examples:

[ex_MSRCXSet.nxc](#).

**8.3.3.433 void MSRCXSetDirection (const byte *outputs*, const byte *dir*)
[inline]**

MSRCXSetDirection function. Send the SetDirection command to an RCX to configure the direction of the specified outputs.

Parameters:

outputs The RCX output(s) to set direction. See [RCX output constants](#).
dir The RCX output direction. See [RCX output direction constants](#).

Examples:

[ex_MSRCXSetDirection.nxc](#).

**8.3.3.434 void MSRCXSetEvent (const byte *evt*, const byte *src*, const byte *type*)
[inline]**

MSRCXSetEvent function. Send the SetEvent command to an RCX.

Parameters:

evt The event number to set.
src The RCX source. See [RCX and Scout source constants](#).
type The event type.

Examples:

[ex_MSRCXSetEvent.nxc](#).

**8.3.3.435 void MSRCXSetGlobalDirection (const byte *outputs*, const byte *dir*)
[inline]**

MSRCXSetGlobalDirection function. Send the SetGlobalDirection command to an RCX.

Parameters:

outputs The RCX output(s) to set global direction. See [RCX output constants](#).
dir The RCX output direction. See [RCX output direction constants](#).

Examples:

[ex_MSRCXSetGlobalDirection.nxc](#).

**8.3.3.436 void MSRCXSetGlobalOutput (const byte *outputs*, const byte *mode*)
[inline]**

MSRCXSetGlobalOutput function. Send the SetGlobalOutput command to an RCX.

Parameters:

outputs The RCX output(s) to set global mode. See [RCX output constants](#).
mode The RCX output mode. See [RCX output mode constants](#).

Examples:

[ex_MSRCXSetGlobalOutput.nxc](#).

**8.3.3.437 void MSRCXSetMaxPower (const byte *outputs*, const byte *pwrsrc*,
const byte *pwrvval*) [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](#).
pwrvval The RCX value.

Examples:

[ex_MSRCXSetMaxPower.nxc](#).

8.3.3.438 void MSRCXSetMessage (const byte *msg*) [inline]

MSRCXSetMessage function. Send the SetMessage command to an RCX.

Parameters:

msg The numeric message to send.

Examples:

[ex_MSRCXSetMessage.nxc](#).

8.3.3.439 void MSRCXSetNRLinkPort (const byte *port*, const byte *i2caddr*) [inline]

MSRCXSetIRLinkPort 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 Lowspeed port before using any of the mindsensors RCX and Scout NRLink functions.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Examples:

[ex_MSRCXSetNRLinkPort.nxc](#).

8.3.3.440 void MSRCXSetOutput (const byte *outputs*, const byte *mode*) [inline]

MSRCXSetOutput function. Send the SetOutput command to an RCX to configure the mode of the specified outputs

Parameters:

outputs The RCX output(s) to set mode. See [RCX output constants](#).

mode The RCX output mode. See [RCX output mode constants](#).

Examples:

[ex_MSRCXSetOutput.nxc](#).

8.3.3.441 void MSRCXSetPower (const byte *outputs*, const byte *pwrsrc*, const byte *pwrval*) [inline]

MSRCXSetPower function. Send the SetPower command to an RCX to configure the power level of the specified outputs.

Parameters:

outputs The RCX output(s) to set power. See [RCX output constants](#).

pwrsrc The RCX source. See [RCX and Scout source constants](#).

pwrval The RCX value.

Examples:

[ex_MSRCXSetPower.nxc](#).

8.3.3.442 void MSRCXSetPriority (const byte *p*) [inline]

MSRCXSetPriority function. Send the SetPriority command to an RCX.

Parameters:

p The new task priority.

Examples:

[ex_MSRCXSetPriority.nxc](#).

**8.3.3.443 void MSRCXSetSensorMode (const byte *port*, const byte *mode*)
[inline]**

MSRCXSetSensorMode function. Send the SetSensorMode command to an RCX.

Parameters:

port The RCX sensor port.
mode The RCX sensor mode.

Examples:

[ex_MSRCXSetSensorMode.nxc](#).

**8.3.3.444 void MSRCXSetSensorType (const byte *port*, const byte *type*)
[inline]**

MSRCXSetSensorType function. Send the SetSensorType command to an RCX.

Parameters:

port The RCX sensor port.
type The RCX sensor type.

Examples:

[ex_MSRCXSetSensorType.nxc](#).

8.3.3.445 void MSRCXSetSleepTime (const byte *t*) [inline]

MSRCXSetSleepTime function. Send the SetSleepTime command to an RCX.

Parameters:

t The new sleep time value.

Examples:

[ex_MSRCXSetSleepTime.nxc](#).

8.3.3.446 void MSRCXSetTxPower (const byte *pwr*) [inline]

MSRCXSetTxPower function. Send the SetTxPower command to an RCX.

Parameters:

pwr The IR transmit power level.

Examples:

[ex_MSRCXSetTxPower.nxc](#).

8.3.3.447 void MSRCXSetUserDisplay (const byte *src*, const unsigned int *value*, const byte *precision*) [inline]

MSRCXSetUserDisplay function. Send the SetUserDisplay command to an RCX.

Parameters:

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

precision The number of digits of precision.

Examples:

[ex_MSRCXSetUserDisplay.nxc](#).

8.3.3.448 void MSRCXSetVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSetVar function. Send the SetVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXSetVar.nxc](#).

**8.3.3.449 void MSRCXSetWatch (const byte *hours*, const byte *minutes*)
[inline]**

MSRCXSetWatch function. Send the SetWatch command to an RCX.

Parameters:

hours The new watch time hours value.
minutes The new watch time minutes value.

Examples:

[ex_MSRCXSetWatch.nxc](#).

**8.3.3.450 void MSRCXSgnVar (const byte *varnum*, const byte *src*, const
unsigned int *value*) [inline]**

MSRCXSgnVar function. Send the SgnVar command to an RCX.

Parameters:

varnum The variable number to change.
src The RCX source. See [RCX and Scout source constants](#).
value The RCX value.

Examples:

[ex_MSRCXSgnVar.nxc](#).

8.3.3.451 void MSRCXStartTask (const byte *t*) [inline]

MSRCXStartTask function. Send the StartTask command to an RCX.

Parameters:

t The task number to start.

Examples:

[ex_MSRCXStartTask.nxc](#).

8.3.3.452 void MSRCXStopAllTasks (void) [inline]

MSRCXStopAllTasks function. Send the StopAllTasks command to an RCX.

Examples:

[ex_MSRCXStopAllTasks.nxc](#).

8.3.3.453 void MSRCXStopTask (const byte *t*) [inline]

MSRCXStopTask function. Send the StopTask command to an RCX.

Parameters:

t The task number to stop.

Examples:

[ex_MSRCXStopTask.nxc](#).

8.3.3.454 void MSRCXSubVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSubVar function. Send the SubVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXSubVar.nxc](#).

8.3.3.455 void MSRCXSumVar (const byte *varnum*, const byte *src*, const unsigned int *value*) [inline]

MSRCXSumVar function. Send the SumVar command to an RCX.

Parameters:

varnum The variable number to change.

src The RCX source. See [RCX and Scout source constants](#).

value The RCX value.

Examples:

[ex_MSRCXSumVar.nxc](#).

8.3.3.456 void MSRCXToggle (const byte *outputs*) [inline]

MSRCXToggle function. Send commands to an RCX to toggle the direction of the specified outputs.

Parameters:

outputs The RCX output(s) to toggle. See [RCX output constants](#).

Examples:

[ex_MSRCXToggle.nxc](#).

8.3.3.457 void MSRCXUnlock (void) [inline]

MSRCXUnlock function. Send the Unlock command to an RCX.

Examples:

[ex_MSRCXUnlock.nxc](#).

8.3.3.458 void MSRCXUnmuteSound (void) [inline]

MSRCXUnmuteSound function. Send the UnmuteSound command to an RCX.

Examples:

[ex_MSRCXUnmuteSound.nxc](#).

8.3.3.459 int MSReadValue (const byte *port*, const byte *i2caddr*, 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](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

reg The device register to read.

numbytes The number of bytes to read. Only 1 or 2 byte values are supported.

Returns:

The function call result.

Examples:

[ex_MSReadValue.nxc](#).

8.3.3.460 void MSScoutCalibrateSensor (void) [inline]

MSScoutCalibrateSensor function. Send the CalibrateSensor command to a Scout.

Examples:

[ex_MSScoutCalibrateSensor.nxc](#).

8.3.3.461 void MSScoutMuteSound (void) [inline]

MSScoutMuteSound function. Send the MuteSound command to a Scout.

Examples:

[ex_MSScoutMuteSound.nxc](#).

8.3.3.462 void MSScoutSelectSounds (const byte *grp*) [inline]

MSScoutSelectSounds function. Send the SelectSounds command to a Scout.

Parameters:

grp The Scout sound group to select.

Examples:

[ex_MSScoutSelectSounds.nxc](#).

8.3.3.463 void MSScoutSendVLL (const byte *src*, const unsigned int *value*) [inline]

MSScoutSendVLL function. Send the SendVLL command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSendVLL.nxc](#).

8.3.3.464 void MSScoutSetCounterLimit (const byte *ctr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetCounterLimit function. Send the SetCounterLimit command to a Scout.

Parameters:

ctr The counter for which to set the limit.

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSetCounterLimit.nxc](#).

8.3.3.465 void MSScoutSetEventFeedback (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetEventFeedback function. Send the SetEventFeedback command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSetEventFeedback.nxc](#).

8.3.3.466 void MSScoutSetLight (const byte *x*) [inline]

MSScoutSetLight function. Send the SetLight command to a Scout.

Parameters:

x Set the light on or off using this value. See [Scout light constants](#).

Examples:

[ex_MSScoutSetLight.nxc](#).

8.3.3.467 void MSScoutSetScoutMode (const byte *mode*) [inline]

MSScoutSetScoutMode function. Send the SetScoutMode command to a Scout.

Parameters:

mode Set the scout mode. See [Scout mode constants](#).

Examples:

[ex_MSScoutSetScoutMode.nxc](#).

8.3.3.468 void MSScoutSetScoutRules (const byte *m*, const byte *t*, const byte *l*, const byte *tm*, const byte *fx*) [inline]

MSScoutSetScoutRules function. Send the SetScoutRules command to a Scout.

Parameters:

- m* Scout motion rule. See [Scout motion rule constants](#).
- t* Scout touch rule. See [Scout touch rule constants](#).
- l* Scout light rule. See [Scout light rule constants](#).
- tm* Scout transmit rule. See [Scout transmit rule constants](#).
- fx* Scout special effects rule. See [Scout special effect constants](#).

Examples:

[ex_MSScoutSetScoutRules.nxc](#).

8.3.3.469 void MSScoutSetSensorClickTime (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorClickTime function. Send the SetSensorClickTime command to a Scout.

Parameters:

- src* The Scout source. See [RCX and Scout source constants](#).
- value* The Scout value.

Examples:

[ex_MSScoutSetSensorClickTime.nxc](#).

8.3.3.470 void MSScoutSetSensorHysteresis (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorHysteresis function. Send the SetSensorHysteresis command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSetSensorHysteresis.nxc](#).

8.3.3.471 void MSScoutSetSensorLowerLimit (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorLowerLimit function. Send the SetSensorLowerLimit command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSetSensorLowerLimit.nxc](#).

8.3.3.472 void MSScoutSetSensorUpperLimit (const byte *src*, const unsigned int *value*) [inline]

MSScoutSetSensorUpperLimit function. Send the SetSensorUpperLimit command to a Scout.

Parameters:

src The Scout source. See [RCX and Scout source constants](#).

value The Scout value.

Examples:

[ex_MSScoutSetSensorUpperLimit.nxc](#).

8.3.3.473 void MSScoutSetTimerLimit (const byte *tmr*, const byte *src*, const unsigned int *value*) [inline]

MSScoutSetTimerLimit function. Send the SetTimerLimit command to a Scout.

Parameters:

- tmr* The timer for which to set a limit.
- src* The Scout source. See [RCX and Scout source constants](#).
- value* The Scout value.

Examples:

[ex_MSScoutSetTimerLimit.nxc](#).

8.3.3.474 void MSScoutUnmuteSound (void) [inline]

MSScoutUnmuteSound function. Send the UnmuteSound command to a Scout.

Examples:

[ex_MSScoutUnmuteSound.nxc](#).

8.3.3.475 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_multdiv32.nxc](#).

8.3.3.476 char NRLink2400 (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in 2400 baud mode. Configure the mindsensors NRLink device in 2400 baud mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLink2400.nxc](#).

8.3.3.477 char NRLink4800 (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in 4800 baud mode. Configure the mindsensors NRLink device in 4800 baud mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLink4800.nxc](#).

8.3.3.478 char NRLinkFlush (const byte *port*, const byte *i2caddr*) [inline]

Flush NRLink buffers. Flush the mindsensors NRLink device buffers. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkFlush.nxc](#).

8.3.3.479 char NRLinkIRLong (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in IR long mode. Configure the mindsensors NRLink device in IR long mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkIRLong.nxc](#).

8.3.3.480 char NRLinkIRShort (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLink in IR short mode. Configure the mindsensors NRLink device in IR short mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLLinkIRShort.nxc](#).

8.3.3.481 char NRLLinkSetPF (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLLink in power function mode. Configure the mindsensors NRLLink device in power function mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLLinkSetPF.nxc](#).

8.3.3.482 char NRLLinkSetRCX (const byte *port*, const byte *i2caddr*) [inline]

Configure NRLLink in RCX mode. Configure the mindsensors NRLLink device in RCX mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLLinkSetRCX.nxc](#).

**8.3.3.483 char NRLLinkSetTrain (const byte *port*, const byte *i2caddr*)
[inline]**

Configure NRLLink in IR train mode. Configure the mindsensors NRLLink device in IR train mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLLinkSetTrain.nxc](#).

8.3.3.484 byte NRLLinkStatus (const byte *port*, const byte *i2caddr*) [inline]

Read NRLLink status. Read the status of the mindsensors NRLLink device. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The mindsensors NRLLink status.

Examples:

[ex_NRLLinkStatus.nxc](#).

**8.3.3.485 char NRLinkTxRaw (const byte *port*, const byte *i2caddr*)
[inline]**

Configure NRLink in raw IR transmit mode. Configure the mindsensors NRLink device in raw IR transmit mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_NRLinkTxRaw.nxc](#).

8.3.3.486 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_dispgaout.nxc](#), [ex_dispgout.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_FlattenVar.nxc](#), [ex_getchar.nxc](#), [ex_isNAN.nxc](#), [ex_labs.nxc](#), [ex_ldiv.nxc](#), [ex_memcmp.nxc](#), [ex_NumOut.nxc](#), [ex_readdressof.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](#), [ex_SysWriteSemData.nxc](#), and [ex_UnflattenVar.nxc](#).

8.3.3.487 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](#).

8.3.3.488 void Off (byte *outputs*) [inline]

Turn motors off. Turn the specified outputs off (with braking).

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

Examples:[ex_off.nxc.](#)**8.3.3.489 void OffEx (byte *outputs*, const byte *reset*) [inline]**

Turn motors off and reset counters. Turn the specified outputs off (with braking).

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:[ex_offex.nxc.](#)**8.3.3.490 byte OnBrickProgramPointer (void) [inline]**

Read the on brick program pointer value. Return the current OBP (on-brick program) step

Returns:

On brick program pointer (step).

Examples:[ex_OnBrickProgramPointer.nxc.](#)**8.3.3.491 void OnFwd (byte *outputs*, char *pwr*) [inline]**

Run motors forward. Set outputs to forward direction and turn them on.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

Examples:

[ex_onfwd.nxc](#), [ex_yield.nxc](#), and [util_rpm.nxc](#).

8.3.3.492 void OnFwdEx (byte *outputs*, char *pwr*, const byte *reset*) [inline]

Run motors forward and reset counters. Set outputs to forward direction and turn them on.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onfwdex.nxc](#).

8.3.3.493 void OnFwdReg (byte *outputs*, char *pwr*, byte *regmode*) [inline]

Run motors forward regulated. Run the specified outputs forward using the specified regulation mode.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

Examples:

[ex_onfwdreg.nxc](#).

8.3.3.494 void OnFwdRegEx (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*) [inline]

Run motors forward regulated and reset counters. Run the specified outputs forward using the specified regulation mode.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onfwdregex.nxc](#).

8.3.3.495 void OnFwdRegExPID (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*, byte *p*, byte *i*, byte *d*) [inline]

Run motors forward regulated and reset counters with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onfwdregexpid.nxc](#).

8.3.3.496 void OnFwdRegPID (byte *outputs*, char *pwr*, byte *regmode*, byte *p*, byte *i*, byte *d*) [inline]

Run motors forward regulated with PID factors. Run the specified outputs forward using the specified regulation mode. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onfwdregpid.nxc](#).

8.3.3.497 void OnFwdSync (byte *outputs*, char *pwr*, char *turnpct*) [inline]

Run motors forward synchronised. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

Examples:

[ex_onfwdsync.nxc](#).

8.3.3.498 void OnFwdSyncEx (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*) [inline]

Run motors forward synchronised and reset counters. Run the specified outputs forward with regulated synchronization using the specified turn ratio.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onfwdsyncex.nxc](#).

8.3.3.499 void OnFwdSyncExPID (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*, byte *p*, byte *i*, byte *d*) [inline]

Run motors forward synchronised and reset counters with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onfwdsyncexpid.nxc](#).

8.3.3.500 void OnFwdSyncPID (byte *outputs*, char *pwr*, char *turnpct*, byte *p*, byte *i*, byte *d*) [inline]

Run motors forward synchronised with PID factors. Run the specified outputs forward with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onfwdsyncpid.nxc](#).

8.3.3.501 void OnRev (byte *outputs*, char *pwr*) [inline]

Run motors backward. Set outputs to reverse direction and turn them on.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

Examples:

[ex_onrev.nxc](#).

8.3.3.502 void OnRevEx (byte *outputs*, char *pwr*, const byte *reset*) [inline]

Run motors backward and reset counters. Set outputs to reverse direction and turn them on.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onrevex.nxc](#).

8.3.3.503 void OnRevReg (byte *outputs*, char *pwr*, byte *regmode*) [inline]

Run motors forward regulated. Run the specified outputs in reverse using the specified regulation mode.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

Examples:

[ex_onrevreg.nxc](#).

8.3.3.504 void OnRevRegEx (byte *outputs*, char *pwr*, byte *regmode*, const byte *reset*) [inline]

Run motors backward regulated and reset counters. Run the specified outputs in reverse using the specified regulation mode.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onrevregex.nxc](#).

8.3.3.505 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_onrevregpid.nxc](#).

8.3.3.506 void OnRevRegPID (byte *outputs*, char *pwr*, byte *regmode*, byte *p*, byte *i*, byte *d*) [inline]

Run motors reverse regulated with PID factors. Run the specified outputs in reverse using the specified regulation mode. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

regmode Regulation mode, see [Output port regulation mode constants](#).

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onrevregpid.nxc](#).

8.3.3.507 void OnRevSync (byte *outputs*, char *pwr*, char *turnpct*) [inline]

Run motors backward synchronised. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

Examples:

[ex_onrevsync.nxc](#).

8.3.3.508 void OnRevSyncEx (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*) [inline]

Run motors backward synchronised and reset counters. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

Examples:

[ex_onrevsyncex.nxc](#).

8.3.3.509 void OnRevSyncExPID (byte *outputs*, char *pwr*, char *turnpct*, const byte *reset*, byte *p*, byte *i*, byte *d*) [[inline](#)]

Run motors backward synchronised and reset counters with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

reset Position counters reset control. It must be a constant, see [Tachometer counter reset flags](#).

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onrevsyncexpid.nxc](#).

8.3.3.510 void OnRevSyncPID (byte *outputs*, char *pwr*, char *turnpct*, byte *p*, byte *i*, byte *d*) [inline]

Run motors backward synchronised with PID factors. Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_onrevsyncpid.nxc](#).

8.3.3.511 unsigned int OpenFileAppend (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]

Open a file for appending. Open an existing file with the specified filename for writing. The file size is returned in the second parameter, which must be a variable. The file

handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

Parameters:

- fname* The name of the file to open.
- fsize* The size of the file returned by the function.
- handle* The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_file_system.nxc](#), and [ex_OpenFileAppend.nxc](#).

8.3.3.512 `unsigned int OpenFileRead (string fname, unsigned int & fsize, byte & handle) [inline]`

Open a file for reading. Open an existing file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

Parameters:

- fname* The name of the file to open.
- fsize* The size of the file returned by the function.
- handle* The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_file_system.nxc](#), and [ex_OpenFileRead.nxc](#).

8.3.3.513 unsigned int OpenFileReadLinear (string *fname*, unsigned int & *fsize*, byte & *handle*) [inline]

Open a linear file for reading. Open an existing linear file with the specified filename for reading. The file size is returned in the second parameter, which must be a variable. The file handle is returned in the last parameter, which must be a variable. The loader result code is returned as the value of the function call. The filename parameter must be a constant or a variable.

Parameters:

- fname* The name of the file to open.
- fsize* The size of the file returned by the function.
- handle* The file handle output from the function call.

Returns:

The function call result. See [Loader module error codes](#).

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_OpenFileReadLinear.nxc](#).

8.3.3.514 char PlayFile (string *filename*) [inline]

Play a file. Play the specified file. The filename may be any valid string expression. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

Parameters:

- filename* The name of the sound or melody file to play.

Examples:

[ex_PlayFile.nxc](#).

**8.3.3.515 char PlayFileEx (string *filename*, byte *volume*, bool *loop*)
[inline]**

Play a file with extra options. Play the specified file. The filename may be any valid string expression. Volume should be a number from 0 (silent) to 4 (loudest). Play the file repeatedly if loop is true. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

Parameters:

filename The name of the sound or melody file to play.

volume The desired tone volume.

loop A boolean flag indicating whether to play the file repeatedly.

Examples:

[ex_PlayFileEx.nxc](#).

8.3.3.516 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](#).

<i>aCode</i>	Resulting Sound
SOUND_CLICK	key click sound
SOUND_DOUBLE_BEEP	double beep
SOUND_DOWN	sweep down
SOUND_UP	sweep up
SOUND_LOW_BEEP	error sound
SOUND_FAST_UP	fast sweep up

Parameters:

aCode The system sound to play. See [RCX and Scout sound constants](#).

Examples:

[ex_playsound.nxc](#).

**8.3.3.517 char PlayTone (unsigned int *frequency*, unsigned int *duration*)
[inline]**

Play a tone. Play a single tone of the specified frequency and duration. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). The tone is played at the loudest sound level supported by the firmware and it is not looped.

Parameters:

frequency The desired tone frequency, in Hz.
duration The desired tone duration, in ms.

Examples:

[alternating_tasks.nxc](#), [ex_file_system.nxc](#), [ex_PlayTone.nxc](#), and [ex_yield.nxc](#).

**8.3.3.518 char PlayToneEx (unsigned int *frequency*, unsigned int *duration*,
byte *volume*, bool *loop*) [inline]**

Play a tone with extra options. Play a single tone of the specified frequency, duration, and volume. The frequency is in Hz (see the [Tone constants](#) group). The duration is in 1000ths of a second (see the [Time constants](#) group). Volume should be a number from 0 (silent) to 4 (loudest). Play the tone repeatedly if loop is true.

Parameters:

frequency The desired tone frequency, in Hz.
duration The desired tone duration, in ms.
volume The desired tone volume.
loop A boolean flag indicating whether to play the tone repeatedly.

Examples:

[ex_PlayToneEx.nxc](#).

8.3.3.519 void PlayTones (Tone *tones*[])

Play multiple tones. Play a series of tones contained in the tones array. Each element in the array is an instance of the [Tone](#) structure, containing a frequency and a duration.

Parameters:

tones The array of tones to play.

Examples:

[ex_playtones.nxc](#).

8.3.3.520 char PointOut (int *x*, int *y*, unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw a point. This function lets you draw a point on the screen at *x*, *y*. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawPoint](#), [DrawPointType](#)

Parameters:

- x* The *x* value for the point.
- y* The *y* value for the point.
- options* The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_PointOut.nxc](#), [ex_sin_cos.nxc](#), and [ex_sind_cosd.nxc](#).

8.3.3.521 char PolyOut (LocationType *points*[], unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw a polygon. This function lets you draw a polygon on the screen using an array of points. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawPolygon](#), [DrawPolygonType](#)

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

points An array of [LocationType](#) points that define the polygon.

options The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_PolyOut.nxc](#).

8.3.3.522 float pow (float *base*, float *exponent*) [inline]

Raise to power. Computes base raised to the power exponent.

Parameters:

base Floating point value.

exponent Floating point value.

Returns:

The result of raising base to the power exponent.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_pow.nxc](#).

8.3.3.523 void PowerDown () [inline]

Power down the NXT. This function powers down the NXT. The running program will terminate as a result of this action.

Examples:

[ex_powerdown.nxc](#).

8.3.3.524 void Precedes (task *task1*, task *task2*, ..., task *taskN*) [inline]

Declare tasks that this task precedes. Schedule the listed tasks for execution once the current task has completed executing. The tasks will all execute simultaneously unless other dependencies prevent them from doing so. This statement should be used once within a task - preferably at the start of the task definition. Any number of tasks may be listed in the Precedes statement.

Parameters:

task1 The first task to start executing after the current task ends.

task2 The second task to start executing after the current task ends.

taskN The last task to start executing after the current task ends.

Examples:

[alternating_tasks.nxc](#), [ex_Precedes.nxc](#), and [ex_yield.nxc](#).

8.3.3.525 void printf (string *format*, variant *value*) [inline]

Print formatted data to stdout. Writes to the LCD at 0, LCD_LINE1 a sequence of data formatted as the format argument specifies. After the format parameter, the function expects one value argument.

Parameters:

format A string specifying the desired format.

value A value to be formatted for writing to the LCD.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_printf.nxc](#).

**8.3.3.526 char PSPNxAnalog (const byte & *port*, const byte & *i2caddr*)
[inline]**

Configure PSPNx in analog mode. Configure the mindsensors PSPNx device in analog mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_PSPNxAnalog.nxc](#).

**8.3.3.527 char PSPNxDigital (const byte & *port*, const byte & *i2caddr*)
[inline]**

Configure PSPNx in digital mode. Configure the mindsensors PSPNx device in digital mode. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The function call result.

Examples:

[ex_PSPNxDigital.nxc](#).

8.3.3.528 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](#).

8.3.3.529 int Random (unsigned int *n* = 0) [inline]

Generate random number. Return a signed or unsigned 16-bit random number. If the optional argument n is not provided the function will return a signed value. Otherwise the returned value will range between 0 and n (exclusive).

Parameters:

n The maximum unsigned value desired (optional).

Returns:

A random number

Examples:

[ex_ArrayMax.nxc](#), [ex_CircleOut.nxc](#), [ex_dispgoutex.nxc](#), [ex_EllipseOut.nxc](#), [ex_file_system.nxc](#), [ex_Random.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [ex_string.nxc](#), [ex_sysdrawellipse.nxc](#), and [ex_wait.nxc](#).

8.3.3.530 unsigned int Read (byte *handle*, variant & *value*) [inline]

Read a value from a file. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read.

Parameters:

handle The file handle.

value The variable to store the data read from the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_file_system.nxc](#), and [ex_Read.nxc](#).

**8.3.3.531 char ReadButtonEx (const byte *btn*, bool *reset*, bool & *pressed*,
unsigned int & *count*) [inline]**

Read button information. Read the specified button. Set the pressed and count parameters with the current state of the button. Optionally reset the press count after reading it.

Parameters:

btn The button to check. See [Button name constants](#).

reset Whether or not to reset the press counter.

pressed The button pressed state.

count The button press count.

Returns:

The function call result.

Examples:

[ex_ReadButtonEx.nxc](#).

**8.3.3.532 unsigned int ReadBytes (byte *handle*, unsigned int & *length*, byte &
buf[]) [inline]**

Read bytes from a file. Read the specified number of bytes from the file associated with the specified handle. The handle parameter must be a variable. The length parameter must be a variable. The buf parameter must be an array or a string variable. The actual number of bytes read is returned in the length parameter.

Parameters:

handle The file handle.

length The number of bytes to read. Returns the number of bytes actually read.

buf The byte array where the data is stored on output.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_ReadBytes.nxc](#).

**8.3.3.533 char ReadI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte & *out*)
[inline]**

Read I2C register. Read a single byte from an I2C device register.

Parameters:

port The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.

i2caddr The I2C device address.

reg The I2C device register from which to read a single byte.

out The single byte read from the I2C device.

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

Examples:

[ex_readi2cregister.nxc](#).

8.3.3.534 unsigned int ReadLn (byte *handle*, variant & *value*) [inline]

Read a value from a file plus line ending. Read a value from the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a variable. The type of the value parameter determines the number of bytes of data read. The ReadLn function reads two additional bytes from the file which it assumes are a carriage return and line feed pair.

Parameters:

handle The file handle.

value The variable to store the data read from the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_ReadLn.nxc](#).

**8.3.3.535 unsigned int ReadLnString (byte *handle*, string & *output*)
[inline]**

Read a string from a file plus line ending. Read a string from the file associated with the specified handle. The handle parameter must be a variable. The output parameter must be a variable. Appends bytes to the output variable until a line ending (CRLF) is reached. The line ending is also read but it is not appended to the output parameter.

Parameters:

handle The file handle.

output The variable to store the string read from the file.

Returns:

The function call result. See [Loader module error codes](#).

8.3.3.536 bool ReadNRLinkBytes (const byte *port*, const byte *i2caddr*, byte & *data*[]) [inline]

Read data from NRLink. Read data from the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

data A byte array that will contain the data read from the device on output.

Returns:

The function call result.

Examples:

[ex_ReadNRLLinkBytes.nxc](#).

8.3.3.537 int ReadSensorColorEx (const byte & *port*, int & *colorval*, unsigned int & *raw*[], unsigned int & *norm*[], int & *scaled*[]) [inline]

Read LEGO color sensor extra. This function lets you read the LEGO color sensor. It returns the color value, and three arrays containing raw, normalized, and scaled color values for red, green, blue, and none indices.

Parameters:

port The sensor port. See [Input port constants](#).

colorval The color value. See [Color values](#).

raw An array containing four raw color values. See [Color sensor array indices](#).

norm An array containing four normalized color values. See [Color sensor array indices](#).

scaled An array containing four scaled color values. See [Color sensor array indices](#).

Returns:

The function call result.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_ReadSensorColorEx.nxc](#).

8.3.3.538 int ReadSensorColorRaw (const byte & *port*, unsigned int & *rawVals*[]) [inline]

Read LEGO color sensor raw values. This function lets you read the LEGO color sensor. It returns an array containing raw color values for red, green, blue, and none indices.

Parameters:

port The sensor port. See [Input port constants](#).

rawVals An array containing four raw color values. See [Color sensor array indices](#).

Returns:

The function call result.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_ReadSensorColorRaw.nxc](#).

8.3.3.539 `char ReadSensorEMeter (const byte port, float & vIn, float & aIn,
float & vOut, float & aOut, int & joules, float & wIn, float & wOut)
[inline]`

Read the LEGO EMeter values. Read all the LEGO EMeter register values. They must all be read at once to ensure data coherency.

Parameters:

port The port to which the LEGO EMeter sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

vIn Input voltage

aIn Input current

vOut Output voltage

aOut Output current

joules The number of joules stored in the EMeter

wIn The number of watts generated

wOut The number of watts consumed

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

**8.3.3.540 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 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_ReadSensorHTAccel.nxc](#).

**8.3.3.541 bool ReadSensorHTColor (const byte *port*, byte & *ColorNum*, byte
& *Red*, byte & *Green*, byte & *Blue*) [inline]**

Read HiTechnic Color values. Read color number, red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

- port* The sensor port. See [Input port constants](#).
- ColorNum* The output color number.
- Red* The red color value.
- Green* The green color value.
- Blue* The blue color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTColor.nxc](#).

8.3.3.542 bool ReadSensorHTColor2Active (byte *port*, byte & *ColorNum*, byte & *Red*, byte & *Green*, byte & *Blue*, byte & *White*) [inline]

Read HiTechnic Color2 active values. Read color number, red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).
ColorNum The output color number.
Red The red color value.
Green The green color value.
Blue The blue color value.
White The white color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTColor2Active.nxc](#).

8.3.3.543 bool ReadSensorHTIRReceiver (const byte *port*, char & *pfdatal*[]) [inline]

Read HiTechnic IRReceiver Power Function bytes. Read Power Function bytes from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

pfdata Eight bytes of power function remote IR data.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRReceiver.nxc](#).

8.3.3.544 bool ReadSensorHTIRReceiverEx (const byte *port*, const byte *offset*, char & *pfchar*) [inline]

Read HiTechnic IRReceiver Power Function value. Read a Power Function byte from the HiTechnic IRReceiver sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

offset The power function data offset. See [HiTechnic IRReceiver constants](#).

pfchar A single byte of power function remote IR data.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRReceiverEx.nxc](#).

8.3.3.545 bool ReadSensorHTIRSeeker (const byte *port*, byte & *dir*, byte & *s1*, byte & *s3*, byte & *s5*, byte & *s7*, byte & *s9*) [inline]

Read HiTechnic IRSSeeker values. Read direction, and five signal strength values from the HiTechnic IRSSeeker sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

dir The direction.

s1 The signal strength from sensor 1.

s3 The signal strength from sensor 3.

s5 The signal strength from sensor 5.

s7 The signal strength from sensor 7.

s9 The signal strength from sensor 9.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRSeeker.nxc](#).

8.3.3.546 bool ReadSensorHTIRSeeker2AC (const byte *port*, byte & *dir*, byte & *s1*, byte & *s3*, byte & *s5*, byte & *s7*, byte & *s9*) [inline]

Read HiTechnic IRSeeker2 AC values. Read direction, and five signal strength values from the HiTechnic IRSeeker2 sensor in AC mode. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

dir The direction.

s1 The signal strength from sensor 1.

s3 The signal strength from sensor 3.

s5 The signal strength from sensor 5.

s7 The signal strength from sensor 7.

s9 The signal strength from sensor 9.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTIRSeeker2AC.nxc](#).

8.3.3.547 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](#).

8.3.3.548 bool ReadSensorHTNormalizedColor (const byte *port*, byte & *ColorIdx*, byte & *Red*, byte & *Green*, byte & *Blue*) [inline]

Read HiTechnic Color normalized values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

ColorIdx The output color index.

Red The normalized red color value.

Green The normalized green color value.

Blue The normalized blue color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTNormalizedColor.nxc](#).

8.3.3.549 bool ReadSensorHTNormalizedColor2Active (const byte *port*, byte & *ColorIdx*, byte & *Red*, byte & *Green*, byte & *Blue*) [inline]

Read HiTechnic Color2 normalized active values. Read the color index and the normalized red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

ColorIdx The output color index.

Red The normalized red color value.

Green The normalized green color value.

Blue The normalized blue color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTNormalizedColor2Active.nxc](#).

8.3.3.550 bool ReadSensorHTRawColor (const byte *port*, unsigned int & *Red*, unsigned int & *Green*, unsigned int & *Blue*) [inline]

Read HiTechnic Color raw values. Read the raw red, green, and blue values from the HiTechnic Color sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

Red The raw red color value.

Green The raw green color value.

Blue The raw blue color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTRawColor.nxc](#).

**8.3.3.551 bool ReadSensorHTRawColor2 (const byte *port*, unsigned int & *Red*,
unsigned int & *Green*, unsigned int & *Blue*, unsigned int & *White*)
[inline]**

Read HiTechnic Color2 raw values. Read the raw red, green, and blue values from the HiTechnic Color2 sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

Red The raw red color value.

Green The raw green color value.

Blue The raw blue color value.

White The raw white color value.

Returns:

The function call result.

Examples:

[ex_ReadSensorHTRawColor2.nxc](#).

**8.3.3.552 void ReadSensorHTTouchMultiplexer (const byte *port*, byte & *t1*,
byte & *t2*, byte & *t3*, byte & *t4*) [inline]**

Read HiTechnic touch multiplexer. Read touch sensor values from the HiTechnic touch multiplexer device.

Parameters:

- port* The sensor port. See [Input port constants](#).
- t1* The value of touch sensor 1.
- t2* The value of touch sensor 2.
- t3* The value of touch sensor 3.
- t4* The value of touch sensor 4.

Examples:

[ex_ReadSensorHTTouchMultiplexer.nxc](#).

**8.3.3.553 bool ReadSensorMSAccel (const byte *port*, const byte *i2caddr*, int &
x, int & *y*, int & *z*) [inline]**

Read mindsensors acceleration values. Read X, Y, and Z axis acceleration values from the mindsensors Accelerometer sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

- port* The sensor port. See [Input port constants](#).
- i2caddr* The sensor I2C address. See sensor documentation for this value.
- x* The output x-axis acceleration.
- y* The output y-axis acceleration.
- z* The output z-axis acceleration.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSAccel.nxc](#).

**8.3.3.554 bool ReadSensorMSPlayStation (const byte *port*, const byte *i2caddr*,
byte & *btnset1*, byte & *btnset2*, byte & *xleft*, byte & *yleft*, byte &
xright, byte & *yright*) [inline]**

Read mindsensors playstation controller values. Read playstation controller values from the mindsensors playstation sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

btnset1 The button set 1 values. See [MindSensors PSP-Nx button set 1 constants](#).

btnset2 The button set 2 values. See [MindSensors PSP-Nx button set 2 constants](#).

xleft The left joystick x value.

yleft The left joystick y value.

xright The right joystick x value.

yright The right joystick y value.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSPlayStation.nxc](#).

**8.3.3.555 bool ReadSensorMSRTClock (const byte *port*, byte & *sec*, byte &
min, byte & *hrs*, byte & *dow*, byte & *date*, byte & *month*, byte &
year) [inline]**

Read mindsensors RTClock values. Read real-time clock values from the Mindsensors RTClock sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

sec The seconds.

min The minutes.
hrs The hours.
dow The day of week number.
date The day.
month The month.
year The year.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSRTClock.nxc](#).

8.3.3.556 bool ReadSensorMSTilt (const byte *port*, const byte *i2caddr*, byte & *x*, byte & *y*, byte & *z*) [inline]

Read mindsensors tilt values. Read X, Y, and Z axis tilt values from the mindsensors tilt sensor. Returns a boolean value indicating whether or not the operation completed successfully. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).
i2caddr The sensor I2C address. See sensor documentation for this value.
x The output x-axis tilt.
y The output y-axis tilt.
z The output z-axis tilt.

Returns:

The function call result.

Examples:

[ex_ReadSensorMSTilt.nxc](#).

8.3.3.557 char ReadSensorUSEx (const byte *port*, byte & *values*[]) [inline]

Read multiple ultrasonic sensor values. Return eight ultrasonic sensor distance values.

Parameters:

port The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

values An array of bytes that will contain the 8 distance values read from the ultrasonic sensor.

Returns:

A status code indicating whether the read completed successfully or not. See [CommLSReadType](#) for possible result values.

Examples:

[ex_readsensorusex.nxc](#).

8.3.3.558 void RebootInFirmwareMode () [inline]

Reboot the NXT in firmware download mode. This function lets you reboot the NXT into SAMBA or firmware download mode. The running program will terminate as a result of this action.

Examples:

[ex_rebootinfirmwaremode.nxc](#).

8.3.3.559 char ReceiveMessage (byte *queue*, bool *clear*, string & *msg*) [inline]

Read a message from a queue/mailbox. Read a message from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

clear A flag indicating whether to remove the message from the mailbox after it has been read.

msg The message that is read from the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

**8.3.3.560 char ReceiveRemoteBool (byte *queue*, bool *clear*, bool & *bval*)
[inline]**

Read a boolean value from a queue/mailbox. Read a boolean value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

clear A flag indicating whether to remove the message from the mailbox after it has been read.

bval The boolean value that is read from the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_ReceiveRemoteBool.nxc](#), and [ex_ReceiveRemoteNumber.nxc](#).

**8.3.3.561 char ReceiveRemoteMessageEx (byte *queue*, bool *clear*, string & *str*,
long & *val*, bool & *bval*) [inline]**

Read a value from a queue/mailbox. Read a value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number. Output the value in string, number, and boolean form.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

clear A flag indicating whether to remove the message from the mailbox after it has been read.

str The string value that is read from the mailbox.

val The numeric value that is read from the mailbox.

bval The boolean value that is read from the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_ReceiveRemoteMessageEx.nxc](#).

**8.3.3.562 char ReceiveRemoteNumber (byte *queue*, bool *clear*, long & *val*)
[inline]**

Read a numeric value from a queue/mailbox. Read a numeric value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

clear A flag indicating whether to remove the message from the mailbox after it has been read.

val The numeric value that is read from the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

**8.3.3.563 char ReceiveRemoteString (byte *queue*, bool *clear*, string & *str*)
[inline]**

Read a string value from a queue/mailbox. Read a string value from a mailbox and optionally remove it. If the local mailbox is empty and this NXT is the master then it attempts to poll one of its slave NXTs for a message from the response mailbox that corresponds to the specified local mailbox number.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

clear A flag indicating whether to remove the message from the mailbox after it has been read.

str The string value that is read from the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_ReceiveRemoteString.nxc](#).

8.3.3.564 bool RechargeableBattery (void) [inline]

Read battery type. Return whether the NXT has a rechargeable battery installed or not.

Returns:

Whether the battery is rechargeable or not. (false = no, true = yes)

Examples:

[ex_RechargeableBattery.nxc](#).

8.3.3.565 char RectOut (int *x*, int *y*, int *width*, int *height*, unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw a rectangle. This function lets you draw a rectangle on the screen at *x*, *y* with the specified width and height. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawRect](#), [DrawRectType](#)

Parameters:

x The x value for the top left corner of the rectangle.

y The y value for the top left corner of the rectangle.

width The width of the rectangle.

height The height of the rectangle.

options The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_RectOut.nxc](#).

8.3.3.566 unsigned long reladdressOf (variant *data*) [inline]

Get the relative address of a variable. Get the relative address of a variable and return it to the calling routine as an unsigned long value. The relative address is an offset from the Command module's MemoryPool address.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Parameters:

data A variable whose address you wish to get.

Returns:

The relative address of the variable.

Examples:

[ex_reladdressof.nxc](#).

8.3.3.567 void Release (mutex *m*) [inline]

Acquire a mutex. Release the specified mutex variable. Use this to relinquish a mutex so that it can be acquired by another task. Release should always be called after a matching call to Acquire and as soon as possible after a shared resource is no longer needed.

Parameters:

m The mutex to release.

Examples:

[ex_Acquire.nxc](#), and [ex_Release.nxc](#).

8.3.3.568 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](#).

8.3.3.569 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](#).

**8.3.3.570 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](#).

**8.3.3.571 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](#).

8.3.3.572 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](#).

8.3.3.573 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](#).

8.3.3.574 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](#).

8.3.3.575 char RemoteSetInputMode (byte *conn*, byte *port*, byte *type*, byte *mode*) [inline]

Send a SetInputModule message. Send the SetInputModule 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_RemoteSetInputModule.nxc](#).

8.3.3.576 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](#).

8.3.3.577 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](#).

8.3.3.578 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](#).

8.3.3.579 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](#).

8.3.3.580 int remove (string *filename*) [inline]

Remove file. Delete the specified file. The loader result code is returned as the value of the function call.

Parameters:

filename The name of the file to be deleted.

Returns:

The loader result code.

8.3.3.581 int rename (string *old*, string *new*) [inline]

Rename file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call.

Parameters:

old The name of the file to be renamed.

new The new name for the file.

Returns:

The loader result code.

Examples:

[ex_rename.nxc](#).

8.3.3.582 unsigned int RenameFile (string *oldname*, string *newname*) [inline]

Rename a file. Rename a file from the old filename to the new filename. The loader result code is returned as the value of the function call. The filename parameters must be constants or variables.

Parameters:

oldname The old filename.

newname The new filename.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_RenameFile.nxc](#).

8.3.3.583 void ResetAllTachoCounts (byte *outputs*) [inline]

Reset all tachometer counters. Reset all three position counters and reset the current tachometer limit goal for the specified outputs.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

Examples:

[ex_resetalltachocounts.nxc](#).

8.3.3.584 void ResetBlockTachoCount (byte *outputs*) [inline]

Reset block-relative counter. Reset the block-relative position counter for the specified outputs.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

Examples:

[ex_resetblocktachocount.nxc](#).

8.3.3.585 void ResetRotationCount (byte *outputs*) [inline]

Reset program-relative counter. Reset the program-relative position counter for the specified outputs.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

Examples:

[ex_resetrotationcount.nxc](#).

8.3.3.586 void ResetScreen () [inline]

Reset LCD screen. This function lets you restore the standard NXT running program screen.

Examples:

[ex_ResetScreen.nxc](#).

8.3.3.587 void ResetSensor (const byte & *port*) [inline]

Reset the sensor port. Sets the invalid data flag on the specified port and waits for it to become valid again. After changing the type or the mode of a sensor port you must call this function to give the firmware time to reconfigure the sensor port.

Parameters:

port The port to reset. See [Input port constants](#).

Examples:

[ex_ResetSensor.nxc](#).

8.3.3.588 long ResetSleepTimer () [inline]

Reset the sleep timer. This function lets you reset the sleep timer.

Returns:

The result of resetting the sleep timer.

Examples:

[ex_ResetSleepTimer.nxc](#).

8.3.3.589 void ResetTachoCount (byte *outputs*) [inline]

Reset tachometer counter. Reset the tachometer count and tachometer limit goal for the specified outputs.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

Examples:

[ex_resettachocount.nxc](#).

8.3.3.590 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](#).

8.3.3.591 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](#).

8.3.3.592 void rewind (byte *handle*) [inline]

Set position indicator to the beginning. Sets the position indicator associated with stream to the beginning of the file.

Parameters:

handle The handle of the file.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_rewind.nxc](#).

8.3.3.593 string RightStr (string str, unsigned int size) [inline]

Copy a portion from the end of a string. Returns the substring of a specified length that appears at the end of a string.

Parameters:

str A string

size The size or length of the substring.

Returns:

The substring of a specified length that appears at the end of a string.

Examples:

[ex_rightstr.nxc](#).

8.3.3.594 void RotateMotor (byte outputs, char pwr, long angle) [inline]

Rotate motor. Run the specified outputs forward for the specified number of degrees.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

angle Angle limit, in degree. Can be negative to reverse direction.

Examples:

[ex_rotatemotor.nxc](#).

8.3.3.595 void RotateMotorEx (byte outputs, char pwr, long angle, char turnpct, bool sync, bool stop) [inline]

Rotate motor. Run the specified outputs forward for the specified number of degrees.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

angle Angle limit, in degree. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

sync Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.

stop Specify whether the motor(s) should brake at the end of the rotation.

Examples:

[ex_rotatemotorex.nxc](#).

8.3.3.596 void RotateMotorExPID (byte *outputs*, char *pwr*, long *angle*, char *turnpct*, bool *sync*, bool *stop*, byte *p*, byte *i*, byte *d*) [inline]

Rotate motor. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

angle Angle limit, in degree. Can be negative to reverse direction.

turnpct Turn ratio, -100 to 100. The direction of your vehicle will depend on its construction.

sync Synchronise two motors. Should be set to true if a non-zero turn percent is specified or no turning will occur.

stop Specify whether the motor(s) should brake at the end of the rotation.

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_rotatemotorexpid.nxc](#).

8.3.3.597 void RotateMotorPID (byte *outputs*, char *pwr*, long *angle*, byte *p*, byte *i*, byte *d*) [inline]

Rotate motor with PID factors. Run the specified outputs forward for the specified number of degrees. Specify proportional, integral, and derivative factors.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). If you use a variable and want to control multiple outputs in a single call you need to use a byte array rather than a byte and store the output port values in the byte array before passing it into this function.

pwr Output power, 0 to 100. Can be negative to reverse direction.

angle Angle limit, in degree. Can be negative to reverse direction.

p Proportional factor used by the firmware's PID motor control algorithm. See [PID constants](#).

i Integral factor used by the firmware's PID motor control algorithm. See [PID constants](#).

d Derivative factor used by the firmware's PID motor control algorithm. See [PID constants](#).

Examples:

[ex_rotatemotorpid.nxc](#).

8.3.3.598 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 SysCommH-SControl 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](#).

8.3.3.599 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](#).

8.3.3.600 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](#).

8.3.3.601 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](#).

8.3.3.602 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](#).

8.3.3.603 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](#).

8.3.3.604 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](#).

8.3.3.605 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](#).

8.3.3.606 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](#).

8.3.3.607 char RunNRLinkMacro (const byte *port*, const byte *i2caddr*, const byte *macro*) [inline]

Run NRLink macro. Run the specified mindsensors NRLink device macro. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

macro The address of the macro to execute.

Returns:

The function call result.

Examples:

[ex_RunNRLLinkMacro.nxc](#).

8.3.3.608 char SendMessage (byte *queue*, string *msg*) [inline]

Send a message to a queue/mailbox. Write a message into a local mailbox.

Parameters:

queue The mailbox number. See [Mailbox constants](#).

msg The message to write to the mailbox.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_sendmessage.nxc](#).

8.3.3.609 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](#).

8.3.3.610 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](#).

8.3.3.611 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](#).

8.3.3.612 char SendResponseBool (byte *queue*, bool *bval*) [inline]

Write a boolean value to a local response mailbox. Write a boolean value to a response mailbox (the mailbox number + 10).

Parameters:

queue The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

bval The boolean value to write.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_SendResponseBool.nxc](#).

8.3.3.613 char SendResponseNumber (byte *queue*, long *val*) [inline]

Write a numeric value to a local response mailbox. Write a numeric value to a response mailbox (the mailbox number + 10).

Parameters:

queue The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

val The numeric value to write.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_SendResponseNumber.nxc](#).

8.3.3.614 char SendResponseString (byte *queue*, string *str*) [inline]

Write a string value to a local response mailbox. Write a string value to a response mailbox (the mailbox number + 10).

Parameters:

queue The mailbox number. See [Mailbox constants](#). This function shifts the specified value into the range of response mailbox numbers by adding 10.

str The string value to write.

Returns:

A char value indicating whether the function call succeeded or not.

Examples:

[ex_SendResponseString.nxc](#).

8.3.3.615 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](#).

8.3.3.616 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](#).

8.3.3.617 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](#).

8.3.3.618 unsigned int Sensor (const byte & *port*) [inline]

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR_1](#)).

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's scaled value.

Examples:

[ex_Sensor.nxc](#), and [ex_SysComputeCalibValue.nxc](#).

8.3.3.619 bool SensorBoolean (const byte *port*) [inline]

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling SetSensorMode.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

Returns:

The sensor's boolean value.

Examples:

[ex_SensorBoolean.nxc](#).

8.3.3.620 byte SensorDigiPinsDirection (const byte *port*) [inline]

Read sensor digital pins direction. Return the digital pins direction value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

Returns:

The sensor's digital pins direction.

Examples:

[ex_SensorDigiPinsDirection.nxc](#).

8.3.3.621 byte SensorDigiPinsOutputLevel (const byte *port*) [inline]

Read sensor digital pins output level. Return the digital pins output level value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

Returns:

The sensor's digital pins output level.

Examples:

[ex_SensorDigiPinsOutputLevel.nxc](#).

8.3.3.622 byte SensorDigiPinsStatus (const byte *port*) [inline]

Read sensor digital pins status. Return the digital pins status value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

Returns:

The sensor's digital pins status.

Examples:

[ex_SensorDigiPinsStatus.nxc](#).

8.3.3.623 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](#).

8.3.3.624 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](#).

8.3.3.625 int SensorHTEOPD (const byte & *port*) [inline]

Read HiTechnic EOPD sensor. Read the HiTechnic EOPD sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#).

Returns:

The EOPD sensor reading.

Examples:

[ex_SensorHTEOPD.nxc](#).

8.3.3.626 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](#).

8.3.3.627 int SensorHTIRSeeker2ACDir (const byte & *port*) [inline]

Read HiTechnic IRSeeker2 AC direction. Read the AC direction value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

Returns:

The IRSeeker2 AC direction.

Examples:

[ex_SensorHTIRSeeker2ACDir.nxc](#).

8.3.3.628 int SensorHTIRSeeker2Addr (const byte & *port*, const byte *reg*) [inline]

Read HiTechnic IRSeeker2 register. Read a register value from the HiTechnic IR Seeker2 on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

reg The register address. See [HiTechnic IRSeeker2 constants](#).

Returns:

The IRSeeker2 register value.

Examples:

[ex_SensorHTIRSeeker2Addr.nxc](#).

8.3.3.629 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](#).

8.3.3.630 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](#).

8.3.3.631 bool SensorInvalid (const byte & *port*) [inline]

Read sensor invalid data flag. Return the value of the InvalidData flag of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's invalid data flag.

Examples:

[ex_SensorInvalid.nxc](#).

8.3.3.632 byte SensorMode (const byte & *port*) [inline]

Read sensor mode. Return the mode of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's mode. See [Sensor mode constants](#).

Examples:

[ex_SensorMode.nxc](#).

**8.3.3.633 int SensorMSCompass (const byte & *port*, const byte *i2caddr*)
[inline]**

Read mindsensors compass value. Return the Mindsensors Compass sensor value.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

Returns:

The mindsensors compass value

Examples:

[ex_SensorMSCompass.nxc](#).

8.3.3.634 int SensorMSDROD (const byte & *port*) [inline]

Read mindsensors DROD value. Return the Mindsensors DROD sensor value.

Parameters:

port The sensor port. See [Input port constants](#).

Returns:

The mindsensors DROD value

Examples:

[ex_SensorMSDROD.nxc](#).

8.3.3.635 int SensorMSPressure (const byte & *port*) [inline]

Read mindsensors pressure sensor. Read the pressure sensor value of the mindsensors pressure sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#).

Returns:

The pressure reading.

Examples:

[ex_SensorMSPressure.nxc](#).

8.3.3.636 int SensorMSPressureRaw (const byte & *port*) [inline]

Read mindsensors raw pressure value. Return the Mindsensors pressure sensor raw value.

Parameters:

port The sensor port. See [Input port constants](#).

Returns:

The mindsensors raw pressure value

Examples:

[ex_SensorMSPressureRaw.nxc](#).

8.3.3.637 unsigned int SensorNormalized (const byte & *port*) [inline]

Read sensor normalized value. Return the normalized value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's normalized value.

Examples:

[ex_SensorNormalized.nxc](#).

8.3.3.638 unsigned int SensorRaw (const byte & port) [inline]

Read sensor raw value. Return the raw value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's raw value.

Examples:

[ex_SensorRaw.nxc](#).

8.3.3.639 unsigned int SensorScaled (const byte & port) [inline]

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR_1](#)) or the [Sensor](#) function.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's scaled value.

Examples:

[ex_SensorScaled.nxc](#).

8.3.3.640 byte SensorType (const byte & port) [inline]

Read sensor type. Return the type of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's type. See [Sensor type constants](#).

Examples:

[ex_SensorType.nxc](#).

8.3.3.641 byte SensorUS (const byte *port*) [inline]

Read ultrasonic sensor value. Return the ultrasonic sensor distance value. Since an ultrasonic sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The port to which the ultrasonic sensor is attached. See the [Input port constants](#) group. You may use a constant or a variable.

Returns:

The ultrasonic sensor distance value (0..255)

Examples:

[ex_sensorus.nxc](#).

8.3.3.642 unsigned int SensorValue (const byte & *port*) [inline]

Read sensor scaled value. Return the processed sensor reading for a sensor on the specified port. This is the same value that is returned by the sensor value names (e.g. [SENSOR_1](#)) or the [Sensor](#) function.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's scaled value.

Examples:

[ex_SensorValue.nxc](#).

8.3.3.643 bool SensorValueBool (const byte *port*) [inline]

Read sensor boolean value. Return the boolean value of a sensor on the specified port. Boolean conversion is either done based on preset cutoffs, or a slope parameter specified by calling SetSensorMode.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

Returns:

The sensor's boolean value.

Examples:

[ex_SensorValueBool.nxc](#).

8.3.3.644 unsigned int SensorValueRaw (const byte & *port*) [inline]

Read sensor raw value. Return the raw value of a sensor on the specified port.

Parameters:

port The sensor port. See [Input port constants](#). A variable whose value is the desired sensor port may also be used.

Returns:

The sensor's raw value.

Examples:

[ex_SensorValueRaw.nxc](#).

8.3.3.645 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](#).

8.3.3.646 char SetACCLNxSensitivity (const byte *port*, const byte *i2caddr*, byte *slevel*) [inline]

Set ACCL-Nx sensitivity. Reset the mindsensors ACCL-Nx sensor calibration to factory settings. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

slevel The sensitivity level. See [MindSensors ACCL-Nx sensitivity level constants](#).

Returns:

The function call result.

Examples:

[ex_SetACCLNxSensitivity.nxc](#).

8.3.3.647 void SetBatteryState (byte *state*) [inline]

Set battery state. Set battery state information.

Parameters:

state The desired battery state (0..4).

Examples:

[ex_SetBatteryState.nxc](#).

8.3.3.648 void SetBluetoothState (byte *state*) [inline]

Set bluetooth state. Set the Bluetooth state.

Parameters:

state The desired bluetooth state. See [BluetoothState constants](#).

Examples:

[ex_SetBluetoothState.nxc](#).

8.3.3.649 void SetBTInputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set bluetooth input buffer data. Write *cnt* bytes of data to the bluetooth input buffer at *offset*.

Parameters:

offset A constant offset into the input buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetBTInputBuffer.nxc](#).

8.3.3.650 void SetBTInputBufferInPtr (byte *n*) [inline]

Set bluetooth input buffer in-pointer. Set the value of the input buffer in-pointer.

Parameters:

n The new in-pointer value (0..127).

Examples:

[ex_SetBTInputBufferInPtr.nxc](#).

8.3.3.651 void SetBTInputBufferOutPtr (byte *n*) [inline]

Set bluetooth input buffer out-pointer. Set the value of the input buffer out-pointer.

Parameters:

n The new out-pointer value (0..127).

Examples:

[ex_SetBTInputBufferOutPtr.nxc](#).

8.3.3.652 void SetBTOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set bluetooth output buffer data. Write *cnt* bytes of data to the bluetooth output buffer at offset.

Parameters:

offset A constant offset into the output buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetBTOutputBuffer.nxc](#).

8.3.3.653 void SetBTOutputBufferInPtr (byte *n*) [inline]

Set bluetooth output buffer in-pointer. Set the value of the output buffer in-pointer.

Parameters:

n The new in-pointer value (0..127).

Examples:

[ex_SetBTOutputBufferInPtr.nxc](#).

8.3.3.654 void SetBTOutputBufferOutPtr (byte *n*) [inline]

Set bluetooth output buffer out-pointer. Set the value of the output buffer out-pointer.

Parameters:

n The new out-pointer value (0..127).

Examples:

[ex_SetBTOutputBufferOutPtr.nxc](#).

8.3.3.655 void SetButtonLongPressCount (const byte *btn*, const byte *n*) [inline]

Set button long press count. Set the long press count of the specified button.

Parameters:

btn The button number. See [Button name constants](#).

n The new long press count value.

Examples:

[ex_SetButtonLongPressCount.nxc](#).

**8.3.3.656 void SetButtonLongReleaseCount (const byte *btn*, const byte *n*)
[inline]**

Set button long release count. Set the long release count of the specified button.

Parameters:

btn The button number. See [Button name constants](#).
n The new long release count value.

Examples:

[ex_SetButtonLongReleaseCount.nxc](#).

**8.3.3.657 void SetButtonModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Button module IOMap value. Set one of the fields of the Button module IOMap structure to a new value. You provide the offset into the Button module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Button module IOMap structure where the new value should be written. See [Button module IOMAP offsets](#).
value A variable containing the new value to write to the Button module IOMap.

8.3.3.658 void SetButtonPressCount (const byte *btn*, const byte *n*) [inline]

Set button press count. Set the press count of the specified button.

Parameters:

btn The button number. See [Button name constants](#).
n The new press count value.

Examples:

[ex_SetButtonPressCount.nxc](#).

**8.3.3.659 void SetButtonReleaseCount (const byte *btn*, const byte *n*)
[inline]**

Set button release count. Set the release count of the specified button.

Parameters:

btn The button number. See [Button name constants](#).
n The new release count value.

Examples:

[ex_SetButtonReleaseCount.nxc](#).

**8.3.3.660 void SetButtonShortReleaseCount (const byte *btn*, const byte *n*)
[inline]**

Set button short release count. Set the short release count of the specified button.

Parameters:

btn The button number. See [Button name constants](#).
n The new short release count value.

Examples:

[ex_SetButtonShortReleaseCount.nxc](#).

8.3.3.661 void SetButtonState (const byte *btn*, const byte *state*) [inline]

Set button state. Set the state of the specified button.

Parameters:

btn The button to check. See [Button name constants](#).
state The new button state. See [ButtonState constants](#).

Examples:

[ex_SetButtonState.nxc](#).

8.3.3.662 void SetCommandFlags (const byte *cmdFlags*) [inline]

Set command flags. Set the command flags.

Parameters:

cmdFlags The new command flags. See [CommandFlags constants](#).

Examples:

[ex_SetCommandFlags.nxc](#).

8.3.3.663 void SetCommandModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Command module IOMap bytes. Modify one or more bytes of data in the Command module IOMap structure. You provide the offset into the Command module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the data should be written. See [Command module IOMAP offsets](#).

count The number of bytes to write at the specified Command module IOMap offset.

data The byte array containing the data to write to the Command module IOMap.

8.3.3.664 void SetCommandModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Command module IOMap value. Set one of the fields of the Command module IOMap structure to a new value. You provide the offset into the Command module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Command module IOMap structure where the new value should be written. See [Command module IOMAP offsets](#).

value A variable containing the new value to write to the Command module IOMap.

8.3.3.665 void SetCommModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Comm module IOMap bytes. Modify one or more bytes of data in an IOMap structure. You provide the offset into the Comm module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the data should be written. See [Comm module IOMAP offsets](#).

count The number of bytes to write at the specified Comm module IOMap offset.

data The byte array containing the data to write to the Comm module IOMap.

8.3.3.666 void SetCommModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Comm module IOMap value. Set one of the fields of the Comm module IOMap structure to a new value. You provide the offset into the Comm module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Comm module IOMap structure where the new value should be written. See [Comm module IOMAP offsets](#).

value A variable containing the new value to write to the Comm module IOMap.

8.3.3.667 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](#).

8.3.3.668 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](#).

8.3.3.669 void SetCustomSensorZeroOffset (const byte *port*, const int & *zeroOffset*) [inline]

Set custom zero offset. Sets the zero offset value of a custom sensor.

Parameters:

port The sensor port. See [Input port constants](#). Must be a constant.

zeroOffset The new zero offset value.

Examples:

[ex_SetCustomSensorZeroOffset.nxc](#).

8.3.3.670 void SetDisplayContrast (byte *contrast*) [inline]

Set the display contrast. This function lets you set the display contrast setting.

Parameters:

contrast The desired display contrast.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_contrast.nxc](#), and [ex_SetDisplayContrast.nxc](#).

8.3.3.671 void SetDisplayDisplay (unsigned long *dispaddr*) [inline]

Set the display memory address. This function lets you set the current display memory address.

Parameters:

dispaddr The new display memory address.

Examples:

[ex_dispmisc.nxc](#), and [ex_SetDisplayDisplay.nxc](#).

8.3.3.672 void SetDisplayEraseMask (unsigned long *eraseMask*) [inline]

Set the display erase mask. This function lets you set the current display erase mask.

Parameters:

eraseMask The new display erase mask.

Examples:

[ex_dispmisc.nxc](#), and [ex_SetDisplayEraseMask.nxc](#).

8.3.3.673 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_dispmiss.nxc](#), and [ex_SetDisplayFlags.nxc](#).

8.3.3.674 void SetDisplayFont (unsigned long *fontaddr*) [inline]

Set the display font memory address. This function lets you set the current display font memory address.

Parameters:

fontaddr The new display font memory address.

Examples:

[ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_DisplayFont.nxc](#), and [ex_SetDisplayFont.nxc](#).

8.3.3.675 void SetDisplayModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Display module IOMap bytes. Modify one or more bytes of data in the Display module IOMap structure. You provide the offset into the Display module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the data should be written. See [Display module IOMAP offsets](#).

count The number of bytes to write at the specified Display module IOMap offset.

data The byte array containing the data to write to the Display module IOMap.

**8.3.3.676 void SetDisplayModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Display module IOMap value. Set one of the fields of the Display module IOMap structure to a new value. You provide the offset into the Display module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Display module IOMap structure where the new value should be written. See [Display module IOMAP offsets](#).

value A variable containing the new value to write to the Display module IOMap.

8.3.3.677 void SetDisplayNormal (const byte *x*, const byte *line*, unsigned int *cnt*, byte *data*[]) [inline]

Write pixel data to the normal display buffer. Write "cnt" bytes to the normal display memory from the data array. Start writing at the specified x, line coordinate. Each byte of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE_1 through TEXTLINE_8 for the "line" parameter.

Parameters:

x The desired x position where you wish to write pixel data.

line The desired line where you wish to write pixel data.

cnt The number of bytes of pixel data to write.

data The array of bytes from which pixel data is read.

Examples:

[ex_SetDisplayNormal.nxc](#).

8.3.3.678 void SetDisplayPopup (const byte *x*, const byte *line*, unsigned int *cnt*, byte *data*[]) [inline]

Write pixel data to the popup display buffer. Write "cnt" bytes to the popup display memory from the data array. Start writing at the specified x, line coordinate. Each byte

of data is a vertical strip of 8 bits at the desired location. Each bit represents a single pixel on the LCD screen. Use TEXTLINE_1 through TEXTLINE_8 for the "line" parameter.

Parameters:

- x* The desired x position where you wish to write pixel data.
- line* The desired line where you wish to write pixel data.
- cnt* The number of bytes of pixel data to write.
- data* The array of bytes from which pixel data is read.

Examples:

[ex_SetDisplayPopup.nxc](#).

8.3.3.679 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_dispmiss.nxc](#), and [ex_SetDisplayTextLinesCenterFlags.nxc](#).

8.3.3.680 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_dispmiss.nxc](#), and [ex_SetDisplayUpdateMask.nxc](#).

8.3.3.681 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](#).

8.3.3.682 void SetHSInputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set hi-speed port input buffer data. Write *cnt* bytes of data to the hi-speed port input buffer at *offset*.

Parameters:

offset A constant offset into the input buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetHSInputBuffer.nxc](#).

8.3.3.683 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](#).

8.3.3.684 void SetHSInputBufferOutPtr (byte *n*) [inline]

Set hi-speed port input buffer out-pointer. Set the value of the input buffer out-pointer.

Parameters:

n The new out-pointer value (0..127).

Examples:

[ex_SetHSInputBufferOutPtr.nxc](#).

8.3.3.685 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](#).

8.3.3.686 void SetHSOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set hi-speed port output buffer data. Write *cnt* bytes of data to the hi-speed port output buffer at *offset*.

Parameters:

offset A constant offset into the output buffer
cnt The number of bytes to write
data A byte array containing the data to write

Examples:

[ex_SetHSOutputBuffer.nxc](#).

8.3.3.687 void SetHSOutputBufferInPtr (byte *n*) [inline]

Set hi-speed port output buffer in-pointer. Set the value of the output buffer in-pointer.

Parameters:

n The new in-pointer value (0..127).

Examples:

[ex_SetHSOutputBufferInPtr.nxc](#).

8.3.3.688 void SetHSOutputBufferOutPtr (byte *n*) [inline]

Set hi-speed port output buffer out-pointer. Set the value of the output buffer out-pointer.

Parameters:

n The new out-pointer value (0..127).

Examples:

[ex_SetHSOutputBufferOutPtr.nxc](#).

8.3.3.689 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](#).

8.3.3.690 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](#).

8.3.3.691 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](#).

8.3.3.692 char SetHTIRSeeker2Mode (const byte & *port*, const byte *mode*) [inline]

Set HiTechnic IRSeeker2 mode. Set the mode of the HiTechnic IRSeeker2 sensor on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

mode The IRSeeker2 mode. See [HiTechnic IRSeeker2 constants](#).

Returns:

The function call result. [NO_ERR](#) or [Communications specific errors](#).

Examples:

[ex_SetHTIRSeeker2Mode.nxc](#), and [ex_SetSensorBoolean.nxc](#).

**8.3.3.693 void SetInput (const byte & *port*, const int *field*, variant *value*)
[inline]**

Set an input field value. Set the specified field of the sensor on the specified port to the value provided.

Parameters:

port The sensor port. See [Input port constants](#). A constant or a variable may be used (no expressions).

field An input field constant. See [Input field constants](#).

value The new value, which may be any valid expression.

Examples:

[ex_SetInput.nxc](#).

**8.3.3.694 void SetInputModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Input module IOMap value. Set one of the fields of the Input module IOMap structure to a new value. You provide the offset into the Input module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Input module IOMap structure where the new value should be written. See [Input module IOMAP offsets](#).

value A variable containing the new value to write to the Input module IOMap.

**8.3.3.695 void SetIOCtlModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set IOCtl module IOMap value. Set one of the fields of the IOCtl module IOMap structure to a new value. You provide the offset into the IOCtl module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the IOCtl module IOMap structure where the new value should be written. See [IOCtl module IOMAP offsets](#).

value A variable containing the new value to write to the IOCtl module IOMap.

**8.3.3.696 void SetIOMapBytes (string *moduleName*, unsigned int *offset*,
unsigned int *count*, byte *data*[]) [inline]**

Set IOMap bytes by name. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

moduleName The module name of the IOMap to modify. See [NXT firmware module names](#).

offset The number of bytes offset from the start of the IOMap structure where the data should be written

count The number of bytes to write at the specified IOMap offset.

data The byte array containing the data to write to the IOMap

**8.3.3.697 void SetIOMapBytesByID (unsigned long *moduleId*, unsigned int
offset, unsigned int *count*, byte *data*[]) [inline]**

Set IOMap bytes by ID. Modify one or more bytes of data in an IOMap structure. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

moduleId The module ID of the IOMap to modify. See [NXT firmware module IDs](#).
offset The number of bytes offset from the start of the IOMap structure where the data should be written.
count The number of bytes to write at the specified IOMap offset.
data The byte array containing the data to write to the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

8.3.3.698 void SetIOMapValue (string *moduleName*, unsigned int *offset*, variant *value*) [inline]

Set IOMap value by name. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its module name. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

moduleName The module name of the IOMap to modify. See [NXT firmware module names](#).
offset The number of bytes offset from the start of the IOMap structure where the new value should be written
value A variable containing the new value to write to the IOMap

8.3.3.699 void SetIOMapValueByID (unsigned long *moduleId*, unsigned int *offset*, variant *value*) [inline]

Set IOMap value by ID. Set one of the fields of an IOMap structure to a new value. The IOMap structure is specified by its Module ID. You also provide the offset into the IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

moduleId The module ID of the IOMap to modify. See [NXT firmware module IDs](#).

offset The number of bytes offset from the start of the IOMap structure where the new value should be written.

value A variable containing the new value to write to the IOMap.

Warning:

This function requires the enhanced NBC/NXC firmware.

**8.3.3.700 void SetLoaderModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Loader module IOMap value. Set one of the fields of the Loader module IOMap structure to a new value. You provide the offset into the Loader module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Loader module IOMap structure where the new value should be written. See [Loader module IOMAP offsets](#).

value A variable containing the new value to write to the Loader module IOMap.

8.3.3.701 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](#).

8.3.3.702 void SetLowSpeedModuleBytes (unsigned int *offset*, unsigned int *count*, byte *data*[]) [inline]

Set Lowspeed module IOMap bytes. Modify one or more bytes of data in the Lowspeed module IOMap structure. You provide the offset into the Lowspeed module IOMap structure where you want to start writing, the number of bytes to write at that location, and a byte array containing the new data.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the data should be written. See [Low speed module IOMAP offsets](#).

count The number of bytes to write at the specified Lowspeed module IOMap offset.

data The byte array containing the data to write to the Lowspeed module IOMap.

8.3.3.703 void SetLowSpeedModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Lowspeed module IOMap value. Set one of the fields of the Lowspeed module IOMap structure to a new value. You provide the offset into the Lowspeed module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Lowspeed module IOMap structure where the new value should be written. See [Low speed module IOMAP offsets](#).

value A variable containing the new value to write to the Lowspeed module IOMap.

8.3.3.704 void SetMotorPwnFreq (byte *n*) [inline]

Set motor PWN frequency. Set the motor PWN frequency.

Parameters:

n The motor pwn frequency

Examples:[ex_setmotorpwnfreq.nxc.](#)**8.3.3.705 void SetOnBrickProgramPointer (byte *obpStep*) [inline]**

Set on-brick program pointer. Set the current OBP (on-brick program) step.

Parameters:

obpStep The new on-brick program step.

Examples:[ex_SetOnBrickProgramPointer.nxc.](#)**8.3.3.706 void SetOutput (byte *outputs*, byte *field1*, variant *val1*, ..., byte *fieldN*, variant *valN*) [inline]**

Set output fields. Set the specified field of the outputs to the value provided. The field must be a valid output field constant. This function takes a variable number of field/value pairs.

Parameters:

outputs Desired output ports. Can be a constant or a variable, see [Output port constants](#). For multiple outputs at the same time you need to add single output port values into a byte array and pass the array instead of a single numeric value.

field1 The 1st output port field to access, this should be a constant, see [Output field constants](#).

val1 Value to set for the 1st field.

fieldN The Nth output port field to access, this should be a constant, see [Output field constants](#).

valN The value to set for the Nth field.

Examples:[ex_setoutput.nxc.](#)

**8.3.3.707 void SetOutputModuleValue (unsigned int *offset*, variant *value*)
[inline]**

Set Output module IOMap value. Set one of the fields of the Output module IOMap structure to a new value. You provide the offset into the Output module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Output module IOMap structure where the new value should be written. See [Output module IOMAP offsets](#).

value A variable containing the new value to write to the Output module IOMap.

**8.3.3.708 void SetSensor (const byte & *port*, const unsigned int *config*)
[inline]**

Set sensor configuration. Set the type and mode of the given sensor to the specified configuration, which must be a special constant containing both type and mode information.

See also:

[SetSensorType\(\)](#), [SetSensorMode\(\)](#), and [ResetSensor\(\)](#)

Parameters:

port The port to configure. See [Input port constants](#).

config The configuration constant containing both the type and mode. See [Combined sensor type and mode constants](#).

Examples:

[ex_SetSensor.nxc](#).

**8.3.3.709 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.

8.3.3.710 void SetSensorColorBlue (const byte & *port*) [inline]

Configure an NXT 2.0 blue light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in blue light mode. Requires an NXT 2.0 compatible firmware.

Parameters:

port The port to configure. See [Input port constants](#).

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SetSensorColorBlue.nxc](#).

8.3.3.711 void SetSensorColorFull (const byte & *port*) [inline]

Configure an NXT 2.0 full color sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in full color mode. Requires an NXT 2.0 compatible firmware.

Parameters:

port The port to configure. See [Input port constants](#).

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SetSensorColorFull.nxc](#), and [ex_SysColorSensorRead.nxc](#).

8.3.3.712 void SetSensorColorGreen (const byte & *port*) [inline]

Configure an NXT 2.0 green light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in green light mode. Requires an NXT 2.0 compatible firmware.

Parameters:

port The port to configure. See [Input port constants](#).

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SetSensorColorGreen.nxc](#).

8.3.3.713 void SetSensorColorNone (const byte & *port*) [inline]

Configure an NXT 2.0 no light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in no light mode. Requires an NXT 2.0 compatible firmware.

Parameters:

port The port to configure. See [Input port constants](#).

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SetSensorColorNone.nxc](#).

8.3.3.714 void SetSensorColorRed (const byte & *port*) [inline]

Configure an NXT 2.0 red light sensor. Configure the sensor on the specified port as an NXT 2.0 color sensor in red light mode. Requires an NXT 2.0 compatible firmware.

Parameters:

port The port to configure. See [Input port constants](#).

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SetSensorColorRed.nxc](#).

8.3.3.715 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](#).

8.3.3.716 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](#).

**8.3.3.717 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](#).

**8.3.3.718 void SetSensorHTEOPD (const byte & *port*, bool *bStandard*)
[inline]**

Set sensor as HiTechnic EOPD. Configure the sensor on the specified port as a HiTechnic EOPD sensor.

Parameters:

port The sensor port. See [Input port constants](#).

bStandard Configure in standard or long-range mode.

Examples:

[ex_SetSensorHTEOPD.nxc](#).

8.3.3.719 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](#).

**8.3.3.720 void SetSensorLight (const byte & *port*, bool *bActive* = true)
[inline]**

Configure a light sensor. Configure the sensor on the specified port as an NXT light sensor.

Parameters:

port The port to configure. See [Input port constants](#).

bActive A boolean flag indicating whether to configure the port as an active or inactive light sensor. The default value for this optional parameter is true.

Examples:

[ex_SetSensorLight.nxc](#).

**8.3.3.721 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_I2CDeviceInfo.nxc](#),
[ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), and [ex_SetSensorLowspeed.nxc](#).

8.3.3.722 void SetSensorMode (const byte & *port*, byte *mode*) [inline]

Set sensor mode. Set a sensor's mode, which should be one of the predefined sensor mode constants. A slope parameter for boolean conversion, if desired, may be added to the mode. After changing the type or the mode of a sensor port you must call [ResetSensor](#) to give the firmware time to reconfigure the sensor port.

See also:

[SetSensorType\(\)](#), [SetSensor\(\)](#)

Parameters:

port The port to configure. See [Input port constants](#).

mode The desired sensor mode. See [Sensor mode constants](#).

Examples:

[ex_SetSensorMode.nxc](#).

**8.3.3.723 void SetSensorMSDROD (const byte & *port*, bool *bActive*)
[inline]**

Configure a mindsensors DROD sensor. Configure the specified port for a mindsensors DROD sensor.

Parameters:

port The port to configure. See [Input port constants](#).

bActive A flag indicating whether to configure the sensor in active or inactive mode.

Examples:

[ex_SetSensorMSDROD.nxc](#).

8.3.3.724 void SetSensorMSPressure (const byte & *port*) [inline]

Configure a mindsensors pressure sensor. Configure the specified port for a mindsensors pressure sensor.

Parameters:

port The port to configure. See [Input port constants](#).

Examples:

[ex_SetSensorMSPressure.nxc](#).

8.3.3.725 void SetSensorSound (const byte & *port*, bool *bDBScaling* = true) [inline]

Configure a sound sensor. Configure the sensor on the specified port as a sound sensor.

Parameters:

port The port to configure. See [Input port constants](#).

bDBScaling A boolean flag indicating whether to configure the port as a sound sensor with dB or dBA scaling. The default value for this optional parameter is true, meaning dB scaling.

Examples:

[ex_SetSensorSound.nxc](#).

8.3.3.726 void SetSensorTouch (const byte & *port*) [inline]

Configure a touch sensor. Configure the sensor on the specified port as a touch sensor.

Parameters:

port The port to configure. See [Input port constants](#).

Examples:

[ex_ReadSensorHTTouchMultiplexer.nxc](#), and [ex_SetSensorTouch.nxc](#).

8.3.3.727 void SetSensorType (const byte & *port*, byte *type*) [inline]

Set sensor type. Set a sensor's type, which must be one of the predefined sensor type constants. After changing the type or the mode of a sensor port you must call [Reset-Sensor](#) to give the firmware time to reconfigure the sensor port.

See also:

[SetSensorMode\(\)](#), [SetSensor\(\)](#)

Parameters:

port The port to configure. See [Input port constants](#).

type The desired sensor type. See [Sensor type constants](#).

Examples:

[ex_SetSensorType.nxc](#).

8.3.3.728 void SetSleepTime (const byte *n*) [inline]

Set sleep time. Set the NXT sleep timeout value to the specified number of minutes.

Parameters:

n The minutes to wait before sleeping.

See also:

[SetSleepTimeout](#), [SleepTimeout](#)

Examples:

[ex_SetSleepTime.nxc](#).

8.3.3.729 void SetSleepTimeout (const byte *n*) [inline]

Set sleep timeout. Set the NXT sleep timeout value to the specified number of minutes.

Parameters:

n The minutes to wait before sleeping.

Examples:

[ex_SetSleepTimeout.nxc](#).

8.3.3.730 void SetSleepTimer (const byte *n*) [inline]

Set the sleep timer. Set the system sleep timer to the specified number of minutes.

Parameters:

n The minutes left on the timer.

Examples:

[ex_SetSleepTimer.nxc](#).

8.3.3.731 void SetSoundDuration (unsigned int *duration*) [inline]

Set sound duration. Set the sound duration.

See also:

[SoundDuration\(\)](#)

Parameters:

duration The new sound duration

Examples:

[ex_SetSoundDuration.nxc](#).

8.3.3.732 void SetSoundFlags (byte *flags*) [inline]

Set sound module flags. Set the sound module flags. See the [SoundFlags constants group](#).

See also:

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

Parameters:

flags The new sound module flags

Examples:

[ex_SetSoundFlags.nxc](#).

8.3.3.733 void SetSoundFrequency (unsigned int *frequency*) [inline]

Set sound frequency. Set the sound frequency.

See also:

[SoundFrequency\(\)](#)

Parameters:

frequency The new sound frequency

Examples:

[ex_SetSoundFrequency.nxc](#).

8.3.3.734 void SetSoundMode (byte *mode*) [inline]

Set sound mode. Set the sound mode. See the [SoundMode constants](#) group.

See also:

[SoundMode\(\)](#)

Parameters:

mode The new sound mode

Examples:

[ex_SetSoundMode.nxc](#).

8.3.3.735 void SetSoundModuleState (byte *state*) [inline]

Set sound module state. Set the sound module state. See the [SoundState constants](#) group.

See also:

[SoundState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

Parameters:

state The new sound state

Examples:

[ex_SetSoundModuleState.nxc](#).

8.3.3.736 void SetSoundModuleValue (unsigned int *offset*, variant *value*) [inline]

Set Sound module IOMap value. Set one of the fields of the Sound module IOMap structure to a new value. You provide the offset into the Sound module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Sound module IOMap structure where the new value should be written. See [Sound module IOMAP offsets](#).

value A variable containing the new value to write to the Sound module IOMap.

8.3.3.737 void SetSoundSampleRate (unsigned int *sampleRate*) [inline]

Set sample rate. Set the sound sample rate.

See also:

[SoundSampleRate\(\)](#)

Parameters:

sampleRate The new sample rate

Examples:

[ex_SetSoundSampleRate.nxc](#).

8.3.3.738 void SetSoundVolume (byte *volume*) [inline]

Set sound volume. Set the sound volume.

See also:[SoundVolume\(\)](#)**Parameters:**

volume The new volume

Examples:[ex_SetSoundVolume.nxc.](#)**8.3.3.739 void SetUIButton (byte *btn*) [inline]**

Set UI button. Set user interface button information.

Parameters:

btn A user interface button value. See [UIButton constants](#).

Examples:[ex_SetUIButton.nxc.](#)**8.3.3.740 void SetUIModuleValue (unsigned int *offset*, variant *value*) [inline]**

Set Ui module IOMap value. Set one of the fields of the Ui module IOMap structure to a new value. You provide the offset into the Ui module IOMap structure where you want to write the value along with a variable containing the new value.

Parameters:

offset The number of bytes offset from the start of the Ui module IOMap structure where the new value should be written. See [Ui module IOMAP offsets](#).

value A variable containing the new value to write to the Ui module IOMap.

8.3.3.741 void SetUIState (byte *state*) [inline]

Set UI state. Set the user interface state.

Parameters:

state A user interface state value. See [UIState constants](#).

Examples:

[ex_SetUIState.nxc](#).

8.3.3.742 void SetUSBInputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set USB input buffer data. Write *cnt* bytes of data to the USB input buffer at *offset*.

Parameters:

offset A constant offset into the input buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetUSBInputBuffer.nxc](#).

8.3.3.743 void SetUSBInputBufferInPtr (byte *n*) [inline]

Set USB input buffer in-pointer. Set the value of the input buffer in-pointer.

Parameters:

n The new in-pointer value (0..63).

Examples:

[ex_SetUSBInputBufferInPtr.nxc](#).

8.3.3.744 void SetUSBInputBufferOutPtr (byte *n*) [inline]

Set USB input buffer out-pointer. Set the value of the input buffer out-pointer.

Parameters:

n The new out-pointer value (0..63).

Examples:

[ex_SetUSBInputBufferOutPtr.nxc](#).

8.3.3.745 void SetUSBOutputBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set USB output buffer data. Write *cnt* bytes of data to the USB output buffer at *offset*.

Parameters:

offset A constant offset into the output buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetUSBOutputBuffer.nxc](#).

8.3.3.746 void SetUSBOutputBufferInPtr (byte *n*) [inline]

Set USB output buffer in-pointer. Set the value of the output buffer in-pointer.

Parameters:

n The new in-pointer value (0..63).

Examples:

[ex_SetUSBOutputBufferInPtr.nxc](#).

8.3.3.747 void SetUSBOutputBufferOutPtr (byte *n*) [inline]

Set USB output buffer out-pointer. Set the value of the output buffer out-pointer.

Parameters:

n The new out-pointer value (0..63).

Examples:

[ex_SetUSBOutputBufferOutPtr.nxc](#).

8.3.3.748 void SetUSBPollBuffer (const byte *offset*, byte *cnt*, byte *data*[]) [inline]

Set USB poll buffer data. Write *cnt* bytes of data to the USB poll buffer at *offset*.

Parameters:

offset A constant offset into the poll buffer

cnt The number of bytes to write

data A byte array containing the data to write

Examples:

[ex_SetUSBPollBuffer.nxc](#).

8.3.3.749 void SetUSBPollBufferInPtr (byte *n*) [inline]

Set USB poll buffer in-pointer. Set the value of the poll buffer in-pointer.

Parameters:

n The new in-pointer value (0..63).

Examples:

[ex_SetUSBPollBufferInPtr.nxc](#).

8.3.3.750 void SetUSBPollBufferOutPtr (byte *n*) [inline]

Set USB poll buffer out-pointer. Set the value of the poll buffer out-pointer.

Parameters:

n The new out-pointer value (0..63).

Examples:

[ex_SetUSBPollBufferOutPtr.nxc](#).

8.3.3.751 void SetUSBState (byte *usbState*) [inline]

Set USB state. This method sets the value of the USB state.

Parameters:

usbState The USB state.

Examples:

[ex_SetUSBState.nxc](#).

8.3.3.752 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](#).

8.3.3.753 void SetVolume (byte *volume*) [inline]

Set volume. Set the user interface volume level. Valid values are from 0 to 4.

Parameters:

volume The new volume level.

Examples:

[ex_SetVolume.nxc](#).

8.3.3.754 char sign (variant *num*) [inline]

Sign value. Return the sign of the value argument (-1, 0, or 1). Any scalar type can be passed into this function.

Parameters:

num The numeric value for which to calculate its sign value.

Returns:

-1 if the parameter is negative, 0 if the parameter is zero, or 1 if the parameter is positive.

Examples:

[ex_sign.nxc](#).

8.3.3.755 float sin (float *x*) [inline]

Compute sine. Computes the sine of an angle of *x* radians.

Parameters:

x Floating point value representing an angle expressed in radians.

Returns:

Sine of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sin_cos.nxc](#).

8.3.3.756 float sind (float *x*) [inline]

Compute sine (degrees). Computes the sine of an angle of *x* degrees.

Parameters:

x Floating point value representing an angle expressed in degrees.

Returns:

Sine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sind_cosd.nxc](#).

8.3.3.757 float sinh (float x) [inline]

Compute hyperbolic sine. Computes the hyperbolic sine of x, expressed in radians.

Parameters:

x Floating point value.

Returns:

Hyperbolic sine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sinh.nxc](#).

8.3.3.758 float sinhd (float x) [inline]

Compute hyperbolic sine (degrees). Computes the hyperbolic sine of x, expressed in degrees.

Parameters:

x Floating point value.

Returns:

Hyperbolic sine of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

8.3.3.759 void SleepNow () [inline]

Put the brick to sleep immediately. This function lets you immediately put the NXT to sleep. The running program will terminate as a result of this action.

Examples:

[ex_sleepnow.nxc](#).

8.3.3.760 byte SleepTime (void) [inline]

Read sleep time. Return the number of minutes that the NXT will remain on before it automatically shuts down.

Returns:

The sleep time value

See also:

[SleepTimeout](#)

Examples:

[ex_SleepTime.nxc](#).

8.3.3.761 byte SleepTimeout (void) [inline]

Read sleep timeout. Return the number of minutes that the NXT will remain on before it automatically shuts down.

Returns:

The sleep timeout value

Examples:

[ex_SleepTimeout.nxc](#).

8.3.3.762 byte SleepTimer (void) [inline]

Read sleep timer. Return the number of minutes left in the countdown to zero from the original SleepTimeout value. When the SleepTimer value reaches zero the NXT will shutdown.

Returns:

The sleep timer value

Examples:

[ex_SleepTimer.nxc](#).

8.3.3.763 unsigned int SoundDuration () [inline]

Get sound duration. Return the current sound duration.

See also:

[SetSoundDuration\(\)](#)

Returns:

The current sound duration.

Examples:

[ex_SoundDuration.nxc](#).

8.3.3.764 byte SoundFlags () [inline]

Get sound module flags. Return the current sound module flags. See the [SoundFlags constants](#) group.

See also:

[SetSoundFlags\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

Returns:

The current sound module flags.

Examples:

[ex_SoundFlags.nxc](#).

8.3.3.765 unsigned int SoundFrequency () [inline]

Get sound frequency. Return the current sound frequency.

See also:

[SetSoundFrequency\(\)](#)

Returns:

The current sound frequency.

Examples:

[ex_SoundFrequency.nxc](#).

8.3.3.766 byte SoundMode () [inline]

Get sound mode. Return the current sound mode. See the [SoundMode constants](#) group.

See also:

[SetSoundMode\(\)](#)

Returns:

The current sound mode.

Examples:

[ex_SoundMode.nxc](#).

8.3.3.767 unsigned int SoundSampleRate () [inline]

Get sample rate. Return the current sound sample rate.

See also:

[SetSoundSampleRate\(\)](#)

Returns:

The current sound sample rate.

Examples:

[ex_SoundSampleRate.nxc](#).

8.3.3.768 byte SoundState () [inline]

Get sound module state. Return the current sound module state. See the [SoundState constants](#) group.

See also:

[SetSoundModuleState\(\)](#), [SysSoundSetState\(\)](#), [SysSoundGetState\(\)](#)

Returns:

The current sound module state.

Examples:

[ex_SoundState.nxc](#).

8.3.3.769 byte SoundVolume () [inline]

Get volume. Return the current sound volume.

See also:

[SetSoundVolume\(\)](#)

Returns:

The current sound volume.

Examples:

[ex_SoundVolume.nxc](#).

8.3.3.770 void sprintf (string & str, string format, variant value) [inline]

Write formatted data to string. Writes a sequence of data formatted as the format argument specifies to a string. After the format parameter, the function expects one value argument.

Parameters:

str The string to write to.

format A string specifying the desired format.

value A value to be formatted for writing to the string.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sprintf.nxc](#).

8.3.3.771 float sqrt (float x) [inline]

Compute square root. Computes the square root of x.

Parameters:

x Floating point value.

Returns:

Square root of x.

Examples:

[ex_isNAN.nxc](#), [ex_labs.nxc](#), and [ex_sqrt.nxc](#).

8.3.3.772 void StartTask (task *t*) [inline]

Start a task. Start the specified task.

Parameters:

t The task to start.

Examples:

[ex_starttask.nxc](#).

8.3.3.773 void Stop (bool *bvalue*) [inline]

Stop the running program. Stop the running program if *bvalue* is true. This will halt the program completely, so any code following this command will be ignored.

Parameters:

bvalue If this value is true the program will stop executing.

Examples:

[ex_file_system.nxc](#), and [ex_stop.nxc](#).

8.3.3.774 void StopAllTasks () [inline]

Stop all tasks. Stop all currently running tasks. This will halt the program completely, so any code following this command will be ignored.

Examples:

[ex_stopalltasks.nxc](#).

8.3.3.775 byte StopSound () [inline]

Stop sound. Stop playing of the current tone or file.

Returns:

The result

Todo

?

Examples:

[ex_StopSound.nxc](#).

8.3.3.776 void StopTask (task *t*) [inline]

Stop a task. Stop the specified task.

Parameters:

t The task to stop.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_stoptask.nxc](#).

8.3.3.777 string streat (string & *dest*, const string & *src*) [inline]

Concatenate strings. Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a new null-character is appended at the end of the new string formed by the concatenation of both in destination. The destination string is returned.

Parameters:

dest The destination string.

src The string to be appended.

Returns:

The destination string.

Examples:

[ex_strcat.nxc](#).

8.3.3.778 string StrCat (string str1, string str2, string strN) [inline]

Concatenate strings. Return a string which is the result of concatenating all of the string arguments together. This function accepts any number of parameters which may be string variables, constants, or expressions.

Parameters:

- str1* The first string.
- str2* The second string.
- strN* The Nth string.

Returns:

The concatenated string.

Examples:

[ex_GetBrickDataAddress.nxc](#), [ex_StrCatOld.nxc](#), [ex_string.nxc](#), [ex_-StrReplace.nxc](#), and [util_battery_1.nxc](#).

8.3.3.779 int strcmp (const string & str1, const string & str2) [inline]

Compare two strings. Compares the string str1 to the string str2.

Parameters:

- str1* A string to be compared.
- str2* A string to be compared.

Returns:

Returns an integral value indicating the relationship between the strings. A zero value indicates that both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2. A value less than zero indicates the opposite.

Examples:

[ex_strcmp.nxc](#).

8.3.3.780 string strcpy (string & dest, const string & src) [inline]

Copy string. Copies the string pointed by source into the array pointed by destination, including the terminating null character. The destination string is returned.

Parameters:

dest The destination string.
src The string to be appended.

Returns:

The destination string.

Examples:

[ex_strcpy.nxc](#).

8.3.3.781 byte StrIndex (string str, unsigned int idx) [inline]

Extract a character from a string. Return the numeric value of the character in the specified string at the specified index. The input string parameter may be a variable, constant, or expression.

Parameters:

str A string.
idx The index of the character to retrieve.

Returns:

The numeric value of the character at the specified index.

Examples:

[ex_StrIndex.nxc](#), and [ex_string.nxc](#).

8.3.3.782 int strlen (const string & str) [inline]

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string.

Parameters:

str A string.

Returns:

The length of the string.

Examples:

[ex_string.nxc](#), and [ex_strlen.nxc](#).

8.3.3.783 unsigned int StrLen (string *str*) [inline]

Get string length. Return the length of the specified string. The length of a string does not include the null terminator at the end of the string. The input string parameter may be a variable, constant, or expression.

Parameters:

str A string.

Returns:

The length of the string.

Examples:

[ex_string.nxc](#), and [ex_StrLenOld.nxc](#).

8.3.3.784 string strncat (string & *dest*, const string & *src*, unsigned int *num*) [inline]

Append characters from string. Appends the first num characters of source to destination, plus a terminating null-character. If the length of the string in source is less than num, only the content up to the terminating null-character is copied. The destination string is returned.

Parameters:

dest The destination string.

src The string to be appended.

num The maximum number of characters to be appended.

Returns:

The destination string.

Examples:

[ex_strncat.nxc](#).

8.3.3.785 int strncmp (const string & str1, const string & str2, unsigned int num) [inline]

Compare characters of two strings. Compares up to num characters of the string str1 to those of the string str2.

Parameters:

str1 A string to be compared.

str2 A string to be compared.

num The maximum number of characters to be compared.

Returns:

Returns an integral value indicating the relationship between the strings. A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2. A value less than zero indicates the opposite.

Examples:

[ex_strncmp.nxc](#).

8.3.3.786 string strnncpy (string & dest, const string & src, unsigned int num) [inline]

Copy characters from string. Copies the first num characters of source to destination. The destination string is returned.

Parameters:

dest The destination string.

src The string to be appended.

num The maximum number of characters to be appended.

Returns:

The destination string.

Examples:

[ex_strncpy.nxc](#).

8.3.3.787 string StrReplace (string *str*, unsigned int *idx*, string *strnew*) [inline]

Replace a portion of a string. Return a string with the part of the string replaced (starting at the specified index) with the contents of the new string value provided in the third argument. The input string parameters may be variables, constants, or expressions.

Parameters:

str A string.

idx The starting point for the replace operation.

strnew The replacement string.

Returns:

The modified string.

Examples:

[ex_string.nxc](#), and [ex_StrReplace.nxc](#).

8.3.3.788 void StrToByteArray (string *str*, byte & *data*[]) [inline]

Convert a string to a byte array. Convert the specified string to an array of byte by removing the null terminator at the end of the string. The output array variable must be a one-dimensional array of byte.

See also:

[ByteArrayToStr](#), [ByteArrayToStrEx](#)

Parameters:

str A string

data A byte array reference which, on output, will contain str without its null terminator.

Examples:

[ex_string.nxc](#), and [ex_StrToByteArray.nxc](#).

8.3.3.789 float strtod (const string & str, string & endptr) [inline]

Convert string to float. Parses the string str interpreting its content as a floating point number and returns its value as a float.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax resembling that of floating point literals, and interprets them as a numerical value. A string containing the rest of the string after the last valid character is stored in endptr.

A valid floating point number for atof is formed by a succession of:

- An optional plus or minus sign
- A sequence of digits, optionally containing a decimal-point character
- An optional exponent part, which itself consists on an 'e' or 'E' character followed by an optional sign and a sequence of digits.

If the first sequence of non-whitespace characters in str does not form a valid floating-point number as just defined, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String beginning with the representation of a floating-point number.

endptr Reference to a string, whose value is set by the function to the remaining characters in str after the numerical value.

Returns:

On success, the function returns the converted floating point number as a float value. If no valid conversion could be performed a zero value (0.0) is returned.

Examples:

[ex_strtod.nxc](#).

**8.3.3.790 long strtol (const string & str, string & endptr, int base = 10)
[inline]**

Convert string to long integer. Parses the C string str interpreting its content as an integral number of the specified base, which is returned as a long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in str is stored in endptr.

If the first sequence of non-whitespace characters in str does not form a valid integral number, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String beginning with the representation of an integral number.

endptr Reference to a string, whose value is set by the function to the remaining characters in str after the numerical value.

base Optional and ignored if specified.

Returns:

On success, the function returns the converted integral number as a long int value. If no valid conversion could be performed a zero value is returned.

Warning:

Only base = 10 is currently supported.

Examples:

[ex_strtol.nxc](#).

8.3.3.791 variant StrToNum (string str) [inline]

Convert string to number. Return the numeric value specified by the string passed to the function. If the content of the string is not a numeric value then this function returns zero. The input string parameter may be a variable, constant, or expression.

Parameters:

str String beginning with the representation of a number.

str A string.

Returns:

A number.

Examples:

[ex_string.nxc](#), and [ex_StrToNum.nxc](#).

8.3.3.792 long strtoul (const string & str, string & endptr, int base = 10) [inline]

Convert string to unsigned long integer. Parses the C string str interpreting its content as an unsigned integral number of the specified base, which is returned as an unsigned long int value.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. A string containing the rest of the characters following the integer representation in str is stored in endptr.

If the first sequence of non-whitespace characters in str does not form a valid integral number, or if no such sequence exists because either str is empty or contains only whitespace characters, no conversion is performed.

Parameters:

str String containing the representation of an unsigned integral number.

endptr Reference to a string, whose value is set by the function to the remaining characters in str after the numerical value.

base Optional and ignored if specified.

Returns:

On success, the function returns the converted integral number as an unsigned long int value. If no valid conversion could be performed a zero value is returned.

Warning:

Only base = 10 is currently supported.

Examples:

[ex_strtoul.nxc](#).

**8.3.3.793 string SubStr (string *str*, unsigned int *idx*, unsigned int *len*)
[inline]**

Extract a portion of a string. Return a sub-string from the specified input string starting at *idx* and including the specified number of characters. The input string parameter may be a variable, constant, or expression.

Parameters:

str A string.
idx The starting point of the sub-string.
len The length of the sub-string.

Returns:

The sub-string extracted from parameter str.

Examples:

[ex_StrCatOld.nxc](#), [ex_string.nxc](#), and [ex_SubStr.nxc](#).

8.3.3.794 void SysCall (byte *funcID*, variant & *args*) [inline]

Call any system function. This generic macro can be used to call any system function. No type checking is performed so you need to make sure you use the correct structure type given the selected system function ID. This is, however, the fastest possible way to call a system function in NXC.

Valid function ID constants are defined in the [System Call function constants](#) group.

Parameters:

funcID The function ID constant corresponding to the function to be called.
args The structure containing the needed parameters.

Examples:

[ex_dispgout.nxc](#), and [ex_syscall.nxc](#).

**8.3.3.795 void SysColorSensorRead (ColorSensorReadType & *args*)
[inline]**

Read LEGO color sensor. This function lets you read the LEGO color sensor given the parameters you pass in via the [ColorSensorReadType](#) structure.

Parameters:

args The [ColorSensorReadType](#) structure containing the required parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysColorSensorRead.nxc](#).

8.3.3.796 void SysCommBTCheckStatus (CommBTCheckStatusType & *args*)

Check Bluetooth connection status. This function lets you check the status of a Bluetooth connection using the values specified via the [CommBTCheckStatusType](#) structure.

Parameters:

args The [CommBTCheckStatusType](#) structure containing the needed parameters.

Examples:

[ex_syscommbtcheckstatus.nxc](#).

**8.3.3.797 void SysCommBTConnection (CommBTConnectionType & *args*)
[inline]**

Connect or disconnect a bluetooth device. This function lets you connect or disconnect a bluetooth device using the values specified via the [CommBTConnectionType](#) structure.

Parameters:

args The [CommBTConnectionType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysCommBTConnection.nxc](#).

8.3.3.798 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](#).

8.3.3.799 void SysCommBTWrite (CommBTWriteType & args)

Write data to a Bluetooth connection. This function lets you write to a Bluetooth connection using the values specified via the [CommBTWriteType](#) structure.

Parameters:

args The [CommBTWriteType](#) structure containing the needed parameters.

Examples:

[ex_syscommbtwrite.nxc](#).

8.3.3.800 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_syscommexecufunction.nxc](#).

8.3.3.801 void SysCommHSCheckStatus (CommHSCheckStatusType & args) [inline]

Check the hi-speed port status. This function lets you check the hi-speed port status using the values specified via the [CommHSCheckStatusType](#) structure.

Parameters:

args The [CommHSCheckStatusType](#) structure containing the needed parameters.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_SysCommHSCheckStatus.nxc](#).

8.3.3.802 void SysCommHSControl (CommHSControlType & args) [inline]

Control the hi-speed port. This function lets you control the hi-speed port using the values specified via the [CommHSControlType](#) structure.

Parameters:

args The [CommHSControlType](#) structure containing the needed parameters.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_SysCommHSControl.nxc](#).

**8.3.3.803 void SysCommHSRead (CommHSReadWriteType & *args*)
[inline]**

Read from the hi-speed port. This function lets you read from the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

Parameters:

args The [CommHSReadWriteType](#) structure containing the needed parameters.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_SysCommHSRead.nxc](#).

**8.3.3.804 void SysCommHSWrite (CommHSReadWriteType & *args*)
[inline]**

Write to the hi-speed port. This function lets you write to the hi-speed port using the values specified via the [CommHSReadWriteType](#) structure.

Parameters:

args The [CommHSReadWriteType](#) structure containing the needed parameters.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_SysCommHSWrite.nxc](#).

8.3.3.805 void SysCommLSCheckStatus (CommLSCheckStatusType & args) [inline]

Check Lowspeed sensor status. This function lets you check the status of an I2C (Lowspeed) sensor transaction using the values specified via the [CommLSCheckStatusType](#) structure.

Parameters:

args The [CommLSCheckStatusType](#) structure containing the needed parameters.

Examples:

[ex_syscommlscheckstatus.nxc](#).

8.3.3.806 void SysCommLSRead (CommLSReadType & args) [inline]

Read from a Lowspeed sensor. This function lets you read from an I2C (Lowspeed) sensor using the values specified via the [CommLSReadType](#) structure.

Parameters:

args The [CommLSReadType](#) structure containing the needed parameters.

Examples:

[ex_syscommlsread.nxc](#).

8.3.3.807 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 [CommLSSWriteType](#) structure containing the needed parameters.

Examples:

[ex_syscommlsswrite.nxc](#).

**8.3.3.808 void SysCommLSSWriteEx (CommLSSWriteExType & *args*)
[inline]**

Write to a Lowspeed sensor (extra). This function lets you write to an I2C (Lowspeed) sensor using the values specified via the [CommLSSWriteExType](#) structure. This is the same as the SysCommLSSWrite 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 [CommLSSWriteExType](#) structure containing the desired parameters.

Examples:

[ex_syscommlsswriteex.nxc](#).

**8.3.3.809 void SysComputeCalibValue (ComputeCalibValueType & *args*)
[inline]**

Compute calibration values. This function lets you compute calibration values using the values specified via the [ComputeCalibValueType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [ComputeCalibValueType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysComputeCalibValue.nxc](#).

8.3.3.810 void SysDatalogGetTimes (DatalogGetTimesType & *args*) [inline]

Get datalog times. This function lets you get datalog times using the values specified via the [DatalogGetTimesType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [DatalogGetTimesType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysDatalogGetTimes.nxc](#).

8.3.3.811 void SysDatalogWrite (DatalogWriteType & *args*) [inline]

Write to the datalog. This function lets you write to the datalog using the values specified via the [DatalogWriteType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [DatalogWriteType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysDatalogWrite.nxc](#).

8.3.3.812 void SysDisplayExecuteFunction (DisplayExecuteFunctionType & args) [inline]

Execute any Display module command. This function lets you directly execute the Display module's primary drawing function using the values specified via the [DisplayExecuteFunctionType](#) structure.

Parameters:

args The [DisplayExecuteFunctionType](#) structure containing the drawing parameters.

Examples:

[ex_dispfunc.nxc](#), and [ex_sysdisplayexecutefunction.nxc](#).

8.3.3.813 void SysDrawCircle (DrawCircleType & args) [inline]

Draw a circle. This function lets you draw a circle on the NXT LCD given the parameters you pass in via the [DrawCircleType](#) structure.

Parameters:

args The [DrawCircleType](#) structure containing the drawing parameters.

Examples:

[ex_sysdrawcircle.nxc](#).

8.3.3.814 void SysDrawEllipse (DrawEllipseType & args) [inline]

Draw an ellipse. This function lets you draw an ellipse on the NXT LCD given the parameters you pass in via the [DrawEllipseType](#) structure.

Parameters:

args The [DrawEllipseType](#) structure containing the drawing parameters.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_sysdrawellipse.nxc](#).

8.3.3.815 void SysDrawFont (DrawFontType & args) [inline]

Draw text using a custom font. This function lets you draw text on the NXT LCD using a custom font with parameters you pass in via the [DrawFontType](#) structure.

Parameters:

args The [DrawFontType](#) structure containing the drawing parameters.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_dispftout.nxc](#), and [ex_sysdrawfont.nxc](#).

8.3.3.816 void SysDrawGraphic (DrawGraphicType & args) [inline]

Draw a graphic (RIC file). This function lets you draw a graphic image (RIC file) on the NXT LCD given the parameters you pass in via the [DrawGraphicType](#) structure.

Parameters:

args The [DrawGraphicType](#) structure containing the drawing parameters.

Examples:

[ex_sysdrawgraphic.nxc](#).

8.3.3.817 void SysDrawGraphicArray (DrawGraphicArrayType & args) [inline]

Draw a graphic image from a byte array. This function lets you draw a graphic image on the NXT LCD given the parameters you pass in via the [DrawGraphicArrayType](#) structure.

Parameters:

args The `DrawGraphicArrayType` structure containing the drawing parameters.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_sysdrawgraphicarray.nxc](#).

8.3.3.818 void SysDrawLine (DrawLineType & *args*) [inline]

Draw a line. This function lets you draw a line on the NXT LCD given the parameters you pass in via the `DrawLineType` structure.

Parameters:

args The `DrawLineType` structure containing the drawing parameters.

Examples:

[ex_sysdrawline.nxc](#).

8.3.3.819 void SysDrawPoint (DrawPointType & *args*) [inline]

Draw a point. This function lets you draw a pixel on the NXT LCD given the parameters you pass in via the `DrawPointType` structure.

Parameters:

args The `DrawPointType` structure containing the drawing parameters.

Examples:

[ex_sysdrawpoint.nxc](#).

8.3.3.820 void SysDrawPolygon (DrawPolygonType & args) [inline]

Draw a polygon. This function lets you draw a polygon on the NXT LCD given the parameters you pass in via the [DrawPolygonType](#) structure.

Parameters:

args The [DrawPolygonType](#) structure containing the drawing parameters.

Warning:

This function requires the enhanced NBC/NXC firmware version 1.28+.

Examples:

[ex_sysdrawpolygon.nxc](#).

8.3.3.821 void SysDrawRect (DrawRectType & args) [inline]

Draw a rectangle. This function lets you draw a rectangle on the NXT LCD given the parameters you pass in via the [DrawRectType](#) structure.

Parameters:

args The [DrawRectType](#) structure containing the drawing parameters.

Examples:

[ex_sysdrawrect.nxc](#).

8.3.3.822 void SysDrawText (DrawTextType & args) [inline]

Draw text. This function lets you draw text on the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

Parameters:

args The [DrawTextType](#) structure containing the drawing parameters.

Examples:

[ex_sysdrawtext.nxc](#).

8.3.3.823 void SysFileClose (FileCloseType & args) [inline]

Close file handle. This function lets you close a file using the values specified via the [FileCloseType](#) structure.

Parameters:

args The [FileCloseType](#) structure containing the needed parameters.

Examples:

[ex_sysfileclose.nxc](#).

8.3.3.824 void SysFileDelete (FileDeleteType & args) [inline]

Delete file. This function lets you delete a file using the values specified via the [FileDeleteType](#) structure.

Parameters:

args The [FileDeleteType](#) structure containing the needed parameters.

Examples:

[ex_sysfiledelete.nxc](#).

8.3.3.825 void SysFileFindFirst (FileFindType & args) [inline]

Start finding files. This function lets you begin iterating through files stored on the NXT.

Parameters:

args The [FileFindType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfilefindfirst.nxc](#).

8.3.3.826 void SysFileFindNext (FileFindType & args) [inline]

Continue finding files. This function lets you continue iterating through files stored on the NXT.

Parameters:

args The [FileFindType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfilefindnext.nxc](#).

8.3.3.827 void SysFileOpenAppend (FileOpenType & args) [inline]

Open file for writing at end of file. This function lets you open an existing file that you can write to using the values specified via the [FileOpenType](#) structure.

The available length remaining in the file is returned via the Length member.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Examples:

[ex_sysfileopenappend.nxc](#).

8.3.3.828 void SysFileOpenRead (FileOpenType & args) [inline]

Open file for reading. This function lets you open an existing file for reading using the values specified via the [FileOpenType](#) structure.

The number of bytes that can be read from the file is returned via the Length member.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Examples:

[ex_sysfileopenread.nxc](#).

8.3.3.829 void SysFileOpenReadLinear (FileOpenType & args) [inline]

Open linear file for reading. This function lets you open an existing linear file for reading using the values specified via the [FileOpenType](#) structure.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfileopenreadlinear.nxc](#).

8.3.3.830 void SysFileOpenWrite (FileOpenType & args) [inline]

Open and create file for writing. This function lets you create a file that you can write to using the values specified via the [FileOpenType](#) structure.

The desired maximum file capacity in bytes is specified via the Length member.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Examples:

[ex_sysfileopenwrite.nxc](#).

8.3.3.831 void SysFileOpenWriteLinear (FileOpenType & args) [inline]

Open and create linear file for writing. This function lets you create a linear file that you can write to using the values specified via the [FileOpenType](#) structure.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfileopenwritelinear.nxc](#).

8.3.3.832 void SysFileOpenWriteNonLinear ([FileOpenType](#) & *args*) [inline]

Open and create non-linear file for writing. This function lets you create a non-linear linear file that you can write to using the values specified via the [FileOpenType](#) structure.

Parameters:

args The [FileOpenType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfileopenwritenonlinear.nxc](#).

8.3.3.833 void SysFileRead ([FileReadWriteType](#) & *args*) [inline]

Read from file. This function lets you read from a file using the values specified via the [FileReadWriteType](#) structure.

Parameters:

args The [FileReadWriteType](#) structure containing the needed parameters.

Examples:

[ex_sysfileread.nxc](#).

8.3.3.834 void SysFileRename (FileRenameType & args) [inline]

Rename file. This function lets you rename a file using the values specified via the [FileRenameType](#) structure.

Parameters:

args The [FileRenameType](#) structure containing the needed parameters.

Examples:

[ex_sysfilerename.nxc](#).

8.3.3.835 void SysFileResize (FileResizeType & args) [inline]

Resize a file. This function lets you resize a file using the values specified via the [FileResizeType](#) structure.

Parameters:

args The [FileResizeType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware. It has not yet been implemented at the firmware level.

Examples:

[ex_sysfileresize.nxc](#).

8.3.3.836 void SysFileResolveHandle (FileResolveHandleType & args) [inline]

File resolve handle. This function lets you resolve the handle of a file using the values specified via the [FileResolveHandleType](#) structure. This will find a previously opened file handle.

Parameters:

args The [FileResolveHandleType](#) structure containing the needed parameters.

Examples:

[ex_sysfileresolvehandle.nxc](#).

8.3.3.837 void SysFileSeek (FileSeekType & args) [inline]

Seek to file position. This function lets you seek to a specific file position using the values specified via the [FileSeekType](#) structure.

Parameters:

args The [FileSeekType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysfileseek.nxc](#).

8.3.3.838 void SysFileWrite (FileReadWriteType & args) [inline]

File write. This function lets you write to a file using the values specified via the [FileReadWriteType](#) structure.

Parameters:

args The [FileReadWriteType](#) structure containing the needed parameters.

Examples:

[ex_sysfilewrite.nxc](#).

8.3.3.839 void SysGetStartTick (GetStartTickType & args) [inline]

Get start tick. This function lets you obtain the tick value at the time your program began executing via the [GetStartTickType](#) structure.

Parameters:

args The [GetStartTickType](#) structure receiving results.

Examples:

[ex_sysgetstarttick.nxc](#).

8.3.3.840 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_systiomapread.nxc](#).

8.3.3.841 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_relassressof.nxc](#), and [ex_systiomapreadbyid.nxc](#).

8.3.3.842 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_systiomapwrite.nxc](#).

8.3.3.843 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_readdressof.nxc](#), and [ex_systiomapwritebyid.nxc](#).

8.3.3.844 void SysKeepAlive (KeepAliveType & args) [inline]

Keep alive. This function lets you reset the sleep timer via the [KeepAliveType](#) structure.

Parameters:

args The [KeepAliveType](#) structure receiving results.

Examples:

[ex_syskeepalive.nxc](#).

8.3.3.845 void SysListFiles (ListFilesType & args) [inline]

List files. This function lets you retrieve a list of files on the NXT using the values specified via the [ListFilesType](#) structure.

Parameters:

args The [ListFilesType](#) structure containing the needed parameters.

Examples:

[ex_syslistfiles.nxc](#).

8.3.3.846 void SysLoaderExecuteFunction (LoaderExecuteFunctionType & args) [inline]

Execute any Loader module command. This function lets you directly execute the Loader module's primary function using the values specified via the [LoaderExecuteFunctionType](#) structure.

Parameters:

args The [LoaderExecuteFunctionType](#) structure containing the needed parameters.

Warning:

This function requires the extended firmware.

Examples:

[ex_sysloaderexecutefunction.nxc](#).

8.3.3.847 void SysMessageRead (MessageReadType & args)

Read message. This function lets you read a message from a queue (aka mailbox) using the values specified via the [MessageReadType](#) structure.

Parameters:

args The [MessageReadType](#) structure containing the needed parameters.

Examples:

[ex_sysmessageread.nxc](#).

8.3.3.848 void SysMessageWrite (MessageWriteType & *args*)

Write message. This function lets you write a message to a queue (aka mailbox) using the values specified via the [MessageWriteType](#) structure.

Parameters:

args The [MessageWriteType](#) structure containing the needed parameters.

Examples:

[ex_sysmessagewrite.nxc](#).

8.3.3.849 void SysRandomNumber (RandomNumberType & *args*) [inline]

Draw a random number. This function lets you obtain a random number via the [RandomNumberType](#) structure.

Parameters:

args The [RandomNumberType](#) structure receiving results.

Examples:

[ex_sysrandomnumber.nxc](#).

8.3.3.850 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](#).

8.3.3.851 void SysReadSemData (ReadSemDataType & *args*) [inline]

Read semaphore data. This function lets you read global motor semaphore data using the values specified via the [ReadSemDataType](#) structure.

Parameters:

args The [ReadSemDataType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysReadSemData.nxc](#).

8.3.3.852 void SysSetScreenMode (SetScreenModeType & *args*) [inline]

Set the screen mode. This function lets you set the screen mode of the NXT LCD given the parameters you pass in via the [DrawTextType](#) structure.

Parameters:

args The [SetScreenModeType](#) structure containing the screen mode parameters.

Examples:

[ex_syssetscreenmode.nxc](#).

8.3.3.853 void SysSetSleepTimeout (SetSleepTimeoutType & args) [inline]

Set system sleep timeout. This function lets you set the system sleep timeout value given the parameters you pass in via the [SetSleepTimeoutType](#) structure.

Parameters:

args The [SetSleepTimeoutType](#) structure containing the required parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysSetSleepTimeout.nxc](#).

8.3.3.854 void SysSoundGetState (SoundGetStateType & args) [inline]

Get sound state. This function lets you retrieve information about the sound module state via the [SoundGetStateType](#) structure.

Parameters:

args The [SoundGetStateType](#) structure containing the needed parameters.

Examples:

[ex_syssoundgetstate.nxc](#).

8.3.3.855 void SysSoundPlayFile (SoundPlayFileType & args) [inline]

Play sound file. This function lets you play a sound file given the parameters you pass in via the [SoundPlayFileType](#) structure. The sound file can either be an RSO file containing PCM or compressed ADPCM samples or it can be an NXT melody (RMD) file containing frequency and duration values.

Parameters:

args The [SoundPlayFileType](#) structure containing the needed parameters.

Examples:

[ex_syssoundplayfile.nxc](#).

8.3.3.856 void SysSoundPlayTone (SoundPlayToneType & args) [inline]

Play tone. This function lets you play a tone given the parameters you pass in via the [SoundPlayToneType](#) structure.

Parameters:

args The [SoundPlayToneType](#) structure containing the needed parameters.

Examples:

[ex_syssoundplaytone.nxc](#).

8.3.3.857 void SysSoundSetState (SoundSetStateType & args) [inline]

Set sound state. This function lets you set sound module state settings via the [SoundSetStateType](#) structure.

Parameters:

args The [SoundSetStateType](#) structure containing the needed parameters.

Examples:

[ex_syssoundsetstate.nxc](#).

8.3.3.858 void SysUpdateCalibCacheInfo (UpdateCalibCacheInfoType & args) [inline]

Update calibration cache information. This function lets you update calibration cache information using the values specified via the [UpdateCalibCacheInfoType](#) structure.

Todo

figure out what this function is intended for

Parameters:

args The [UpdateCalibCacheInfoType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysUpdateCalibCacheInfo.nxc](#).

8.3.3.859 void SysWriteSemData (WriteSemDataType & *args*) [inline]

Write semaphore data. This function lets you write global motor semaphore data using the values specified via the [WriteSemDataType](#) structure.

Parameters:

args The [WriteSemDataType](#) structure containing the needed parameters.

Warning:

This function requires an NXT 2.0 compatible firmware.

Examples:

[ex_SysWriteSemData.nxc](#).

8.3.3.860 float tan (float *x*) [inline]

Compute tangent. Computes the tangent of an angle of *x* radians.

Parameters:

x Floating point value representing an angle expressed in radians.

Returns:

Tangent of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_tan.nxc](#).

8.3.3.861 float tand (float x) [inline]

Compute tangent (degrees). Computes the tangent of an angle of x degrees.

Parameters:

x Floating point value representing an angle expressed in degrees.

Returns:

Tangent of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_tand.nxc](#).

8.3.3.862 float tanh (float x) [inline]

Compute hyperbolic tangent. Computes the hyperbolic tangent of x, expressed in radians.

Parameters:

x Floating point value.

Returns:

Hyperbolic tangent of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_tanh.nxc](#).

8.3.3.863 float tanhd (float *x*) [inline]

Compute hyperbolic tangent (degrees). Computes the hyperbolic tangent of *x*, expressed in degrees.

Parameters:

x Floating point value.

Returns:

Hyperbolic tangent of *x*.

Warning:

This function requires the enhanced NBC/NXC firmware.

8.3.3.864 char TextOut (int *x*, int *y*, string *str*, unsigned long *options* = DRAW_OPT_NORMAL) [inline]

Draw text. Draw a text value on the screen at the specified *x* and *y* location. The *y* value must be a multiple of 8. Valid line number constants are listed in the [Line number constants](#) group. Optionally specify drawing options. If this argument is not specified it defaults to [DRAW_OPT_NORMAL](#). Valid display option constants are listed in the [Drawing option constants](#) group.

See also:

[SysDrawText](#), [DrawTextType](#)

Parameters:

x The *x* value for the start of the text output.
y The text line number for the text output.
str The text to output to the LCD screen.
options The optional drawing options.

Returns:

The result of the drawing operation.

Examples:

[ex_acos.nxc](#), [ex_acosd.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_asin.nxc](#), [ex_asind.nxc](#), [ex_atan.nxc](#), [ex_atan2.nxc](#), [ex_atan2d.nxc](#), [ex_atand.nxc](#), [ex_ClearLine.nxc](#), [ex_Copy.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_dispgout.nxc](#), [ex_DisplayFont.nxc](#), [ex_file_system.nxc](#), [ex_GetBrickDataAddress.nxc](#), [ex_I2CDeviceId.nxc](#), [ex_I2CDeviceInfo.nxc](#), [ex_I2CVendorId.nxc](#), [ex_I2CVersion.nxc](#), [ex_isNAN.nxc](#), [ex_labs.nxc](#), [ex_leftstr.nxc](#), [ex_midstr.nxc](#), [ex_ReadSensorHTTouchMultiplexer.nxc](#), [ex_readdressof.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_SysCommBTOFF.nxc](#), [ex_SysCommHSCheckStatus.nxc](#), [ex_SysCommHSControl.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_SysComputeCalibValue.nxc](#), [ex_SysDatalogWrite.nxc](#), [ex_sysfilefindfirst.nxc](#), [ex_sysfilefindnext.nxc](#), [ex_sysfileread.nxc](#), [ex_syslistfiles.nxc](#), [ex_sysmessageread.nxc](#), [ex_tan.nxc](#), [ex_tand.nxc](#), [ex_TextOut.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

8.3.3.865 int tolower (int c) [inline]

Convert uppercase letter to lowercase. Converts parameter c to its lowercase equivalent if c is an uppercase letter and has a lowercase equivalent. If no such conversion is possible, the value returned is c unchanged.

Parameters:

c Uppercase letter character to be converted.

Returns:

The lowercase equivalent to c, if such value exists, or c (unchanged) otherwise..

Examples:

[ex_ctype.nxc](#), and [ex_tolower.nxc](#).

8.3.3.866 int toupper (int c) [inline]

Convert lowercase letter to uppercase. Converts parameter c to its uppercase equivalent if c is a lowercase letter and has an uppercase equivalent. If no such conversion is possible, the value returned is c unchanged.

Parameters:

c Lowercase letter character to be converted.

Returns:

The uppercase equivalent to c, if such value exists, or c (unchanged) otherwise..

Examples:

[ex_ctype.nxc](#), and [ex_toupper.nxc](#).

8.3.3.867 long trunc (float x) [inline]

Compute integral part. Computes the integral part of x.

Parameters:

x Floating point value.

Returns:

Integral part of x.

Warning:

This function requires the enhanced NBC/NXC firmware.

Examples:

[ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), and [ex_trunc.nxc](#).

8.3.3.868 byte UIButton (void) [inline]

Read UI button. Return user interface button information.

Returns:

A UI button value. See [UIButton constants](#).

Examples:

[ex_UIButton.nxc](#).

8.3.3.869 byte UIState (void) [inline]

Get UI module state. Return the user interface state.

Returns:

The UI module state. See [UIState constants](#).

Examples:

[ex_UIState.nxc](#).

8.3.3.870 int UnflattenVar (string str, variant & x) [inline]

Unflatten a string into a data type. Convert a string containing the byte representation of the specified variable back into the original variable type.

See also:

[FlattenVar](#), [Flatten](#)

Parameters:

str A string containing flattened data.

x A variable reference where the unflattened data is stored.

Returns:

A boolean value indicating whether the operation succeeded or not.

Examples:

[ex_FlattenVar.nxc](#), [ex_string.nxc](#), and [ex_UnflattenVar.nxc](#).

8.3.3.871 byte USBInputBufferInPtr (void) [inline]

Get usb port input buffer in-pointer. This method returns the value of the input pointer of the usb port input buffer.

Returns:

The USB port input buffer's in-pointer value.

Examples:

[ex_USBInputBufferInPtr.nxc](#).

8.3.3.872 byte USBInputBufferOutPtr (void) [inline]

Get usb port input buffer out-pointer. This method returns the value of the output pointer of the usb port input buffer.

Returns:

The USB port input buffer's out-pointer value.

Examples:

[ex_USBInputBufferOutPtr.nxc](#).

8.3.3.873 byte USBOutputBufferInPtr (void) [inline]

Get usb port output buffer in-pointer. This method returns the value of the input pointer of the usb port output buffer.

Returns:

The USB port output buffer's in-pointer value.

Examples:

[ex_USBOutputBufferInPtr.nxc](#).

8.3.3.874 byte USBOutputBufferOutPtr (void) [inline]

Get usb port output buffer out-pointer. This method returns the value of the output pointer of the usb port output buffer.

Returns:

The USB port output buffer's out-pointer value.

Examples:

[ex_USBOutputBufferOutPtr.nxc](#).

8.3.3.875 byte USBPollBufferInPtr (void) [inline]

Get usb port poll buffer in-pointer. This method returns the value of the input pointer of the usb port poll buffer.

Returns:

The USB port poll buffer's in-pointer value.

Examples:

[ex_USBPollBufferInPtr.nxc](#).

8.3.3.876 byte USBPollBufferOutPtr (void) [inline]

Get usb port poll buffer out-pointer. This method returns the value of the output pointer of the usb port poll buffer.

Returns:

The USB port poll buffer's out-pointer value.

Examples:

[ex_USBPollBufferOutPtr.nxc](#), and [ex_USBState.nxc](#).

8.3.3.877 byte UsbState (void) [inline]

Get UI module USB state. This method returns the UI module USB state.

Returns:

The UI module USB state. (0=disconnected, 1=connected, 2=working)

Examples:

[ex_UiUsbState.nxc](#).

8.3.3.878 byte USBState (void) [inline]

Get USB state. This method returns the value of the USB state.

Returns:

The USB state.

8.3.3.879 byte VMRunState (void) [inline]

Read VM run state. Return VM run state information.

Returns:

VM run state. See [VM run state constants](#).

Examples:

[ex_VMRunState.nxc](#).

8.3.3.880 byte Volume (void) [inline]

Read volume. Return the user interface volume level. Valid values are from 0 to 4.

Returns:

The UI module volume. (0..4)

Examples:

[ex_Volume.nxc](#).

8.3.3.881 void Wait (unsigned long *ms*) [inline]

Wait some milliseconds. Make a task sleep for specified amount of time (in 1000ths of a second).

Parameters:

ms The number of milliseconds to sleep.

Examples:

[alternating_tasks.nxc](#), [ex_addressof.nxc](#), [ex_addressofex.nxc](#), [ex_ArrayMax.nxc](#), [ex_ArrayMean.nxc](#), [ex_ArrayMin.nxc](#), [ex_ArrayOp.nxc](#), [ex_ArraySort.nxc](#), [ex_ArrayStd.nxc](#), [ex_ArraySum.nxc](#), [ex_ArraySumSqr.nxc](#), [ex_atof.nxc](#), [ex_atoi.nxc](#), [ex_atol.nxc](#), [ex_CircleOut.nxc](#), [ex_ClearLine.nxc](#), [ex_ClearScreen.nxc](#), [ex_contrast.nxc](#), [ex_Copy.nxc](#), [ex_ctype.nxc](#), [ex_delete_data_file.nxc](#), [ex_dispfout.nxc](#), [ex_dispfunc.nxc](#), [ex_dispgaout.nxc](#), [ex_dispgout.nxc](#), [ex_dispgoutex.nxc](#), [ex_DisplayFont.nxc](#), [ex_dispmisc.nxc](#), [ex_div.nxc](#), [ex_file_system.nxc](#), [ex_FlattenVar.nxc](#), [ex_getchar.nxc](#), [ex_I2CDeviceInfo.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_reladdressof.nxc](#), [ex_rightstr.nxc](#), [ex_SensorHTGyro.nxc](#), [ex_SetDisplayFont.nxc](#), [ex_sin_cos.nxc](#), [ex_sind_cosd.nxc](#), [ex_StrCatOld.nxc](#), [ex_StrIndex.nxc](#), [ex_string.nxc](#), [ex_StrLenOld.nxc](#), [ex_StrReplace.nxc](#), [ex strtod.nxc](#), [ex strtol.nxc](#), [ex strtoul.nxc](#), [ex_SubStr.nxc](#), [ex_SysCommBTConnection.nxc](#), [ex_SysCommHSControl.nxc](#), [ex_SysCommHSRead.nxc](#), [ex_SysDatalogGetTimes.nxc](#), [ex_sysdrawfont.nxc](#), [ex_sysdrawgraphicarray.nxc](#), [ex_sysdrawpolygon.nxc](#), [ex_syslistfiles.nxc](#), [ex_UnflattenVar.nxc](#), [ex_wait.nxc](#), [ex_yield.nxc](#), [glBoxDemo.nxc](#), [glScaleDemo.nxc](#), [util_battery_1.nxc](#), [util_battery_2.nxc](#), and [util_rpm.nxc](#).

8.3.3.882 unsigned int Write (byte *handle*, const variant & *value*) [inline]

Write value to file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written.

Parameters:

handle The file handle.

value The value to write to the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_file_system.nxc](#), and [ex_Write.nxc](#).

8.3.3.883 unsigned int WriteBytes (byte *handle*, const byte & *buf*[], unsigned int & *cnt*) [inline]

Write bytes to file. Write the contents of the data array to the file associated with the specified handle. The handle parameter must be a variable. The cnt parameter must be a variable. The data parameter must be a byte array. The actual number of bytes written is returned in the cnt parameter.

Parameters:

- handle* The file handle.
- buf* The byte array or string containing the data to write.
- cnt* The number of bytes actually written to the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_WriteBytes.nxc](#).

8.3.3.884 unsigned int WriteBytesEx (byte *handle*, unsigned int & *len*, const byte & *buf*[]) [inline]

Write bytes to a file with limit. Write the specified number of bytes to the file associated with the specified handle. The handle parameter must be a variable. The len parameter must be a variable. The buf parameter must be a byte array or a string variable or string constant. The actual number of bytes written is returned in the len parameter.

Parameters:

- handle* The file handle.
- len* The maximum number of bytes to write on input. Returns the actual number of bytes written.
- buf* The byte array or string containing the data to write.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_WriteBytesEx.nxc](#).

**8.3.3.885 char WriteI2CRegister (byte *port*, byte *i2caddr*, byte *reg*, byte *val*)
[inline]**

Write I2C register. Write a single byte to an I2C device register.

Parameters:

- port* The port to which the I2C device is attached. See the [Input port constants](#) group. You may use a constant or a variable.
i2caddr The I2C device address.
reg The I2C device register to which to write a single byte.
val The byte to write to the I2C device.

Returns:

A status code indicating whether the write completed successfully or not. See [CommLSCheckStatusType](#) for possible result values.

Examples:

[ex_writei2cregister.nxc](#).

**8.3.3.886 unsigned int WriteLn (byte *handle*, const variant & *value*)
[inline]**

Write a value and new line to a file. Write a value to the file associated with the specified handle. The handle parameter must be a variable. The value parameter must be a constant, a constant expression, or a variable. The type of the value parameter determines the number of bytes of data written. This function also writes a carriage return and a line feed to the file following the numeric data.

Parameters:

- handle* The file handle.
value The value to write to the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_Writeln.nxc](#).

**8.3.3.887 `unsigned int WriteLnString (byte handle, const string & str,
unsigned int & cnt) [inline]`**

Write string and new line to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. This function also writes a carriage return and a line feed to the file following the string data. The total number of bytes written is returned in the cnt parameter.

Parameters:

handle The file handle.

str The string to write to the file.

cnt The number of bytes actually written to the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_WriteLnString.nxc](#).

**8.3.3.888 `char WriteNRLinkBytes (const byte port, const byte i2caddr, const
byte data[]) [inline]`**

Write data to NRLink. Write data to the mindsensors NRLink device on the specified port. The port must be configured as a Lowspeed port before using this function.

Parameters:

port The sensor port. See [Input port constants](#).

i2caddr The sensor I2C address. See sensor documentation for this value.

data A byte array containing the data to write.

Returns:

The function call result.

Examples:

[ex_WriteNRLinkBytes.nxc](#).

8.3.3.889 `unsigned int WriteString (byte handle, const string & str, unsigned int & cnt) [inline]`

Write string to a file. Write the string to the file associated with the specified handle. The handle parameter must be a variable. The count parameter must be a variable. The str parameter must be a string variable or string constant. The actual number of bytes written is returned in the cnt parameter.

Parameters:

handle The file handle.

str The string to write to the file.

cnt The number of bytes actually written to the file.

Returns:

The function call result. See [Loader module error codes](#).

Examples:

[ex_WriteString.nxc](#).

8.3.3.890 `void Yield () [inline]`

Yield to another task. Make a task yield to another concurrently running task.

Examples:

[ex_yield.nxc](#).

9 Example Documentation

9.1 alternating_tasks.nxc

This is an example of how to use the [ExitTo](#) function.

```
// When run, this program alternates between task A and task B until halted
// by pressing the gray button.

task B();
```

```
void beep(const int tone)
{
    PlayTone(tone, MS_500);
    Wait(SEC_1);
}

task A()
{
    beep(TONE_C4);
    ExitTo(B);
}

task B()
{
    beep(TONE_C6);
    ExitTo(A);
}

task main()
{
    // ExitTo(B) would work as well here.
    Precedes(B);
}
```

9.2 ex_abort.nxc

This is an example of how to use the [abort](#) function.

```
abort(); // stop the program
```

9.3 ex_AbortFlag.nxc

This is an example of how to use the [AbortFlag](#) function.

```
byte af = AbortFlag();
```

9.4 ex_abs.nxc

This is an example of how to use the [abs](#) function.

```
float val = abs(x); // return the absolute value of x
```

9.5 ex_ACCLNxCalibrateX.nxc

This is an example of how to use the [ACCLNxCalibrateX](#) function.

```
result = ACCLNxCalibrateX(S1, MS_ADDR_ACCLNX);
```

9.6 ex_ACCLNxCalibrateXEnd.nxc

This is an example of how to use the [ACCLNxCalibrateXEnd](#) function.

```
result = ACCLNxCalibrateXEnd(S1, MS_ADDR_ACCLNX);
```

9.7 ex_ACCLNxCalibrateY.nxc

This is an example of how to use the [ACCLNxCalibrateY](#) function.

```
result = ACCLNxCalibrateY(S1, MS_ADDR_ACCLNX);
```

9.8 ex_ACCLNxCalibrateYEnd.nxc

This is an example of how to use the [ACCLNxCalibrateYEnd](#) function.

```
result = ACCLNxCalibrateYEnd(S1, MS_ADDR_ACCLNX);
```

9.9 ex_ACCLNxCalibrateZ.nxc

This is an example of how to use the [ACCLNxCalibrateZ](#) function.

```
result = ACCLNxCalibrateZ(S1, MS_ADDR_ACCLNX);
```

9.10 ex_ACCLNxCalibrateZEnd.nxc

This is an example of how to use the [ACCLNxCalibrateZEnd](#) function.

```
result = ACCLNxCalibrateZEnd(S1, MS_ADDR_ACCLNX);
```

9.11 ex_ACCLNxResetCalibration.nxc

This is an example of how to use the [ACCLNxResetCalibration](#) function.

```
result = ACCLNxResetCalibration(S1, MS_ADDR_ACCLNX);
```

9.12 ex_ACCLNxSensitivity.nxc

This is an example of how to use the [ACCLNxSensitivity](#) function.

```
result = ACCLNxSensitivity(S1, MS_ADDR_ACCLNX);
```

9.13 ex_ACCLNxXOffset.nxc

This is an example of how to use the [ACCLNxXOffset](#) function.

```
result = ACCLNxXOffset(S1, MS_ADDR_ACCLNX);
```

9.14 ex_ACCLNxXRange.nxc

This is an example of how to use the [ACCLNxXRange](#) function.

```
result = ACCLNxXRange(S1, MS_ADDR_ACCLNX);
```

9.15 ex_ACCLNxYOffset.nxc

This is an example of how to use the [ACCLNxYOffset](#) function.

```
result = ACCLNxYOffset(S1, MS_ADDR_ACCLNX);
```

9.16 ex_ACCLNxYRange.nxc

This is an example of how to use the [ACCLNxYRange](#) function.

```
result = ACCLNxYRange(S1, MS_ADDR_ACCLNX);
```

9.17 ex_ACCLNxZOffset.nxc

This is an example of how to use the [ACCLNxZOffset](#) function.

```
result = ACCLNxZOffset(S1, MS_ADDR_ACCLNX);
```

9.18 ex_ACCLNxZRange.nxc

This is an example of how to use the [ACCLNxZRange](#) function.

```
result = ACCLNxZRange(S1, MS_ADDR_ACCLNX);
```

9.19 ex_acos.nxc

This is an example of how to use the `acos` function.

```
// ex_acos.nxc
// Display values of the acos API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_acos(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%7.4f RAD", acos(val)));
}

task main()
{
    show_acos(MIN_VAL, LCD_LINE1); // shows 3.1416 RAD
    show_acos(MID_VAL, LCD_LINE2); // shows 1.5708 RAD
    show_acos(MAX_VAL, LCD_LINE3); // shows 0.0000 RAD
    // An invalid value returns not-a-number (nan).
    show_acos(INVALID, LCD_LINE4); // shows -nan RAD
    while (true);
}
```

9.20 ex_acosd.nxc

This is an example of how to use the `acosd` function.

```
// ex_acosd.nxc
// Display values of the acosd API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_acos(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", acosd(val)));
}

task main()
{
    show_acos(MIN_VAL, LCD_LINE1); // shows 180.00 DEG
    show_acos(MID_VAL, LCD_LINE2); // shows 90.00 DEG
    show_acos(MAX_VAL, LCD_LINE3); // shows 0.00 DEG
    // An invalid value returns not-a-number (nan).
```

```

    show_acos(INVALID, LCD_LINE4); // shows -nan DEG
    while (true);
}

```

9.21 ex_Acquire.nxc

This is an example of how to use the [Acquire](#) function.

```

mutex motorMutex;
// ...
Acquire(motorMutex); // make sure we have exclusive access
// use the motors
Release(motorMutex);

```

9.22 ex_addressof.nxc

This is an example of how to use the [addressOf](#) function.

```

const byte NewFont[] =
{
    0x04,0x00, // Graphics Format
    0x02,0x40, // Graphics DataSize
    0x10, // Graphics Count X
    0x06, // Graphics Count Y
    0x06, // Graphics Width
    0x08, // Graphics Height
    0x00,0x00,0x00,0x00,0x00,0x06,0x5F,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,0x41,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x
    08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x00,0x08,0x08,0x0
    8,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,
    0x3E,0x51,0x49,0x45,0x3E,0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x62,0x51,0x49,0x49
        ,0x46,0x00,0x22,0x49,0x49,0x49,0x36,0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x2F,0x49,
    0x49,0x49,0x31,0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
    x36,0x49,0x49,0x49,0x36,0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00,0x6C,0x6C,0x00,0x
    00,0x00,0xEC,0x6C,0x00,0x00,0x08,0x14,0x22,0x41,0x00,0x24,0x24,0x2
    4,0x24,0x24,0x00,0x41,0x22,0x14,0x08,0x00,0x02,0x01,0x59,0x09,0x06,0x00,
    0x3E,0x41,0x5D,0x55,0x1E,0x00,0x7E,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,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
    x7F,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,0x09,0x09,0x06,0x00,0x3E,0x41,0x21,0x5E,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,0x09,0x09,0x00,0x18,0x24,0xA4,0xA4,0xFC,0x00,0
        x7F,0x04,0x04,0x78,0x00,0x00,0x00,0x7D,0x40,0x00,0x00,0x40,0x80,0x84,0x7D,0x
        00,0x00,0x7F,0x10,0x28,0x44,0x00,0x00,0x00,0x7F,0x40,0x00,0x00,0x7C,0x04,0x1
        8,0x04,0x78,0x00,0x7C,0x04,0x04,0x78,0x00,0x00,0x38,0x44,0x44,0x38,0x00,
        0xFC,0x44,0x44,0x44,0x38,0x00,0x38,0x44,0x44,0x44,0xFC,0x00,0x44,0x78,0x44,0x04
        ,0x08,0x00,0x08,0x54,0x54,0x20,0x00,0x04,0x3E,0x44,0x24,0x00,0x00,0x3C,0x40,
        0x20,0x7C,0x00,0x00,0x1C,0x20,0x40,0x20,0x1C,0x00,0x3C,0x60,0x30,0x60,0x3C,0x00,0
        x6C,0x10,0x10,0x6C,0x00,0x00,0x9C,0xA0,0x60,0x3C,0x00,0x00,0x64,0x54,0x54,0x4C,0x
        00,0x00,0x08,0x3E,0x41,0x41,0x00,0x00,0x00,0x00,0x77,0x00,0x00,0x00,0x00,0x41,0x4
        1,0x3E,0x08,0x00,0x02,0x01,0x02,0x01,0x00,0x00,0x10,0x20,0x40,0x38,0x07,0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    ptr = addressOf(NewFont);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

9.23 ex_addressofex.nxc

This is an example of how to use the `addressOfEx` function.

```

const byte NewFont[] =
{
    0x04,0x00, // Graphics Format
    0x02,0x40, // Graphics DataSize
    0x10,       // Graphics Count X
    0x06,       // Graphics Count Y
    0x06,       // Graphics Width
    0x08,       // Graphics Height
    0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x5F,0x06,0x00,0x00,0x07,0x03,0x00,0x07
        ,0x03,0x00,0x24,0x7E,0x24,0x7E,0x24,0x00,0x24,0x2B,0x6A,0x12,0x00,0x00,0x63,0x13,
        0x08,0x64,0x63,0x00,0x30,0x4C,0x52,0x22,0x50,0x00,0x00,0x07,0x03,0x00,0x00,0x00,0
        x00,0x3E,0x41,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x
        08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x00,0x08,0x08,0x0
        8,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,
    0x3E,0x51,0x49,0x45,0x3E,0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x62,0x51,0x49,0x49
        ,0x46,0x00,0x22,0x49,0x49,0x49,0x36,0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x2F,0x49,
        0x49,0x49,0x31,0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
        x36,0x49,0x49,0x49,0x36,0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00,0x6C,0x6C,0x00,0x
        00,0x00,0xEC,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,0x41,0x41,0x3E,0x00,0x7F,0x49,
        0x49,0x49,0x41,0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
        x7F,0x08,0x08,0x08,0x7F,0x00,0x00,0x41,0x7F,0x41,0x00,0x00,0x30,0x40,0x40,0x40,0x
        3F,0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x7F,0x40,0x40,0x40,0x00,0x7F,0x02,0x00
};

```

```

        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, 0x00, 0x00, 0x00,
0x00, 0x02, 0x05, 0x02, 0x00, 0x20, 0x54, 0x54, 0x54, 0x78, 0x00, 0x7F, 0x44, 0x44, 0x44
    , 0x38, 0x00, 0x38, 0x44, 0x44, 0x44, 0x28, 0x00, 0x38, 0x44, 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, 0x7F, 0x40, 0x00, 0x00, 0x7C, 0x04, 0x1
8, 0x04, 0x78, 0x00, 0x7C, 0x04, 0x04, 0x78, 0x00, 0x00, 0x38, 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, 0x44, 0x44, 0x24, 0x00, 0x00, 0x3C, 0x40,
0x20, 0x7C, 0x00, 0x00, 0x1C, 0x20, 0x40, 0x1C, 0x20, 0x3C, 0x00, 0x30, 0x60, 0x30, 0x60, 0x3C, 0x00, 0
x6C, 0x10, 0x10, 0x6C, 0x00, 0x00, 0x9C, 0xA0, 0x60, 0x3C, 0x00, 0x00, 0x64, 0x54, 0x54, 0x4C, 0x
00, 0x00, 0x08, 0x3E, 0x41, 0x41, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x41, 0x4
1, 0x3E, 0x08, 0x00, 0x02, 0x01, 0x02, 0x01, 0x00, 0x00, 0x10, 0x20, 0x40, 0x38, 0x07, 0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    ptr = addressOfEx(NewFont, false);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

9.24 ex_arraybuild.nxc

This is an example of how to use the [ArrayBuild](#) function.

```

task main()
{
    byte myArray[];
    byte src1 = 0x45, src2 = 0x1f, srcN = 0x7a;

    ArrayBuild(myArray, src1, src2, srcN);
    // myArray = {0x45, 0x1f, 0x7a};

    int abSample[];
    int s1[] = {0, 1, 2, 3};
    int s2 = 4, s3 = 5, s4 = 6, sN[] = {7, 8};
    ArrayBuild(abSample, s1, s2, s3, s4, sN);
    // abSample = {0, 1, 2, 3, 4, 5, 6, 7, 8};
    NumOut(0, LCD_LINE4, myArray[2]);
    NumOut(0, LCD_LINE5, abSample[1]);
}

```

9.25 ex_arrayinit.nxc

This is an example of how to use the [ArrayInit](#) function.

```
ArrayInit(myArray, 0, 10); // 10 elements == zero
```

9.26 ex_arraylen.nxc

This is an example of how to use the [ArrayLen](#) function.

```
x = ArrayLen(myArray);
```

9.27 ex_ArrayMax.nxc

This is an example of how to use the [ArrayMax](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = Random();
    TextOut(0, LCD_LINE1, "Max value = ");
    int x = ArrayMax(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.28 ex_ArrayMean.nxc

This is an example of how to use the [ArrayMean](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Mean value = ");
    int x = ArrayMean(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.29 ex_ArrayMin.nxc

This is an example of how to use the [ArrayMin](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Min value = ");
    int x = ArrayMin(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.30 ex_ArrayOp.nxc

This is an example of how to use the [ArrayOp](#) function.

```
task main()
{
    int data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Max value = ");
    int x;
    ArrayOp(OPARR_MAX, x, data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.31 ex_ArraySort.nxc

This is an example of how to use the [ArraySort](#) function.

```
task main()
{
    int data[40];
    int tmp[];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    ArraySort(tmp, data, NA, NA); // start at 0 and go to length(data);
    TextOut(0, LCD_LINE1, "Min value = ");
    NumOut(0, LCD_LINE2, tmp[0]);
    TextOut(0, LCD_LINE3, "Max value = ");
    NumOut(0, LCD_LINE4, tmp[39]);
    TextOut(0, LCD_LINE5, "Min value = ");
    NumOut(0, LCD_LINE6, ArrayMin(data, NA, NA));
    TextOut(0, LCD_LINE7, "Max value = ");
    NumOut(0, LCD_LINE8, ArrayMax(data, NA, NA));
    Wait(SEC_3);
}
```

9.32 ex_ArrayStd.nxc

This is an example of how to use the [ArrayStd](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "StdDev values = ");
    long x = ArrayStd(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.33 ex_arraysubset.nxc

This is an example of how to use the [ArraySubset](#) function.

```
// copy 5 elements starting with the 3rd element, i.e., srcArray[2]
ArraySubset(myArray, srcArray, 2, 5);
```

9.34 ex_ArraySum.nxc

This is an example of how to use the [ArraySum](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "Sum of values = ");
    long x = ArraySum(data, NA, NA); // start at 0 and go to length(data);
    NumOut(0, LCD_LINE2, x);
    Wait(SEC_3);
}
```

9.35 ex_ArraySumSqr.nxc

This is an example of how to use the [ArraySumSqr](#) function.

```
task main()
{
    long data[40];
    for (int i = 0; i < 40; i++)
        data[i] = rand();
    TextOut(0, LCD_LINE1, "SumSqr values = ");
```

```
long x = ArraySumSqr(data, NA, NA); // start at 0 and go to length(data);
NumOut(0, LCD_LINE2, x);
Wait(SEC_3);
}
```

9.36 ex_asin.nxc

This is an example of how to use the [asin](#) function.

```
// ex_asin.nxc
// Display values of the asin API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_asin(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%7.4f RAD", asin(val)));
}

task main()
{
    show_asin(MIN_VAL, LCD_LINE1); // shows -1.5708 RAD
    show_asin(MID_VAL, LCD_LINE2); // shows 0.0000 RAD
    show_asin(MAX_VAL, LCD_LINE3); // shows 1.5708 RAD
    // An invalid value returns not-a-number (nan).
    show_asin(INVALID, LCD_LINE4); // shows -nan RAD
    while (true);
}
```

9.37 ex_asind.nxc

This is an example of how to use the [asind](#) function.

```
// ex_asind.nxc
// Display values of the asind API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define MIN_VAL -1.0
#define MID_VAL 0.0
#define MAX_VAL 1.0
#define INVALID 2.0

inline void show_asin(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", asind(val)));
}
```

```
task main()
{
    show_asin(MIN_VAL, LCD_LINE1); // shows -90.00 DEG
    show_asin(MID_VAL, LCD_LINE2); // shows 0.00 DEG
    show_asin(MAX_VAL, LCD_LINE3); // shows 90.00 DEG
    // An invalid value returns not-a-number (nan).
    show_asin(INVALID, LCD_LINE4); // shows -nan DEG
    while (true);
}
```

9.38 ex_atan.nxc

This is an example of how to use the `atan` function.

```
// ex_atan.nxc
// Display values of the atan API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define BIG_NEG_VAL -1000.0
#define NEG_VAL -1.0
#define POS_VAL 1.0
#define BIG_POS_VAL 1000.0

inline void show_atan(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%7.4f RAD", atan(val)));
}

task main()
{
    show_atan(BIG_NEG_VAL, LCD_LINE1); // shows -1.5698 RAD
    show_atan(NEG_VAL, LCD_LINE2); // shows -0.7854 RAD
    show_atan(0.0, LCD_LINE3); // shows 0.0000 RAD
    show_atan(POS_VAL, LCD_LINE4); // shows 0.7854 RAD
    show_atan(BIG_POS_VAL, LCD_LINE5); // shows 1.5698 RAD
    while (true);
}
```

9.39 ex_atan2.nxc

This is an example of how to use the `atan2` function.

```
// ex_atan2.nxc
// Display values of the atan2 API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

// The following two arrays comprise the x and y coordinates of the corners and
// the mid-points of the sides of a square centered at the origin and having
// sides two units long.
```

```

const float y_coord[] = {-1.0, -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0};
const float x_coord[] = {-1.0, 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0};

// Display the angles made by lines from the origin to the points on the square
// as specified above.
task main()
{
    const int pts = ArrayLen(y_coord);
    for (int i = 0; i < pts; ++i)
    {
        float angle = atan2(y_coord[i], x_coord[i]);
        TextOut(0, 56 - 8 * i, FormatNum("%7.4f RAD", angle));
    }
    while (true);
}

```

9.40 ex_atan2d.nxc

This is an example of how to use the [atan2d](#) function.

```

// ex_atan2d.nxc
// Display values of the atan2d API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

// The following two arrays comprise the x and y coordinates of the corners and
// the mid-points of the sides of a square centered at the origin and having
// sides two units long.
const float y_coord[] = {-1.0, -1.0, -1.0, 0.0, 1.0, 1.0, 1.0, 0.0};
const float x_coord[] = {-1.0, 0.0, 1.0, 1.0, 1.0, 0.0, -1.0, -1.0};

// Display the angles made by lines from the origin to the points on the square
// as specified above.
task main()
{
    const int pts = ArrayLen(y_coord);
    for (int i = 0; i < pts; ++i)
    {
        float angle = atan2d(y_coord[i], x_coord[i]);
        TextOut(0, 56 - 8 * i, FormatNum("%7.2f DEG", angle));
    }
    while (true);
}

```

9.41 ex_atand.nxc

This is an example of how to use the [atand](#) function.

```

// ex_atand.nxc
// Display values of the atand API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

```

```
#define BIG_NEG_VAL -1000.0
#define NEG_VAL -1.0
#define POS_VAL 1.0
#define BIG_POS_VAL 1000.0

inline void show_atan(const float val, const int screen_y)
{
    TextOut(0, screen_y, FormatNum("%6.2f DEG", atan(val)));
}

task main()
{
    show_atan(BIG_NEG_VAL, LCD_LINE1); // shows -89.94 DEG
    show_atan(NEG_VAL, LCD_LINE2); // shows -45.00 DEG
    show_atan(0.0, LCD_LINE3); // shows 0.00 DEG
    show_atan(POS_VAL, LCD_LINE4); // shows 45.00 DEG
    show_atan(BIG_POS_VAL, LCD_LINE5); // shows 89.94 DEG
    while (true);
}
```

9.42 ex_atof.nxc

This is an example of how to use the [atof](#) function.

```
task main()
{
    float f = atof("3.14159e2");
    NumOut(0, LCD_LINE1, f);
    Wait(SEC_5);
}
```

9.43 ex_atoi.nxc

This is an example of how to use the [atoi](#) function.

```
task main()
{
    NumOut(0, LCD_LINE1, atoi("3.14159"));
    Wait(SEC_5);
}
```

9.44 ex_atol.nxc

This is an example of how to use the [atol](#) function.

```
task main()
{
    NumOut(0, LCD_LINE1, atol("3.142e2"));
    Wait(SEC_5);
}
```

9.45 ex_BatteryState.nxc

This is an example of how to use the [BatteryState](#) function.

```
x = BatteryState();
```

9.46 ex_bcd2dec.nxc

This is an example of how to use the [bcd2dec](#) function.

```
// convert binary-coded decimal byte to decimal.  
byte dec = bcd2dec(0x3a);
```

9.47 ex_BluetoothState.nxc

This is an example of how to use the [BluetoothState](#) function.

```
x = BluetoothState();
```

9.48 ex_bluetoothstatus.nxc

This is an example of how to use the [BluetoothStatus](#) function.

```
x = BluetoothStatus(1);
```

9.49 ex_bluetoothwrite.nxc

This is an example of how to use the [BluetoothWrite](#) function.

```
x = BluetoothWrite(1, data);
```

9.50 ex_BrickDataBluecoreVersion.nxc

This is an example of how to use the [BrickDataBluecoreVersion](#) function.

```
int bv = BrickDataBluecoreVersion();
```

9.51 ex_BrickDataBtHardwareStatus.nxc

This is an example of how to use the [BrickDataBtHardwareStatus](#) function.

```
int x = BrickDataBtHardwareStatus();
```

9.52 ex_BrickDataBtStateStatus.nxc

This is an example of how to use the [BrickDataBtStateStatus](#) function.

```
int x = BrickDataBtStateStatus();
```

9.53 ex_BrickDataName.nxc

This is an example of how to use the [BrickDataName](#) function.

```
string name = BrickDataName();
```

9.54 ex_BrickDataTimeoutValue.nxc

This is an example of how to use the [BrickDataTimeoutValue](#) function.

```
int x = BrickDataTimeoutValue();
```

9.55 ex_BTConnectionClass.nxc

This is an example of how to use the [BTConnectionClass](#) function.

```
long class = BTConnectionClass(1);
```

9.56 ex_BTConnectionHandleNum.nxc

This is an example of how to use the [BTConnectionHandleNum](#) function.

```
byte handlenum = BTConnectionHandleNum(idx);
```

9.57 ex_BTConnectionLinkQuality.nxc

This is an example of how to use the [BTConnectionLinkQuality](#) function.

```
byte linkquality = BTConnectionLinkQuality(1);
```

9.58 ex_BTConnectionName.nxc

This is an example of how to use the [BTConnectionName](#) function.

```
string name = BTConnectionName(0);
```

9.59 ex_BTConnectionPinCode.nxc

This is an example of how to use the [BTConnectionPinCode](#) function.

```
string pincode = BTConnectionPinCode(0);
```

9.60 ex_BTConnectionStreamStatus.nxc

This is an example of how to use the [BTConnectionStreamStatus](#) function.

```
byte streamstatus = BTConnectionStreamStatus(1);
```

9.61 ex_BTDeviceClass.nxc

This is an example of how to use the [BTDeviceClass](#) function.

```
long class = BTDeviceClass(1);
```

9.62 ex_BTDeviceCount.nxc

This is an example of how to use the [BTDeviceCount](#) function.

```
byte x = BTDeviceCount();
```

9.63 ex_BTDeviceName.nxc

This is an example of how to use the [BTDeviceName](#) function.

```
string name = BTDeviceName(0);
```

9.64 ex_BTDeviceNameCount.nxc

This is an example of how to use the [BTDeviceNameCount](#) function.

```
byte x = BTDeviceNameCount();
```

9.65 ex_BTDeviceStatus.nxc

This is an example of how to use the [BTDeviceStatus](#) function.

```
byte status = BTDeviceStatus(1);
```

9.66 ex_BTInputBufferInPtr.nxc

This is an example of how to use the [BTInputBufferInPtr](#) function.

```
byte x = BTInputBufferInPtr();
```

9.67 ex_BTInputBufferOutPtr.nxc

This is an example of how to use the [BTInputBufferOutPtr](#) function.

```
byte x = BTInputBufferOutPtr();
```

9.68 ex_BTOutputBufferInPtr.nxc

This is an example of how to use the [BTOutputBufferInPtr](#) function.

```
byte x = BTOutputBufferInPtr();
```

9.69 ex_BTOutputBufferOutPtr.nxc

This is an example of how to use the [BTOutputBufferOutPtr](#) function.

```
byte x = BTOutputBufferOutPtr();
```

9.70 ex_ButtonCount.nxc

This is an example of how to use the [ButtonCount](#) function.

```
value = ButtonCount(BTN1, true);
```

9.71 ex_ButtonLongPressCount.nxc

This is an example of how to use the [ButtonLongPressCount](#) function.

```
value = ButtonLongPressCount(BTN1);
```

9.72 ex_ButtonLongReleaseCount.nxc

This is an example of how to use the [ButtonLongReleaseCount](#) function.

```
value = ButtonLongReleaseCount(BTN1);
```

9.73 ex_ButtonPressCount.nxc

This is an example of how to use the [ButtonPressCount](#) function.

```
value = ButtonPressCount(BTN1);
```

9.74 ex_ButtonPressed.nxc

This is an example of how to use the [ButtonPressed](#) function.

```
task main()
{
#ifndef __ENHANCED_FIRMWARE
    SetLongAbort(true);
```

```
#endif
    while(true)
    {
#define __ENHANCED_FIRMWARE
        NumOut(0, LCD_LINE1, ButtonPressed(BTNEXIT, false));
#endif
        NumOut(0, LCD_LINE2, ButtonPressed(BTNRIGHT, false));
        NumOut(0, LCD_LINE3, ButtonPressed(BTNLEFT, false));
        NumOut(0, LCD_LINE4, ButtonPressed(BTNCENTER, false));
    }
}
```

9.75 ex_ButtonReleaseCount.nxc

This is an example of how to use the [ButtonReleaseCount](#) function.

```
value = ButtonReleaseCount(BTN1);
```

9.76 ex_ButtonShortReleaseCount.nxc

This is an example of how to use the [ButtonShortReleaseCount](#) function.

```
value = ButtonShortReleaseCount(BTN1);
```

9.77 ex_ButtonState.nxc

This is an example of how to use the [ButtonState](#) function.

```
value = ButtonState(BTN1);
```

9.78 ex_ByteArrayToStr.nxc

This is an example of how to use the [ByteArrayToStr](#) function.

```
myStr = ByteArrayToStr(myArray);
```

9.79 ex_ByteArrayToStrEx.nxc

This is an example of how to use the [ByteArrayToStrEx](#) function.

```
ByteArrayToStrEx(myArray, myStr);
```

9.80 ex_ceil.nxc

This is an example of how to use the [ceil](#) function.

```
y = ceil(x);
```

9.81 ex_CircleOut.nxc

This is an example of how to use the [CircleOut](#), [Random](#), and [Wait](#) functions.

```
task main()
{
    for (int i=0; i < 50; i++) {
        int x = Random(10)+50;
        int y = Random(10)+32;
        int r = Random(20);
        CircleOut(x, y, r, DRAW_OPT_NORMAL+DRAW_OPT_LOGICAL_XOR+DRAW_OPT_FILL_SHAPE);

        Wait(MS_50);
    }
    CircleOut(20, 50, 20);
    Wait(SEC_2);
}
```

9.82 ex_ClearLine.nxc

This is an example of how to use the [TextOut](#), [ClearLine](#), and [Wait](#) functions.

```
task main()
{
    TextOut(0, LCD_LINE1, "testing 1234567890");
    Wait(SEC_5);
    ClearLine(LCD_LINE1);
    Wait(SEC_5);
    TextOut(0, LCD_LINE1, "testing 1234567890");
    Wait(SEC_5);
}
```

9.83 ex_ClearScreen.nxc

This is an example of how to use the [ClearScreen](#) and [Wait](#) functions.

```
task main()
{
    ClearScreen();
    Wait(SEC_10);
}
```

9.84 ex_ClearSensor.nxc

This is an example of how to use the [ClearSensor](#) function.

```
ClearSensor(S1);
```

9.85 ex_CloseFile.nxc

This is an example of how to use the [CloseFile](#) function.

```
result = CloseFile(handle);
```

9.86 ex_coast.nxc

This is an example of how to use the [Coast](#) function.

```
Coast(OUT_A); // coast output A
```

9.87 ex_coastex.nxc

This is an example of how to use the [CoastEx](#) function.

```
CoastEx(OUT_A, RESET_NONE); // coast output A
```

9.88 ex_ColorADRaw.nxc

This is an example of how to use the [ColorADRaw](#) function.

```
unsigned int rawRed = ColorADRaw(S1, INPUT_RED);
```

9.89 ex_ColorBoolean.nxc

This is an example of how to use the [ColorBoolean](#) function.

```
bool bRed = ColorBoolean(S1, INPUT_RED);
```

9.90 ex_ColorCalibration.nxc

This is an example of how to use the [ColorCalibration](#) function.

```
long value = ColorCalibration(S1, INPUT_CAL_POINT_0, INPUT_RED);
```

9.91 ex_ColorCalibrationState.nxc

This is an example of how to use the [ColorCalibrationState](#) function.

```
byte value = ColorCalibrationState(S1);
```

9.92 ex_ColorCalLimits.nxc

This is an example of how to use the [ColorCalLimits](#) function.

```
unsigned int limit = ColorCalLimits(S1, INPUT_CAL_POINT_0);
```

9.93 ex_ColorSensorRaw.nxc

This is an example of how to use the [ColorSensorRaw](#) function.

```
unsigned int rawRed = ColorSensorRaw(S1, INPUT_RED);
```

9.94 ex_ColorSensorValue.nxc

This is an example of how to use the [ColorSensorValue](#) function.

```
unsigned int valRed = ColorSensorValue(S1, INPUT_RED);
```

9.95 ex_CommandFlags.nxc

This is an example of how to use the [CommandFlags](#) function.

```
x = CommandFlags();
```

9.96 ex_contrast.nxc

This is an example of how to use the [DisplayContrast](#) and [SetDisplayContrast](#) functions.

```
task main()
{
    for (byte contrast = 0; contrast < DISPLAY_CONTRAST_MAX; contrast++)
    {
        SetDisplayContrast(contrast);
        NumOut(0, LCD_LINE1, DisplayContrast());
```

```
    Wait(100);
}
for (byte contrast = DISPLAY_CONTRAST_MAX; contrast > 0; contrast--)
{
    SetDisplayContrast(contrast);
    NumOut(0, LCD_LINE1, DisplayContrast());
    Wait(100);
}
SetDisplayContrast(DISPLAY_CONTRAST_DEFAULT);
NumOut(0, LCD_LINE1, DisplayContrast());
while(true);
}
```

9.97 ex_Copy.nxc

This is an example of how to use the [Copy](#) function.

```
task main()
{
    string s = "Now is the winter of our discontent";
    TextOut(0, LCD_LINE1, Copy(s, 12, 5));
    Wait(SEC_4);
}
```

9.98 ex_cosh.nxc

This is an example of how to use the [cosh](#) function.

```
x = cosh(y);
```

9.99 ex_CreateFile.nxc

This is an example of how to use the [CreateFile](#) function.

```
result = CreateFile("data.txt", 1024, handle);
```

9.100 ex_CreateFileLinear.nxc

This is an example of how to use the [CreateFileLinear](#) function.

```
result = CreateFileLinear("data.txt", 1024, handle);
```

9.101 ex_CreateFileNonLinear.nxc

This is an example of how to use the [CreateFileNonLinear](#) function.

```
result = CreateFileNonLinear("data.txt", 1024, handle);
```

9.102 ex_cstdio.nxc

This is an example of how to use the cstdio API functions: [fopen](#), [fprintf](#), [fputc](#), [fputs](#), [fseek](#), [ftell](#), [fclose](#), [feof](#), [fflush](#), [fgetc](#), [fgets](#), [getc](#), [putc](#), [rewind](#), [printf](#), [sprintf](#), [rename](#), and [remove](#).

```
task main()
{
/*
fclose(byte handle)
feof(byte handle)
fflush(byte handle)
fgetc(byte handle)
fgets(string & str, int num, byte handle)
fopen(string filename, const string mode)
fprintf(byte handle, const string & format, variant value)
fputc(char ch, byte handle)
fputs(string str, byte handle)
fseek(byte handle, long offset, int origin)
ftell(byte handle)
getc(byte handle)
putc(char ch, byte handle)
remove(string fname)
rename(string oldname, string newname)
rewind(byte handle)

*/
```

9.103 ex_cstring.nxc

This is an example of how to use the cstring API functions: [strcat](#), [strcmp](#), [strcpy](#), [strlen](#), [strncat](#), [strncmp](#), [strncpy](#), [memcp](#)y, [memmove](#), and [memcmp](#).

```
task main()
{
/*
inline variant StrToNum(string str);
inline unsigned int StrLen(string str);
inline byte StrIndex(string str, unsigned int idx);
inline string NumToStr(variant num);
```

```

inline string StrCat(string str1, string str2, string str3, string strN);
inline string SubStr(string str, unsigned int idx, unsigned int len);
inline string Flatten(variant num);
inline string StrReplace(string str, unsigned int idx, string strnew);
inline string FormatNum(string fmt, variant number);

inline string FlattenVar(variant x);
inline int UnflattenVar(string str, variant & variable);
inline string ByteArrayToStr(byte data[]);
inline void ByteArrayToStrEx(byte data[], string & str);
inline void StrToByteArray(string str, byte & data[]);

strcat(string & dest, const string & src)
strcmp(const string & str1, const string & str2)
strcpy(string & dest, const string & src)
strlen(const string & str)
strncat(string & dest, const string & src, const int num)
strncmp(const string & str1, const string & str2, unsigned int num)
strncpy(string & dest, const string & src, const int num)
*/
}

```

9.104 ex_ctype.nxc

This is an example of how to use the ctype API functions: [isupper](#), [islower](#), [isalpha](#), [isdigit](#), [isalnum](#), [isspace](#), [iscntrl](#), [isprint](#), [isgraph](#), [ispunct](#), [isxdigit](#), [tolower](#), and [toupper](#).

```

task main()
{
    string tmp = "a1B2.G% ";
    TextOut(0, LCD_LINE1, tmp);
    NumOut(0, LCD_LINE2, isalnum(tmp[0])); // 1
    NumOut(0, LCD_LINE3, isalpha(tmp[1])); // 0
    NumOut(0, LCD_LINE4, iscntrl(tmp[2])); // 0
    NumOut(0, LCD_LINE5, isdigit(tmp[3])); // 1
    NumOut(0, LCD_LINE6, isgraph(tmp[4])); // 1
    NumOut(0, LCD_LINE7, islower(tmp[5])); // 0
    NumOut(0, LCD_LINE8, isprint(tmp[6])); // 1

    NumOut(40, LCD_LINE2, ispunct(tmp[0])); // 0
    NumOut(40, LCD_LINE3, isspace(tmp[1])); // 0
    NumOut(40, LCD_LINE4, isupper(tmp[2])); // 1
    NumOut(40, LCD_LINE5, isxdigit(tmp[3])); // 1
    NumOut(40, LCD_LINE6, tolower(tmp[4])); // 46
    NumOut(40, LCD_LINE7, toupper(tmp[5])); // 71

    Wait(SEC_5);
}

```

9.105 ex_CurrentTick.nxc

This is an example of how to use the [CurrentTick](#) function.

```
unsigned int x = CurrentTick();
```

9.106 ex_CustomSensorActiveStatus.nxc

This is an example of how to use the [CustomSensorActiveStatus](#) function.

```
x = CustomSensorActiveStatus(S1);
```

9.107 ex_CustomSensorPercentFullScale.nxc

This is an example of how to use the [CustomSensorPercentFullScale](#) function.

```
x = CustomSensorPercentFullScale(S1);
```

9.108 ex_CustomSensorZeroOffset.nxc

This is an example of how to use the [CustomSensorZeroOffset](#) function.

```
x = CustomSensorZeroOffset(S1);
```

9.109 ex_delete_data_file.nxc

This is an example of how to use the [DeleteFile](#), [TextOut](#), [FormatNum](#), and [Wait](#) functions. It is useful for deleting the circles.dat file created by the program described in the [ex_file_system::nxc](#) example.

```
// ex_delete_data_file.nxc
// Demonstates the use of the DeleteFile API call.
// Useful for deleting the circles.dat file created by the program described
// in the ex_file_system.nxc example.

#define FILE_NAME "circles.dat"

// Display a return code from a file system API call on the NXT screen.
// The codes most likely to be displayed are are:
//     LDR_SUCCESS          0x0000
//     LDR_FILENOTFOUND    0x8700
// See "Loader module error codes" to interpret any other code that appears.
void rtn_code_out(const unsigned int code)
{
    TextOut(0, LCD_LINE1, "code           ");
    TextOut(50, LCD_LINE1, FormatNum("%04x", code));
```

```
}

task main()
{
    unsigned int rtn_code = DeleteFile(FILE_NAME);
    rtn_code_out(rtn_code);
    Wait(SEC_5);
}
```

9.110 ex_DeleteFile.nxc

This is an example of how to use the [DeleteFile](#) function.

```
result = DeleteFile("data.txt");
```

9.111 ex_dispnfout.nxc

This is an example of how to use the [FontNumOut](#) function.

```
#download "PropTiny.ric"

task main()
{
    FontNumOut(0, 40, "PropTiny.RIC", PI);
    while( 1 ) ;
}
```

9.112 ex_disptout.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", "WiWiWiWiWi", DRAW_OPT_NORMAL|
    DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,48,"PropTiny.RIC", "WiWiWiWiWi", DRAW_OPT_INVERT|
    DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,40,"PropTiny.RIC", "WiWiWiWiWi", DRAW_OPT_LOGICAL_OR|
    DRAW_OPT_FONT_DIR_L2RB );
FontTextOut( 50,32,"PropTiny.RIC", "WiWiWiWiWi", DRAW_OPT_INVERT|
    DRAW_OPT_LOGICAL_AND|DRAW_OPT_FONT_DIR_L2RB );
Wait(SEC_5);
ClearScreen();
Wait(SEC_4);
}

```

9.113 ex_dispfunc.nxc

This is an example of how to use the [SysDisplayExecuteFunction](#) and [Wait](#) functions along with the [DisplayExecuteFunctionType](#) structure.

```

task main()
{
    DisplayExecuteFunctionType defArgs;
    defArgs.Cmd = DISPLAY_HORIZONTAL_LINE;
    defArgs.On = DRAW_OPT_NORMAL;
    defArgs.X1 = 20;
    defArgs.Y1 = 20;
    defArgs.X2 = 40;
    SysDisplayExecuteFunction(defArgs);
    Wait(SEC_15);
}

```

9.114 ex_dispgaout.nxc

This is an example of how to use the [GraphicArrayOut](#), [NumOut](#), and [Wait](#) function. It also demonstrates how to use the [RICOpSprite](#), [RICSpriteData](#), [RICOpCopyBits](#), [RICImgRect](#), and [RICImgPoint](#) macros.

```

byte ric_data[] = {
    RICOpSprite(1, 64, 2,
        RICSpriteData(0xFF, 0xFF, 0x80, 0x01, 0x80, 0x41,
            0x80, 0x21, 0x80, 0x11, 0x80, 0x09,
            0x80, 0x05, 0x80, 0x09, 0x80, 0x11,
            0x80, 0x21, 0x80, 0x41, 0x80, 0x81,
            0x81, 0x01, 0x82, 0x01, 0x84, 0x01,
            0x88, 0x01, 0x90, 0x01, 0xA0, 0x01,
            0x90, 0x01, 0x88, 0x01, 0x84, 0x01,
            0x82, 0x01, 0x81, 0x01, 0x80, 0x81,
            0x80, 0x41, 0x80, 0x21, 0x80, 0x11,
            0x80, 0x09, 0x80, 0x05, 0x80, 0x09,
            0x80, 0x11, 0x80, 0x21, 0x80, 0x41,
            0x80, 0x81, 0x81, 0x01, 0x82, 0x01,

```

```

        0x84, 0x01, 0x88, 0x01, 0x90, 0x01,
        0xA0, 0x01, 0x90, 0x01, 0x88, 0x01,
        0x84, 0x01, 0x82, 0x01, 0x81, 0x01,
        0x80, 0x81, 0x80, 0x41, 0x80, 0x21,
        0x80, 0x11, 0x80, 0x09, 0x80, 0x05,
        0x80, 0x09, 0x80, 0x11, 0x80, 0x21,
        0x80, 0x41, 0x80, 0x81, 0x81, 0x01,
        0x82, 0x01, 0x84, 0x01, 0x88, 0x01,
        0x90, 0x01, 0xA0, 0x01, 0x80, 0x01,
        0xFF, 0xFF)),
    RICOpCopyBits(0, 1,
        RICImgRect(
            RICImgPoint(0, 0), 16, 64),
            RICImgPoint(0, 0))
    );
}

void Animate()
{
    int i;
    byte a;
    byte b;

    a = ric_data[12];
    b = ric_data[13];

    for( i=12; i<132; i++ )
        ric_data[i] = ric_data[i+2];

    ric_data[ 132 ] = a;
    ric_data[ 133 ] = b;
}

task main()
{
    int counter = 0;

    while( 1 )
    {
        Animate();

        GraphicArrayOut(0, 0, ric_data);
        NumOut( 50, LCD_LINE1, ++counter );
        Wait(MS_20);
    }
}

```

9.115 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, 0x82, 0xBA, 0x86, 0xC6,
                      0x86, 0x86, 0xC2, 0xBE, 0xBA, 0xC6, 0xEE, 0xFE,
                      0xD6, 0xEE, 0xB6, 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, 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, 0x00, 0x28, 0x10, 0x48,
                      0x44, 0x48, 0x48, 0x40, 0x04, 0x78, 0x48, 0x08,
                      0x00, 0x00, 0x00, 0x10, 0x44, 0x78, 0x34, 0x44,
                      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                      0x00, 0x00, 0x00, 0x00, 0x00)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(0), 7), 7, 7), RICImgPoint(0, 0
        )),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(1), 0), 7, 7), RICImgPoint(7, 0
        )),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(2), 7), 7, 7), RICImgPoint(14,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(3), 0), 7, 7), RICImgPoint(21,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(4), 7), 7, 7), RICImgPoint(28,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(5), 0), 7, 7), RICImgPoint(35,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(6), 7), 7, 7), RICImgPoint(42,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(7), 0), 7, 7), RICImgPoint(49,
        0)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(8), 0), 7, 7), RICImgPoint(0, 7
        )),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(9), 7), 7, 7), RICImgPoint(7, 7
        )),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(10), 0), 7, 7), RICImgPoint(14,
        7)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(11), 7), 7, 7), RICImgPoint(21,
        7)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(12), 0), 7, 7), RICImgPoint(28,
        7)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(13), 7), 7, 7), RICImgPoint(35,
        7)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(14), 0), 7, 7), RICImgPoint(42,
        7)),
    RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(15), 7), 7, 7), RICImgPoint(49,
```

```
    7)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(16), 7), 7, 7), RICImgPoint(0,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(17), 0), 7, 7), RICImgPoint(7,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(18), 7), 7, 7), RICImgPoint(14,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(19), 0), 7, 7), RICImgPoint(21,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(20), 7), 7, 7), RICImgPoint(28,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(21), 0), 7, 7), RICImgPoint(35,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(22), 7), 7, 7), RICImgPoint(42,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(23), 0), 7, 7), RICImgPoint(49,
    14)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(24), 0), 7, 7), RICImgPoint(0,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(25), 7), 7, 7), RICImgPoint(7,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(26), 0), 7, 7), RICImgPoint(14,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(27), 7), 7, 7), RICImgPoint(21,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(28), 0), 7, 7), RICImgPoint(28,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(29), 7), 7, 7), RICImgPoint(35,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(30), 0), 7, 7), RICImgPoint(42,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(31), 7), 7, 7), RICImgPoint(49,
    21)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(32), 7), 7, 7), RICImgPoint(0,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(33), 0), 7, 7), RICImgPoint(7,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(34), 7), 7, 7), RICImgPoint(14,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(35), 0), 7, 7), RICImgPoint(21,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(36), 7), 7, 7), RICImgPoint(28,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(37), 0), 7, 7), RICImgPoint(35,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(38), 7), 7, 7), RICImgPoint(42,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(39), 0), 7, 7), RICImgPoint(49,
    28)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(40), 0), 7, 7), RICImgPoint(0,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(41), 7), 7, 7), RICImgPoint(7,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(42), 0), 7, 7), RICImgPoint(14,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(43), 7), 7, 7), RICImgPoint(21,
    35)),
```

```
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(44), 0), 7, 7), RICImgPoint(28,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(45), 7), 7, 7), RICImgPoint(35,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(46), 0), 7, 7), RICImgPoint(42,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(47), 7), 7, 7), RICImgPoint(49,
    35)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(48), 7), 7, 7), RICImgPoint(0,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(49), 0), 7, 7), RICImgPoint(7,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(50), 7), 7, 7), RICImgPoint(14,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(51), 0), 7, 7), RICImgPoint(21,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(52), 7), 7, 7), RICImgPoint(28,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(53), 0), 7, 7), RICImgPoint(35,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(54), 7), 7, 7), RICImgPoint(42,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(55), 0), 7, 7), RICImgPoint(49,
    42)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(56), 0), 7, 7), RICImgPoint(0,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(57), 7), 7, 7), RICImgPoint(7,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(58), 0), 7, 7), RICImgPoint(14,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(59), 7), 7, 7), RICImgPoint(21,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(60), 0), 7, 7), RICImgPoint(28,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(61), 7), 7, 7), RICImgPoint(35,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(62), 0), 7, 7), RICImgPoint(42,
    49)),
RICOpCopyBits(0, 1, RICImgRect(RICImgPoint(RICArg(63), 7), 7, 7), RICImgPoint(49,
    49))
};

#define A 0
#define B 1
#define C 2
#define D 3
#define E 4
#define F 5
#define G 6
#define H 8

#define P(_file, _rank) (((_rank)-1)*8)+(_file)

#define A1 P(A, 1)
#define A2 P(A, 2)
#define A3 P(A, 3)
#define A4 P(A, 4)
```

```
#define A5 P(A, 5)
#define A6 P(A, 6)
#define A7 P(A, 7)
#define A8 P(A, 8)

#define B1 P(B, 1)
#define B2 P(B, 2)
#define B3 P(B, 3)
#define B4 P(B, 4)
#define B5 P(B, 5)
#define B6 P(B, 6)
#define B7 P(B, 7)
#define B8 P(B, 8)

#define C1 P(C, 1)
#define C2 P(C, 2)
#define C3 P(C, 3)
#define C4 P(C, 4)
#define C5 P(C, 5)
#define C6 P(C, 6)
#define C7 P(C, 7)
#define C8 P(C, 8)

#define D1 P(D, 1)
#define D2 P(D, 2)
#define D3 P(D, 3)
#define D4 P(D, 4)
#define D5 P(D, 5)
#define D6 P(D, 6)
#define D7 P(D, 7)
#define D8 P(D, 8)

#define E1 P(E, 1)
#define E2 P(E, 2)
#define E3 P(E, 3)
#define E4 P(E, 4)
#define E5 P(E, 5)
#define E6 P(E, 6)
#define E7 P(E, 7)
#define E8 P(E, 8)

#define F1 P(F, 1)
#define F2 P(F, 2)
#define F3 P(F, 3)
#define F4 P(F, 4)
#define F5 P(F, 5)
#define F6 P(F, 6)
#define F7 P(F, 7)
#define F8 P(F, 8)

#define G1 P(G, 1)
#define G2 P(G, 2)
#define G3 P(G, 3)
#define G4 P(G, 4)
#define G5 P(G, 5)
#define G6 P(G, 6)
#define G7 P(G, 7)
```

```

#define G8 P(G, 8)

#define H1 P(H, 1)
#define H2 P(H, 2)
#define H3 P(H, 3)
#define H4 P(H, 4)
#define H5 P(H, 5)
#define H6 P(H, 6)
#define H7 P(H, 7)
#define H8 P(H, 8)

int b[] =
{
    64, 72, 80, 88, 96, 80, 72, 64, // 1
    56, 56, 56, 56, 56, 56, 56, // 2
    48, 48, 48, 48, 48, 48, 48, // 3
    48, 48, 48, 48, 48, 48, 48, // 4
    48, 48, 48, 48, 48, 48, 48, // 5
    48, 48, 48, 48, 48, 48, 48, // 6
    40, 40, 40, 40, 40, 40, 40, // 7
    32, 24, 16, 8, 0, 16, 24, 32 // 8
};

// A   B   C   D   E   F   G   H

#define Vacant 48

#define Move(_from, _to) \
b[_to] = b[_from]; \
b[_from] = Vacant; \
GraphicArrayOutEx( 8,8,Chess1_data , b, true); \
Wait(SEC_2);

task main()
{
    // setup board
    GraphicArrayOutEx( 8,8,Chess1_data, b, true);
    WaitSEC_2();

    Move(A2, A3); // white pawn from A2 to A3
    Move(B7, B5); // black pawn from B7 to B5
    Move(A3, A4); // white pawn from A3 to A4
    Move(B5, B4); // black pawn from B5 to B4
    Move(A4, A5); // white pawn from A4 to A5
    Move(B4, B3); // black pawn from B4 to B3
    Move(A5, A6); // white pawn from A5 to A6
    while( true );
}

```

9.116 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", "21.ric"};
task main()
{
    long tick;
    TextOut(0, LCD_LINE1, "testing");
    tick = CurrentTick();
    GraphicOut(10, 10, names[0]);
    tick = CurrentTick()-tick;
    NumOut(0, LCD_LINE8, tick);
    Wait(SEC_5);
    ClearScreen();
    Wait(MS_500);
    TextOut(0, LCD_LINE1, "testing");
    tick = CurrentTick();
    dgaArgs.Location.X = 10;
    dgaArgs.Location.Y = 10;
    dgaArgs.Options = 0;
    dgaArgs.Data = tmpData2;
    SysCall(DrawGraphicArray, dgaArgs);
    tick = CurrentTick()-tick;
    NumOut(0, LCD_LINE8, tick);
    Wait(SEC_5);
}
```

9.117 ex_dispoutex.nxc

This is an example of how to use the [GraphicOutEx](#) and [Wait](#) functions.

```
#download "letters.ric"

string fnames[] = {"letters.ric", "letter2.ric"};
int Values[] = {0};
void Display( int n )
{
    Values[0]=n*10;
    GraphicOutEx(Values[0], Random(30), fnames[0], Values,
        DRAW_OPT_CLEAR_WHOLE_SCREEN);
    Wait(MS_200);
}

task main()
{
    while( true )
    {
        for( int i=0; i<9; i++ )
            Display( i );
```

```
    }  
}
```

9.118 ex_DisplayDisplay.nxc

This is an example of how to use the [DisplayDisplay](#) function.

```
x = DisplayDisplay();
```

9.119 ex_DisplayEraseMask.nxc

This is an example of how to use the [DisplayEraseMask](#) function.

```
x = DisplayEraseMask();
```

9.120 ex_DisplayFlags.nxc

This is an example of how to use the [DisplayFlags](#) function.

```
x = DisplayFlags();
```

9.121 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  
    0x00,0x00,0x00,0x00,0x00,0x06,0x5F,0x06,0x00,0x00,0x07,0x03,0x00,0x07,  
    ,0x03,0x00,0x24,0x7E,0x24,0x7E,0x24,0x00,0x24,0x2B,0x6A,0x12,0x00,0x00,0x63,0x13,  
    0x08,0x64,0x63,0x00,0x30,0x4C,0x52,0x22,0x50,0x00,0x00,0x07,0x03,0x00,0x00,0x00,0  
    x00,0x3E,0x41,0x00,0x00,0x00,0x00,0x41,0x3E,0x00,0x00,0x00,0x08,0x3E,0x1C,0x3E,0x  
    08,0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x80,0x60,0x60,0x00,0x00,0x00,0x08,0x08,0x0  
    8,0x08,0x08,0x00,0x00,0x60,0x60,0x00,0x00,0x20,0x10,0x08,0x04,0x02,0x00,  
    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,0x49,0x30,0x00,0x01,0x71,0x09,0x05,0x03,0x00,0
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,0x11,0x11,0x7E,0x00,0x7F,0x49,0x49,0x49
,0x36,0x00,0x3E,0x41,0x41,0x22,0x00,0x7F,0x41,0x41,0x41,0x3E,0x00,0x7F,0x49,
0x49,0x49,0x41,0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x3E,0x41,0x49,0x49,0x7A,0x00,0
x7F,0x08,0x08,0x08,0x7F,0x00,0x00,0x41,0x7F,0x41,0x00,0x00,0x30,0x40,0x40,0x40,0x
3F,0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x7F,0x40,0x40,0x40,0x00,0x7F,0x02,0x0
4,0x02,0x7F,0x00,0x7F,0x02,0x04,0x08,0x7F,0x00,0x3E,0x41,0x41,0x41,0x3E,0x00,
0x7F,0x09,0x09,0x09,0x06,0x00,0x41,0x51,0x21,0x5E,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,0x41,0x41,0x00,0x02,0x04,0x08,0x10,0x20,0x00,0x00,0x41,0x4
1,0x7F,0x00,0x00,0x04,0x02,0x01,0x02,0x04,0x00,0x80,0x80,0x80,0x80,0x00,
0x00,0x02,0x05,0x02,0x00,0x20,0x54,0x54,0x54,0x78,0x00,0x7F,0x44,0x44,0x44
,0x38,0x00,0x38,0x44,0x44,0x28,0x00,0x38,0x44,0x44,0x44,0x44,0x7F,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,0x41,0x41,0x00,0x00,0x00,0x00,0x77,0x00,0x00,0x00,0x00,0x00,0x41,0x4
1,0x3E,0x08,0x00,0x02,0x01,0x02,0x01,0x00,0x20,0x40,0x38,0x07,0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    byte myData[800];
    ptr = addr(NewFont);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

9.122 ex_DisplayTextLinesCenterFlags.nxc

This is an example of how to use the [DisplayTextLinesCenterFlags](#) function.

```
x = DisplayTextLinesCenterFlags();
```

9.123 ex_DisplayUpdateMask.nxc

This is an example of how to use the [DisplayUpdateMask](#) function.

```
x = DisplayUpdateMask();
```

9.124 ex_dispmisc.nxc

This is an example of how to use the [DisplayEraseMask](#), [DisplayUpdateMask](#), [DisplayDisplay](#), [DisplayFlags](#), [DisplayTextLinesCenterFlags](#) functions, [SetDisplayEraseMask](#), [SetDisplayUpdateMask](#), [SetDisplayDisplay](#), [SetDisplayFlags](#), and [SetDisplayTextLinesCenterFlags](#) functions,

```
task main()
{
    unsigned long addr = DisplayDisplay();
    NumOut(0, LCD_LINE1, DisplayEraseMask());
    NumOut(0, LCD_LINE2, DisplayUpdateMask());
    NumOut(0, LCD_LINE3, addr);
    NumOut(0, LCD_LINE4, DisplayFlags());
    NumOut(0, LCD_LINE5, DisplayTextLinesCenterFlags());
    Wait(SEC_4);
    // setting the display address function can be ... dangerous
    SetDisplayDisplay(addr);
    // fiddling with the display flags is also dangerous
    unsigned long flags = DisplayFlags();
    flags |= DISPLAY_POPUP;
    SetDisplayFlags(flags);
    Wait(SEC_2);
    flags = flags & (~DISPLAY_POPUP);
    SetDisplayFlags(flags);
    Wait(SEC_1);
    SetDisplayEraseMask(DisplayEraseMask());
    SetDisplayUpdateMask(DisplayUpdateMask());
    SetDisplayTextLinesCenterFlags(DisplayTextLinesCenterFlags());
    Wait(SEC_2);
}
```

9.125 ex_DISTNxDistance.nxc

This is an example of how to use the [DISTNxDistance](#) function.

```
int dist = DISTNxDistance(S1, MS_ADDR_DISTNX);
```

9.126 ex_DISTNxGP2D12.nxc

This is an example of how to use the [DISTNxGP2D12](#) function.

```
char result = DISTNxGP2D12(S1, MS_ADDR_DISTNX);
```

9.127 ex_DISTNxGP2D120.nxc

This is an example of how to use the [DISTNxGP2D120](#) function.

```
char result = DISTNxGP2D120(S1, MS_ADDR_DISTNX);
```

9.128 ex_DISTNxGP2YA02.nxc

This is an example of how to use the [DISTNxGP2YA02](#) function.

```
char result = DISTNxGP2YA02(S1, MS_ADDR_DISTNX);
```

9.129 ex_DISTNxGP2YA21.nxc

This is an example of how to use the [DISTNxGP2YA21](#) function.

```
char result = DISTNxGP2YA21(S1, MS_ADDR_DISTNX);
```

9.130 ex_DISTNxMaxDistance.nxc

This is an example of how to use the [DISTNxMaxDistance](#) function.

```
int dist = DISTNxMaxDistance(S1, MS_ADDR_DISTNX);
```

9.131 ex_DISTNxMinDistance.nxc

This is an example of how to use the [DISTNxMinDistance](#) function.

```
int dist = DISTNxMinDistance(S1, MS_ADDR_DISTNX);
```

9.132 ex_DISTNxModuleType.nxc

This is an example of how to use the [DISTNxModuleType](#) function.

```
int modtype = DISTNxModuleType(S1, MS_ADDR_DISTNX);
```

9.133 ex_DISTNxNumPoints.nxc

This is an example of how to use the [DISTNxNumPoints](#) function.

```
int numpoints = DISTNxNumPoints(S1, MS_ADDR_DISTNX);
```

9.134 ex_DISTNxVoltage.nxc

This is an example of how to use the [DISTNxVoltage](#) function.

```
int volt = DISTNxVoltage(S1, MS_ADDR_DISTNX);
```

9.135 ex_div.nxc

This is an example of how to use the [div](#) function.

```
task main()
{
    long x, y;
    x = 31464;
    y = 33;
    div_t r;
    r = div(x, y);
    NumOut(0, LCD_LINE1, r.quot);
    NumOut(0, LCD_LINE2, r.rem);
    Wait(SEC_3);
}
```

9.136 ex_EllipseOut.nxc

This is an example of how to use the [EllipseOut](#) and [Random](#) functions.

```
task main()
{
    repeat (10)
        EllipseOut(50, 32, 20+Random(15), 20+Random(10), DRAW_OPT_FILL_SHAPE |
                   DRAW_OPT_LOGICAL_XOR);
    while(true);
}
```

9.137 ex_exp.nxc

This is an example of how to use the [exp](#) function.

```
y = exp(x);
```

9.138 ex_fclose.nxc

This is an example of how to use the [fclose](#) function.

```
result = fclose(handle);
```

9.139 ex_feof.nxc

This is an example of how to use the [feof](#) function.

```
int i = feof(handle);
```

9.140 ex_fflush.nxc

This is an example of how to use the [fflush](#) function.

```
int i = fflush(handle);
```

9.141 ex_fgetc.nxc

This is an example of how to use the [fgetc](#) function.

```
char val = fgetc(handle);
```

9.142 ex_fgets.nxc

This is an example of how to use the [fgets](#) function.

```
fgets(msg, 10, handle);
```

9.143 ex_file_system.nxc

This is an example of how to use the [PlayTone](#), [Wait](#), [Stop](#), [TextOut](#), [OpenFileAppend](#), [CloseFile](#), [OpenFileRead](#), [FormatNum](#), [Write](#), [Read](#), and [CircleOut](#) functions. This program is intended to serve as an introduction to data files on the NXT. It focuses on handling the codes returned by the file system's API calls, which is an important aspect of the API that is all too often neglected by programmers. The program deals

with a data file describing circles. On each run, it adds a new circle record to the data file. Then it reads in the whole data file and displays all the circles on NXT screen. It creates the data file if doesn't already exist. If you run it several times in secession, you will fill the data file and get a file-is-full exception. The data file created by this program is not visible on the NXT. To delete the file, circles.dat, you can use the NeXT Explorer or the example program [ex_delete_data_file::nxc](#).

```
// ex_file_system.nxc
// This program is intended to serve as an introduction to data files on the
// NXT. It focuses on handling the codes returned by the file system's API
// calls, which is an important aspect of the API that is all too often
// neglected by programmers.
//
// The program deals with a data file describing circles. On each run, it adds
// a new circle record to the data file. Then it reads in the whole data file
// and displays all the circles on NXT screen. It creates the data file if
// doesn't already exist. If you run it several times in secession, you will
// fill the data file and get a file-is-full exception.
//
// The data file created by this program is not visible on the NXT. To delete
// the file, circles.dat, you can use the NeXT Explorer or the example program
// ex_delete_data_file.nxc.

#define MIN_R 10
#define MAX_R 30
#define MIN_X 20
#define MAX_X 80
#define MIN_Y 10
#define MAX_Y 54

byte handle = 0; // file handle

#define FILE_NAME "circles.dat"
// The file size is made small so it will fill up quickly.
#define RECORDS 4
#define RECORD_SIZE 3
#define FILE_SIZE (RECORD_SIZE * RECORDS)

// This struct defines the data records.
struct circle
{
    byte r; // radius
    byte cx; // center x-coordinate
    byte cy; // center y-coordinate
};

// Initialize a circle with random radius r and center (cx, cy).
void init_circle(circle & c)
{
    c.r = MIN_R + Random(MAX_R - MIN_R);
    c.cx = MIN_X + Random(MAX_X - MIN_X);
    c.cy = MIN_Y + Random(MAX_Y - MIN_Y);
}

// Make sure file is closed whether or not file operations succeed or fail.
void shutdown(const int delay)
```

```
{  
    if (handle) CloseFile(handle);  
    // Get user's attention.  
    PlayTone(TONE_C5, SEC_1);  
    // Give the user time to read screen messages.  
    Wait(delay);  
    Stop(true);  
}  
  
// Display a return code from a file system API call on the NXT screen.  
void rtn_code_out(const unsigned int code)  
{  
    TextOut(0, LCD_LINE2, "code      ");  
    TextOut(50, LCD_LINE2, FormatNum("%04x", code));  
}  
  
// Open the data file for writing.  
void open_for_write()  
{  
    unsigned int file_size = FILE_SIZE;  
    handle = 0;  
    // Start with the assumption the file doesn't exist and needs to be created.  
    unsigned int rtn_code = CreateFile(FILE_NAME, file_size, handle);  
    // If the file already exists, open it with the intent of adding to the data  
    // that is already there.  
    if (rtn_code == LDR_FILEEXISTS)  
        rtn_code = OpenFileAppend(FILE_NAME, file_size, handle);  
    // Return code handling  
    switch (rtn_code)  
    {  
        case LDR_SUCCESS:  
            return;  
        case LDR_FILEISFULL:  
            TextOut(0, LCD_LINE1, "file is full      ");  
            break;  
        default:  
            // Unanticipated exception.  
            TextOut(0, LCD_LINE1, "write open      ");  
            rtn_code_out(rtn_code);  
            break;  
    }  
    shutdown(SEC_8);  
}  
  
// Open the data file for reading.  
void open_for_read()  
{  
    unsigned int file_size = FILE_SIZE;  
    handle = 0;  
    unsigned int rtn_code = OpenFileRead(FILE_NAME, file_size, handle);  
    // Return code handling  
    if (rtn_code != LDR_SUCCESS)  
    {  
        // Unanticipated exception.  
        TextOut(0, LCD_LINE1, "read open      ");  
        rtn_code_out(rtn_code);  
        shutdown(SEC_8);  
    }
```

```
    }

}

// Write one circle record to the data file.
void write_recd(const circle recd)
{
    unsigned int rtn_code = Write(handle, recd);
    // Return code handling
    if (rtn_code != LDR_SUCCESS)
    {
        switch (rtn_code)
        {
        case LDR_EOFEXPECTED:
            TextOut(0, LCD_LINE1, "no more space      ");
            break;
        default:
            // Unanticipated exception.
            TextOut(0, LCD_LINE1, "write failed      ");
            rtn_code_out(rtn_code);
            break;
        }
        shutdown(SEC_8);
    }
}

// Read all the circle records from the data file. Display each circle as it is
// read.
void read_all(circle & recd)
{
    while (true)
    {
        unsigned int rtn_code = Read(handle, recd);
        // rtn_code_out(rtn_code);
        // Return code handling
        switch (rtn_code)
        {
        case LDR_SUCCESS:
            // Record has been read. Display circle described by it.
            CircleOut(recd.cx, recd.cy, recd.r);
            Wait(SEC_2);
            break;
        case LDR_ENDOFFILE:
            // No more data to read.
            return;
        default:
            // Unanticipated exception.
            TextOut(0, LCD_LINE1, "read failed      ");
            rtn_code_out(rtn_code);
            shutdown(SEC_8);
        }
    }
}

task main()
{
    circle c;
    open_for_write();
```

```
    init_circle(c);
    write_recd(c);
    CloseFile(handle);
    open_for_read();
    read_all(c);
    shutdown(SEC_8);
}
```

9.144 ex_FindFirstFile.nxc

This is an example of how to use the [FindFirstFile](#) function.

```
result = FindFirstFile(fname, handle);
```

9.145 ex_FindNextFile.nxc

This is an example of how to use the [FindNextFile](#) function.

```
result = FindNextFile(fname, handle);
```

9.146 ex_FirstTick.nxc

This is an example of how to use the [FirstTick](#) function.

```
unsigned int x = FirstTick();
```

9.147 ex_Flatten.nxc

This is an example of how to use the [Flatten](#) function.

```
msg = Flatten(48); // returns "0" since 48 == ascii("0")
```

9.148 ex_FlattenVar.nxc

This is an example of how to use the [FlattenVar](#) function.

```
task main()
{
    long data[] = {-50123, 68142, 128176, -45123};
    long data2[4];
```

```
float fdata[] = {12.123, 3.14159, 2.68};
float fdata2[3];
NumOut(0, LCD_LINE1, data[0]);
NumOut(0, LCD_LINE2, fdata[1]);
string sdata = FlattenVar(data);
string tmp;
// transfer the string to another NXT
tmp = sdata;
UnflattenVar(tmp, data2);
NumOut(0, LCD_LINE3, data2[0]);
sdata = FlattenVar(fdata);
// transfer the string to another NXT
tmp = sdata;
UnflattenVar(tmp, fdata2);
NumOut(0, LCD_LINE4, fdata2[1]);
Wait(SEC_5);
}
```

9.149 ex_float.nxc

This is an example of how to use the [Float](#) function.

```
Float(OUT_A); // float output A
```

9.150 ex_floor.nxc

This is an example of how to use the [floor](#) function.

```
y = floor(x);
```

9.151 ex_Follows.nxc

This is an example of how to use the [Follows](#) statement.

```
Follows(main);
```

9.152 ex_fopen.nxc

This is an example of how to use the [fopen](#) function.

```
byte handle = fopen("test.txt", "r");
```

9.153 ex_ForceOff.nxc

This is an example of how to use the [ForceOff](#) function.

```
ForceOff(true);
```

9.154 ex_FormatNum.nxc

This is an example of how to use the [FormatNum](#) function.

```
msg = FormatNum("value = %d", x);
```

9.155 ex_fprintf.nxc

This is an example of how to use the [fprintf](#) function.

```
fprintf(handle, "value = %d", value);
```

9.156 ex_fputc.nxc

This is an example of how to use the [fputc](#) function.

```
fputc(ch, handle);
```

9.157 ex_fputs.nxc

This is an example of how to use the [fputs](#) function.

```
fputs(msg, handle);
```

9.158 ex_frac.nxc

This is an example of how to use the [frac](#) function.

```
y = frac(x);
```

9.159 ex_FreeMemory.nxc

This is an example of how to use the [FreeMemory](#) function.

```
x = FreeMemory();
```

9.160 ex_fseek.nxc

This is an example of how to use the [fseek](#) function.

```
fseek(handle, 10, SEEK_CUR);
```

9.161 ex_ftell.nxc

This is an example of how to use the [ftell](#) function.

```
long i = ftell(handle);
```

9.162 ex_GetBrickDataAddress.nxc

This is an example of how to use the [GetBrickDataAddress](#) function.

```
task main()
{
    byte data[];
    GetBrickDataAddress(data);
    // 6 bytes plus null
    TextOut(0, LCD_LINE1, StrCat(
        FormatNum("%2.2x", data[0]),
        FormatNum("%2.2x", data[1]),
        FormatNum("%2.2x", data[2]),
        FormatNum("%2.2x", data[3]),
        FormatNum("%2.2x", data[4]),
        FormatNum("%2.2x", data[5])));
    while (true);
}
```

9.163 ex_GetBTConnectionAddress.nxc

This is an example of how to use the [GetBTConnectionAddress](#) function.

```
GetBTConnectionAddress(0, buffer);
```

9.164 ex_BTDeviceAddress.nxc

This is an example of how to use the [GetBTDeviceAddress](#) function.

```
GetBTDeviceAddress(0, buffer);
```

9.165 ex_BTInputBuffer.nxc

This is an example of how to use the [GetBTInputBuffer](#) function.

```
GetBTInputBuffer(0, 10, buffer);
```

9.166 ex_BTOutputBuffer.nxc

This is an example of how to use the [GetBTOutputBuffer](#) function.

```
GetBTOutputBuffer(0, 10, buffer);
```

9.167 ex_getc.nxc

This is an example of how to use the [getc](#) function.

```
char val = getc(handle);
```

9.168 ex_getchar.nxc

This is an example of how to use the [getchar](#) function.

```
task main()
{
    SetLongAbort(true);
    while (true) {
        NumOut(0, LCD_LINE1, getchar(), true);
        Wait(MS_5);
    }
}
```

9.169 ex_GetDisplayNormal.nxc

This is an example of how to use the [GetDisplayNormal](#) function.

```
GetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

9.170 ex_GetDisplayPopup.nxc

This is an example of how to use the [GetDisplayPopup](#) function.

```
GetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

9.171 ex_GetHSInputBuffer.nxc

This is an example of how to use the [GetHSInputBuffer](#) function.

```
GetHSInputBuffer(0, 10, buffer);
```

9.172 ex_GetHSOutputBuffer.nxc

This is an example of how to use the [GetHSOutputBuffer](#) function.

```
GetHSOutputBuffer(0, 10, buffer);
```

9.173 ex_GetInput.nxc

This is an example of how to use the [GetInput](#) function.

```
x = GetInput(S1, Type);
```

9.174 ex_GetLSInputBuffer.nxc

This is an example of how to use the [GetLSInputBuffer](#) function.

```
GetLSInputBuffer(S1, 0, 8, buffer);
```

9.175 ex_LSOutputBuffer.nxc

This is an example of how to use the [GetLSOutputBuffer](#) function.

```
GetLSOutputBuffer(S1, 0, 8, outbuffer);
```

9.176 ex_getoutput.nxc

This is an example of how to use the [GetOutput](#) function.

```
x = GetOutput(OUT_A, TachoLimit);
```

9.177 ex_GetUSBInputBuffer.nxc

This is an example of how to use the [GetUSBInputBuffer](#) function.

```
GetUSBInputBuffer(0, 10, buffer);
```

9.178 ex_GetUSBOutputBuffer.nxc

This is an example of how to use the [GetUSBOutputBuffer](#) function.

```
GetUSBOutputBuffer(0, 10, buffer);
```

9.179 ex_GetUSBPollBuffer.nxc

This is an example of how to use the [GetUSBPollBuffer](#) function.

```
GetUSBPollBuffer(0, 10, buffer);
```

9.180 ex_GraphicOut.nxc

This is an example of how to use the [GraphicOut](#) function.

```
GraphicOut(40, 40, "image.ric");
```

9.181 ex_GraphicOutEx.nxc

This is an example of how to use the [GraphicOutEx](#) function.

```
GraphicOutEx(40, 40, "image.ric", variables);
```

9.182 ex_HSFlags.nxc

This is an example of how to use the [HSFlags](#) function.

```
byte x = HSFlags();
```

9.183 ex_HSInputBufferInPtr.nxc

This is an example of how to use the [HSInputBufferInPtr](#) function.

```
byte x = HSInputBufferInPtr();
```

9.184 ex_HSInputBufferOutPtr.nxc

This is an example of how to use the [HSInputBufferOutPtr](#) function.

```
byte x = HSInputBufferOutPtr();
```

9.185 ex_HSMODE.nxc

This is an example of how to use the [HSMODE](#) function.

```
int mode = HSMODE();
```

9.186 ex_HSOOutputBufferInPtr.nxc

This is an example of how to use the [HSOutputBufferInPtr](#) function.

```
byte x = HSOutputBufferInPtr();
```

9.187 ex_HSOutputBufferOutPtr.nxc

This is an example of how to use the [HSOutputBufferOutPtr](#) function.

```
byte x = HSOutputBufferOutPtr();
```

9.188 ex_HSSpeed.nxc

This is an example of how to use the [HSSpeed](#) function.

```
byte x = HSSpeed();
```

9.189 ex_HSState.nxc

This is an example of how to use the [HSState](#) function.

```
byte x = HSState();
```

9.190 ex_HTIRTrain.nxc

This is an example of how to use the [HTIRTrain](#) function.

```
HTIRTrain(S1, TRAIN_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

9.191 ex_HTPFComboDirect.nxc

This is an example of how to use the [HTPFComboDirect](#) function.

```
HTPFComboDirect(S1, PF_CHANNEL_1, PF_CMD_STOP, PF_CMD_FWD);
```

9.192 ex_HTPFComboPWM.nxc

This is an example of how to use the [HTPFComboPWM](#) function.

```
HTPFComboPWM(S1, PF_CHANNEL_1, PF_PWM_REV4, PF_PWM_FWD5);
```

9.193 ex_HTPFRawOutput.nxc

This is an example of how to use the [HTPFRawOutput](#) function.

```
HTPFRawOutput(S1, 0x0a, 0x01, 0x02);
```

9.194 ex_HTPFRepeat.nxc

This is an example of how to use the [HTPFRepeat](#) function.

```
HTPFRepeat(S1, 5, 100);
```

9.195 ex_HTPFSingleOutputCST.nxc

This is an example of how to use the [HTPFSingleOutputCST](#) function.

```
HTPFSingleOutputCST(S1, PF_CHANNEL_1, PF_OUT_A, PF_CST_SET1_SET2);
```

9.196 ex_HTPFSingleOutputPWM.nxc

This is an example of how to use the [HTPFSingleOutputPWM](#) function.

```
HTPFSingleOutputPWM(S1, PF_CHANNEL_1, PF_OUT_A, PF_PWM_FWD5);
```

9.197 ex_HTPFSinglePin.nxc

This is an example of how to use the [HTPFSinglePin](#) function.

```
HTPFSinglePin(S1, PF_CHANNEL_1, PF_OUT_A, PF_PIN_C1, PF_FUNC_SET, true);
```

9.198 ex_HTPFTrain.nxc

This is an example of how to use the [HTPFTTrain](#) function.

```
HTPFTTrain(S1, PF_CHANNEL_1, TRAIN_FUNC_INCR_SPEED);
```

9.199 ex_HTRCXAddToDatalog.nxc

This is an example of how to use the [HTRCXAddToDatalog](#) function.

```
HTRCXAddToDatalog(RCX_InputValueSrc, S1);
```

9.200 ex_HTRCXBatteryLevel.nxc

This is an example of how to use the [HTRCXBatteryLevel](#) function.

```
x = HTRCXBatteryLevel();
```

9.201 ex_HTRCXClearAllEvents.nxc

This is an example of how to use the [HTRCXClearAllEvents](#) function.

```
HTRCXClearAllEvents();
```

9.202 ex_HTRCXClearCounter.nxc

This is an example of how to use the [HTRCXClearCounter](#) function.

```
HTRCXClearCounter(0);
```

9.203 ex_HTRCXClearMsg.nxc

This is an example of how to use the [HTRCXClearMsg](#) function.

```
HTRCXClearMsg();
```

9.204 ex_HTRCXClearSensor.nxc

This is an example of how to use the [HTRCXClearSensor](#) function.

```
HTRCXClearSensor(S1);
```

9.205 ex_HTRCXClearSound.nxc

This is an example of how to use the [HTRCXClearSound](#) function.

```
HTRCXClearSound();
```

9.206 ex_HTRCXClearTimer.nxc

This is an example of how to use the [HTRCXClearTimer](#) function.

```
HTRCXClearTimer(0);
```

9.207 ex_HTRCXCreateDatalog.nxc

This is an example of how to use the [HTRCXCreateDatalog](#) function.

```
HTRCXCreateDatalog(50);
```

9.208 ex_HTRCXDecCounter.nxc

This is an example of how to use the [HTRCXDecCounter](#) function.

```
HTRCXDecCounter(0);
```

9.209 ex_HTRCXDeleteSub.nxc

This is an example of how to use the [HTRCXDeleteSub](#) function.

```
HTRCXDeleteSub(2);
```

9.210 ex_HTRCXDeleteSubs.nxc

This is an example of how to use the [HTRCXDeleteSubs](#) function.

```
HTRCXDeleteSubs();
```

9.211 ex_HTRCXDeleteTask.nxc

This is an example of how to use the [HTRCXDeleteTask](#) function.

```
HTRCXDeleteTask(3);
```

9.212 ex_HTRCXDeleteTasks.nxc

This is an example of how to use the [HTRCXDeleteTasks](#) function.

```
HTRCXDeleteTasks();
```

9.213 ex_HTRCXDisableOutput.nxc

This is an example of how to use the [HTRCXDisableOutput](#) function.

```
HTRCXDisableOutput(RCX_OUT_A);
```

9.214 ex_HTRCXEnableOutput.nxc

This is an example of how to use the [HTRCXEnableOutput](#) function.

```
HTRCXEnableOutput(RCX_OUT_A);
```

9.215 ex_HTRCXEvent.nxc

This is an example of how to use the [HTRCXEvent](#) function.

```
HTRCXEvent(RCX_ConstantSrc, 2);
```

9.216 ex_HTRCXFloat.nxc

This is an example of how to use the [HTRCXFloat](#) function.

```
HTRCXFloat(RCX_OUT_A);
```

9.217 ex_HTRCXFwd.nxc

This is an example of how to use the [HTRCXFwd](#) function.

```
HTRCXFwd (RCX_OUT_A) ;
```

9.218 ex_HTRCXIncCounter.nxc

This is an example of how to use the [HTRCXIncCounter](#) function.

```
HTRCXIncCounter (0) ;
```

9.219 ex_HTRCXInvertOutput.nxc

This is an example of how to use the [HTRCXInvertOutput](#) function.

```
HTRCXInvertOutput (RCX_OUT_A) ;
```

9.220 ex_HTRCXMuteSound.nxc

This is an example of how to use the [HTRCXMuteSound](#) function.

```
HTRCXMuteSound () ;
```

9.221 ex_HTRCXObvertOutput.nxc

This is an example of how to use the [HTRCXObvertOutput](#) function.

```
HTRCXObvertOutput (RCX_OUT_A) ;
```

9.222 ex_HTRCXOff.nxc

This is an example of how to use the [HTRCXOff](#) function.

```
HTRCXOff (RCX_OUT_A) ;
```

9.223 ex_HTRCXOn.nxc

This is an example of how to use the [HTRCXOn](#) function.

```
HTRCXOn (RCX_OUT_A) ;
```

9.224 ex_HTRCXOnFor.nxc

This is an example of how to use the [HTRCXOnFor](#) function.

```
HTRCXOnFor (RCX_OUT_A, 100) ;
```

9.225 ex_HTRCXOnFwd.nxc

This is an example of how to use the [HTRCXOnFwd](#) function.

```
HTRCXOnFwd (RCX_OUT_A) ;
```

9.226 ex_HTRCXOnRev.nxc

This is an example of how to use the [HTRCXOnRev](#) function.

```
HTRCXOnRev (RCX_OUT_A) ;
```

9.227 ex_HTRCXPBTurnOff.nxc

This is an example of how to use the [HTRCXPBTurnOff](#) function.

```
HTRCXPBTurnOff () ;
```

9.228 ex_HTRCXPing.nxc

This is an example of how to use the [HTRCXPing](#) function.

```
HTRCXPing () ;
```

9.229 ex_HTRCXPlaySound.nxc

This is an example of how to use the [HTRCXPlaySound](#) function.

```
HTRCXPlaySound(RCX_SOUND_UP);
```

9.230 ex_HTRCXPlayTone.nxc

This is an example of how to use the [HTRCXPlayTone](#) function.

```
HTRCXPlayTone(440, 100);
```

9.231 ex_HTRCXPlayToneVar.nxc

This is an example of how to use the [HTRCXPlayToneVar](#) function.

```
HTRCXPlayToneVar(0, 50);
```

9.232 ex_HTRCXPoll.nxc

This is an example of how to use the [HTRCXPoll](#) function.

```
x = HTRCXPoll(RCX_VariableSrc, 0);
```

9.233 ex_HTRCXPollMemory.nxc

This is an example of how to use the [HTRCXPollMemory](#) function.

```
HTRCXPollMemory(0, 10);
```

9.234 ex_HTRCXRemote.nxc

This is an example of how to use the [HTRCXRemote](#) function.

```
HTRCXRemote(RCX_RemotePlayASound);
```

9.235 ex_HTRCXRev.nxc

This is an example of how to use the [HTRCXRev](#) function.

```
HTRCXRev (RCX_OUT_A) ;
```

9.236 ex_HTRCXSelectDisplay.nxc

This is an example of how to use the [HTRCXSelectDisplay](#) function.

```
HTRCXSelectDisplay (RCX_VariableSrc, 2) ;
```

9.237 ex_HTRCXSelectProgram.nxc

This is an example of how to use the [HTRCXSelectProgram](#) function.

```
HTRCXSelectProgram (3) ;
```

9.238 ex_HTRCXSendSerial.nxc

This is an example of how to use the [HTRCXSendSerial](#) function.

```
HTRCXSendSerial (0, 10) ;
```

9.239 ex_HTRCXSetDirection.nxc

This is an example of how to use the [HTRCXSetDirection](#) function.

```
HTRCXSetDirection (RCX_OUT_A, RCX_OUT_FWD) ;
```

9.240 ex_HTRCXSetEvent.nxc

This is an example of how to use the [HTRCXSetEvent](#) function.

```
HTRCXSetEvent (0, RCX_ConstantSrc, 5) ;
```

9.241 ex_HTRCXSetGlobalDirection.nxc

This is an example of how to use the [HTRCXSetGlobalDirection](#) function.

```
HTRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

9.242 ex_HTRCXSetGlobalOutput.nxc

This is an example of how to use the [HTRCXSetGlobalOutput](#) function.

```
HTRCXSetGlobalOutput(RCX_OUT_A, RCX_OUT_ON);
```

9.243 ex_HTRCXSetIRLinkPort.nxc

This is an example of how to use the [HTRCXSetIRLinkPort](#) function.

```
SetSensorLowspeed(S1);
```

9.244 ex_HTRCXSetMaxPower.nxc

This is an example of how to use the [HTRCXSetMaxPower](#) function.

```
HTRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

9.245 ex_HTRCXSetMessage.nxc

This is an example of how to use the [HTRCXSetMessage](#) function.

```
HTRCXSetMessage(20);
```

9.246 ex_HTRCXSetOutput.nxc

This is an example of how to use the [HTRCXSetOutput](#) function.

```
HTRCXSetOutput(RCX_OUT_A, RCX_OUT_ON);
```

9.247 ex_HTRCXSetPower.nxc

This is an example of how to use the [HTRCXSetPower](#) function.

```
HTRCXSetPower(RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

9.248 ex_HTRCXSetPriority.nxc

This is an example of how to use the [HTRCXSetPriority](#) function.

```
HTRCXSetPriority(2);
```

9.249 ex_HTRCXSetSensorMode.nxc

This is an example of how to use the [HTRCXSetSensorMode](#) function.

```
HTRCXSetSensorMode(S1, SENSOR_MODE_BOOL);
```

9.250 ex_HTRCXSetSensorType.nxc

This is an example of how to use the [HTRCXSetSensorType](#) function.

```
HTRCXSetSensorType(S1, SENSOR_TYPE_TOUCH);
```

9.251 ex_HTRCXSetSleepTime.nxc

This is an example of how to use the [HTRCXSetSleepTime](#) function.

```
HTRCXSetSleepTime(4);
```

9.252 ex_HTRCXSetTxPower.nxc

This is an example of how to use the [HTRCXSetTxPower](#) function.

```
HTRCXSetTxPower(0);
```

9.253 ex_HTRCXSetWatch.nxc

This is an example of how to use the [HTRCXSetWatch](#) function.

```
HTRCXSetWatch(3, 30);
```

9.254 ex_HTRCXStartTask.nxc

This is an example of how to use the [HTRCXStartTask](#) function.

```
HTRCXStartTask(2);
```

9.255 ex_HTRCXStopAllTasks.nxc

This is an example of how to use the [HTRCXStopAllTasks](#) function.

```
HTRCXStopAllTasks();
```

9.256 ex_HTRCXStopTask.nxc

This is an example of how to use the [HTRCXStopTask](#) function.

```
HTRCXStopTask(1);
```

9.257 ex_HTRCXToggle.nxc

This is an example of how to use the [HTRCXToggle](#) function.

```
HTRCXToggle(RCX_OUT_A);
```

9.258 ex_HTRCXUnmuteSound.nxc

This is an example of how to use the [HTRCXUnmuteSound](#) function.

```
HTRCXUnmuteSound();
```

9.259 ex_HTScoutCalibrateSensor.nxc

This is an example of how to use the [HTScoutCalibrateSensor](#) function.

```
HTScoutCalibrateSensor();
```

9.260 ex_HTScoutMuteSound.nxc

This is an example of how to use the [HTScoutMuteSound](#) function.

```
HTScoutMuteSound();
```

9.261 ex_HTScoutSelectSounds.nxc

This is an example of how to use the [HTScoutSelectSounds](#) function.

```
HTScoutSelectSounds(0);
```

9.262 ex_HTScoutSendVLL.nxc

This is an example of how to use the [HTScoutSendVLL](#) function.

```
HTScoutSendVLL(RCX_ConstantSrc, 0x30);
```

9.263 ex_HTScoutSetEventFeedback.nxc

This is an example of how to use the [HTScoutSetEventFeedback](#) function.

```
HTScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

9.264 ex_HTScoutSetLight.nxc

This is an example of how to use the [HTScoutSetLight](#) function.

```
HTScoutSetLight(SCOUT_LIGHT_ON);
```

9.265 ex_HTScoutSetScoutMode.nxc

This is an example of how to use the [HTScoutSetScoutMode](#) function.

```
HTScoutSetScoutMode (SCOUT_MODE_POWER);
```

9.266 ex_HTScoutSetSensorClickTime.nxc

This is an example of how to use the [HTScoutSetSensorClickTime](#) function.

```
HTScoutSetSensorClickTime (RCX_ConstantSrc, 200);
```

9.267 ex_HTScoutSetSensorHysteresis.nxc

This is an example of how to use the [HTScoutSetSensorHysteresis](#) function.

```
HTScoutSetSensorHysteresis (RCX_ConstantSrc, 50);
```

9.268 ex_HTScoutSetSensorLowerLimit.nxc

This is an example of how to use the [HTScoutSetSensorLowerLimit](#) function.

```
HTScoutSetSensorLowerLimit (RCX_VariableSrc, 0);
```

9.269 ex_HTScoutSetSensorUpperLimit.nxc

This is an example of how to use the [HTScoutSetSensorUpperLimit](#) function.

```
HTScoutSetSensorUpperLimit (RCX_VariableSrc, 0);
```

9.270 ex_HTScoutUnmuteSound.nxc

This is an example of how to use the [HTScoutUnmuteSound](#) function.

```
HTScoutUnmuteSound();
```

9.271 ex_I2CBytes.nxc

This is an example of how to use the [I2CBytes](#) function.

```
x = I2CBytes(S4, writebuf, cnt, readbuf);
```

9.272 ex_I2CBytesReady.nxc

This is an example of how to use the [I2CBytesReady](#) function.

```
x = I2CBytesReady(S1);
```

9.273 ex_I2CCheckStatus.nxc

This is an example of how to use the [I2CCheckStatus](#) function.

```
x = I2CCheckStatus(S1);
```

9.274 ex_I2CDeviceId.nxc

This is an example of how to use the [I2CDeviceId](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    while (true) {
        TextOut(0, LCD_LINE1, I2CVendorId(S1, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
    }
}
```

9.275 ex_I2CDeviceInfo.nxc

This is an example of how to use the [I2CDeviceInfo](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    TextOut(0, LCD_LINE1, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_DEVICE_ID));
    TextOut(0, LCD_LINE2, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_VENDOR_ID));
    TextOut(0, LCD_LINE3, I2CDeviceInfo(S1, I2C_ADDR_DEFAULT, I2C_REG_VERSION));
    Wait(SEC_10);
}
```

9.276 ex_I2CRead.nxc

This is an example of how to use the [I2CRead](#) function.

```
x = I2CRead(S1, 1, outbuffer);
```

9.277 ex_I2CSendCommand.nxc

This is an example of how to use the [I2CSendCommand](#) function.

```
long result = I2CSendCommand(S1, I2C_ADDR_DEFAULT, HT_CMD_COLOR2_ACTIVE);
```

9.278 ex_I2CStatus.nxc

This is an example of how to use the [I2CStatus](#) function.

```
x = I2CStatus(S1, nRead);
```

9.279 ex_I2CVendorId.nxc

This is an example of how to use the [I2CVendorId](#) function.

```
task main()
{
    SetSensorLowspeed(S1);
    while (true) {
        TextOut(0, LCD_LINE1, I2CVendorId(S1, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
    }
}
```

9.280 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, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE2, I2CDeviceId(S1, I2C_ADDR_DEFAULT));
        TextOut(0, LCD_LINE3, I2CVersion(S1, I2C_ADDR_DEFAULT));
    }
}
```

9.281 ex_i2cwrite.nxc

This is an example of how to use the [I2CWrite](#) function.

```
x = I2CWrite(S1, 1, inbuffer);
```

9.282 ex_isalnum.nxc

This is an example of how to use the [isalnum](#) function.

```
i = isalnum(x);
```

9.283 ex_isalpha.nxc

This is an example of how to use the [isalpha](#) function.

```
i = isalpha(x);
```

9.284 ex_iscntrl.nxc

This is an example of how to use the [iscntrl](#) function.

```
i = iscntrl(x);
```

9.285 ex_isdigit.nxc

This is an example of how to use the [isdigit](#) function.

```
i = isdigit(x);
```

9.286 ex_isgraph.nxc

This is an example of how to use the [isgraph](#) function.

```
i = isgraph(x);
```

9.287 ex_islower.nxc

This is an example of how to use the [islower](#) function.

```
i = islower(x);
```

9.288 ex_isNAN.nxc

This is an example of how to use the [isnan](#) function.

```
task main()
{
    float j = -1;
    float f = sqrt(j);
    if (isnan(f))
        TextOut(0, LCD_LINE1, "not a number");
    else
        NumOut(0, LCD_LINE1, f);
    NumOut(0, LCD_LINE2, f);
    Wait(SEC_5);
}
```

9.289 ex_isprint.nxc

This is an example of how to use the [isprint](#) function.

```
i = isprint(x);
```

9.290 ex_ispunct.nxc

This is an example of how to use the [ispunct](#) function.

```
i = ispunct(x);
```

9.291 ex_isspace.nxc

This is an example of how to use the [isspace](#) function.

```
i = isspace(x);
```

9.292 ex_isupper.nxc

This is an example of how to use the [isupper](#) function.

```
i = isupper(x);
```

9.293 ex_isxdigit.nxc

This is an example of how to use the [isxdigit](#) function.

```
i = isxdigit(x);
```

9.294 ex_labs.nxc

This is an example of how to use the [labs](#) function.

```
task main()
{
    float j = -1;
    float f = sqrt(j);
    if (isnan(f))
        TextOut(0, LCD_LINE1, "not a number");
    else
        NumOut(0, LCD_LINE1, f);
    NumOut(0, LCD_LINE2, f);
    Wait(SEC_5);
}
```

9.295 ex_ldiv.nxc

This is an example of how to use the [ldiv](#) function.

```
task main()
{
    long x, y;
    x = 314564;
    y = 33;
    ldiv_t r;
    r = ldiv(x, y);
    NumOut(0, LCD_LINE1, r.quot);
    NumOut(0, LCD_LINE2, r.rem);
    Wait(SEC_3);
}
```

9.296 ex_leftstr.nxc

This is an example of how to use the [LeftStr](#) function.

```
task main()
{
    string s = "Now is the winter of our discontent";
    TextOut(0, LCD_LINE1, LeftStr(s, 12));
    Wait(SEC_4);
}
```

9.297 ex_LineOut.nxc

This is an example of how to use the [LineOut](#) function.

```
task main()
{
    repeat(10) {
        LineOut(0, 0, DISPLAY_WIDTH, DISPLAY_HEIGHT, DRAW_OPT_LOGICAL_XOR);
        Wait(SEC_2);
    }
}
```

9.298 ex_log.nxc

This is an example of how to use the [log](#) function.

```
y = log(x);
```

9.299 ex_log10.nxc

This is an example of how to use the [log10](#) function.

```
y = log10(x);
```

9.300 ex_LongAbort.nxc

This is an example of how to use the [LongAbort](#) function.

```
x = LongAbort();
```

9.301 ex_LowspeedBytesReady.nxc

This is an example of how to use the [LowspeedBytesReady](#) function.

```
x = LowspeedBytesReady(S1);
```

9.302 ex_LowspeedCheckStatus.nxc

This is an example of how to use the [LowspeedCheckStatus](#) function.

```
x = LowspeedCheckStatus(S1);
```

9.303 ex_LowspeedRead.nxc

This is an example of how to use the [LowspeedRead](#) function.

```
x = LowspeedRead(S1, 1, outbuffer);
```

9.304 ex_lowspeedstatus.nxc

This is an example of how to use the [LowspeedStatus](#) function.

```
x = LowspeedStatus(S1, nRead);
```

9.305 ex_LowspeedWrite.nxc

This is an example of how to use the [LowspeedWrite](#) function.

```
x = LowspeedWrite(S1, 1, inbuffer);
```

9.306 ex_LSChannelState.nxc

This is an example of how to use the [LSChannelState](#) function.

```
x = LSChannelState(S1);
```

9.307 ex_LSErrorType.nxc

This is an example of how to use the [LSErrorType](#) function.

```
x = LSErrorType(S1);
```

9.308 ex_LSInputBufferBytesToRx.nxc

This is an example of how to use the [LSInputBufferBytesToRx](#) function.

```
x = LSInputBufferBytesToRx(S1);
```

9.309 ex_LSInputBufferInPtr.nxc

This is an example of how to use the [LSInputBufferInPtr](#) function.

```
x = LSInputBufferInPtr(S1);
```

9.310 ex_LSInputBufferOutPtr.nxc

This is an example of how to use the [LSInputBufferOutPtr](#) function.

```
x = LSInputBufferOutPtr(S1);
```

9.311 ex_LSMODE.nxc

This is an example of how to use the [LSMode](#) function.

```
x = LSMODE(S1);
```

9.312 ex_LSNorestartOnRead.nxc

This is an example of how to use the [LSNoRestartOnRead](#) function.

```
byte val = LSNoRestartOnRead();
```

9.313 ex_LSOOutputBufferBytesToRx.nxc

This is an example of how to use the [LSOutputBufferBytesToRx](#) function.

```
x = LSOOutputBufferBytesToRx(S1);
```

9.314 ex_LSOOutputBufferInPtr.nxc

This is an example of how to use the [LSOutputBufferInPtr](#) function.

```
x = LSOOutputBufferInPtr(S1);
```

9.315 ex_LSOOutputBufferOutPtr.nxc

This is an example of how to use the [LSOutputBufferOutPtr](#) function.

```
x = LSOOutputBufferOutPtr(S1);
```

9.316 ex_LSSpeed.nxc

This is an example of how to use the [LSSpeed](#) function.

```
x = LSSpeed();
```

9.317 ex_LSState.nxc

This is an example of how to use the [LSState](#) function.

```
x = LSState();
```

9.318 ex_memcmp.nxc

This is an example of how to use the [memcmp](#) function.

```
task main()
{
    byte myArray[] = {1, 2, 3, 4};
    byte x[] = {1, 2, 3, 5};
```

```
int i = 5;
int j;
j = memcmp(myArray, x, 1); // returns -1, 0, or 1
NumOut(0, LCD_LINE1, i);
NumOut(0, LCD_LINE2, j);
NumOut(0, LCD_LINE3, memcmp(i, j, 1));
Wait(SEC_15);
}
```

9.319 ex_memcpy.nxc

This is an example of how to use the [memcpy](#) function.

```
memcpy(myArray, anotherArray, 1);
```

9.320 ex_memmove.nxc

This is an example of how to use the [memmove](#) function.

```
memmove(myArray, anotherArray, 1);
```

9.321 ex_midstr.nxc

This is an example of how to use the [MidStr](#) function.

```
task main()
{
    string s = "Now is the winter of our discontent";
    TextOut(0, LCD_LINE1, MidStr(s, 12, 5));
    Wait(SEC_4);
}
```

9.322 ex_motoractualspeed.nxc

This is an example of how to use the [MotorActualSpeed](#) function.

```
x = MotorActualSpeed(OUT_A);
```

9.323 ex_motorblocktachocount.nxc

This is an example of how to use the [MotorBlockTachoCount](#) function.

```
x = MotorBlockTachoCount(OUT_A);
```

9.324 ex_motormode.nxc

This is an example of how to use the [MotorMode](#) function.

```
x = MotorMode(OUT_A);
```

9.325 ex_motoroverload.nxc

This is an example of how to use the [MotorOverload](#) function.

```
x = MotorOverload(OUT_A);
```

9.326 ex_motorpower.nxc

This is an example of how to use the [MotorPower](#) function.

```
x = MotorPower(OUT_A);
```

9.327 ex_motorpwnfreq.nxc

This is an example of how to use the [MotorPwnFreq](#) function.

```
x = MotorPwnFreq();
```

9.328 ex_motorregdvalue.nxc

This is an example of how to use the [MotorRegDValue](#) function.

```
x = MotorRegDValue(OUT_A);
```

9.329 ex_motorregivalue.nxc

This is an example of how to use the [MotorRegIValue](#) function.

```
x = MotorRegIValue(OUT_A);
```

9.330 ex_motorregpvalue.nxc

This is an example of how to use the [MotorRegPValue](#) function.

```
x = MotorRegPValue(OUT_A);
```

9.331 ex_motorregulation.nxc

This is an example of how to use the [MotorRegulation](#) function.

```
x = MotorRegulation(OUT_A);
```

9.332 ex_motorrotationcount.nxc

This is an example of how to use the [MotorRotationCount](#) function.

```
x = MotorRotationCount(OUT_A);
```

9.333 ex_motorrunstate.nxc

This is an example of how to use the [MotorRunState](#) function.

```
x = MotorRunState(OUT_A);
```

9.334 ex_motortachocount.nxc

This is an example of how to use the [MotorTachoCount](#) function.

```
x = MotorTachoCount(OUT_A);
```

9.335 ex_motortacholimit.nxc

This is an example of how to use the [MotorTachoLimit](#) function.

```
x = MotorTachoLimit(OUT_A);
```

9.336 ex_motorturnratio.nxc

This is an example of how to use the [MotorTurnRatio](#) function.

```
x = MotorTurnRatio(OUT_A);
```

9.337 ex_MSADPAOff.nxc

This is an example of how to use the [MSADPAOff](#) function.

```
char result = MSADPAOff(S1, MS_ADDR_DISTNX);
```

9.338 ex_MSADPAOn.nxc

This is an example of how to use the [MSADPAOn](#) function.

```
char result = MSADPAOn(S1, MS_ADDR_DISTNX);
```

9.339 ex_MSDeenergize.nxc

This is an example of how to use the [MSDeenergize](#) function.

```
char result = MSDeenergize(S1, I2C_ADDR_DEFAULT);
```

9.340 ex_MSEnergize.nxc

This is an example of how to use the [MSEnergize](#) function.

```
char result = MSEnergize(S1, I2C_ADDR_DEFAULT);
```

9.341 ex_MSIRTrain.nxc

This is an example of how to use the [MSIRTrain](#) function.

```
char result = MSIRTrain(S1, I2C_ADDR_DEFAULT, TRAIN_CHANNEL_1,
                        TRAIN_FUNC_INCR_SPEED);
```

9.342 ex_MSPFComboDirect.nxc

This is an example of how to use the [MSPFComboDirect](#) function.

```
char result = MSPFComboDirect(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_CMD_STOP,
                               PF_CMD_FWD);
```

9.343 ex_MSPFComboPWM.nxc

This is an example of how to use the [MSPFComboPWM](#) function.

```
char result = MSPFComboPWM(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_PWM_REV4,  
                           PF_PWM_FWD5);
```

9.344 ex_MSPFRawOutput.nxc

This is an example of how to use the [MSPFRawOutput](#) function.

```
char result = MSPFRawOutput(S1, I2C_ADDR_DEFAULT, 0x0a, 0x01, 0x02);
```

9.345 ex_MSPFRepeat.nxc

This is an example of how to use the [MSPFRepeat](#) function.

```
char result = MSPFRepeat(S1, I2C_ADDR_DEFAULT, 5, 100);
```

9.346 ex_MSPFSingleOutputCST.nxc

This is an example of how to use the [MSPFSingleOutputCST](#) function.

```
char result = MSPFSingleOutputCST(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
                                   PF_CST_SET1_SET2);
```

9.347 ex_MSPFSingleOutputPWM.nxc

This is an example of how to use the [MSPFSingleOutputPWM](#) function.

```
char result = MSPFSingleOutputPWM(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
                                   PF_PWM_FWD5);
```

9.348 ex_MSPFSinglePin.nxc

This is an example of how to use the [MSPFSinglePin](#) function.

```
char result = MSPFSinglePin(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1, PF_OUT_A,  
                           PF_PIN_C1, PF_FUNC_SET, true);
```

9.349 ex_MSPFTrain.nxc

This is an example of how to use the [MSPFTrain](#) function.

```
char result = MSPFTrain(S1, I2C_ADDR_DEFAULT, PF_CHANNEL_1,  
                        TRAIN_FUNC_INCR_SPEED);
```

9.350 ex_MSRCXAbsVar.nxc

This is an example of how to use the [MSRCXAbsVar](#) function.

```
MSRCXAbsVar(0, RCX_VariableSrc, 0);
```

9.351 ex_MSRCXAddToDatalog.nxc

This is an example of how to use the [MSRCXAddToDatalog](#) function.

```
MSRCXAddToDatalog(RCX_InputValueSrc, S1);
```

9.352 ex_MSRCXAndVar.nxc

This is an example of how to use the [MSRCXAndVar](#) function.

```
MSRCXAndVar(0, RCX_ConstantSrc, 0x7f);
```

9.353 ex_MSRCXBatteryLevel.nxc

This is an example of how to use the [MSRCXBatteryLevel](#) function.

```
x = MSRCXBatteryLevel();
```

9.354 ex_MSRCXBoot.nxc

This is an example of how to use the [MSRCXBoot](#) function.

```
MSRCXBoot();
```

9.355 ex_MSRCXCalibrateEvent.nxc

This is an example of how to use the [MSRCXCalibrateEvent](#) function.

```
MSRCXCalibrateEvent (0, 200, 500, 50);
```

9.356 ex_MSRCXClearAllEvents.nxc

This is an example of how to use the [MSRCXClearAllEvents](#) function.

```
MSRCXClearAllEvents();
```

9.357 ex_MSRCXClearCounter.nxc

This is an example of how to use the [MSRCXClearCounter](#) function.

```
MSRCXClearCounter (0);
```

9.358 ex_MSRCXClearMsg.nxc

This is an example of how to use the [MSRCXClearMsg](#) function.

```
MSRCXClearMsg();
```

9.359 ex_MSRCXClearSensor.nxc

This is an example of how to use the [MSRCXClearSensor](#) function.

```
MSRCXClearSensor (S1);
```

9.360 ex_MSRCXClearSound.nxc

This is an example of how to use the [MSRCXClearSound](#) function.

```
MSRCXClearSound();
```

9.361 ex_MSRCXClearTimer.nxc

This is an example of how to use the [MSRCXClearTimer](#) function.

```
MSRCXClearTimer(0);
```

9.362 ex_MSRCXCreateDatalog.nxc

This is an example of how to use the [MSRCXCreateDatalog](#) function.

```
MSRCXCreateDatalog(50);
```

9.363 ex_MSRCXDecCounter.nxc

This is an example of how to use the [MSRCXDecCounter](#) function.

```
MSRCXDecCounter(0);
```

9.364 ex_MSRCXDeleteSub.nxc

This is an example of how to use the [MSRCXDeleteSub](#) function.

```
MSRCXDeleteSub(2);
```

9.365 ex_MSRCXDeleteSubs.nxc

This is an example of how to use the [MSRCXDeleteSubs](#) function.

```
MSRCXDeleteSubs();
```

9.366 ex_MSRCXDeleteTask.nxc

This is an example of how to use the [MSRCXDeleteTask](#) function.

```
MSRCXDeleteTask(3);
```

9.367 ex_MSRCXDeleteTasks.nxc

This is an example of how to use the [MSRCXDeleteTasks](#) function.

```
MSRCXDeleteTasks();
```

9.368 ex_MSRCXDisableOutput.nxc

This is an example of how to use the [MSRCXDisableOutput](#) function.

```
MSRCXDisableOutput(RCX_OUT_A);
```

9.369 ex_MSRCXDivVar.nxc

This is an example of how to use the [MSRCXDivVar](#) function.

```
MSRCXDivVar(0, RCX_ConstantSrc, 2);
```

9.370 ex_MSRCXEnableOutput.nxc

This is an example of how to use the [MSRCXEnableOutput](#) function.

```
MSRCXEnableOutput(RCX_OUT_A);
```

9.371 ex_MSRCXEvent.nxc

This is an example of how to use the [MSRCXEvent](#) function.

```
MSRCXEvent(RCX_ConstantSrc, 2);
```

9.372 ex_MSRCXFloat.nxc

This is an example of how to use the [MSRCXFloat](#) function.

```
MSRCXFloat(RCX_OUT_A);
```

9.373 ex_MSRCXFwd.nxc

This is an example of how to use the [MSRCXFwd](#) function.

```
MSRCXFwd (RCX_OUT_A) ;
```

9.374 ex_MSRCXIncCounter.nxc

This is an example of how to use the [MSRCXIncCounter](#) function.

```
MSRCXIncCounter (0) ;
```

9.375 ex_MSRCXInvertOutput.nxc

This is an example of how to use the [MSRCXInvertOutput](#) function.

```
MSRCXInvertOutput (RCX_OUT_A) ;
```

9.376 ex_MSRCXMulVar.nxc

This is an example of how to use the [MSRCXMulVar](#) function.

```
MSRCXMulVar (0, RCX_VariableSrc, 4);
```

9.377 ex_MSRCXMuteSound.nxc

This is an example of how to use the [MSRCXMuteSound](#) function.

```
MSRCXMuteSound () ;
```

9.378 ex_MSRCXObvertOutput.nxc

This is an example of how to use the [MSRCXObvertOutput](#) function.

```
MSRCXObvertOutput (RCX_OUT_A) ;
```

9.379 ex_MSRCXOff.nxc

This is an example of how to use the [MSRCXOff](#) function.

```
MSRCXOff (RCX_OUT_A) ;
```

9.380 ex_MSRCXOn.nxc

This is an example of how to use the [MSRCXOn](#) function.

```
MSRCXOn (RCX_OUT_A) ;
```

9.381 ex_MSRCXOnFor.nxc

This is an example of how to use the [MSRCXOnFor](#) function.

```
MSRCXOnFor (RCX_OUT_A, 100) ;
```

9.382 ex_MSRCXOnFwd.nxc

This is an example of how to use the [MSRCXOnFwd](#) function.

```
MSRCXOnFwd (RCX_OUT_A) ;
```

9.383 ex_MSRCXOnRev.nxc

This is an example of how to use the [MSRCXOnRev](#) function.

```
MSRCXOnRev (RCX_OUT_A) ;
```

9.384 ex_MSRCXOrVar.nxc

This is an example of how to use the [MSRCXOrVar](#) function.

```
MSRCXOrVar (0, RCX_ConstantSrc, 0xCC) ;
```

9.385 ex_MSRCXPBTurnOff.nxc

This is an example of how to use the [MSRCXPBTurnOff](#) function.

```
MSRCXPBTurnOff();
```

9.386 ex_MSRCXPing.nxc

This is an example of how to use the [MSRCXPing](#) function.

```
MSRCXPing();
```

9.387 ex_MSRCXPlaySound.nxc

This is an example of how to use the [MSRCXPlaySound](#) function.

```
MSRCXPlaySound(RCX_SOUND_UP);
```

9.388 ex_MSRCXPlayTone.nxc

This is an example of how to use the [MSRCXPlayTone](#) function.

```
MSRCXPlayTone(440, 100);
```

9.389 ex_MSRCXPlayToneVar.nxc

This is an example of how to use the [MSRCXPlayToneVar](#) function.

```
MSRCXPlayToneVar(0, 50);
```

9.390 ex_MSRCXPoll.nxc

This is an example of how to use the [MSRCXPoll](#) function.

```
x = MSRCXPoll(RCX_VariableSrc, 0);
```

9.391 ex_MSRCXPollMemory.nxc

This is an example of how to use the [MSRCXPollMemory](#) function.

```
MSRCXPollMemory(0, 10);
```

9.392 ex_MSRCXRemote.nxc

This is an example of how to use the [MSRCXRemote](#) function.

```
MSRCXRemote(RCX_RemotePlayASound);
```

9.393 ex_MSRCXReset.nxc

This is an example of how to use the [MSRCXReset](#) function.

```
MSRCXReset();
```

9.394 ex_MSRCXRev.nxc

This is an example of how to use the [MSRCXRev](#) function.

```
MSRCXRev(RCX_OUT_A);
```

9.395 ex_MSRCXSelectDisplay.nxc

This is an example of how to use the [MSRCXSelectDisplay](#) function.

```
MSRCXSelectDisplay(RCX_VariableSrc, 2);
```

9.396 ex_MSRCXSelectProgram.nxc

This is an example of how to use the [MSRCXSelectProgram](#) function.

```
MSRCXSelectProgram(3);
```

9.397 ex_MSRCXSendSerial.nxc

This is an example of how to use the [MSRCXSendSerial](#) function.

```
MSRCXSendSerial(0, 10);
```

9.398 ex_MSRCXSet.nxc

This is an example of how to use the [MSRCXSet](#) function.

```
MSRCXSet(RCX_VariableSrc, 0, RCX_RandomSrc, 10000);
```

9.399 ex_MSRCXSetDirection.nxc

This is an example of how to use the [MSRCXSetDirection](#) function.

```
MSRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

9.400 ex_MSRCXSetEvent.nxc

This is an example of how to use the [MSRCXSetEvent](#) function.

```
MSRCXSetEvent(0, RCX_ConstantSrc, 5);
```

9.401 ex_MSRCXSetGlobalDirection.nxc

This is an example of how to use the [MSRCXSetGlobalDirection](#) function.

```
MSRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

9.402 ex_MSRCXSetGlobalOutput.nxc

This is an example of how to use the [MSRCXSetGlobalOutput](#) function.

```
MSRCXSetGlobalOutput(RCX_OUT_A, RCX_OUT_ON);
```

9.403 ex_MSRCXSetMaxPower.nxc

This is an example of how to use the [MSRCXSetMaxPower](#) function.

```
MSRCXSetMaxPower (RCX_OUT_A, RCX_ConstantSrc, 5);
```

9.404 ex_MSRCXSetMessage.nxc

This is an example of how to use the [MSRCXSetMessage](#) function.

```
MSRCXSetMessage (20);
```

9.405 ex_MSRCXSetNRLinkPort.nxc

This is an example of how to use the [MSRCXSetNRLinkPort](#) function.

```
MSRCXSetNRLinkPort (S1, MS_ADDR_NRLINK);
```

9.406 ex_MSRCXSetOutput.nxc

This is an example of how to use the [MSRCXSetOutput](#) function.

```
MSRCXSetOutput (RCX_OUT_A, RCX_OUT_ON);
```

9.407 ex_MSRCXSetPower.nxc

This is an example of how to use the [MSRCXSetPower](#) function.

```
MSRCXSetPower (RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

9.408 ex_MSRCXSetPriority.nxc

This is an example of how to use the [MSRCXSetPriority](#) function.

```
MSRCXSetPriority (2);
```

9.409 ex_MSRCXSetSensorMode.nxc

This is an example of how to use the [MSRCXSetSensorMode](#) function.

```
MSRCXSetSensorMode(S1, SENSOR_MODE_BOOL);
```

9.410 ex_MSRCXSetSensorType.nxc

This is an example of how to use the [MSRCXSetSensorType](#) function.

```
MSRCXSetSensorType(S1, SENSOR_TYPE_TOUCH);
```

9.411 ex_MSRCXSetSleepTime.nxc

This is an example of how to use the [MSRCXSetSleepTime](#) function.

```
MSRCXSetSleepTime(4);
```

9.412 ex_MSRCXSetTxPower.nxc

This is an example of how to use the [MSRCXSetTxPower](#) function.

```
MSRCXSetTxPower(0);
```

9.413 ex_MSRCXSetUserDisplay.nxc

This is an example of how to use the [MSRCXSetUserDisplay](#) function.

```
MSRCXSetUserDisplay(RCX_VariableSrc, 0, 2);
```

9.414 ex_MSRCXSetVar.nxc

This is an example of how to use the [MSRCXSetVar](#) function.

```
MSRCXSetVar(0, RCX_VariableSrc, 1);
```

9.415 ex_MSRCXSetWatch.nxc

This is an example of how to use the [MSRCXSetWatch](#) function.

```
MSRCXSetWatch(3, 30);
```

9.416 ex_MSRCXSgnVar.nxc

This is an example of how to use the [MSRCXSgnVar](#) function.

```
MSRCXSgnVar(0, RCX_VariableSrc, 0);
```

9.417 ex_MSRCXStartTask.nxc

This is an example of how to use the [MSRCXStartTask](#) function.

```
MSRCXStartTask(2);
```

9.418 ex_MSRCXStopAllTasks.nxc

This is an example of how to use the [MSRCXStopAllTasks](#) function.

```
MSRCXStopAllTasks();
```

9.419 ex_MSRCXStopTask.nxc

This is an example of how to use the [MSRCXStopTask](#) function.

```
MSRCXStopTask(1);
```

9.420 ex_MSRCXSubVar.nxc

This is an example of how to use the [MSRCXSubVar](#) function.

```
MSRCXSubVar(0, RCX_RandomSrc, 10);
```

9.421 ex_MSRCXSumVar.nxc

This is an example of how to use the [MSRCXSumVar](#) function.

```
MSRCXSumVar(0, RCX_InputValueSrc, S1);
```

9.422 ex_MSRCXToggle.nxc

This is an example of how to use the [MSRCXToggle](#) function.

```
MSRCXToggle(RCX_OUT_A);
```

9.423 ex_MSRCXUnlock.nxc

This is an example of how to use the [MSRCXUnlock](#) function.

```
MSRCXUnlock();
```

9.424 ex_MSRCXUnmuteSound.nxc

This is an example of how to use the [MSRCXUnmuteSound](#) function.

```
MSRCXUnmuteSound();
```

9.425 ex_MSReadValue.nxc

This is an example of how to use the [MSReadValue](#) function.

```
byte value = MSReadValue(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, 1);
```

9.426 ex_MSScoutCalibrateSensor.nxc

This is an example of how to use the [MSScoutCalibrateSensor](#) function.

```
MSScoutCalibrateSensor();
```

9.427 ex_MSScoutMuteSound.nxc

This is an example of how to use the [MSScoutMuteSound](#) function.

```
MSScoutMuteSound();
```

9.428 ex_MSScoutSelectSounds.nxc

This is an example of how to use the [MSScoutSelectSounds](#) function.

```
MSScoutSelectSounds(0);
```

9.429 ex_MSScoutSendVLL.nxc

This is an example of how to use the [MSScoutSendVLL](#) function.

```
MSScoutSendVLL(RCX_ConstantSrc, 0x30);
```

9.430 ex_MSScoutSetCounterLimit.nxc

This is an example of how to use the [MSScoutSetCounterLimit](#) function.

```
MSScoutSetCounterLimit(0, RCX_ConstantSrc, 2000);
```

9.431 ex_MSScoutSetEventFeedback.nxc

This is an example of how to use the [MSScoutSetEventFeedback](#) function.

```
MSScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

9.432 ex_MSScoutSetLight.nxc

This is an example of how to use the [MSScoutSetLight](#) function.

```
MSScoutSetLight(SCOUT_LIGMS_ON);
```

9.433 ex_MSScoutSetScoutMode.nxc

This is an example of how to use the [MSScoutSetScoutMode](#) function.

```
MSScoutSetScoutMode(SCOUT_MODE_POWER);
```

9.434 ex_MSScoutSetScoutRules.nxc

This is an example of how to use the [MSScoutSetScoutRules](#) function.

```
MSScoutSetScoutRules(SCOUT_MR_FORWARD, SCOUT_TR_REVERSE, SCOUT_LR_IGNORE,
SCOUT_TGS_SHORT, SCOUT_FXR_BUG);
```

9.435 ex_MSScoutSetSensorClickTime.nxc

This is an example of how to use the [MSScoutSetSensorClickTime](#) function.

```
MSScoutSetSensorClickTime(RCX_ConstantSrc, 200);
```

9.436 ex_MSScoutSetSensorHysteresis.nxc

This is an example of how to use the [MSScoutSetSensorHysteresis](#) function.

```
MSScoutSetSensorHysteresis(RCX_ConstantSrc, 50);
```

9.437 ex_MSScoutSetSensorLowerLimit.nxc

This is an example of how to use the [MSScoutSetSensorLowerLimit](#) function.

```
MSScoutSetSensorLowerLimit(RCX_VariableSrc, 0);
```

9.438 ex_MSScoutSetSensorUpperLimit.nxc

This is an example of how to use the [MSScoutSetSensorUpperLimit](#) function.

```
MSScoutSetSensorUpperLimit(RCX_VariableSrc, 0);
```

9.439 ex_MSScoutSetTimerLimit.nxc

This is an example of how to use the [MSScoutSetTimerLimit](#) function.

```
MSScoutSetTimerLimit(0, RCX_ConstantSrc, 10000);
```

9.440 ex_MSScoutUnmuteSound.nxc

This is an example of how to use the [MSScoutUnmuteSound](#) function.

```
MSScoutUnmuteSound();
```

9.441 ex_muldiv32.nxc

This is an example of how to use the [muldiv32](#) function.

```
y = muldiv32(a, b, c);
```

9.442 ex_NRLink2400.nxc

This is an example of how to use the [NRLink2400](#) function.

```
char result = NRLink2400(S1, MS_ADDR_NRLINK);
```

9.443 ex_NRLink4800.nxc

This is an example of how to use the [NRLink4800](#) function.

```
char result = NRLink4800(S1, MS_ADDR_NRLINK);
```

9.444 ex_NRLinkFlush.nxc

This is an example of how to use the [NRLinkFlush](#) function.

```
char result = NRLinkFlush(S1, MS_ADDR_NRLINK);
```

9.445 ex_NRLinkIRLong.nxc

This is an example of how to use the [NRLinkIRLong](#) function.

```
char result = NRLinkIRLong(S1, MS_ADDR_NRLINK);
```

9.446 ex_NRLinkIRShort.nxc

This is an example of how to use the [NRLinkIRShort](#) function.

```
char result = NRLinkIRShort(S1, MS_ADDR_NRLINK);
```

9.447 ex_NRLinkSetPF.nxc

This is an example of how to use the [NRLinkSetPF](#) function.

```
char result = NRLinkSetPF(S1, MS_ADDR_NRLINK);
```

9.448 ex_NRLinkSetRCX.nxc

This is an example of how to use the [NRLinkSetRCX](#) function.

```
char result = NRLinkSetRCX(S1, MS_ADDR_NRLINK);
```

9.449 ex_NRLinkSetTrain.nxc

This is an example of how to use the [NRLinkSetTrain](#) function.

```
char result = NRLinkSetTrain(S1, MS_ADDR_NRLINK);
```

9.450 ex_NRLinkStatus.nxc

This is an example of how to use the [NRLinkStatus](#) function.

```
byte result = NRLinkStatus(S1, MS_ADDR_NRLINK);
```

9.451 ex_NRLinkTxRaw.nxc

This is an example of how to use the [NRLinkTxRaw](#) function.

```
byte result = NRLinkTxRaw(S1, MS_ADDR_NRLINK);
```

9.452 ex_NumOut.nxc

This is an example of how to use the [NumOut](#) function.

```
NumOut (0, LCD_LINE1, x);
```

9.453 ex_NumToStr.nxc

This is an example of how to use the [NumToStr](#) function.

```
msg = NumToStr(-2); // returns "-2" in a string
```

9.454 ex_off.nxc

This is an example of how to use the [Off](#) function.

```
Off(OUT_A); // turn off output A
```

9.455 ex_offex.nxc

This is an example of how to use the [OffEx](#) function.

```
OffEx(OUT_A, RESET_NONE); // turn off output A
```

9.456 ex_OnBrickProgramPointer.nxc

This is an example of how to use the [OnBrickProgramPointer](#) function.

```
x = OnBrickProgramPointer();
```

9.457 ex_onfwd.nxc

This is an example of how to use the [OnFwd](#) function.

```
OnFwd(OUT_A, 75);
```

9.458 ex_onfwdxex.nxc

This is an example of how to use the [OnFwdEx](#) function.

```
OnFwdEx(OUT_A, 75, RESET_NONE);
```

9.459 ex_onfwdreg.nxc

This is an example of how to use the [OnFwdReg](#) function.

```
OnFwdReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

9.460 ex_onfwdregex.nxc

This is an example of how to use the [OnFwdRegEx](#) function.

```
OnFwdRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

9.461 ex_onfwdregexpid.nxc

This is an example of how to use the [OnFwdRegExPID](#) function.

```
OnFwdRegExPID(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE, 30, 50, 90);
```

9.462 ex_onfwdregpid.nxc

This is an example of how to use the [OnFwdRegPID](#) function.

```
OnFwdRegPID(OUT_A, 75, OUT_REGMODE_SPEED, 30, 50, 90); // regulate speed
```

9.463 ex_onfwdsync.nxc

This is an example of how to use the [OnFwdSync](#) function.

```
OnFwdSync(OUT_AB, 75, -100); // spin right
```

9.464 ex_onfwdsyncex.nxc

This is an example of how to use the [OnFwdSyncEx](#) function.

```
OnFwdSyncEx(OUT_AB, 75, 0, RESET_NONE);
```

9.465 ex_onfwdsyncexpid.nxc

This is an example of how to use the OnFwdSyncExPID function.

```
OnFwdSyncExPID(OUT_AB, 75, 0, RESET_NONE, 30, 50, 90);
```

9.466 ex_onfwdsynccpid.nxc

This is an example of how to use the OnFwdSyncPID function.

```
OnFwdSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin right
```

9.467 ex_onrev.nxc

This is an example of how to use the [OnRev](#) function.

```
OnRev(OUT_A, 75);
```

9.468 ex_onrevex.nxc

This is an example of how to use the [OnRevEx](#) function.

```
OnRevEx(OUT_A, 75, RESET_NONE);
```

9.469 ex_onrevreg.nxc

This is an example of how to use the [OnRevReg](#) function.

```
OnRevReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulate speed
```

9.470 ex_onrevregex.nxc

This is an example of how to use the [OnRevRegEx](#) function.

```
OnRevRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

9.471 ex_onrevregexpid.nxc

This is an example of how to use the OnRevRegExPID function.

```
OnRevRegExPID(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE, 30, 50, 90);
```

9.472 ex_onrevregpid.nxc

This is an example of how to use the [OnRevRegPID](#) function.

```
OnRevRegPID(OUT_A, 75, OUT_REGMODE_SPEED, 30, 50, 90); // regulate speed
```

9.473 ex_onrevsync.nxc

This is an example of how to use the [OnRevSync](#) function.

```
OnRevSync(OUT_AB, 75, -100); // spin left
```

9.474 ex_onrevsyncex.nxc

This is an example of how to use the [OnRevSyncEx](#) function.

```
OnRevSyncEx(OUT_AB, 75, -100, RESET_NONE); // spin left
```

9.475 ex_onrevsyncexpid.nxc

This is an example of how to use the [OnRevSyncExPID](#) function.

```
OnRevSyncExPID(OUT_AB, 75, -100, RESET_NONE, 30, 50, 90); // spin left
```

9.476 ex_onrevsyncpid.nxc

This is an example of how to use the [OnRevSyncPID](#) function.

```
OnRevSyncPID(OUT_AB, 75, -100, 30, 50, 90); // spin left
```

9.477 ex_OpenFileAppend.nxc

This is an example of how to use the [OpenFileAppend](#) function.

```
result = OpenFileAppend("data.txt", fsize, handle);
```

9.478 ex_OpenFileRead.nxc

This is an example of how to use the [OpenFileRead](#) function.

```
result = OpenFileRead("data.txt", fsize, handle);
```

9.479 ex_OpenFileReadLinear.nxc

This is an example of how to use the [OpenFileReadLinear](#) function.

```
result = OpenFileReadLinear("data.txt", fsize, handle);
```

9.480 ex_PlayFile.nxc

This is an example of how to use the [PlayFile](#) function.

```
PlayFile("startup.rso");
```

9.481 ex_PlayFileEx.nxc

This is an example of how to use the [PlayFileEx](#) function.

```
PlayFileEx("startup.rso", 3, true);
```

9.482 ex_playsound.nxc

This is an example of how to use the [PlaySound](#) function.

```
task main()
{
    PlaySound(SOUND_UP);
    PlaySound(SOUND_DOWN);
    Wait(SEC_1);
    PlaySound(SOUND_LOW_BEEP);
    Wait(MS_500);
    PlaySound(SOUND_FAST_UP);
}
```

9.483 ex_PlayTone.nxc

This is an example of how to use the [PlayTone](#) function.

```
PlayTone(440, 500);      // Play 'A' for one half second
```

9.484 ex_PlayToneEx.nxc

This is an example of how to use the [PlayToneEx](#) function.

```
PlayToneEx(440, 500, 2, false);
```

9.485 ex_playtones.nxc

This is an example of how to use the [PlayTones](#) function along with the [Tone](#) structure.

```
Tone sweepUp[] = {  
    TONE_C4, MS_50,  
    TONE_E4, MS_50,  
    TONE_G4, MS_50,  
    TONE_C5, MS_50,  
    TONE_E5, MS_50,  
    TONE_G5, MS_50,  
    TONE_C6, MS_200  
};  
  
task main()  
{  
    PlayTones(sweepUp);  
    Wait(SEC_1);  
}
```

9.486 ex_PointOut.nxc

This is an example of how to use the [PointOut](#) function.

```
PointOut(40, 40);
```

9.487 ex_PolyOut.nxc

This is an example of how to use the [PolyOut](#) function.

```
LocationType myPoints[] = {16,16, 8,40, 32,52, 20,36, 52,36, 56,52, 64,32, 44,20,  
    24,20};  
  
task main()  
{  
    PolyOut(myPoints, false);  
    Wait(SEC_2);  
    ClearScreen();  
    for(int i=0;i<10;i++) {  
        PolyOut(myPoints, DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE);  
    }  
}
```

```
    Wait(SEC_1);
}
PolyOut(myPoints, true|DRAW_OPT_FILL_SHAPE);
Wait(SEC_2);
ClearScreen();
for (int i=0;i<100;i++) {
    PolyOut(myPoints, DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE);
    Wait(MS_100);
}
Wait(SEC_1);
}
```

9.488 ex_pow.nxc

This is an example of how to use the [pow](#) function.

```
y = pow(x, 3);
```

9.489 ex_powerdown.nxc

This is an example of how to use the [PowerDown](#) functions.

```
PowerDown();
```

9.490 ex_Precedes.nxc

This is an example of how to use the [Precedes](#) statement.

```
Precedes(moving, drawing, playing);
```

9.491 ex_printf.nxc

This is an example of how to use the [printf](#) function.

```
printf("value = %d", value);
```

9.492 ex_PSPNxAnalog.nxc

This is an example of how to use the [PSPNxAnalog](#) function.

```
char result = PSPNxAnalog(S1, MS_ADDR_PSPNX);
```

9.493 ex_PSPNxDigital.nxc

This is an example of how to use the [PSPNxDigital](#) function.

```
char result = PSPNxDigital(S1, MS_ADDR_PSPNX);
```

9.494 ex_putc.nxc

This is an example of how to use the [putc](#) function.

```
putc(ch, handle);
```

9.495 ex_rand.nxc

This is an example of how to use the [rand](#) function.

```
unsigned int x = rand(); // 0..RAND_MAX-1
```

9.496 ex_Random.nxc

This is an example of how to use the [Random](#) function.

```
int x = Random(); // signed int between -32767..32767
unsigned i = Random(100); // 0..99

int ending = 4000, starting = 1000;
unsigned int j = Random(ending-starting)+starting; // 1000..3999
```

9.497 ex_Read.nxc

This is an example of how to use the [Read](#) function.

```
result = Read(handle, value);
```

9.498 ex_ReadButtonEx.nxc

This is an example of how to use the [ReadButtonEx](#) function.

```
ReadButtonEx(BTN1, true, pressed, count);
```

9.499 ex_ReadBytes.nxc

This is an example of how to use the [ReadBytes](#) function.

```
result = ReadBytes(handle, len, buffer);
```

9.500 ex_readi2cregister.nxc

This is an example of how to use the [ReadI2CRegister](#) function.

```
char result = ReadI2CRegister(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, out);
```

9.501 ex_ReadLn.nxc

This is an example of how to use the [ReadLn](#) function.

```
result = ReadLn(handle, value);
```

9.502 ex_ReadNRLinkBytes.nxc

This is an example of how to use the [ReadNRLinkBytes](#) function.

```
bool result = ReadNRLinkBytes(S1, MS_ADDR_NRLINK, data);
```

9.503 ex_ReadSensorColorEx.nxc

This is an example of how to use the [ReadSensorColorEx](#) function.

```
unsigned int rawData[], normData[];
int scaledData[];
int cval;
int result = ReadSensorColorEx(S1, cval, rawData, normData, scaledData);
```

9.504 ex_ReadSensorColorRaw.nxc

This is an example of how to use the [ReadSensorColorRaw](#) function.

```
unsigned int rawData[];
int result = ReadSensorColorRaw(S1, rawData);
```

9.505 ex_ReadSensorHTAccel.nxc

This is an example of how to use the [ReadSensorHTAccel](#) function.

```
bVal = ReadSensorHTAccel(S1, x, y, z);
```

9.506 ex_ReadSensorHTColor.nxc

This is an example of how to use the [ReadSensorHTColor](#) function.

```
bVal = ReadSensorHTColor(S1, c, r, g, b);
```

9.507 ex_ReadSensorHTColor2Active.nxc

This is an example of how to use the [ReadSensorHTColor2Active](#) function.

```
byte cnum, red, green, blue, white;
bool result = ReadSensorHTColor2Active(S1, cnum, red, green, blue, white);
```

9.508 ex_ReadSensorHTIRReceiver.nxc

This is an example of how to use the [ReadSensorHTIRReceiver](#) function.

```
char pfdata[];
bool result = ReadSensorHTIRReceiver(S1, pfdata);
```

9.509 ex_ReadSensorHTIRReceiverEx.nxc

This is an example of how to use the [ReadSensorHTIRReceiverEx](#) function.

```
char pfchar;
bool result = ReadSensorHTIRReceiverEx(S1, HT_CH1_A, pfchar);
```

9.510 ex_ReadSensorHTIRSeeker.nxc

This is an example of how to use the [ReadSensorHTIRSeeker](#) function.

```
bVal = ReadSensorHTIRSeeker(port, dir, s1, s3, s5, s7, s9);
```

9.511 ex_ReadSensorHTIRSeeker2AC.nxc

This is an example of how to use the [ReadSensorHTIRSeeker2AC](#) function.

```
byte s1, s3, s5, s7, s9;
bool result = ReadSensorHTIRSeeker2AC(S1, dir, s1, s3, s5, s7, s9);
```

9.512 ex_ReadSensorHTIRSeeker2DC.nxc

This is an example of how to use the [ReadSensorHTIRSeeker2DC](#) function.

```
byte s1, s3, s5, s7, s9, avg;
bool result = ReadSensorHTIRSeeker2DC(S1, dir, s1, s3, s5, s7, s9, avg);
```

9.513 ex_ReadSensorHTNormalizedColor.nxc

This is an example of how to use the [ReadSensorHTNormalizedColor](#) function.

```
bVal = ReadSensorHTNormalizedColor(S1, c, r, g, b);
```

9.514 ex_ReadSensorHTNormalizedColor2Active.nxc

This is an example of how to use the [ReadSensorHTNormalizedColor2Active](#) function.

```
byte cidx, red, green, blue;
bool result = ReadSensorHTNormalizedColor2Active(S1, cidx, red, green, blue);
```

9.515 ex_ReadSensorHTRawColor.nxc

This is an example of how to use the [ReadSensorHTRawColor](#) function.

```
bVal = ReadSensorHTRawColor(S1, r, g, b);
```

9.516 ex_ReadSensorHTRawColor2.nxc

This is an example of how to use the [ReadSensorHTRawColor2](#) function.

```
unsigned int red, green, blue, white;
bool result = ReadSensorHTRawColor2(S1, red, green, blue, white);
```

9.517 ex_ReadSensorHTTouchMultiplexer.nxc

This is an example of how to use the [ReadSensorHTTouchMultiplexer](#) function.

```
task main()
{
    byte t1, t2, t3, t4;
    SetSensorTouch(S1);
    while (true) {
        ReadSensorHTTouchMultiplexer(S1, t1, t2, t3, t4);
        if (t1)
            TextOut(0, LCD_LINE1, "1 pressed" );
        else
            TextOut(0, LCD_LINE1, "           " );
        if (t2)
            TextOut(0, LCD_LINE2, "2 pressed" );
        else
            TextOut(0, LCD_LINE2, "           " );
        if (t3)
            TextOut(0, LCD_LINE3, "3 pressed" );
        else
            TextOut(0, LCD_LINE3, "           " );
        if (t4)
            TextOut(0, LCD_LINE4, "4 pressed" );
        else
            TextOut(0, LCD_LINE4, "           " );
    }
}
```

9.518 ex_ReadSensorMSAccel.nxc

This is an example of how to use the [ReadSensorMSAccel](#) function.

```
int x, y, z;
bool result = ReadSensorMSAccel(S1, MS_ADDR_ACCLNX, x, y, z);
```

9.519 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, MS_ADDR_PSPNX, btnset1, btnset2, xleft,
                                      yleft, xright, yright);
```

9.520 ex_ReadSensorMSRTClock.nxc

This is an example of how to use the [ReadSensorMSRTClock](#) function.

```
ReadSensorMSRTClock(S1, ss, mm, hh, dow, dd, mon, yy);
```

9.521 ex_ReadSensorMSTilt.nxc

This is an example of how to use the [ReadSensorMSTilt](#) function.

```
byte x, y, z;
bool result = ReadSensorMSTilt(S1, MS_ADDR_ACCLNX, x, y, z);
```

9.522 ex_readsensorusex.nxc

This is an example of how to use the [ReadSensorUSEx](#) function.

```
byte values[];
char result = ReadSensorUSEx(S1, values);
```

9.523 ex_rebootinfirmwaremode.nxc

This is an example of how to use the [RebootInFirmwareMode](#) functions.

```
RebootInFirmwareMode();
```

9.524 ex_receivemessage.nxc

This is an example of how to use the [ReceiveMessage](#) function.

```
x = RecieveMessage(MAILBOX1, true, buffer);
```

9.525 ex_ReceiveRemoteBool.nxc

This is an example of how to use the [ReceiveRemoteBool](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, bvalue);
```

9.526 ex_ReceiveRemoteMessageEx.nxc

This is an example of how to use the [ReceiveRemoteMessageEx](#) function.

```
x = ReceiveRemoteMessageEx(MAILBOX1, true, strval, val, bval);
```

9.527 ex_ReceiveRemoteNumber.nxc

This is an example of how to use the [ReceiveRemoteNumber](#) function.

```
x = ReceiveRemoteBool(MAILBOX1, true, value);
```

9.528 ex_ReceiveRemoteString.nxc

This is an example of how to use the [ReceiveRemoteString](#) function.

```
x = ReceiveRemoteString(queue, true, strval);
```

9.529 ex_RechargeableBattery.nxc

This is an example of how to use the [RechargeableBattery](#) function.

```
x = RechargeableBattery();
```

9.530 ex_RectOut.nxc

This is an example of how to use the [RectOut](#) function.

```
RectOut(40, 40, 30, 10);
```

9.531 ex_reladdressof.nxc

This is an example of how to use the [reladdressOf](#) function.

```
task main()
{
    char x[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    unsigned long ptr = reladdressOf(x);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    IOMapReadByIDType args;
    args.ModuleID = CommandModuleID;
    args.Offset    = CommandOffsetMemoryPool+ptr;
    args.Count     = 10;
    SysIOMapReadByID(args);
    NumOut( 0, LCD_LINE2, x[0]);
    NumOut(20, LCD_LINE2, x[1]);
    NumOut(40, LCD_LINE2, x[2]);
```

```
    NumOut(60, LCD_LINE2, x[3]);
    NumOut(80, LCD_LINE2, x[4]);
    NumOut( 0, LCD_LINE3, args.Buffer[0]);
    NumOut(20, LCD_LINE3, args.Buffer[1]);
    NumOut(40, LCD_LINE3, args.Buffer[2]);
    NumOut(60, LCD_LINE3, args.Buffer[3]);
    NumOut(80, LCD_LINE3, args.Buffer[4]);
    args.Buffer++;
    args.Buffer *= 3;
    IOMapWriteByIDType a2;
    a2.ModuleID = CommandModuleID;
    a2.Offset   = CommandOffsetMemoryPool+ptr;
    a2.Buffer   = args.Buffer;
    SysIOMapWriteByID(a2);
    NumOut( 0, LCD_LINE4, x[0]);
    NumOut(20, LCD_LINE4, x[1]);
    NumOut(40, LCD_LINE4, x[2]);
    NumOut(60, LCD_LINE4, x[3]);
    NumOut(80, LCD_LINE4, x[4]);
    Wait(SEC_10);
}
```

9.532 ex_Release.nxc

This is an example of how to use the [Release](#) function.

```
Acquire(motorMutex); // make sure we have exclusive access
// use the motors
Release(motorMutex); // release mutex for other tasks
```

9.533 ex_RemoteKeepAlive.nxc

This is an example of how to use the [RemoteKeepAlive](#) function.

```
x = RemoteKeepAlive(1);
```

9.534 ex_RemoteMessageRead.nxc

This is an example of how to use the [RemoteMessageRead](#) function.

```
x = RemoteMessageRead(1, 5);
```

9.535 ex_RemoteMessageWrite.nxc

This is an example of how to use the [RemoteMessageWrite](#) function.

```
x = RemoteMessageWrite(1, 5, "test");
```

9.536 ex_RemotePlaySoundFile.nxc

This is an example of how to use the [RemotePlaySoundFile](#) function.

```
x = RemotePlaySoundFile(1, "click.rso", false);
```

9.537 ex_RemotePlayTone.nxc

This is an example of how to use the [RemotePlayTone](#) function.

```
x = RemotePlayTone(1, 440, 1000);
```

9.538 ex_RemoteResetMotorPosition.nxc

This is an example of how to use the [RemoteResetMotorPosition](#) function.

```
x = RemoteResetMotorPosition(1, OUT_A, true);
```

9.539 ex_RemoteResetScaledValue.nxc

This is an example of how to use the [RemoteResetScaledValue](#) function.

```
x = RemoteResetScaledValue(1, S1);
```

9.540 ex_RemoteSetInputMode.nxc

This is an example of how to use the [RemoteSetInputMode](#) function.

```
x = RemoteSetInputMode(1, S1, SENSOR_TYPE_LOWSPEED, SENSOR_MODE_RAW);
```

9.541 ex_RemoteSetOutputState.nxc

This is an example of how to use the [RemoteSetOutputState](#) function.

```
x = RemoteSetOutputState(1, OUT_A, 75, OUT_MODE_MOTORON, OUT_REGMODE_IDLE, 0,  
    OUT_RUNSTATE_RUNNING, 0);
```

9.542 ex_RemoteStartProgram.nxc

This is an example of how to use the [RemoteStartProgram](#) function.

```
x = RemoteStartProgram(1, "myprog.rxe");
```

9.543 ex_RemoteStopProgram.nxc

This is an example of how to use the [RemoteStopProgram](#) function.

```
x = RemoteStopProgram(1);
```

9.544 ex_RemoteStopSound.nxc

This is an example of how to use the [RemoteStopSound](#) function.

```
x = RemoteStopSound(1);
```

9.545 ex_remove.nxc

This is an example of how to use the [remove](#) function.

```
result = remove("data.txt");
```

9.546 ex_rename.nxc

This is an example of how to use the [rename](#) function.

```
result = rename("data.txt", "mydata.txt");
```

9.547 ex_RenameFile.nxc

This is an example of how to use the [RenameFile](#) function.

```
result = RenameFile("data.txt", "mydata.txt");
```

9.548 ex_resetalltachocounts.nxc

This is an example of how to use the [ResetAllTachoCounts](#) function.

```
ResetAllTachoCounts(OUT_AB);
```

9.549 ex_resetblocktachocount.nxc

This is an example of how to use the [ResetBlockTachoCount](#) function.

```
ResetBlockTachoCount(OUT_AB);
```

9.550 ex_resettotationcount.nxc

This is an example of how to use the [ResetRotationCount](#) function.

```
ResetRotationCount(OUT_AB);
```

9.551 ex_ResetScreen.nxc

This is an example of how to use the [ResetScreen](#) function.

```
ResetScreen();
```

9.552 ex_ResetSensor.nxc

This is an example of how to use the [ResetSensor](#) function.

```
ResetSensor(x); // x = S1
```

9.553 ex_ResetSleepTimer.nxc

This is an example of how to use the [ResetSleepTimer](#) function.

```
ResetSleepTimer();
```

9.554 ex_resettachocount.nxc

This is an example of how to use the [ResetTachoCount](#) function.

```
ResetTachoCount(OUT_AB);
```

9.555 ex_ResizeFile.nxc

This is an example of how to use the [ResizeFile](#) function.

```
result = ResizeFile("data.txt", 2048);
```

9.556 ex_ResolveHandle.nxc

This is an example of how to use the [ResolveHandle](#) function.

```
result = ResolveHandle("data.txt", handle, bCanWrite);
```

9.557 ex_rewind.nxc

This is an example of how to use the [rewind](#) function.

```
rewind(handle);
```

9.558 ex_rightstr.nxc

This is an example of how to use the [RightStr](#) function.

```
task main()
{
    string s = "Now is the winter of our discontent";
    TextOut(0, LCD_LINE1, RightStr(s, 12));
    Wait(SEC_4);
}
```

9.559 ex_rotatemotor.nxc

This is an example of how to use the [RotateMotor](#) function.

```
RotateMotor(OUT_A, 75, 45); // forward 45 degrees  
RotateMotor(OUT_A, -75, 45); // reverse 45 degrees
```

9.560 ex_rotatemotorex.nxc

This is an example of how to use the [RotateMotorEx](#) function.

```
RotateMotorEx(OUT_AB, 75, 360, 50, true, true);
```

9.561 ex_rotatemotorexpid.nxc

This is an example of how to use the [RotateMotorExPID](#) function.

```
RotateMotorExPID(OUT_AB, 75, 360, 50, true, true, 30, 50, 90);
```

9.562 ex_rotatemotorpid.nxc

This is an example of how to use the [RotateMotorPID](#) function.

```
RotateMotorPID(OUT_A, 75, 45, 20, 40, 100);
```

9.563 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);
```

9.564 ex_RS485DataAvailable.nxc

This is an example of how to use the [RS485DataAvailable](#) function.

```
bool result = RS485DataAvailable();
```

9.565 ex_RS485Exit.nxc

This is an example of how to use the [RS485Exit](#) function.

```
char result = RS485Exit();
```

9.566 ex_RS485Init.nxc

This is an example of how to use the [RS485Init](#) function.

```
char result = RS485Init();
```

9.567 ex_RS485Read.nxc

This is an example of how to use the [RS485Read](#) function.

```
byte buffer[];
char result = RS485Read(buffer);
```

9.568 ex_RS485SendingData.nxc

This is an example of how to use the [RS485SendingData](#) function.

```
bool result = RS485SendingData()
```

9.569 ex_RS485Status.nxc

This is an example of how to use the [RS485Status](#) function.

```
bool sendingData, dataAvail;
RS485Status(sendingData, dataAvail);
```

9.570 ex_RS485Uart.nxc

This is an example of how to use the [RS485Uart](#) function.

```
char result = RS485Uart(HS_BAUD_115200, HS_MODE_8N1);
```

9.571 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);
```

9.572 ex_RunNRLinkMacro.nxc

This is an example of how to use the [RunNRLinkMacro](#) function.

```
char result = RunNRLinkMacro(S1, MS_ADDR_NRLINK, macro);
```

9.573 ex_sendmessage.nxc

This is an example of how to use the [SendMessage](#) function.

```
x = SendMessage(MAILBOX1, data);
```

9.574 ex_sendremotebool.nxc

This is an example of how to use the [SendRemoteBool](#) function.

```
x = SendRemoteBool(1, MAILBOX1, false);
```

9.575 ex_SendRemoteNumber.nxc

This is an example of how to use the [SendRemoteNumber](#) function.

```
x = SendRemoteNumber(1, MAILBOX1, 123);
```

9.576 ex_SendRemoteString.nxc

This is an example of how to use the [SendRemoteString](#) function.

```
x = SendRemoteString(1, MAILBOX1, "hello world");
```

9.577 ex_SendResponseBool.nxc

This is an example of how to use the [SendResponseBool](#) function.

```
x = SendResponseBool(MAILBOX1, false);
```

9.578 ex_SendResponseNumber.nxc

This is an example of how to use the [SendResponseNumber](#) function.

```
x = SendResponseNumber(MAILBOX1, 123);
```

9.579 ex_SendResponseString.nxc

This is an example of how to use the [SendResponseString](#) function.

```
x = SendResponseString(MAILBOX1, "hello world");
```

9.580 ex_SendRS485Bool.nxc

This is an example of how to use the [SendRS485Bool](#) function.

```
bool bval = true;
char result = SendRS485Bool(bval);
```

9.581 ex_SendRS485Number.nxc

This is an example of how to use the [SendRS485Number](#) function.

```
long val = 12345;
char result = SendRS485Number(val);
```

9.582 ex_SendRS485String.nxc

This is an example of how to use the [SendRS485String](#) function.

```
string msg = "123";
char result = SendRS485String(msg);
```

9.583 ex_Sensor.nxc

This is an example of how to use the [Sensor](#) function.

```
x = Sensor(S1); // read sensor 1
```

9.584 ex_SensorBoolean.nxc

This is an example of how to use the [SensorBoolean](#) function.

```
x = SensorBoolean(S1);
```

9.585 ex_SensorDigiPinsDirection.nxc

This is an example of how to use the [SensorDigiPinsDirection](#) function.

```
x = SensorDigiPinsDirection(S1);
```

9.586 ex_SensorDigiPinsOutputLevel.nxc

This is an example of how to use the [SensorDigiPinsOutputLevel](#) function.

```
x = SensorDigiPinsOutputLevel(S1);
```

9.587 ex_SensorDigiPinsStatus.nxc

This is an example of how to use the [SensorDigiPinsStatus](#) function.

```
x = SensorDigiPinsStatus(S1);
```

9.588 ex_SensorHTColorNum.nxc

This is an example of how to use the [SensorHTColorNum](#) function.

```
x = SensorHTColorNum(S1);
```

9.589 ex_SensorHTCompass.nxc

This is an example of how to use the [SensorHTCompass](#) function.

```
x = SensorHTCompass(S1);
```

9.590 ex_SensorHTEOPD.nxc

This is an example of how to use the [SensorHTEOPD](#) function.

```
int val = SensorHTEOPD(S1);
```

9.591 ex_SensorHTGyro.nxc

This is an example of how to use the [SensorHTGyro](#) function.

```
task main()
{
    int offset = 400;
    SetSensorHTGyro(S1);
    NumOut(0, LCD_LINE1, SensorHTGyro(S1, offset+5));
    Wait(SEC_9);
}
```

9.592 ex_SensorHTIRSeeker2ACDir.nxc

This is an example of how to use the [SensorHTIRSeeker2ACDir](#) function.

```
int val = SensorHTIRSeeker2ACDir(S1);
```

9.593 ex_SensorHTIRSeeker2Addr.nxc

This is an example of how to use the [SensorHTIRSeeker2Addr](#) function.

```
int val = SensorHTIRSeeker2Addr(S1, HTIR2_REG_DCAVG);
```

9.594 ex_SensorHTIRSeeker2DCDir.nxc

This is an example of how to use the [SensorHTIRSeeker2DCDir](#) function.

```
int val = SensorHTIRSeeker2DCDir(S1);
```

9.595 ex_SensorHTIRSeekerDir.nxc

This is an example of how to use the [SensorHTIRSeekerDir](#) function.

```
x = SensorHTIRSeekerDir(S1);
```

9.596 ex_SensorInvalid.nxc

This is an example of how to use the [SensorInvalid](#) function.

```
x = SensorInvalid(S1);
```

9.597 ex_SensorMode.nxc

This is an example of how to use the [SensorMode](#) function.

```
x = SensorMode(S1);
```

9.598 ex_SensorMSCompass.nxc

This is an example of how to use the [SensorMSCompass](#) function.

```
x = SensorMSCompass(S1, MS_ADDR_CMPSNX);
```

9.599 ex_SensorMSDROD.nxc

This is an example of how to use the [SensorMSDROD](#) function.

```
x = SensorMSDROD(S1);
```

9.600 ex_SensorMSPressure.nxc

This is an example of how to use the [SensorMSPressure](#) function.

```
int val = SensorMSPressure(S1);
```

9.601 ex_SensorMSPressureRaw.nxc

This is an example of how to use the [SensorMSPressureRaw](#) function.

```
int val = SensorMSPressureRaw(S1);
```

9.602 ex_SensorNormalized.nxc

This is an example of how to use the [SensorNormalized](#) function.

```
x = SensorNormalized(S1);
```

9.603 ex_SensorRaw.nxc

This is an example of how to use the [SensorRaw](#) function.

```
x = SensorRaw(S1);
```

9.604 ex_SensorScaled.nxc

This is an example of how to use the [SensorScaled](#) function.

```
x = SensorScaled(S1);
```

9.605 ex_SensorType.nxc

This is an example of how to use the [SensorType](#) function.

```
x = SensorType(S1);
```

9.606 ex_sensorus.nxc

This is an example of how to use the [SensorUS](#) function.

```
x = SensorUS(S4); // read sensor 4
```

9.607 ex_SensorValue.nxc

This is an example of how to use the [SensorValue](#) function.

```
unsigned int val = SensorValue(S1);
```

9.608 ex_SensorValueBool.nxc

This is an example of how to use the [SensorValueBool](#) function.

```
bool val = SensorValueBool(S1);
```

9.609 ex_SensorValueRaw.nxc

This is an example of how to use the [SensorValueRaw](#) function.

```
unsigned int val = SensorValueRaw(S1);
```

9.610 ex_SetAbortFlag.nxc

This is an example of how to use the [SetAbortFlag](#) function.

```
SetAbortFlag(BTNSTATE_LONG_PRESSED_EV);
```

9.611 ex_SetACCLNxSensitivity.nxc

This is an example of how to use the [SetACCLNxSensitivity](#) function.

```
result = SetACCLNxSensitivity(S1, MS_ADDR_ACCLNX, ACCL_SENSITIVITY_LEVEL_1);
```

9.612 ex_SetBatteryState.nxc

This is an example of how to use the [SetBatteryState](#) function.

```
SetBatteryState(4);
```

9.613 ex_SetBluetoothState.nxc

This is an example of how to use the [SetBluetoothState](#) function.

```
SetBluetoothState(UI_BT_STATE_OFF);
```

9.614 ex_SetBTInputBuffer.nxc

This is an example of how to use the [SetBTInputBuffer](#) function.

```
SetBTInputBuffer(0, 10, buffer);
```

9.615 ex_SetBTInputBufferInPtr.nxc

This is an example of how to use the [SetBTInputBufferInPtr](#) function.

```
SetBTInputBufferInPtr(0);
```

9.616 ex_SetBTInputBufferOutPtr.nxc

This is an example of how to use the [SetBTInputBufferOutPtr](#) function.

```
SetBTInputBufferOutPtr(0);
```

9.617 ex_SetBTOutputBuffer.nxc

This is an example of how to use the [SetBTOutputBuffer](#) function.

```
SetBTOutputBuffer(0, 10, buffer);
```

9.618 ex_SetBTOutputBufferInPtr.nxc

This is an example of how to use the [SetBTOutputBufferInPtr](#) function.

```
SetBTOutputBufferInPtr(0);
```

9.619 ex_SetBTOutputBufferOutPtr.nxc

This is an example of how to use the [SetBTOutputBufferOutPtr](#) function.

```
SetBTOutputBufferOutPtr(0);
```

9.620 ex_SetButtonLongPressCount.nxc

This is an example of how to use the [SetButtonLongPressCount](#) function.

```
SetButtonLongPressCount (BTN1, value);
```

9.621 ex_SetButtonLongReleaseCount.nxc

This is an example of how to use the [SetButtonLongReleaseCount](#) function.

```
SetButtonLongReleaseCount (BTN1, value);
```

9.622 ex_SetButtonPressCount.nxc

This is an example of how to use the [SetButtonPressCount](#) function.

```
SetButtonPressCount (BTN1, value);
```

9.623 ex_SetButtonReleaseCount.nxc

This is an example of how to use the [SetButtonReleaseCount](#) function.

```
SetButtonReleaseCount (BTN1, value);
```

9.624 ex_SetButtonShortReleaseCount.nxc

This is an example of how to use the [SetButtonShortReleaseCount](#) function.

```
SetButtonShortReleaseCount (BTN1, value);
```

9.625 ex_SetButtonState.nxc

This is an example of how to use the [SetButtonState](#) function.

```
SetButtonState (BTN1, BTNSTATE_PRESSED_EV);
```

9.626 ex_SetCommandFlags.nxc

This is an example of how to use the [SetCommandFlags](#) function.

```
SetCommandFlags (UI_FLAGS_REDRAW_STATUS) ;
```

9.627 ex_SetCustomSensorActiveStatus.nxc

This is an example of how to use the [SetCustomSensorActiveStatus](#) function.

```
SetCustomSensorActiveStatus (S1, true) ;
```

9.628 ex_SetCustomSensorPercentFullScale.nxc

This is an example of how to use the [SetCustomSensorPercentFullScale](#) function.

```
SetCustomSensorPercentFullScale (S1, 100) ;
```

9.629 ex_SetCustomSensorZeroOffset.nxc

This is an example of how to use the [SetCustomSensorZeroOffset](#) function.

```
SetCustomSensorZeroOffset (S1, 12) ;
```

9.630 ex_SetDisplayContrast.nxc

This is an example of how to use the [SetDisplayContrast](#) function.

```
SetDisplayContrast (DISPLAY_CONTRAST_DEFAULT) ;
```

9.631 ex_SetDisplayDisplay.nxc

This is an example of how to use the [SetDisplayDisplay](#) function.

```
SetDisplayDisplay (x) ;
```

9.632 ex_SetDisplayEraseMask.nxc

This is an example of how to use the [SetDisplayEraseMask](#) function.

```
SetDisplayEraseMask(x);
```

9.633 ex_SetDisplayFlags.nxc

This is an example of how to use the [SetDisplayFlags](#) function.

```
SetDisplayFlags(x);
```

9.634 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, 0x41, 0x3E, 0x00, 0x00, 0x08, 0x3E, 0x1C, 0x3E, 0x
    08, 0x00, 0x08, 0x08, 0x3E, 0x08, 0x08, 0x00, 0x80, 0x60, 0x60, 0x00, 0x00, 0x00, 0x08, 0x08, 0x0
    8, 0x08, 0x08, 0x00, 0x00, 0x60, 0x60, 0x00, 0x00, 0x00, 0x20, 0x10, 0x08, 0x04, 0x02, 0x00,
    0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00, 0x00, 0x42, 0x7F, 0x40, 0x00, 0x00, 0x62, 0x51, 0x49, 0x49
    , 0x46, 0x00, 0x22, 0x49, 0x49, 0x49, 0x36, 0x00, 0x18, 0x14, 0x12, 0x7F, 0x10, 0x00, 0x2F, 0x49,
    0x49, 0x49, 0x31, 0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30, 0x00, 0x01, 0x71, 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, 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, 0x01, 0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A, 0x00, 0
    x7F, 0x08, 0x08, 0x08, 0x7F, 0x00, 0x00, 0x41, 0x7F, 0x41, 0x00, 0x00, 0x30, 0x40, 0x40, 0x40, 0x
    3F, 0x00, 0x7F, 0x08, 0x14, 0x22, 0x41, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x00, 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, 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, 0x28, 0x00, 0x38, 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, 0x7D, 0x40, 0x00, 0x00, 0x40, 0x80, 0x84, 0x7D, 0x
00, 0x00, 0x7F, 0x10, 0x28, 0x44, 0x00, 0x00, 0x00, 0x7F, 0x40, 0x00, 0x00, 0x7C, 0x04, 0x1
8, 0x04, 0x78, 0x00, 0x7C, 0x04, 0x04, 0x78, 0x00, 0x00, 0x38, 0x44, 0x44, 0x38, 0x00,
0xFC, 0x44, 0x44, 0x44, 0x00, 0x38, 0x44, 0x44, 0x44, 0x44, 0xFC, 0x00, 0x44, 0x78, 0x44, 0x04
, 0x08, 0x00, 0x08, 0x54, 0x54, 0x54, 0x20, 0x00, 0x04, 0x3E, 0x44, 0x24, 0x00, 0x00, 0x3C, 0x40,
0x20, 0x7C, 0x00, 0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C, 0x00, 0x3C, 0x60, 0x30, 0x60, 0x3C, 0x00, 0
x6C, 0x10, 0x10, 0x6C, 0x00, 0x00, 0x9C, 0xA0, 0x60, 0x3C, 0x00, 0x00, 0x64, 0x54, 0x54, 0x4C, 0x
00, 0x00, 0x08, 0x3E, 0x41, 0x41, 0x00, 0x00, 0x00, 0x00, 0x77, 0x00, 0x00, 0x00, 0x00, 0x41, 0x4
1, 0x3E, 0x08, 0x00, 0x02, 0x01, 0x02, 0x01, 0x00, 0x00, 0x10, 0x20, 0x40, 0x38, 0x07, 0x00
};

task main()
{
    unsigned long ptr, pOldFont;
    byte myData[800];
    ptr = addr(NewFont);
    TextOut(0, LCD_LINE1, FormatNum("%x", ptr));
    pOldFont = DisplayFont();
    SetDisplayFont(ptr);
    TextOut(0, LCD_LINE2, "Testing 1, 2, 3");
    SetDisplayFont(pOldFont);
    TextOut(0, LCD_LINE4, "Testing 1, 2, 3");
    Wait(SEC_10);
}

```

9.635 ex_SetDisplayNormal.nxc

This is an example of how to use the [SetDisplayNormal](#) function.

```
SetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

9.636 ex_SetDisplayPopup.nxc

This is an example of how to use the [SetDisplayPopup](#) function.

```
SetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

9.637 ex_SetDisplayTextLinesCenterFlags.nxc

This is an example of how to use the [SetDisplayTextLinesCenterFlags](#) function.

```
SetDisplayTextLinesCenterFlags(x);
```

9.638 ex_SetDisplayUpdateMask.nxc

This is an example of how to use the [SetDisplayUpdateMask](#) function.

```
SetDisplayUpdateMask (x);
```

9.639 ex_SetHSFlags.nxc

This is an example of how to use the [SetHSFlags](#) function.

```
SetHSFlags (0);
```

9.640 ex_SetHSInputBuffer.nxc

This is an example of how to use the [SetHSInputBuffer](#) function.

```
SetHSInputBuffer(0, 10, buffer);
```

9.641 ex_SetHSInputBufferInPtr.nxc

This is an example of how to use the [SetHSInputBufferInPtr](#) function.

```
SetHSInputBufferInPtr(0);
```

9.642 ex_SetHSInputBufferOutPtr.nxc

This is an example of how to use the [SetHSInputBufferOutPtr](#) function.

```
SetHSInputBufferOutPtr(0);
```

9.643 ex_SetHSMode.nxc

This is an example of how to use the [SetHSMode](#) function.

```
SetHSMode (HS_MODE_8N1);
```

9.644 ex_SetHSOutputBuffer.nxc

This is an example of how to use the [SetHSOutputBuffer](#) function.

```
SetHSOutputBuffer(0, 10, buffer);
```

9.645 ex_SetHSOutputBufferInPtr.nxc

This is an example of how to use the [SetHSOutputBufferInPtr](#) function.

```
SetHSOutputBufferInPtr(0);
```

9.646 ex_SetHSOutputBufferOutPtr.nxc

This is an example of how to use the [SetHSOutputBufferOutPtr](#) function.

```
SetHSOutputBufferOutPtr(0);
```

9.647 ex_SetHSSpeed.nxc

This is an example of how to use the [SetHSSpeed](#) function.

```
SetHSSpeed(1);
```

9.648 ex_SetHSState.nxc

This is an example of how to use the [SetHSState](#) function.

```
SetHSState(1);
```

9.649 ex_SetHTColor2Mode.nxc

This is an example of how to use the [SetHTColor2Mode](#) function.

```
SetHTColor2Mode(S1, HT_CMD_COLOR2_ACTIVE);
```

9.650 ex_SetHTIRSeeker2Mode.nxc

This is an example of how to use the [SetHTIRSeeker2Mode](#) function.

```
SetHTIRSeeker2Mode(S1, HTIR2_MODE_1200);
```

9.651 ex_SetInput.nxc

This is an example of how to use the [SetInput](#) function.

```
SetInput(S1, Type, SENSOR_TYPE_SOUND_DB);
```

9.652 ex_SetLongAbort.nxc

This is an example of how to use the [SetLongAbort](#) function.

```
SetLongAbort(true);
```

9.653 ex_setmotorpwnfreq.nxc

This is an example of how to use the [SetMotorPwnFreq](#) function.

```
SetMotorPwnFreq(x);
```

9.654 ex_SetOnBrickProgramPointer.nxc

This is an example of how to use the [SetOnBrickProgramPointer](#) function.

```
SetOnBrickProgramPointer(2);
```

9.655 ex_setoutput.nxc

This is an example of how to use the [SetOutput](#) function.

```
SetOutput(OUT_AB, TachoLimit, 720); // set tacho limit
```

9.656 ex_SetSensor.nxc

This is an example of how to use the [SetSensor](#) function.

```
SetSensor(S1, SENSOR_TOUCH);
```

9.657 ex_SetSensorBoolean.nxc

This is an example of how to use the [SetSensorBoolean](#) function.

```
SetHTIRSeeker2Mode(S1, HTIR2_MODE_1200);
```

9.658 ex_SetSensorColorBlue.nxc

This is an example of how to use the [SetSensorColorBlue](#) function.

```
SetSensorColorBlue(S1);
```

9.659 ex_SetSensorColorFull.nxc

This is an example of how to use the [SetSensorColorFull](#) function.

```
SetSensorColorFull(S1);
```

9.660 ex_SetSensorColorGreen.nxc

This is an example of how to use the [SetSensorColorGreen](#) function.

```
SetSensorColorGreen(S1);
```

9.661 ex_SetSensorColorNone.nxc

This is an example of how to use the [SetSensorColorNone](#) function.

```
SetSensorColorNone(S1);
```

9.662 ex_SetSensorColorRed.nxc

This is an example of how to use the [SetSensorColorRed](#) function.

```
SetSensorColorRed(S1);
```

9.663 ex_SetSensorDigiPinsDirection.nxc

This is an example of how to use the [SetSensorDigiPinsDirection](#) function.

```
SetSensorDigiPinsDirection(S1, 1);
```

9.664 ex_SetSensorDigiPinsOutputLevel.nxc

This is an example of how to use the [SetSensorDigiPinsOutputLevel](#) function.

```
SetSensorDigiPinsOutputLevel(S1, 100);
```

9.665 ex_SetSensorDigiPinsStatus.nxc

This is an example of how to use the [SetSensorDigiPinsStatus](#) function.

```
SetSensorDigiPinsStatus(S1, false);
```

9.666 ex_SetSensorHTEOPD.nxc

This is an example of how to use the [SetSensorHTEOPD](#) function.

```
SetSensorHTEOPD(S1);
```

9.667 ex_SetSensorHTGyro.nxc

This is an example of how to use the [SetSensorHTGyro](#) function.

```
SetSensorHTGyro(S1);
```

9.668 ex_SetSensorLight.nxc

This is an example of how to use the [SetSensorLight](#) function.

```
SetSensorLight(S1);
```

9.669 ex_SetSensorLowspeed.nxc

This is an example of how to use the [SetSensorLowspeed](#) function.

```
SetSensorLowspeed(S1);
```

9.670 ex_SetSensorMode.nxc

This is an example of how to use the [SetSensorMode](#) function.

```
SetSensorMode(S1, SENSOR_MODE_RAW); // raw mode
```

9.671 ex_SetSensorMSDROD.nxc

This is an example of how to use the [SetSensorMSDROD](#) function.

```
SetSensorMSDROD(S1);
```

9.672 ex_SetSensorMSPressure.nxc

This is an example of how to use the [SetSensorMSPressure](#) function.

```
SetSensorMSPressure(S1);
```

9.673 ex_SetSensorSound.nxc

This is an example of how to use the [SetSensorSound](#) function.

```
SetSensorSound(S1);
```

9.674 ex_SetSensorTouch.nxc

This is an example of how to use the [SetSensorTouch](#) function.

```
SetSensorTouch(S1);
```

9.675 ex_SetSensorType.nxc

This is an example of how to use the [SetSensorType](#) function.

```
SetSensorType(S1, SENSOR_TYPE_TOUCH);
```

9.676 ex_SetSleepTime.nxc

This is an example of how to use the [SetSleepTime](#) function.

```
SetSleepTime(5); // sleep in 5 minutes
```

9.677 ex_SetSleepTimeout.nxc

This is an example of how to use the [SetSleepTimeout](#) function.

```
SetSleepTimeout(8);
```

9.678 ex_SetSleepTimer.nxc

This is an example of how to use the [SetSleepTimer](#) function.

```
SetSleepTimer(3);
```

9.679 ex_SetSoundDuration.nxc

This is an example of how to use the [SetSoundDuration](#) function.

```
SetSoundDuration(500);
```

9.680 ex_SetSoundFlags.nxc

This is an example of how to use the [SetSoundFlags](#) function.

```
SetSoundFlags(SOUND_FLAGS_UPDATE);
```

9.681 ex_SetSoundFrequency.nxc

This is an example of how to use the [SetSoundFrequency](#) function.

```
SetSoundFrequency(440);
```

9.682 ex_SetSoundMode.nxc

This is an example of how to use the [SetSoundMode](#) function.

```
SetSoundMode(SOUND_MODE_ONCE);
```

9.683 ex_SetSoundModuleState.nxc

This is an example of how to use the [SetSoundModuleState](#) function.

```
SetSoundModuleState(SOUND_STATE_STOP);
```

9.684 ex_SetSoundSampleRate.nxc

This is an example of how to use the [SetSoundSampleRate](#) function.

```
SetSoundSampleRate(4000);
```

9.685 ex_SetSoundVolume.nxc

This is an example of how to use the [SetSoundVolume](#) function.

```
SetSoundVolume(3);
```

9.686 ex_SetUIButton.nxc

This is an example of how to use the [SetUIButton](#) function.

```
SetUIButton(UI_BUTTON_ENTER);
```

9.687 ex_SetUIState.nxc

This is an example of how to use the [SetUIState](#) function.

```
SetUIState (UI_STATE_LOW_BATTERY);
```

9.688 ex_SetUSBInputBuffer.nxc

This is an example of how to use the [SetUSBInputBuffer](#) function.

```
SetUSBInputBuffer (0, 10, buffer);
```

9.689 ex_SetUSBInputBufferInPtr.nxc

This is an example of how to use the [SetUSBInputBufferInPtr](#) function.

```
SetUSBInputBufferInPtr (0);
```

9.690 ex_SetUSBInputBufferOutPtr.nxc

This is an example of how to use the [SetUSBInputBufferOutPtr](#) function.

```
SetUSBInputBufferOutPtr (0);
```

9.691 ex_SetUSBOuputBuffer.nxc

This is an example of how to use the [SetUSBOuputBuffer](#) function.

```
SetUSBOuputBuffer (0, 10, buffer);
```

9.692 ex_SetUSBOuputBufferInPtr.nxc

This is an example of how to use the [SetUSBOuputBufferInPtr](#) function.

```
SetUSBOuputBufferInPtr (0);
```

9.693 ex_SetUSBOutputBufferOutPtr.nxc

This is an example of how to use the [SetUSBOutputBufferOutPtr](#) function.

```
SetUSBOutputBufferOutPtr(0);
```

9.694 ex_SetUSBPollBuffer.nxc

This is an example of how to use the [SetUSBPollBuffer](#) function.

```
SetUSBPollBuffer(0, 10, buffer);
```

9.695 ex_SetUSBPollBufferInPtr.nxc

This is an example of how to use the [SetUSBPollBufferInPtr](#) function.

```
SetUSBPollBufferInPtr(0);
```

9.696 ex_SetUSBPollBufferOutPtr.nxc

This is an example of how to use the [SetUSBPollBufferOutPtr](#) function.

```
SetUSBPollBufferOutPtr(0);
```

9.697 ex_SetUSBState.nxc

This is an example of how to use the [SetUSBState](#) function.

```
SetUSBState(0);
```

9.698 ex_SetVMRunState.nxc

This is an example of how to use the [SetVMRunState](#) function.

```
SetVMRunState(0); // stopped
```

9.699 ex_SetVolume.nxc

This is an example of how to use the [SetVolume](#) function.

```
SetVolume(3);
```

9.700 ex_sign.nxc

This is an example of how to use the [sign](#) function.

```
char val = sign(x); // return -1, 0, or 1
```

9.701 ex_sin_cos.nxc

This is an example of how to use the [cos](#) and the [sin](#) functions.

```
// ex_sin_cos.nxc
// Run this program and you will see a circle appear on the NXT screen in a
// strange random way. No two runs will produce the circle in exactly the same
// way.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define SCREEN_WIDH 100
#define SCREEN_HEIGHT 64
#define X_ZERO (SCREEN_WIDH / 2)
#define Y_ZERO (SCREEN_HEIGHT / 2)
#define R (Y_ZERO - 2)
#define MAX_DEG 360

// Convert a float to its nearest integer value.
inline int integer(float x)
{
    return trunc(x + 0.5);
}

task main()
{
    while(true)
    {
        float angle = RADIANS_PER_DEGREE * Random(MAX_DEG);
        float x = X_ZERO + R * cos(angle);
        float y = Y_ZERO + R * sin(angle);
        PointOut(integer(x), integer(y));
        // Without the Wait, the program runs too fast!
        Wait(MS_20);
    }
}
```

9.702 ex_sind_cosd.nxc

This is an example of how to use the [cosd](#) and [sind](#) functions.

```
// ex_sind_cosd.nxc
// Run this program and you will see a circle appear on the NXT screen in a
// strange random way. No two runs will produce the circle in exactly the same
// way.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define SCREEN_WIDTH 100
#define SCREEN_HEIGHT 64
#define X_ZERO (SCREEN_WIDTH / 2)
#define Y_ZERO (SCREEN_HEIGHT / 2)
#define R (Y_ZERO - 2)
#define MAX_DEG 360

// Convert a float to its nearest integer value.
inline int integer(float x)
{
    return trunc(x + 0.5);
}

task main()
{
    while(true)
    {
        float angle = Random(MAX_DEG);
        float x = X_ZERO + R * cosd(angle);
        float y = Y_ZERO + R * sind(angle);
        PointOut(integer(x), integer(y));
        // Without the Wait, the program runs too fast!
        Wait(MS_20);
    }
}
```

9.703 ex_sinh.nxc

This is an example of how to use the [sinh](#) function.

```
x = sinh(y);
```

9.704 ex_sleepnow.nxc

This is an example of how to use the [SleepNow](#) functions.

```
SleepNow();
```

9.705 ex_SleepTime.nxc

This is an example of how to use the [SleepTime](#) function.

```
x = SleepTime(); // read sleep time
```

9.706 ex_SleepTimeout.nxc

This is an example of how to use the [SleepTimeout](#) function.

```
byte x = SleepTimeout();
```

9.707 ex_SleepTimer.nxc

This is an example of how to use the [SleepTimer](#) function.

```
byte x = SleepTimer();
```

9.708 ex_SoundDuration.nxc

This is an example of how to use the [SoundDuration](#) function.

```
x = SoundDuration();
```

9.709 ex_SoundFlags.nxc

This is an example of how to use the [SoundFlags](#) function.

```
x = SoundFlags();
```

9.710 ex_SoundFrequency.nxc

This is an example of how to use the [SoundFrequency](#) function.

```
x = SoundFrequency();
```

9.711 ex_SoundMode.nxc

This is an example of how to use the [SoundMode](#) function.

```
x = SoundMode();
```

9.712 ex_SoundSampleRate.nxc

This is an example of how to use the [SoundSampleRate](#) function.

```
x = SoundSampleRate();
```

9.713 ex_SoundState.nxc

This is an example of how to use the [SoundState](#) function.

```
x = SoundState();
```

9.714 ex_SoundVolume.nxc

This is an example of how to use the [SoundVolume](#) function.

```
x = SoundVolume();
```

9.715 ex_sprintf.nxc

This is an example of how to use the [sprintf](#) function.

```
sprintf(msg, "value = %d", value);
```

9.716 ex_sqrt.nxc

This is an example of how to use the [sqrt](#) function.

```
x = sqrt(x);
```

9.717 ex_starttask.nxc

This is an example of how to use the [StartTask](#) function.

```
StartTask(sound); // start the sound task
```

9.718 ex_stop.nxc

This is an example of how to use the [Stop](#) function.

```
Stop(x == 24); // stop the program if x==24
```

9.719 ex_stopalltasks.nxc

This is an example of how to use the [StopAllTasks](#) function.

```
StopAllTasks(); // stop the program
```

9.720 ex_StopSound.nxc

This is an example of how to use the [StopSound](#) function.

```
StopSound();
```

9.721 ex_stoptask.nxc

This is an example of how to use the [StopTask](#) function.

```
StopTask(sound); // stop the sound task
```

9.722 ex_strcat.nxc

This is an example of how to use the [strcat](#) function.

```
strcat(msg, "foo"); // msg = msg+"foo"
```

9.723 ex_StrCatOld.nxc

This is an example of how to use the [StrCat](#) function.

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    string tmp = "123456";
    string a = "AA", b = "BB", c = "CC";

    TextOut(0, LCD_LINE3, StrCat(a, SubStr(tmp, 2, 3), msgs[0]));
    Wait(SEC_5);
}
```

9.724 ex_strcmp.nxc

This is an example of how to use the [strcmp](#) function.

```
int i = strcmp(msg, "foo"); // returns -1, 0, or 1
```

9.725 ex_strcpy.nxc

This is an example of how to use the [strcpy](#) function.

```
strcpy(msg, "foo"); // msg = "foo"
```

9.726 ex_StrIndex.nxc

This is an example of how to use the [StrIndex](#) function.

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    NumOut(0, LCD_LINE5, StrIndex(msgs[0], 0));
    string msg = "hi there";
    byte x = StrIndex(msg, 2); // return the value of msg[2]
    Wait(SEC_5);
}
```

9.727 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_LINES, StrIndex(msgs[0], 0));
    NumOut(0, LCD_LINE6, StrLen(msgs[0]));
    TextOut(0, LCD_LINE7, FormatNum(fmts[0], Random(34)));
    float val = StrToNum("10.5abc123");
    NumOut(0, LCD_LINE8, val);
    Wait(SEC_5);
    ClearScreen();
    TextOut(0, LCD_LINE1, NumToStr(PI));
    int x = 0x7172;
    string foo = FlattenVar(x);
    TextOut(0, LCD_LINE2, foo);
    TextOut(0, LCD_LINE3, Flatten(0x7374));
    NumOut(0, LCD_LINE4, strlen(foo));
    NumOut(40, LCD_LINE4, UnflattenVar(foo, x));
    TextOut(0, LCD_LINE5, FormatNum(fmts[1], x));
    string bats = tmp; // "123456"
    TextOut(0, LCD_LINE6, bats);
    byte data[];
    StrToByteArray(bats, data);
    TextOut(0, LCD_LINE7, ByteArrayToStr(data));
    ByteArrayToStrEx(data, tmp);
    TextOut(0, LCD_LINE8, tmp);
    Wait(SEC_10);
}

```

9.728 ex_strlen.nxc

This is an example of how to use the [strlen](#) function.

```

task main()
{
    string msg = "hi there";
    byte x = strlen(msg); // return the length of msg
}

```

9.729 ex_StrLenOld.nxc

This is an example of how to use the [StrLen](#) function.

```
task main()
```

```
{  
    string msgs[] = {"please work", "testing, 1, 2, 3"};  
    string msg = "hi there";  
    byte x = StrLen(msg); // return the length of msg  
  
    NumOut(0, LCD_LINE6, StrLen(msgs[0]));  
    Wait(SEC_5);  
}
```

9.730 ex_strncat.nxc

This is an example of how to use the [strncat](#) function.

```
strncat(msg, "foo", 2); // msg = msg+ "fo"
```

9.731 ex_strcmp.nxc

This is an example of how to use the [strcmp](#) function.

```
int i = strcmp(msg, "foo", 2); // returns -1, 0, or 1
```

9.732 ex_strncpy.nxc

This is an example of how to use the [strncpy](#) function.

```
strncpy(msg, "foo", 2); // msg = "fo"
```

9.733 ex_StrReplace.nxc

This is an example of how to use the [StrReplace](#) function.

```
task main()  
{  
    string msgs[] = {"please work", "testing, 1, 2, 3"};  
    string a = "AA", b = "BB", c = "CC";  
    TextOut(0, LCD_LINE4, StrReplace(msgs[0], 2, StrCat(a, b)));  
    string msg = StrReplace("testing", 3, "xx"); // returns "tesxxng"  
  
    Wait(SEC_5);  
}
```

9.734 ex_StrToByteArray.nxc

This is an example of how to use the [StrToByteArray](#) function.

```
StrToByteArray(myStr, myArray);
```

9.735 ex_strtod.nxc

This is an example of how to use the [strtod](#) function.

```
task main()
{
    string str, endptr;
    str = "3.1415926e2abcdefg";
    float f = strtod(str, endptr);
    NumOut(0, LCD_LINE1, f);
    TextOut(0, LCD_LINE2, str);
    TextOut(0, LCD_LINE3, endptr);
    Wait(SEC_6);
}
```

9.736 ex_strtol.nxc

This is an example of how to use the [strtol](#) function.

```
task main()
{
    string str, endptr;
    str = "3.1415926e2abcdefg";
    long l = strtol(str, endptr);
    NumOut(0, LCD_LINE1, l);
    TextOut(0, LCD_LINE2, str);
    TextOut(0, LCD_LINE3, endptr);
    Wait(SEC_6);
}
```

9.737 ex_StrToNum.nxc

This is an example of how to use the [StrToNum](#) function.

```
x = StrToNum(strVal);
```

9.738 ex strtoul.nxc

This is an example of how to use the [strtoul](#) function.

```
task main()
{
    string str, endptr;
    str = "3.1415926e2abcdefg";
    unsigned long l = strtoul(str, endptr);
    NumOut(0, LCD_LINE1, l);
    TextOut(0, LCD_LINE2, str);
    TextOut(0, LCD_LINE3, endptr);
    Wait(SEC_6);
}
```

9.739 ex_SubStr.nxc

This is an example of how to use the [SubStr](#) function.

```
task main()
{
    string msgs[] = {"please work", "testing, 1, 2, 3"};
    TextOut(0, LCD_LINE2, SubStr(msgs[0], 2, 3));
    string msg = SubStr("test", 1, 2); // returns "es"
    Wait(SEC_5);
}
```

9.740 ex_syscall.nxc

This is an example of how to use the [SysCall](#) function.

```
task main()
{
    DrawTextType dtArgs;
    dtArgs.Location.X = 0;
    dtArgs.Location.Y = LCD_LINE1;
    dtArgs.Text = "Please Work";
    SysCall(DrawText, dtArgs);
}
```

9.741 ex_SysColorSensorRead.nxc

This is an example of how to use the [SysColorSensorRead](#) function.

```
task main()
{
    SetSensorColorFull(S1);
    ColorSensorReadType csr;
    csr.Port = S1;
    SysColorSensorRead(csr);
    if (csr.Result == NO_ERR) {
        NumOut(0, LCD_LINE1, csr.ColorValue);
    }
}
```

9.742 ex_syscommbtcheckstatus.nxc

This is an example of how to use the [SysCommBTCheckStatus](#) function along with the [CommBTCheckStatusType](#) structure.

```
task main()
{
    CommBTCheckStatusType args;
    args.Connection = 1;
    SysCommBTCheckStatus(args);
    if (args.Result == LDR_SUCCESS) { /* do something */ }
}
```

9.743 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 above code uses 1)
    args.Action = TRUE; // could use some #define with a non-zero value to connect.
    0 == disconnect
    if (!BluetoothStatus(CONNECTION)==NO_ERR)
    {
        SysCommBTConnection(args); // try to connect.
        for (int i = 0; i < 2000; i++) {
            NumOut(0, LCD_LINE3, args.Result);
            Wait(1);
        }
        // Wait(5000); // let the connection get created
        if (args.Result == LDR_SUCCESS)
        {
            // we are connected
            TextOut(0, LCD_LINE1, "success");
        }
        else {
            TextOut(0, LCD_LINE1, "failure");
            NumOut(0, LCD_LINE2, args.Result);
        }
    }
    Wait(SEC_10);
}
```

9.744 ex_SysCommBTOnOff.nxc

This is an example of how to use the [SysCommBTOnOff](#) function along with the [CommBTOnOffType](#) structure.

```
task main()
{
    CommBTOnOffType bt;
    bt.PowerState = false;
    SysCommBTOnOff(bt);
    if (bt.Result == NO_ERR)
        TextOut(0, LCD_LINE1, "BT is off");
}
```

9.745 ex_syscommbtwrite.nxc

This is an example of how to use the [SysCommBTWrite](#) function along with the [CommBTWriteType](#) structure.

```
task main()
{
    CommBTWriteType args;
    args.Connection = 1;
    args.Buffer = myData;
    SysCommBTWrite(args);
}
```

9.746 ex_syscommexecutefunction.nxc

This is an example of how to use the [SysCommExecuteFunction](#) function along with the [CommExecuteFunctionType](#) structure.

```
task main()
{
    CommExecuteFunctionType args;
    args.Cmd = INTF_BTOFF;
    SysCommExecuteFunction(args);
}
```

9.747 ex_SysCommHSCheckStatus.nxc

This is an example of how to use the [SysCommHSCheckStatus](#) function along with the [CommHSCheckStatusType](#) structure.

```
task main()
{
    CommHSCheckStatusType hsc;
    SysCommHSCheckStatus(hsc);
    if (hsc.SendingData)
        TextOut(0, LCD_LINE1, "sending data");
    else if (hsc.DataAvailable)
        TextOut(0, LCD_LINE1, "data available");
}
```

9.748 ex_SysCommHSControl.nxc

This is an example of how to use the [SysCommHSControl](#) function along with the [CommHSControlType](#) structure.

```
task main()
{
    CommHSControlType hsc;
    hsc.Command = HS_CTRL_INIT;
    SysCommHSControl(hsc);
    if (hsc.Result)
        TextOut(0, LCD_LINE1, "hi-speed initialized");
    Wait(SEC_10);
}
```

9.749 ex_SysCommHSRead.nxc

This is an example of how to use the [SysCommHSRead](#) function along with the [CommHSReadWriteType](#) structure.

```
task main()
{
    CommHSReadWriteType hsr;
    SysCommHSRead(hsr);
    if (hsr.Status == NO_ERR)
        TextOut(0, LCD_LINE1, hsr.Buffer);
    Wait(SEC_1);
}
```

9.750 ex_SysCommHSWrite.nxc

This is an example of how to use the [SysCommHSWrite](#) function along with the [CommHSReadWriteType](#) structure.

```
task main()
{
    // configure the hi-speed port and turn it on
    // ...
    // no write to the port
    CommHSReadWriteType rwt;
    ArrayBuild(rwt.Buffer, 0x01, 0x02, 0x03, 0x04); // four bytes
    SysCommHSWrite(rwt);
    if (rwt.Status == NO_ERR) {
        // do something
    }
}
```

9.751 ex_syscommlscheckstatus.nxc

This is an example of how to use the [SysCommLSCheckStatus](#) function along with the [CommLSCheckStatusType](#) structure.

```
task main()
{
    CommLSCheckStatusType args;
    args.Port = S1;
    SysCommLSCheckStatus(args);
    // is the status (Result) IDLE?
    if (args.Result == LOWSPEED_IDLE) { /* proceed */ }
}
```

9.752 ex_syscommlsread.nxc

This is an example of how to use the [SysCommLSRead](#) function along with the [CommLSReadType](#) structure.

```
task main()
{
    CommLSReadType args;
    args.Port = S1;
    args.Buffer = myBuf;
    args.BufferLen = 8;
    SysCommLSRead(args);
    // check Result for error status & use Buffer contents
}
```

9.753 ex_syscommlswrite.nxc

This is an example of how to use the [SysCommLSWrite](#) function along with the [CommLSWriteType](#) structure.

```
task main()
{
    CommLSWriteType args;
    args.Port = S1;
    args.Buffer = myBuf;
    args.ReturnLen = 8;
    SysCommLSWrite(args);
    // check Result for error status
}
```

9.754 ex_syscommlswriteex.nxc

This is an example of how to use the [SysCommLSWriteEx](#) function along with the [CommLSWriteExType](#) structure.

```
task main()
{
    CommLSWriteExType args;
    args.Port = S1;
    args.Buffer = myBuf;
    args.ReturnLen = 8;
    args.NoRestartOnRead = true;
    SysCommLSWriteEx(args);
    if (args.Result == NO_ERR)
    {
        // do something
    }
}
```

9.755 ex_SysComputeCalibValue.nxc

This is an example of how to use the [SysComputeCalibValue](#) function along with the [ComputeCalibValueType](#) structure.

```
task main()
{
    ComputeCalibValueType args;
    args.Name = "light";
    args.RawVal = Sensor(S1);
    SysComputeCalibValue(args);
    if (args.Result == NO_ERR)
        TextOut(0, LCD_LINE1, "calib computed");
}
```

9.756 ex_SysDatalogGetTimes.nxc

This is an example of how to use the [SysDatalogGetTimes](#) function along with the [DatalogGetTimesType](#) structure.

```
task main()
{
    DatalogGetTimesType args;
    SysDatalogGetTimes(args);
    NumOut(0, LCD_LINE4, args.SyncTime);
    NumOut(0, LCD_LINE5, args.SyncTick);
    Wait(SEC_5);
}
```

9.757 ex_SysDatalogWrite.nxc

This is an example of how to use the [SysDatalogWrite](#) function along with the [Data-logWriteType](#) structure.

```
task main()
```

```
{  
    DatalogWriteType args;  
    ArrayBuild(args.Message, 0x01, 0x02);  
    SysDatalogWrite(args);  
    if (args.Result == NO_ERR)  
        TextOut(0, LCD_LINE1, "success");  
}
```

9.758 ex_sysdisplayexecutefunction.nxc

This is an example of how to use the [SysDisplayExecuteFunction](#) function along with the [DisplayExecuteFunctionType](#) structure.

```
task main()  
{  
    DisplayExecuteFunctionType args;  
    args.Cmd = DISPLAY_ERASE_ALL;  
    SysDisplayExecuteFunction(args);  
}
```

9.759 ex_sysdrawcircle.nxc

This is an example of how to use the [SysDrawCircle](#) function along with the [DrawCircleType](#) structure.

```
task main()  
{  
    DrawCircleType dcArgs;  
    dcArgs.Center.X = 20;  
    dcArgs.Center.Y = 20;  
    dcArgs.Size = 10; // radius  
    dcArgs.Options = 0x01; // clear before drawing  
    SysDrawCircle(dcArgs);  
}
```

9.760 ex_sysdrawellipse.nxc

This is an example of how to use the [SysDrawEllipse](#) function along with the [DrawEllipseType](#) structure.

```
task main()  
{  
    DrawEllipseType args;  
    args.Center.X = 50;  
    args.Center.Y = 32;  
    repeat (10) {  
        args.SizeX = 20+Random(15);  
        args.SizeY = 20+Random(10);  
        args.Options = DRAW_OPT_FILL_SHAPE | DRAW_OPT_LOGICAL_XOR;
```

```

        SysDrawEllipse(args);
    }
    while(true);
}

```

9.761 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);
}

```

9.762 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);
}

```

9.763 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, 0x80, 0x01,
                      0x82, 0x01, 0x84, 0x01, 0x88, 0x01,
                      0x90, 0x01, 0xA0, 0x01, 0x80, 0x01,
                      0xFF, 0xFF)),  

        RICOpCopyBits(0, 1,  

                      RICImgRect(  

                        RICImgPoint(0, 0), 16, 64),  

                        RICImgPoint(0, 0))  

      );  

  

task main()  

{
  DrawGraphicArrayType args;  

  args.Location.X = 0;  

  args.Location.Y = 0;  

  args.Data = ric_data;  

  SysDrawGraphicArray(args);  

  Wait(SEC_5);
}

```

9.764 ex_sysdrawline.nxc

This is an example of how to use the [SysDrawLine](#) function along with the [DrawLine-Type](#) structure.

```

task main()
{
  DrawLineType dlArgs;
  dlArgs.StartLoc.X = 20;
  dlArgs.StartLoc.Y = 20;
  dlArgs.EndLoc.X = 60;
  dlArgs.EndLoc.Y = 60;
  dlArgs.Options = 0x01; // clear before drawing
  SysDrawLine(dlArgs);
}

```

9.765 ex_sysdrawpoint.nxc

This is an example of how to use the [SysDrawPoint](#) function along with the [DrawPointType](#) structure.

```
task main()
{
    DrawPointType dpArgs;
    dpArgs.Location.X = 20;
    dpArgs.Location.Y = 20;
    dpArgs.Options = 0x04; // clear this pixel
    SysDrawPoint(dpArgs);
}
```

9.766 ex_sysdrawpolygon.nxc

This is an example of how to use the [SysDrawPolygon](#) function along with the [DrawPolygonType](#) structure.

```
LocationType myPoints[] = {16,16, 8,40, 32,52, 20,36, 52,36, 56,52, 64,32, 44,20,
                           24,20};

task main()
{
    DrawPolygonType args;
    args.Points = myPoints;
    args.Options = 0x00;
    SysDrawPolygon(args);
    Wait(SEC_2);
    ClearScreen();
    args.Options = DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE;
    for(int i=0;i<10;i++) {
        SysDrawPolygon(args);
        Wait(SEC_1);
    }
    args.Options = true|DRAW_OPT_FILL_SHAPE;
    SysDrawPolygon(args);
    Wait(SEC_2);
    ClearScreen();
    args.Options = DRAW_OPT_LOGICAL_XOR|DRAW_OPT_FILL_SHAPE;
    for (int i=0;i<100;i++) {
        SysDrawPolygon(args);
        Wait(MS_100);
    }
    Wait(SEC_1);
}
```

9.767 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);
}
```

9.768 ex_sysdrawtext.nxc

This is an example of how to use the [SysDrawText](#) function along with the [DrawText-Type](#) structure.

```
task main()
{
    DrawTextType dtArgs;
    dtArgs.Location.X = 0;
    dtArgs.Location.Y = LCD_LINE1;
    dtArgs.Text = "Please Work";
    dtArgs.Options = 0x01; // clear before drawing
    SysDrawText(dtArgs);
}
```

9.769 ex_sysfclose.nxc

This is an example of how to use the [SysFileClose](#) function along with the [FileClose-Type](#) structure.

```
task main()
{
    FileCloseType fcArgs;
    fcArgs.FileHandle = foArgs.FileHandle;
    SysFileClose(fcArgs);
}
```

9.770 ex_sysfiledelete.nxc

This is an example of how to use the [SysFileDelete](#) function along with the [FileDelete-Type](#) structure.

```
task main()
{
    FileDeleteType fdArgs;
    fdArgs.Filename = "myfile.txt";
    SysFileDelete(fdArgs); // delete the file
}
```

9.771 ex_sysfilefindfirst.nxc

This is an example of how to use the [SysFileFindFirst](#) function along with the [FileFindType](#) structure.

```
task main()
{
    FileFindType args;
    args.Filename = "*.*";
    SysFileFindFirst(args);
    TextOut(0, LCD_LINE1, args.Filename);
}
```

9.772 ex_sysfilefindnext.nxc

This is an example of how to use the [SysFileFindNext](#) function along with the [FileFindType](#) structure.

```
task main()
{
    FileFindType args;
    args.FileHandle = prev.FileHandle;
    SysFileFindNext(args);
    TextOut(0, LCD_LINE1, args.Filename);
}
```

9.773 ex_sysfileopenappend.nxc

This is an example of how to use the [SysFileOpenAppend](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    SysFileOpenAppend(foArgs); // open the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
        // up to the remaining available length in Length
    }
}
```

9.774 ex_sysfileopenread.nxc

This is an example of how to use the [SysFileOpenRead](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    SysFileOpenRead(foArgs); // open the file for reading
    if (foArgs.Result == NO_ERR) {
        // read data from the file using FileHandle
    }
}
```

9.775 ex_sysfileopenreadlinear.nxc

This is an example of how to use the [SysFileOpenReadLinear](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.rxe";
    SysFileOpenReadLinear(foArgs); // open the file for
    reading
    if (foArgs.Result == NO_ERR) {
        // read data from the file using FileHandle
    }
}
```

9.776 ex_sysfileopenwrite.nxc

This is an example of how to use the [SysFileOpenWrite](#) function along with the [FileOpenType](#) structure.

```
task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    foArgs.Length = 256; // create with capacity for 256 bytes
    SysFileOpenWrite(foArgs); // create the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
    }
}
```

9.777 ex_sysfileopenwritelinear.nxc

This is an example of how to use the [SysFileOpenWriteLinear](#) function along with the [FileOpenType](#) structure.

```
task main()
{
```

```

FileOpenType foArgs;
foArgs.Filename = "myfile.txt";
foArgs.Length = 256; // create with capacity for 256 bytes
SysFileOpenWriteLinear(foArgs); // create the file
if (foArgs.Result == NO_ERR) {
    // write to the file using FileHandle
}
}

```

9.778 ex_sysfileopenwritenonlinear.nxc

This is an example of how to use the [SysFileOpenWriteNonLinear](#) function along with the [FileOpenType](#) structure.

```

task main()
{
    FileOpenType foArgs;
    foArgs.Filename = "myfile.txt";
    foArgs.Length = 256; // create with capacity for 256 bytes
    SysFileOpenWriteNonLinear(foArgs); // create the file
    if (foArgs.Result == NO_ERR) {
        // write to the file using FileHandle
    }
}

```

9.779 ex_sysfileread.nxc

This is an example of how to use the [SysFileRead](#) function along with the [FileReadWriteType](#) structure.

```

task main()
{
    FileReadWriteType frArgs;
    frArgs.FileHandle = foArgs.FileHandle;
    frArgs.Length = 12; // number of bytes to read
    SysFileRead(frArgs);
    if (frArgs.Result == NO_ERR) {
        TextOut(0, LCD_LINE1, frArgs.Buffer);
        // show how many bytes were actually read
        NumOut(0, LCD_LINE2, frArgs.Length);
    }
}

```

9.780 ex_sysfilerename.nxc

This is an example of how to use the [SysFileRename](#) function along with the [FileRenameType](#) structure.

```
task main()
```

```
{  
    FileRenameType frArgs;  
    frArgs.OldFilename = "myfile.txt";  
    frArgs.NewFilename = "myfile2.txt";  
    SysFileRename(frArgs);  
    if (frArgs.Result == LDR_SUCCESS) { /* do something */ }  
}
```

9.781 ex_sysfileresize.nxc

This is an example of how to use the [SysFileResize](#) function along with the [FileResizeType](#) structure.

```
task main()  
{  
    byte handle;  
    // get a file handle  
    // ...  
    // resize the file  
    FileResizeType args;  
    args.FileHandle = handle;  
    args.NewSize = 2048;  
    SysFileResize(args);  
    if (args.Result == NO_ERR)  
    {  
        // do something  
    }  
}
```

9.782 ex_sysfileresolvehandle.nxc

This is an example of how to use the [SysFileResolveHandle](#) function along with the [FileResolveHandleType](#) structure.

```
task main()  
{  
    FileResolveHandleType frhArgs;  
    frhArgs.Filename = "myfile.txt";  
    SysFileResolveHandle(frhArgs);  
    if (frhArgs.Result == LDR_SUCCESS) {  
        // use the FileHandle as needed  
        if (frhArgs.WriteHandle) {  
            // file is open for writing  
        }  
        else {  
            // file is open for reading  
        }  
    }  
}
```

9.783 ex_sysfileseek.nxc

This is an example of how to use the [SysFileSeek](#) function along with the [FileSeekType](#) structure.

```
task main()
{
    byte handle;
    // get a file handle
    // ...
    FileSeekType args;
    args.FileHandle = handle;
    args.Origin = SEEK_SET;
    args.Length = 65;
    SysFileSeek(args);
    if (args.Result == NO_ERR)
    {
        // do something
    }
}
```

9.784 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);
    }
}
```

9.785 ex_sysgetstarttick.nxc

This is an example of how to use the [SysGetStartTick](#) function along with the [GetStartTickType](#) structure.

```
task main()
{
    GetStartTickType gstArgs;
    SysGetStartTick(gstArgs);
    unsigned long myStart = gstArgs.Result;
}
```

9.786 ex_systiomapread.nxc

This is an example of how to use the [SysIOMapRead](#) function along with the [IOMapReadType](#) structure.

```
task main()
{
    IOMapReadType args;
    args.ModuleName = CommandModuleName;
    args.Offset = CommandOffsetTick;
    args.Count = 4; // this value happens to be 4 bytes long
    SysIOMapRead(args);
    if (args.Result == NO_ERR) { /* do something with data */ }
}
```

9.787 ex_systiomapreadbyid.nxc

This is an example of how to use the [SysIOMapReadByID](#) function along with the [IOMapReadByIDType](#) structure.

```
task main()
{
    IOMapReadByIDType args;
    args.ModuleID = CommandModuleID;
    args.Offset = CommandOffsetTick;
    args.Count = 4; // this value happens to be 4 bytes long
    SysIOMapReadByID(args);
    if (args.Result == NO_ERR) { /* do something with data */ }
}
```

9.788 ex_systiomapwrite.nxc

This is an example of how to use the [SysIOMapWrite](#) function along with the [IOMapWriteType](#) structure.

```
task main()
{
    IOMapWriteType args;
    args.ModuleName = SoundModuleName;
    args.Offset = SoundOffsetSampleRate;
    args.Buffer = theData;
    SysIOMapWrite(args);
}
```

9.789 ex_systiomapwritebyid.nxc

This is an example of how to use the [SysIOMapWriteByID](#) function along with the [IOMapWriteByIDType](#) structure.

```
task main()
{
    IOMapWriteByIDType args;
    args.ModuleID = SoundModuleID;
    args.Offset = SoundOffsetSampleRate;
    args.Buffer = theData;
    SysIOMapWriteByID(args);
}
```

9.790 ex_syskeepalive.nxc

This is an example of how to use the [SysKeepAlive](#) function along with the [KeepAliveType](#) structure.

```
task main()
{
    KeepAliveType kaArgs;
    SysKeepAlive(kaArgs); // reset sleep timer
}
```

9.791 ex_syslistfiles.nxc

This is an example of how to use the [SysListFiles](#) function along with the [ListFileType](#) structure.

```
task main()
{
    ListFileType 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);
}
```

9.792 ex_sysloaderexecufunction.nxc

This is an example of how to use the [SysLoaderExecuteFunction](#) function along with the [LoaderExecuteFunctionType](#) structure.

```
task main()
{
    LoaderExecuteFunctionType args;
    args.Cmd = 0xA0; // delete user flash
    SysLoaderExecuteFunction(args);
}
```

9.793 ex_sysmessageread.nxc

This is an example of how to use the [SysMessageRead](#) function along with the [MessageReadType](#) structure.

```
task main()
{
    MessageReadType args;
    args.QueueID = MAILBOX1; // 0
    args.Remove = true;
    SysMessageRead(args);
    if (args.Result == NO_ERR) {
        TextOut(0, LCD_LINE1, args.Message);
    }
}
```

9.794 ex_sysmessagewrite.nxc

This is an example of how to use the [SysMessageWrite](#) function along with the [MessageWriteType](#) structure.

```
task main()
{
    MessageWriteType args;
    args.QueueID = MAILBOX1; // 0
    args.Message = "testing";
    SysMessageWrite(args);
    // check Result for error status
}
```

9.795 ex_sysrandomnumber.nxc

This is an example of how to use the [SysRandomNumber](#) function along with the [RandomNumberType](#) structure.

```
task main()
{
    RandomNumberType rnArgs;
    SysRandomNumber(rnArgs);
    int myRandomValue = rnArgs.Result;
}
```

9.796 ex_sysreadbutton.nxc

This is an example of how to use the [SysReadButton](#) function along with the [ReadButtonType](#) structure.

```
task main()
{
    ReadButtonType rbArgs;
    rbArgs.Index = BTNRIGHT;
    SysReadButton(rbArgs);
    if (rbArgs.Pressed) /* do something */
}
```

9.797 ex_SysReadSemData.nxc

This is an example of how to use the [SysReadSemData](#) function along with the [ReadSemDataType](#) structure.

```
task main()
{
    ReadSemDataType args;
    args.Request = true;
    SysReadSemData(args);
    NumOut(0, LCD_LINE1, args.SemData);
}
```

9.798 ex_syssetscreenmode.nxc

This is an example of how to use the [SysSetScreenMode](#) function along with the [SetScreenModeType](#) structure.

```
task main()
{
    SetScreenModeType ssmArgs;
    ssmArgs.ScreenMode = 0x00; // restore default NXT screen
    SysSetScreenMode(ssmArgs);
}
```

9.799 ex_SysSetSleepTimeout.nxc

This is an example of how to use the [SysSetSleepTimeout](#) function.

```
task main()
{
    SetSleepTimeoutType args;
    args.TheSleepTimeoutMS = MIN_1*5;
    SysSetSleepTimeout(args);
}
```

9.800 ex_syssoundgetstate.nxc

This is an example of how to use the [SysSoundGetState](#) function along with the [SoundGetStateType](#) structure.

```
task main()
{
    SoundGetStateType sgsArgs;
    SysSoundGetState(sgsArgs);
    if (sgsArgs.State == SOUND_STATE_IDLE) /* do stuff */
}
```

9.801 ex_syssoundplayfile.nxc

This is an example of how to use the [SysSoundPlayFile](#) function along with the [Sound-PlayFileType](#) structure.

```
task main()
{
    SoundPlayFileType spfArgs;
    spfArgs.Filename = "hello.rso";
    spfArgs.Loop = false;
    spfArgs.SoundLevel = 3;
    SysSoundPlayFile(spfArgs);
}
```

9.802 ex_syssoundplaytone.nxc

This is an example of how to use the [SysSoundPlayTone](#) function along with the [Sound-PlayToneType](#) structure.

```
task main()
{
    SoundPlayToneType sptArgs;
    sptArgs.Frequency = 440;
    sptArgs.Duration = 1000; // 1 second
    sptArgs.Loop = false;
    sptArgs.SoundLevel = 3;
    SysSoundPlayTone(sptArgs);
}
```

9.803 ex_syssoundsetstate.nxc

This is an example of how to use the [SysSoundSetState](#) function along with the [Sound-SetStateType](#) structure.

```
task main()
{
    SoundSetStateType sssArgs;
    sssArgs.State = SOUND_STATE_STOP;
    SysSoundSetState(sssArgs);
}
```

9.804 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.MinValue = 0;
    args.MaxValue = 1023;
    SysUpdateCalibCacheInfo(args);
    NumOut(0, LCD_LINE1, args.Result);
}
```

9.805 ex_SysWriteSemData.nxc

This is an example of how to use the [SysWriteSemData](#) function along with the [WriteSemDataType](#) structure.

```
task main()
{
    WriteSemDataType args;
    args.NewVal = 0x4;
    args.Request = true;
    args.ClearBits = false;
    SysWriteSemData(args);
    NumOut(0, LCD_LINE1, args.SemData);
}
```

9.806 ex_tan.nxc

This is an example of how to use the [tan](#) function.

```
// ex_tan.nxc
// Display values generated by the tan API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define DELTA PI / 8

// Angles from -3/8 PI radians to almost PI/2 radians stepped by PI/8 radians.
const float data[] =
{
    -3 * DELTA,
    -2 * DELTA,
    -DELTA,
    0.0,
    DELTA,
    2 * DELTA,
```

```
    3 * DELTA,
    4 * DELTA - 0.01
};

// Display a table of angles and their tangents. The angles are the ones
// specified above.
task main()
{
    const int items = ArrayLen(data);
    for (int i = 0; i < items; ++i)
    {
        int screen_y = 56 - 8 * i;
        float angle = data[i];
        TextOut(0, screen_y, FormatNum("%7.4f", angle));
        TextOut(45, screen_y, FormatNum("%8.4f", tan(angle)));
    }
    while (true);
}
```

9.807 ex_tand.nxc

This is an example of how to use the [tand](#) function.

```
// ex_tand.nxc
// Display values generated by the tand API call.
// This program runs indefinitely -- press gray button to exit.
// Requires enhanced firmware 1.28 or later.

#define DELTA 22.5

// Angles from -67.5 degrees to almost 90.0 degrees stepped by 22.5 degrees.
const float data[] =
{
    -3 * DELTA,
    -2 * DELTA,
    -DELTA,
    0.0,
    DELTA,
    2 * DELTA,
    3 * DELTA,
    4 * DELTA - 0.01
};

// Display a table of angles and their tangents. The angles are the ones
// specified above.
task main()
{
    const int items = ArrayLen(data);
    for (int i = 0; i < items; ++i)
    {
        int screen_y = 56 - 8 * i;
        float angle = data[i];
        TextOut(0, screen_y, FormatNum("%5.1f", angle));
        TextOut(40, screen_y, FormatNum("%8.4f", tand(angle)));
    }
}
```

```
    while (true);  
}
```

9.808 ex_tanh.nxc

This is an example of how to use the `tanh` function.

```
x = tanh(y);
```

9.809 ex_TextOut.nxc

This is an example of how to use the `TextOut` function.

```
TextOut(0, LCD_LINE3, "Hello World!");
```

9.810 ex_tolower.nxc

This is an example of how to use the `tolower` function.

```
i = tolower(x);
```

9.811 ex_toupper.nxc

This is an example of how to use the `toupper` function.

```
i = toupper(x);
```

9.812 ex_trunc.nxc

This is an example of how to use the `trunc` function.

```
y = trunc(x);
```

9.813 ex_UIButton.nxc

This is an example of how to use the `UIButton` function.

```
x = UIButton();
```

9.814 ex_UIState.nxc

This is an example of how to use the [UIState](#) function.

```
x = UIState();
```

9.815 ex_UiUsbState.nxc

This is an example of how to use the [UsbState](#) function.

```
value = UsbState();
```

9.816 ex_UnflattenVar.nxc

This is an example of how to use the [UnflattenVar](#) function.

```
task main()
{
    long data[] = {-50123, 68142, 128176, -45123};
    long data2[4];
    float fdata[] = {12.123, 3.14159, 2.68};
    float fdata2[3];
    NumOut(0, LCD_LINE1, data[0]);
    NumOut(0, LCD_LINE2, fdata[1]);
    string sdata = FlattenVar(data);
    string tmp;
    // transfer the string to another NXT
    tmp = sdata;
    UnflattenVar(tmp, data2);
    NumOut(0, LCD_LINE3, data2[0]);
    sdata = FlattenVar(fdata);
    // transfer the string to another NXT
    tmp = sdata;
    UnflattenVar(tmp, fdata2);
    NumOut(0, LCD_LINE4, fdata2[1]);
    Wait(SEC_5);
}
```

9.817 ex_USBInputBufferInPtr.nxc

This is an example of how to use the [USBInputBufferInPtr](#) function.

```
byte x = USBInputBufferInPtr();
```

9.818 ex_USBInputBufferOutPtr.nxc

This is an example of how to use the [USBInputBufferOutPtr](#) function.

```
byte x = USBInputBufferOutPtr();
```

9.819 ex_USBOutputBufferInPtr.nxc

This is an example of how to use the [USBOutputBufferInPtr](#) function.

```
byte x = USBOutputBufferInPtr();
```

9.820 ex_USBOutputBufferOutPtr.nxc

This is an example of how to use the [USBOutputBufferOutPtr](#) function.

```
byte x = USBOutputBufferOutPtr();
```

9.821 ex_USBPollBufferInPtr.nxc

This is an example of how to use the [USBPollBufferInPtr](#) function.

```
byte x = USBPollBufferInPtr();
```

9.822 ex_USBPollBufferOutPtr.nxc

This is an example of how to use the [USBPollBufferOutPtr](#) function.

```
byte x = USBPollBufferOutPtr();
```

9.823 ex_USBState.nxc

This is an example of how to use the [USBState](#) function.

```
byte x = USBPollBufferOutPtr();
```

9.824 ex_VMRunState.nxc

This is an example of how to use the [VMRunState](#) function.

```
x = VMRunState();
```

9.825 ex_Volume.nxc

This is an example of how to use the [Volume](#) function.

```
x = Volume();
```

9.826 ex_wait.nxc

This is an example of how to use the [Wait](#) function.

```
task main()
{
    Wait(SEC_5); // wait 5 seconds
    Wait(Random(SEC_1)); // wait random time up to 1 second
}
```

9.827 ex_Write.nxc

This is an example of how to use the [Write](#) function.

```
result = Write(handle, value);
```

9.828 ex_WriteBytes.nxc

This is an example of how to use the [WriteBytes](#) function.

```
result = WriteBytes(handle, buffer, count);
```

9.829 ex_WriteBytesEx.nxc

This is an example of how to use the [WriteBytesEx](#) function.

```
result = WriteBytesEx(handle, len, buffer);
```

9.830 ex_writei2cregister.nxc

This is an example of how to use the [WriteI2CRegister](#) function.

```
char result = WriteI2CRegister(S1, I2C_ADDR_DEFAULT, I2C_REG_CMD, US_CMD_OFF);
```

9.831 ex_WriteLn.nxc

This is an example of how to use the [WriteLn](#) function.

```
result = WriteLn(handle, value);
```

9.832 ex_WriteLnString.nxc

This is an example of how to use the [WriteLnString](#) function.

```
result = WriteLnString(handle, "testing", count);
```

9.833 ex_WriteNRLinkBytes.nxc

This is an example of how to use the [WriteNRLinkBytes](#) function.

```
byte data[] = {0x01, 0x02, 0x03};  
char result = WriteNRLinkBytes(S1, MS_ADDR_NRLINK, data);
```

9.834 ex_WriteString.nxc

This is an example of how to use the [WriteString](#) function.

```
result = WriteString(handle, "testing", count);
```

9.835 ex_yield.nxc

This is an example of how to use the [Yield](#) function.

```
task play() {  
    while (true) {  
        PlayTone(TONE_A4, MS_500);  
        Wait(SEC_1);  
    }  
}
```

```
}

task drive()
{
    while (true) {
        OnFwd(OUT_A, 50);
        Yield();
    }
}

task main()
{
    Precedes(drive, play);
}
```

9.836 glBoxDemo.nxc

This is an example of how to use the [glInit](#), [glBeginObject](#), [glBegin](#), [glAddVertex](#), [glEnd](#), [glEndObject](#), [glSetAngleX](#), [glBeginRender](#), [glAddToAngleY](#), [glCallObject](#), and [glFinishRender](#) functions.

```
/*-----
; File          : glBoxDemo.nbc
; Description   : A program demonstrating a 3D box...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
-----*/
task main()
{
    // Initialize the 3D engine...
    glInit();
    // Create a cube, this is the first object which will be object id 0...
    glCube(GL_POLYGON, 20);
    glBox(GL_POLYGON, 20, 30, 40);
    // Set the main view x-angle...
    glSetAngleX(45);
    while (true)
    {
        // Rotate the main view....
        glAddToAngleY(8);
        glAddToAngleX(4);
        // Setup for rendering...
        glBeginRender();
        // translate object 1
        glObjectAction(0, GL_TRANSLATE_X, 20);
        glObjectAction(1, GL_TRANSLATE_X, -20);
        // Call the object with id 0...
        glCallObject(0);
        glCallObject(1);
        // Finish, clear the screen, rotate and render the called objects...
        glFinishRender();
        Wait(MS_20);
    }
}
```

9.837 glCircleDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glBeginRender](#), [glAddToAngleY](#), [glAddToAngleX](#) [glCallObject](#), [glSet](#), and [glFinishRender](#) functions.

```
/*
-----+
; File          : glBoxDemo.nbc
; Description   : A program demonstrating a 3D box with circles on the edges...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----+*/
task main()
{
    // Initialize the 3D engine...
    glInit();
    // Create a box, this is the first object which will be object id 0...
    glBox(GL_POLYGON, 30, 30, 30);
    // Create a box, this is the second object which will be object id 1...
    glBox(GL_CIRCLE, 30, 30, 30);
    // Set the main view x-angle...
    glSetAngleX(45);
    while (true)
    {
        // Rotate the main view....
        glAddToAngleY(3);
        glAddToAngleX(5);
        // Setup for rendering...
        glBeginRender();
        glSet(GL_CULL_MODE, GL_CULL_NONE);
        // Call the object with id 0...
        glCallObject(0);
        // Call the object with id 1...
        glCallObject(1);
        // Finish, clear the screen, rotate and render the called objects...
        glFinishRender();
    }
}
```

9.838 glRotateDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glBeginRender](#), [glCallObject](#), [glObjectAction](#), and [glFinishRender](#) functions.

```
/*
-----+
; File          : glRotateDemo.nbc
; Description   : A program demonstrating two 3D boxes with rotate actions...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
;-----+*/
int angleX, angleY;

task main()
{
```

```
// Initialize the 3D engine...
glInit();
// Create a box, this is the first object which will be object id 0...
glBox(GL_POLYGON, 20, 20, 20);
// Create a box, this is the second object which will be object id 1...
glBox(GL_POLYGON, 40, 40, 40);
glSetAngleX(30);
angleX = 0;
angleY = 0;
while (true) {
    // Setup for rendering...
    glBeginRender();
    // Call the object with id 0...
    glCallObject(0);
    glObjectAction(0, GL_ROTATE_X, angleX);
    // Call the object with id 1...
    glCallObject(1);
    glObjectAction(1, GL_ROTATE_Y, angleY);
    // Finish, clear the screen, rotate and render the called objects...
    glFinishRender();
    angleX += 3;
    angleX %= 360;
    angleY += 5;
    angleY %= 360;
}
}
```

9.839 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);
}
}
```

9.840 glTranslateDemo.nxc

This is an example of how to use the [glInit](#), [glBox](#), [glSetAngleX](#), [glAddToAngleY](#), [glBeginRender](#), [glSet](#), [glCallObject](#), [globjectAction](#), and [glFinishRender](#) functions.

```
/*-----
; File          : glTranslateDemo.nbc
; Description   : A program demonstrating two 3D boxes with translate actions...
; Programmed by : Arno van der Vegt, avandervegt@home.nl
-----*/
int translateX, translateXStep,
    translateY, translateYStep,
    translateZ, translateZStep;

task main()
{
    // Initialize the 3D engine...
    glInit();

    // Create a box, this is the first object which will be object id 0...
    glBox(GL_POLYGON, 20, 20, 20);

    // Create a box, this is the second object which will be object id 1...
    glBox(GL_POLYGON, 40, 40, 40);

    glSetAngleX(30);

    translateX      = 0;
    translateXStep  = 1;
    translateY      = 0;
    translateYStep  = 1;
    translateZ      = 0;
    translateZStep  = 2;
    while (true)
    {
        // Rotate the main view....
        glAddToAngleY(2);
```

```
// Setup for rendering...
glBeginRender();
glSet(GL_CULL_MODE, GL_CULL_NONE);

// Call the object with id 0...
glCallObject(0);
glObjectAction(0, GL_TRANSLATE_X, translateX);
glObjectAction(0, GL_TRANSLATE_Z, translateZ);
glObjectAction(0, GL_TRANSLATE_Y, translateY);

// Call the object with id 1...
glCallObject(1);

// Finish, clear the screen, rotate and render the called objects...
glFinishRender();

translateX += translateXStep;
if (translateX >= 10)
    translateXStep = -1;
else if (translateX <= -10)
    translateXStep = 1;

translateZ += translateZStep;
if (translateZ >= 10)
    translateZStep = -2;
else if (translateZ <= -10)
    translateZStep = 2;

translateY += translateYStep;
if (translateY >= 10)
    translateYStep = -1;
else if (translateY <= -10)
    translateYStep = 1;
}
}
```

9.841 util_battery_1.nxc

This is an example of how to use the [BatteryLevel](#) function.

```
// Display battery voltage for four seconds.
// This version does not use floats and will work on NXTs with firmware 1.07.

// Note: the BatteryLevel API call returns an unsigned integer giving the
// battery level in millivolts.
task main()
{
    unsigned int millivolts = BatteryLevel();
    unsigned int int_volts = millivolts / 1000;
    millivolts %= 1000;
    string left = FormatNum("Battery: %d", int_volts);
    string right = FormatNum("%.03d", millivolts);
    TextOut(0, LCD_LINE2, StrCat(left, right), true);
    Wait(SEC_4);
```

```
}
```

9.842 util_battery_2.nxc

This is an example of how to use the [BatteryLevel](#) function.

```
// Display battery voltage for four seconds.  
// This version uses floats and requires NXTs with firmware 1.28 or later.  
  
// Note: the BatteryLevel API call returns an unsigned integer giving the  
// battery level in millivolts.  
task main()  
{  
    float volts = BatteryLevel() / 1000.0;  
    TextOut(0, LCD_LINE2, FormatNum("Battery: %5.3f", volts), true);  
    Wait(SEC_4);  
}
```

9.843 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, 1160
 SensorTypeModes, 213

A simple 3D graphics library, 194

abort
 cstdlibAPI, 519
 NXCDefs.h, 1183

AbortFlag
 NXCDefs.h, 1183
 UiModuleFunctions, 440

abs
 cstdlibAPI, 519
 NXCDefs.h, 1183

ACCL_CMD_RESET_CAL
 MSACCLNx, 788
 NBCCCommon.h, 917

ACCL_CMD_X_CAL
 MSACCLNx, 788
 NBCCCommon.h, 917

ACCL_CMD_X_CAL_END
 MSACCLNx, 788
 NBCCCommon.h, 917

ACCL_CMD_Y_CAL
 MSACCLNx, 788
 NBCCCommon.h, 917

ACCL_CMD_Y_CAL_END
 MSACCLNx, 788
 NBCCCommon.h, 918

ACCL_CMD_Z_CAL
 MSACCLNx, 789
 NBCCCommon.h, 918

ACCL_CMD_Z_CAL_END
 MSACCLNx, 789
 NBCCCommon.h, 918

ACCL_REG_SENS_LVL
 MSACCLNx, 789
 NBCCCommon.h, 918

ACCL_REG_X_ACCEL
 MSACCLNx, 789
 NBCCCommon.h, 918

ACCL_REG_X_OFFSET
 MSACCLNx, 789

NBCCCommon.h, 918

ACCL_REG_X_RANGE
 MSACCLNx, 789

ACCL_REG_X_TILT
 MSACCLNx, 789

ACCL_REG_Y_ACCEL
 MSACCLNx, 789

ACCL_REG_Y_OFFSET
 MSACCLNx, 789

ACCL_REG_Y_RANGE
 MSACCLNx, 789

ACCL_REG_Y_TILT
 MSACCLNx, 790

ACCL_REG_Z_ACCEL
 MSACCLNx, 790

ACCL_REG_Z_OFFSET
 MSACCLNx, 790

ACCL_REG_Z_RANGE
 MSACCLNx, 790

ACCL_REG_Z_TILT
 MSACCLNx, 790

ACCL_SENSITIVITY_LEVEL_1
 MSACCLNxSLevel, 791

ACCL_SENSITIVITY_LEVEL_2
 MSACCLNxSLevel, 791

ACCL_SENSITIVITY_LEVEL_3
 MSACCLNxSLevel, 791

ACCL_SENSITIVITY_LEVEL_4
 MSACCLNxSLevel, 791

NBCCCommon.h, 920
ACCLNxCalibrateX
 MindSensorsAPI, 119
 NXCDefs.h, 1184
ACCLNxCalibrateXEnd
 MindSensorsAPI, 119
 NXCDefs.h, 1184
ACCLNxCalibrateY
 MindSensorsAPI, 119
 NXCDefs.h, 1185
ACCLNxCalibrateYEnd
 MindSensorsAPI, 120
 NXCDefs.h, 1185
ACCLNxCalibrateZ
 MindSensorsAPI, 120
 NXCDefs.h, 1186
ACCLNxCalibrateZEnd
 MindSensorsAPI, 121
 NXCDefs.h, 1186
ACCLNxResetCalibration
 MindSensorsAPI, 121
 NXCDefs.h, 1186
ACCLNxSensitivity
 MindSensorsAPI, 122
 NXCDefs.h, 1187
ACCLNxXOffset
 MindSensorsAPI, 122
 NXCDefs.h, 1187
ACCLNxXRange
 MindSensorsAPI, 122
 NXCDefs.h, 1188
ACCLNxYOffset
 MindSensorsAPI, 123
 NXCDefs.h, 1188
ACCLNxYRange
 MindSensorsAPI, 123
 NXCDefs.h, 1189
ACCLNxZOffset
 MindSensorsAPI, 124
 NXCDefs.h, 1189
ACCLNxZRange
 MindSensorsAPI, 124
 NXCDefs.h, 1189
Acos
 cmathAPI, 479
 NXCDefs.h, 1160
acos
 cmathAPI, 491
 NXCDefs.h, 1190
AcosD
 cmathAPI, 479
 NXCDefs.h, 1161
acosd
 cmathAPI, 491
 NXCDefs.h, 1190
Acquire
 CommandModuleFunctions, 339
 NXCDefs.h, 1191
Action
 CommBTConnectionType, 802
ActualSpeed
 NBCCCommon.h, 920
 OutputFieldConstants, 650
addressOf
 cstringAPI, 530
 NXCDefs.h, 1191
addressOfEx
 cstringAPI, 530
 NXCDefs.h, 1192
Array API functions, 364
Array operation constants, 551
ArrayBuild
 ArrayFunctions, 366
 NXCDefs.h, 1192
ArrayFunctions
 ArrayBuild, 366
 ArrayInit, 366
 ArrayLen, 366
 ArrayMax, 367
 ArrayMean, 367
 ArrayMin, 368
 ArrayOp, 368
 ArraySort, 369
 ArrayStd, 369
 ArraySubset, 370
 ArraySum, 370
 ArraySumSqr, 371
ArrayInit
 ArrayFunctions, 366
 NXCDefs.h, 1193
ArrayLen
 ArrayFunctions, 366

NXCDefs.h, 1193
ArrayMax
 ArrayFunctions, 367
 NXCDefs.h, 1194
ArrayMean
 ArrayFunctions, 367
 NXCDefs.h, 1194
ArrayMin
 ArrayFunctions, 368
 NXCDefs.h, 1195
ArrayOp
 ArrayFunctions, 368
 NXCDefs.h, 1195
ArrayOpConstants
 OPARR_MAX, 552
 OPARR_MEAN, 552
 OPARR_MIN, 552
 OPARR_SORT, 552
 OPARR_STD, 552
 OPARR_SUM, 552
 OPARR_SUMSQR, 552
ArraySort
 ArrayFunctions, 369
 NXCDefs.h, 1196
ArrayStd
 ArrayFunctions, 369
 NXCDefs.h, 1196
ArraySubset
 ArrayFunctions, 370
 NXCDefs.h, 1197
ArraySum
 ArrayFunctions, 370
 NXCDefs.h, 1197
ArraySumSqr
 ArrayFunctions, 371
 NXCDefs.h, 1198
Asin
 cmathAPI, 480
 NXCDefs.h, 1161
asin
 cmathAPI, 492
 NXCDefs.h, 1198
AsinD
 cmathAPI, 480
 NXCDefs.h, 1162
asind
 cmathAPI, 492
 NXCDefs.h, 1162
 NXCDefs.h, 1199
Atan
 cmathAPI, 480
 NXCDefs.h, 1162
atan
 cmathAPI, 493
 NXCDefs.h, 1199
Atan2
 cmathAPI, 481
 NXCDefs.h, 1162
atan2
 cmathAPI, 494
 NXCDefs.h, 1200
Atan2D
 cmathAPI, 481
 NXCDefs.h, 1163
atan2d
 cmathAPI, 494
 NXCDefs.h, 1200
AtanD
 cmathAPI, 482
 NXCDefs.h, 1163
atand
 cmathAPI, 495
 NXCDefs.h, 1201
atof
 cstdlibAPI, 520
 NXCDefs.h, 1201
atoi
 cstdlibAPI, 521
 NXCDefs.h, 1202
atol
 cstdlibAPI, 521
 NXCDefs.h, 1203
Basic analog sensor value names, 240
BasicSensorValues
 SENSOR_1, 241
 SENSOR_2, 241
 SENSOR_3, 241
 SENSOR_4, 241
BatteryLevel
 NXCDefs.h, 1203
 UiModuleFunctions, 441
BatteryState

NXCDefs.h, 1204
UiModuleFunctions, 441
BaudRate
 CommHSControlType, 809
bcd2dec
 cmathAPI, 495
 NXCDefs.h, 1204
BITMAP_1
 DisplayModuleConstants, 676
 NBCCommon.h, 920
BITMAP_2
 DisplayModuleConstants, 676
 NBCCommon.h, 920
BITMAP_3
 DisplayModuleConstants, 676
 NBCCommon.h, 920
BITMAP_4
 DisplayModuleConstants, 676
 NBCCommon.h, 920
BITMAPS
 DisplayModuleConstants, 676
 NBCCommon.h, 920
BlockTachoCount
 NBCCommon.h, 920
 OutputFieldConstants, 650
Bluetooth hardware status constants, 698
Bluetooth State constants, 697
Bluetooth state status constants, 697
BluetoothState
 NXCDefs.h, 1204
 UiModuleFunctions, 441
BluetoothState constants, 620
BluetoothStatus
 CommModuleFunctions, 383
 NXCDefs.h, 1205
BluetoothWrite
 CommModuleFunctions, 383
 NXCDefs.h, 1205
BREAKOUT_REQ
 CommandVMState, 575
 NBCCommon.h, 921
BrickDataBluecoreVersion
 CommModuleFunctions, 384
 NXCDefs.h, 1206
BrickDataBtHardwareStatus
 CommModuleFunctions, 384
NXCDefs.h, 1206
BrickDataBtStateStatus
 CommModuleFunctions, 384
 NXCDefs.h, 1206
BrickDataName
 CommModuleFunctions, 385
 NXCDefs.h, 1207
BrickDataTimeoutValue
 CommModuleFunctions, 385
 NXCDefs.h, 1207
BT_ARM_CMD_MODE
 CommBtStateConstants, 697
 NBCCommon.h, 921
BT_ARM_DATA_MODE
 CommBtStateConstants, 697
 NBCCommon.h, 921
BT_ARM_OFF
 CommBtStateConstants, 697
 NBCCommon.h, 921
BT_BRICK_PORT_OPEN
 CommBtStateStatusConstants, 698
 NBCCommon.h, 921
BT_BRICK_VISIBILITY
 CommBtStateStatusConstants, 698
 NBCCommon.h, 921
BT_CMD_BYTE
 CommMiscConstants, 695
 NBCCommon.h, 921
BT_CMD_READY
 CommStatusCodesConstants, 712
 NBCCommon.h, 921
BT_CONNECTION_0_ENABLE
 CommBtStateStatusConstants, 698
 NBCCommon.h, 921
BT_CONNECTION_1_ENABLE
 CommBtStateStatusConstants, 698
 NBCCommon.h, 922
BT_CONNECTION_2_ENABLE
 CommBtStateStatusConstants, 698
 NBCCommon.h, 922
BT_CONNECTION_3_ENABLE
 CommBtStateStatusConstants, 698
 NBCCommon.h, 922
BT_DEFAULT_INQUIRY_MAX
 CommMiscConstants, 695
 NBCCommon.h, 922

BT_DEFAULT_INQUIRY_TIMEOUT_-
 LO
 CommMiscConstants, 695
 NBCCommon.h, 922

BT_DEVICE_AWAY
 CommDeviceStatusConstants, 708
 NBCCommon.h, 922

BT_DEVICE_EMPTY
 CommDeviceStatusConstants, 708
 NBCCommon.h, 922

BT_DEVICE_KNOWN
 CommDeviceStatusConstants, 708
 NBCCommon.h, 922

BT_DEVICE_NAME
 CommDeviceStatusConstants, 709
 NBCCommon.h, 922

BT_DEVICE_UNKNOWN
 CommDeviceStatusConstants, 709
 NBCCommon.h, 922

BT_DISABLE
 CommBtHwStatusConstants, 699
 NBCCommon.h, 923

BT_ENABLE
 CommBtHwStatusConstants, 699
 NBCCommon.h, 923

BTConnectionClass
 CommModuleFunctions, 385
 NXCDdefs.h, 1207

BTConnectionHandleNum
 CommModuleFunctions, 386
 NXCDdefs.h, 1208

BTConnectionLinkQuality
 CommModuleFunctions, 386
 NXCDdefs.h, 1208

BTConnectionName
 CommModuleFunctions, 386
 NXCDdefs.h, 1208

BTConnectionPinCode
 CommModuleFunctions, 387
 NXCDdefs.h, 1209

BTConnectionStreamStatus
 CommModuleFunctions, 387
 NXCDdefs.h, 1209

BTDeviceClass
 CommModuleFunctions, 388
 NXCDdefs.h, 1210

 BTDeviceCount
 CommModuleFunctions, 388
 NXCDdefs.h, 1210

 BTDeviceName
 CommModuleFunctions, 388
 NXCDdefs.h, 1210

 BTDeviceNameCount
 CommModuleFunctions, 389
 NXCDdefs.h, 1211

 BTDeviceStatus
 CommModuleFunctions, 389
 NXCDdefs.h, 1211

 BTInputBufferInPtr
 CommModuleFunctions, 389
 NXCDdefs.h, 1211

 BTInputBufferOutPtr
 CommModuleFunctions, 390
 NXCDdefs.h, 1212

 BTN1
 ButtonNameConstants, 610
 NBCCommon.h, 923

 BTN2
 ButtonNameConstants, 610
 NBCCommon.h, 923

 BTN3
 ButtonNameConstants, 610
 NBCCommon.h, 923

 BTN4
 ButtonNameConstants, 610
 NBCCommon.h, 923

 BTNCENTER
 ButtonNameConstants, 611
 NBCCommon.h, 923

 BTNEXTIT
 ButtonNameConstants, 611
 NBCCommon.h, 924

 BTNLEFT
 ButtonNameConstants, 611
 NBCCommon.h, 924

 BTNRIGHT
 ButtonNameConstants, 611
 NBCCommon.h, 924

 BTNSTATE_LONG_PRESSED_EV
 ButtonStateConstants, 612
 NBCCommon.h, 924

 BTNSTATE_LONG_RELEASED_EV

ButtonStateConstants, 612
 NBCCCommon.h, 924

BTNSTATE_NONE
 ButtonStateConstants, 612
 NBCCCommon.h, 924

BTNSTATE_PRESSED_EV
 ButtonStateConstants, 612
 NBCCCommon.h, 925

BTNSTATE_PRESSED_STATE
 ButtonStateConstants, 613
 NBCCCommon.h, 925

BTNSTATE_SHORT_RELEASED_EV
 ButtonStateConstants, 613
 NBCCCommon.h, 925

BTOOutputBufferInPtr
 CommModuleFunctions, 390
 NXCDefs.h, 1212

BTOOutputBufferOutPtr
 CommModuleFunctions, 390
 NXCDefs.h, 1212

Buffer
 CommBTWriteType, 804
 CommHSReadWriteType, 810
 CommLSReadType, 813
 CommLSWriteExType, 814
 CommLSWriteType, 816
 FileReadWriteType, 843
 IOMapReadByIDType, 851
 IOMapReadType, 853
 IOMapWriteByIDType, 854
 IOMapWriteType, 856
 LoaderExecuteFunctionType, 862

BufferLen
 CommLSReadType, 813

Button module, 53

Button module constants, 609

Button module functions, 430

Button module IOMAP offsets, 613

Button module types, 430

Button name constants, 609

ButtonCount
 ButtonModuleFunctions, 432
 NXCDefs.h, 1213

ButtonIOMAP
 ButtonOffsetLongPressCnt, 613
 ButtonOffsetLongRelCnt, 613

 ButtonOffsetPressedCnt, 614
 ButtonOffsetRelCnt, 614
 ButtonOffsetShortRelCnt, 614
 ButtonOffsetState, 614

ButtonLongPressCount
 ButtonModuleFunctions, 432
 NXCDefs.h, 1213

ButtonLongReleaseCount
 ButtonModuleFunctions, 432
 NXCDefs.h, 1214

ButtonModuleFunctions
 ButtonCount, 432
 ButtonLongPressCount, 432
 ButtonLongReleaseCount, 432
 ButtonPressCount, 433
 ButtonPressed, 433
 ButtonReleaseCount, 433
 ButtonShortReleaseCount, 434
 ButtonState, 434
 ReadButtonEx, 435
 SetButtonLongPressCount, 435
 SetButtonLongReleaseCount, 435
 SetButtonPressCount, 436
 SetButtonReleaseCount, 436
 SetButtonShortReleaseCount, 436
 SetButtonState, 437
 SysReadButton, 437

ButtonModuleID
 ModuleIDConstants, 190
 NBCCCommon.h, 925

ButtonModuleName
 ModuleNameConstants, 188
 NBCCCommon.h, 925

ButtonNameConstants
 BTN1, 610
 BTN2, 610
 BTN3, 610
 BTN4, 610
 BTNCENTER, 611
 BTNEXTIT, 611
 BTNLEFT, 611
 BTNRIGHT, 611
 NO_OF_BTNS, 611

ButtonOffsetLongPressCnt
 ButtonIOMAP, 613
 NBCCCommon.h, 925

ButtonOffsetLongRelCnt
 ButtonIOMAP, 613
 NBCCCommon.h, 925

ButtonOffsetPressedCnt
 ButtonIOMAP, 614
 NBCCCommon.h, 925

ButtonOffsetRelCnt
 ButtonIOMAP, 614
 NBCCCommon.h, 925

ButtonOffsetShortRelCnt
 ButtonIOMAP, 614
 NBCCCommon.h, 926

ButtonOffsetState
 ButtonIOMAP, 614
 NBCCCommon.h, 926

ButtonPressCount
 ButtonModuleFunctions, 433
 NXCDefs.h, 1214

ButtonPressed
 ButtonModuleFunctions, 433
 NXCDefs.h, 1214

ButtonReleaseCount
 ButtonModuleFunctions, 433
 NXCDefs.h, 1215

ButtonShortReleaseCount
 ButtonModuleFunctions, 434
 NXCDefs.h, 1215

ButtonState
 ButtonModuleFunctions, 434
 NXCDefs.h, 1216

ButtonState constants, 612

ButtonStateConstants
 BTNSTATE_LONG_PRESSED_-
 EV, 612
 BTNSTATE_LONG_RELEASED_-
 EV, 612
 BTNSTATE_NONE, 612
 BTNSTATE_PRESSED_EV, 612
 BTNSTATE_PRESSED_STATE,
 613
 BTNSTATE_SHORT_-
 RELEASED_EV, 613

ByteArrayToStr
 cstringAPI, 531
 NXCDefs.h, 1216

ByteArrayToStrEx

cstringAPI, 531
 NXCDefs.h, 1216

BytesReady
 CommLSCheckStatusType, 811

Ceil
 cmathAPI, 482
 NXCDefs.h, 1164

ceil
 cmathAPI, 496
 NXCDefs.h, 1217

Center
 DrawCircleType, 824
 DrawEllipseType, 826

CHAR_BIT
 NBCCCommon.h, 926
 NXTLimits, 792

CHAR_MAX
 NBCCCommon.h, 926
 NXTLimits, 792

CHAR_MIN
 NBCCCommon.h, 926
 NXTLimits, 792

CircleOut
 DisplayModuleFunctions, 274
 NXCDefs.h, 1217

ClearBits
 WriteSemDataType, 881

ClearLine
 DisplayModuleFunctions, 275
 NXCDefs.h, 1218

ClearScreen
 DisplayModuleFunctions, 275
 NXCDefs.h, 1218

ClearSensor
 InputModuleFunctions, 219
 NXCDefs.h, 1219

CloseFile
 LoaderModuleFunctions, 455
 NXCDefs.h, 1219

CLUMP_DONE
 CommandVMState, 575
 NBCCCommon.h, 926

CLUMP_SUSPEND
 CommandVMState, 575
 NBCCCommon.h, 926

cmath API, 474
cmathAPI
 Acos, 479
 acos, 491
 AcosD, 479
 acosd, 491
 Asin, 480
 asin, 492
 AsinD, 480
 asind, 492
 Atan, 480
 atan, 493
 Atan2, 481
 atan2, 494
 Atan2D, 481
 atan2d, 494
 AtanD, 482
 atand, 495
 bcd2dec, 495
 Ceil, 482
 ceil, 496
 Cos, 482
 cos, 496
 CosD, 483
 cosd, 496
 Cosh, 483
 cosh, 497
 CoshD, 484
 coshd, 497
 Exp, 484
 exp, 498
 Floor, 484
 floor, 498
 Frac, 485
 frac, 499
 isNAN, 499
 Log, 485
 log, 500
 Log10, 486
 log10, 500
 MulDiv32, 486
 muldiv32, 501
 Pow, 487
 pow, 501
 sign, 502
 Sin, 487
 sin, 502
 SinD, 487
 sind, 503
 Sinh, 488
 sinh, 503
 SinhD, 488
 sinhd, 504
 Sqrt, 489
 sqrt, 504
 Tan, 489
 tan, 504
 TanD, 489
 tand, 505
 Tanh, 490
 tanh, 505
 TanhD, 490
 tanhd, 506
 Trunc, 491
 trunc, 506
Cmd
 CommExecuteFunctionType, 806
 DisplayExecuteFunctionType, 821
 LoaderExecuteFunctionType, 862
Coast
 NXCDefs.h, 1219
 OutputModuleFunctions, 246
CoastEx
 NXCDefs.h, 1220
 OutputModuleFunctions, 246
Color calibration constants, 635
Color calibration state constants, 634
Color sensor array indices, 632
Color values, 633
ColorADRaw
 InputModuleFunctions, 219
 NXCDefs.h, 1220
ColorBoolean
 InputModuleFunctions, 220
 NXCDefs.h, 1221
ColorCalibration
 InputModuleFunctions, 220
 NXCDefs.h, 1221
ColorCalibrationState
 InputModuleFunctions, 221
 NXCDefs.h, 1222
ColorCalLimits

InputModuleFunctions, 221
NXCDefs.h, 1222

ColorSensorRaw
InputModuleFunctions, 222
NXCDefs.h, 1223

ColorSensorRead
NBCCommon.h, 926
SysCallConstants, 554

ColorSensorReadType, 798
ColorValue, 799
Invalid, 799
NormalizedArray, 799
Port, 799
RawArray, 800
Result, 800
ScaledArray, 800

ColorSensorValue
InputModuleFunctions, 222
NXCDefs.h, 1223

ColorValue
ColorSensorReadType, 799

COM_CHANNEL_FOUR_ACTIVE
LowSpeedStateConstants, 659
NBCCommon.h, 926

COM_CHANNEL_NONE_ACTIVE
LowSpeedStateConstants, 659
NBCCommon.h, 926

COM_CHANNEL_ONE_ACTIVE
LowSpeedStateConstants, 659
NBCCommon.h, 927

COM_CHANNEL_THREE_ACTIVE
LowSpeedStateConstants, 659
NBCCommon.h, 927

COM_CHANNEL_TWO_ACTIVE
LowSpeedStateConstants, 659
NBCCommon.h, 927

Combined sensor type and mode constants, 211

Comm module, 52

Comm module constants, 693

Comm module functions, 374

Comm module interface function constants, 709

Comm module IOMAP offsets, 713

Comm module status code constants, 712

Comm module types, 373

Command
CommHSControlType, 809

Command module, 50

Command module constants, 50

Command module functions, 334

Command module IOMAP offsets, 581

Command module types, 333

CommandCommErrors
ERR_COMM_BUFFER_FULL, 579
ERR_COMM_BUS_ERR, 579
ERR_COMM_CHAN_INVALID, 579
ERR_COMM_CHAN_NOT_READY, 579

CommandFatalErrors
ERR_ARG, 576
ERR_BAD_POOL_SIZE, 576
ERR_BAD_PTR, 576
ERR_CLUMP_COUNT, 576
ERR_DEFAULT_OFFSETS, 576
ERR_FILE, 576
ERR_INSANE_OFFSET, 576
ERR_INSTR, 577
ERR_LOADER_ERR, 577
ERR_MEM, 577
ERR_MEMMGR_FAIL, 577
ERR_NO_ACTIVE_CLUMP, 577
ERR_NO_CODE, 577
ERR_NON_FATAL, 577
ERR_SPOTCHECK_FAIL, 577
ERR_VER, 577

CommandFlags
NXCDefs.h, 1224
UiModuleFunctions, 442

CommandFlags constants, 615

CommandGenErrors
ERR_INVALID_FIELD, 578
ERR_INVALID_PORT, 578
ERR_INVALID_QUEUE, 578
ERR_INVALID_SIZE, 578
ERR_NO_PROG, 578

CommandIOMAP
CommandOffsetActivateFlag, 582
CommandOffsetAwake, 582
CommandOffsetDeactivateFlag, 582

CommandOffsetFileName, 582
CommandOffsetFormatString, 582
CommandOffsetMemoryPool, 582
CommandOffsetOffsetDS, 583
CommandOffsetOffsetDVA, 583
CommandOffsetPRCHandler, 583
CommandOffsetProgStatus, 583
CommandOffsetSyncTick, 583
CommandOffsetSyncTime, 583
CommandOffsetTick, 583
CommandModuleConstants
 NO_ERR, 51
 STAT_COMM_PENDING, 52
 STAT_MSG_EMPTY_MAILBOX,
 52
CommandModuleFunctions
 Acquire, 339
 CurrentTick, 339
 ExitTo, 340
 FirstTick, 340
 Follows, 340
 GetButtonModuleValue, 341
 GetCommandModuleBytes, 341
 GetCommandModuleValue, 342
 GetCommModuleBytes, 342
 GetCommModuleValue, 342
 GetDisplayModuleBytes, 343
 GetDisplayModuleValue, 343
 GetInputModuleValue, 344
 GetIOMapBytes, 344
 GetIOMapBytesByID, 344
 GetIOMapValue, 345
 GetIOMapValueByID, 345
 GetLoaderModuleValue, 346
 GetLowSpeedModuleBytes, 346
 GetLowSpeedModuleValue, 347
 GetOutputModuleValue, 347
 GetSoundModuleValue, 347
 GetUIModuleValue, 348
 Precedes, 348
 Release, 348
 ResetSleepTimer, 349
 SetButtonModuleValue, 349
 SetCommandModuleBytes, 349
 SetCommandModuleValue, 350
 SetCommModuleBytes, 350
 SetCommModuleValue, 351
 SetDisplayModuleBytes, 351
 SetDisplayModuleValue, 351
 SetInputModuleValue, 352
 SetIOCctrlModuleValue, 352
 SetIOMapBytes, 352
 SetIOMapBytesByID, 353
 SetIOMapValue, 353
 SetIOMapValueByID, 354
 SetLoaderModuleValue, 354
 SetLowSpeedModuleBytes, 355
 SetLowSpeedModuleValue, 355
 SetOutputModuleValue, 356
 SetSoundModuleValue, 356
 SetUIModuleValue, 356
 StartTask, 357
 Stop, 357
 StopAllTasks, 357
 StopTask, 357
 SysCall, 358
 SysComputeCalibValue, 358
 SysDatalogGetTimes, 359
 SysDatalogWrite, 359
 SysGetStartTick, 360
 SysIOMapRead, 360
 SysIOMapReadByID, 360
 SysIOMapWrite, 361
 SysIOMapWriteByID, 361
 SysKeepAlive, 362
 SysReadSemData, 362
 SysUpdateCalibCacheInfo, 362
 SysWriteSemData, 363
 Wait, 363
 Yield, 364
CommandModuleID
 ModuleIDConstants, 190
 NBCCCommon.h, 927
CommandModuleName
 ModuleNameConstants, 188
 NBCCCommon.h, 927
CommandOffsetActivateFlag
 CommandIOMAP, 582
 NBCCCommon.h, 927
CommandOffsetAwake
 CommandIOMAP, 582
 NBCCCommon.h, 927

CommandOffsetDeactivateFlag
 CommandIOMAP, 582
 NBCCCommon.h, 927

CommandOffsetFileName
 CommandIOMAP, 582
 NBCCCommon.h, 928

CommandOffsetFormatString
 CommandIOMAP, 582
 NBCCCommon.h, 928

CommandOffsetMemoryPool
 CommandIOMAP, 582
 NBCCCommon.h, 928

CommandOffsetOffsetDS
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetOffsetDVA
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetPRCHandler
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetProgStatus
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetSyncTick
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetSyncTime
 CommandIOMAP, 583
 NBCCCommon.h, 928

CommandOffsetTick
 CommandIOMAP, 583
 NBCCCommon.h, 929

CommandProgStatus
 PROG_ABORT, 581
 PROG_ERROR, 581
 PROG_IDLE, 581
 PROG_OK, 581
 PROG_RESET, 581
 PROG_RUNNING, 581

CommandRCErrors
 ERR_RC_BAD_PACKET, 580
 ERR_RC_FAILED, 580
 ERR_RC_ILLEGAL_VAL, 580
 ERR_RC_UNKNOWN_CMD, 580

CommandVMState

BREAKOUT_REQ, 575

CLUMP_DONE, 575

CLUMP_SUSPEND, 575

ROTATE_QUEUE, 575

STOP_REQ, 575

TIMES_UP, 575

CommBTCheckStatus
 NBCCCommon.h, 929
 SysCallConstants, 554

CommBTCheckStatusType, 800
 Connection, 801
 Result, 801

CommBTConnection
 NBCCCommon.h, 929
 SysCallConstants, 554

CommBTConnectionType, 801
 Action, 802
 ConnectionSlot, 802
 Name, 802
 Result, 802

CommBtHwStatusConstants
 BT_DISABLE, 699
 BT_ENABLE, 699

CommBTOnOff
 NBCCCommon.h, 929
 SysCallConstants, 555

CommBTOnOffType, 803
 PowerState, 803
 Result, 803

CommBTRead
 NBCCCommon.h, 929
 SysCallConstants, 555

CommBtStateConstants
 BT_ARM_CMD_MODE, 697
 BT_ARM_DATA_MODE, 697
 BT_ARM_OFF, 697

CommBtStateStatusConstants
 BT_BRICK_PORT_OPEN, 698
 BT_BRICK_VISIBILITY, 698
 BT_CONNECTION_0_ENABLE,
 698
 BT_CONNECTION_1_ENABLE,
 698
 BT_CONNECTION_2_ENABLE,
 698

BT_CONNECTION_3_ENABLE, 698
CommBTWrite
 NBCCCommon.h, 929
 SysCallConstants, 555
CommBTWriteType, 804
 Buffer, 804
 Connection, 804
 Result, 805
CommDeviceStatusConstants
 BT_DEVICE_AWAY, 708
 BT_DEVICE_EMPTY, 708
 BT_DEVICE_KNOWN, 708
 BT_DEVICE_NAME, 709
 BT_DEVICE_UNKNOWN, 709
CommExecuteFunction
 NBCCCommon.h, 929
 SysCallConstants, 555
CommExecuteFunctionType, 805
 Cmd, 806
 Name, 807
 Param1, 807
 Param2, 807
 Param3, 807
 Result, 807
 RetVal, 807
CommHiSpeedBaudConstants
 HS_BAUD_115200, 703
 HS_BAUD_1200, 703
 HS_BAUD_14400, 703
 HS_BAUD_19200, 703
 HS_BAUD_230400, 703
 HS_BAUD_2400, 703
 HS_BAUD_28800, 704
 HS_BAUD_3600, 704
 HS_BAUD_38400, 704
 HS_BAUD_460800, 704
 HS_BAUD_4800, 704
 HS_BAUD_57600, 704
 HS_BAUD_7200, 704
 HS_BAUD_76800, 704
 HS_BAUD_921600, 704
 HS_BAUD_9600, 704
CommHiSpeedCombinedConstants
 HS_MODE_7E1, 708
 HS_MODE_8N1, 708
 CommHiSpeedCtrlConstants
 HS_CTRL_EXIT, 702
 HS_CTRL_INIT, 702
 HS_CTRL_UART, 702
 CommHiSpeedDataBitsConstants
 HS_MODE_5_DATA, 705
 HS_MODE_6_DATA, 705
 HS_MODE_7_DATA, 705
 HS_MODE_8_DATA, 705
 CommHiSpeedFlagsConstants
 HS_UPDATE, 700
 CommHiSpeedParityConstants
 HS_MODE_E_PARITY, 707
 HS_MODE_M_PARITY, 707
 HS_MODE_N_PARITY, 707
 HS_MODE_O_PARITY, 707
 HS_MODE_S_PARITY, 707
 CommHiSpeedStateConstants
 HS_DISABLE, 701
 HS_ENABLE, 701
 HS_INIT_RECEIVER, 701
 HS_INITIALISE, 701
 HS_SEND_DATA, 701
 CommHiSpeedStopBitsConstants
 HS_MODE_10_STOP, 706
 HS_MODE_15_STOP, 706
 HS_MODE_20_STOP, 706
 CommHSCheckStatus
 NBCCCommon.h, 929
 SysCallConstants, 555
 CommHSCheckStatusType, 807
 DataAvailable, 808
 SendingData, 808
 CommHSControl
 NBCCCommon.h, 929
 SysCallConstants, 555
 CommHSControlType, 808
 BaudRate, 809
 Command, 809
 Mode, 809
 Result, 809
 CommHSRead
 NBCCCommon.h, 930
 SysCallConstants, 555
 CommHSReadWriteType, 810
 Buffer, 810

Status, 810
CommHSWrite
 NBCCCommon.h, 930
 SysCallConstants, 555
CommInterfaceConstants
 INTF_BTOFF, 710
 INTF_BTTON, 710
 INTF_CONNECT, 710
 INTF_CONNECTBYNAME, 710
 INTF_CONNECTREQ, 710
 INTF_DISCONNECT, 710
 INTF_DISCONNECTALL, 710
 INTF_EXTREAD, 710
 INTF_FACTORYRESET, 711
 INTF_OPENSTREAM, 711
 INTF_PINREQ, 711
 INTF_REMOVEDevice, 711
 INTF_SEARCH, 711
 INTF_SENDDATA, 711
 INTF_SENDFILE, 711
 INTF_SETBTNAME, 711
 INTF_SETCMDMODE, 711
 INTF_STOPSEARCH, 711
 INTF_VISIBILITY, 712
CommIOMAP
 CommOffsetBrickDataBdAddr, 714
 CommOffsetBrickDataBluecoreVersion, 714
 CommOffsetBrickDataBtHwStatus, 715
 CommOffsetBrickDataBtStateStatus, 715
 CommOffsetBrickDataName, 715
 CommOffsetBrickDataTimeOutValue, 715
 CommOffsetBtConnectTableBdAddr, 715
 CommOffsetBtConnectTableClassOfDevice, 715
 CommOffsetBtConnectTableHandleNr, 715
 CommOffsetBtConnectTableLinkQuality, 715
 CommOffsetBtConnectTableName, 715
CommOffsetBtConnectTablePinCode, 715
CommOffsetBtConnectTableStatus, 716
CommOffsetBtDeviceCnt, 716
CommOffsetBtDeviceNameCnt, 716
CommOffsetBtDeviceTableBdAddr, 716
CommOffsetBtDeviceTableClassOfDevice, 716
CommOffsetBtDeviceTableDeviceStatus, 716
CommOffsetBtDeviceTableName, 716
CommOffsetBtInBufBuf, 716
CommOffsetBtInBufInPtr, 716
CommOffsetBtInBufOutPtr, 716
CommOffsetBtOutBufBuf, 717
CommOffsetBtOutBufInPtr, 717
CommOffsetBtOutBufOutPtr, 717
CommOffsetHsFlags, 717
CommOffsetHsInBufBuf, 717
CommOffsetHsInBufInPtr, 717
CommOffsetHsInBufOutPtr, 717
CommOffsetHsMode, 717
CommOffsetHsOutBufBuf, 717
CommOffsetHsOutBufInPtr, 717
CommOffsetHsOutBufOutPtr, 718
CommOffsetHsSpeed, 718
CommOffsetHsState, 718
CommOffsetPFunc, 718
CommOffsetPFuncTwo, 718
CommOffsetUsbInBufBuf, 718
CommOffsetUsbInBufInPtr, 718
CommOffsetUsbInBufOutPtr, 718
CommOffsetUsbOutBufBuf, 718
CommOffsetUsbOutBufInPtr, 718
CommOffsetUsbOutBufOutPtr, 719
CommOffsetUsbPollBufBuf, 719
CommOffsetUsbPollBufInPtr, 719
CommOffsetUsbPollBufOutPtr, 719
CommOffsetUsbState, 719
CommLSCheckStatus
 NBCCCommon.h, 930
 SysCallConstants, 555
CommLSCheckStatusType, 811

BytesReady, 811
Port, 811
Result, 812
CommLSRead
 NBCCCommon.h, 930
 SysCallConstants, 555
CommLSReadType, 812
 Buffer, 813
 BufferLen, 813
 Port, 813
 Result, 813
CommLSWrite
 NBCCCommon.h, 930
 SysCallConstants, 556
CommLSWriteEx
 NBCCCommon.h, 930
 SysCallConstants, 556
CommLSWriteExType, 814
 Buffer, 814
 NoRestartOnRead, 814
 Port, 814
 Result, 815
 ReturnLen, 815
CommLSWriteType, 815
 Buffer, 816
 Port, 816
 Result, 816
 ReturnLen, 816
CommMiscConstants
 BT_CMD_BYT, 695
 BT_DEFAULT_INQUIRY_MAX, 695
 BT_DEFAULT_INQUIRY_- TIMEOUT_LO, 695
 MAX_BT_MSG_SIZE, 695
 SIZE_OF_BDADDR, 695
 SIZE_OF_BRICK_NAME, 695
 SIZE_OF_BT_CONNECT_- TABLE, 696
 SIZE_OF_BT_DEVICE_TABLE, 696
 SIZE_OF_BT_NAME, 696
 SIZE_OF_BT_PINCODE, 696
 SIZE_OF_BTBUF, 696
 SIZE_OF_CLASS_OF_DEVICE, 696
 SIZE_OF_HSBUF, 696
 SIZE_OF_USBBUF, 696
 SIZE_OF_USBDATA, 696
 USB_PROTOCOL_OVERHEAD, 696
CommModuleFunctions
 BluetoothStatus, 383
 BluetoothWrite, 383
 BrickDataBluecoreVersion, 384
 BrickDataBtHardwareStatus, 384
 BrickDataBtStateStatus, 384
 BrickDataName, 385
 BrickDataTimeoutValue, 385
 BTConnectionClass, 385
 BTConnectionHandleNum, 386
 BTConnectionLinkQuality, 386
 BTConnectionName, 386
 BTConnectionPinCode, 387
 BTConnectionStreamStatus, 387
 BTDeviceClass, 388
 BTDeviceCount, 388
 BTDeviceName, 388
 BTDeviceNameCount, 389
 BTDeviceStatus, 389
 BTInputBufferInPtr, 389
 BTInputBufferOutPtr, 390
 BTOutputBufferInPtr, 390
 BTOutputBufferOutPtr, 390
 GetBrickDataAddress, 391
 GetBTConnectionAddress, 391
 GetBTDeviceAddress, 391
 GetBTInputBuffer, 392
 GetBTOutputBuffer, 392
 GetHSInputBuffer, 393
 GetHSOutputBuffer, 393
 GetUSBInputBuffer, 394
 GetUSBOutputBuffer, 394
 GetUSBPollBuffer, 394
 HSFlags, 395
 HSInputBufferInPtr, 395
 HSInputBufferOutPtr, 395
 HSMode, 396
 HSOutputBufferInPtr, 396
 HSOutputBufferOutPtr, 396
 HSSpeed, 397
 HSState, 397

ReceiveMessage, 397
ReceiveRemoteBool, 398
ReceiveRemoteMessageEx, 398
ReceiveRemoteNumber, 399
ReceiveRemoteString, 399
RemoteKeepAlive, 400
RemoteMessageRead, 400
RemoteMessageWrite, 401
RemotePlaySoundFile, 401
RemotePlayTone, 402
RemoteResetMotorPosition, 402
RemoteResetScaledValue, 403
RemoteSetInputMode, 403
RemoteSetOutputState, 404
RemoteStartProgram, 405
RemoteStopProgram, 405
RemoteStopSound, 405
RS485Control, 406
RS485DataAvailable, 406
RS485Exit, 407
RS485Init, 407
RS485Read, 408
RS485SendingData, 408
RS485Status, 408
RS485Uart, 409
RS485Write, 409
SendMessage, 410
SendRemoteBool, 410
SendRemoteNumber, 411
SendRemoteString, 411
SendResponseBool, 412
SendResponseNumber, 412
SendResponseString, 413
SendRS485Bool, 413
SendRS485Number, 413
SendRS485String, 414
SetBTInputBuffer, 414
SetBTInputBufferInPtr, 415
SetBTInputBufferOutPtr, 415
SetBTOutputBuffer, 415
SetBTOutputBufferInPtr, 416
SetBTOutputBufferOutPtr, 416
SetHSFlags, 416
SetHSInputBuffer, 417
SetHSInputBufferInPtr, 417
SetHSInputBufferOutPtr, 417
SetHSMode, 418
SetHSOutputBuffer, 418
SetHSOutputBufferInPtr, 418
SetHSOutputBufferOutPtr, 419
SetHSSpeed, 419
SetHSState, 419
SetUSBInputBuffer, 420
SetUSBInputBufferInPtr, 420
SetUSBInputBufferOutPtr, 420
SetUSBOutputBuffer, 421
SetUSBOutputBufferInPtr, 421
SetUSBOutputBufferOutPtr, 421
SetUSBPollBuffer, 422
SetUSBPollBufferInPtr, 422
SetUSBPollBufferOutPtr, 422
SetUSBState, 423
SysCommBTCheckStatus, 423
SysCommBTConnection, 423
SysCommBTOnOff, 424
SysCommBTWrite, 424
SysCommExecuteFunction, 425
SysCommHSCheckStatus, 425
SysCommHSControl, 425
SysCommHSRead, 426
SysCommHSWrite, 426
SysMessageRead, 427
SysMessageWrite, 427
USBInputBufferInPtr, 427
USBInputBufferOutPtr, 428
USBOutputBufferInPtr, 428
USBOutputBufferOutPtr, 428
USBPollBufferInPtr, 429
USBPollBufferOutPtr, 429
USBState, 429
CommModuleID
 ModuleIDConstants, 190
 NBCCommon.h, 930
CommModuleName
 ModuleNameConstants, 188
 NBCCommon.h, 930
CommOffsetBrickDataBdAddr
 CommIOMAP, 714
 NBCCommon.h, 930
CommOffsetBrickDataBluecoreVersion
 CommIOMAP, 714
 NBCCommon.h, 930

CommOffsetBrickDataBtHwStatus
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBrickDataBtStateStatus
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBrickDataName
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBrickDataTimeOutValue
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableBdAddr
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableClassOfDevice
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableHandleNr
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableLinkQuality
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableName
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTablePinCode
 CommIOMAP, 715
 NBCCCommon.h, 931

CommOffsetBtConnectTableStreamStatus
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceCnt
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceNameCnt
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceTableBdAddr
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceTableClassOfDevice
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceTableDeviceStatus
 CommIOMAP, 716
 NBCCCommon.h, 932

CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtDeviceTableName
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtInBufBuf
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtInBufInPtr
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtInBufOutPtr
 CommIOMAP, 716
 NBCCCommon.h, 932

CommOffsetBtOutBufBuf
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetBtOutBufInPtr
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetBtOutBufOutPtr
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsFlags
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsInBufBuf
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsInBufInPtr
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsInBufOutPtr
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsMode
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsOutBufBuf
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsOutBufInPtr
 CommIOMAP, 717
 NBCCCommon.h, 933

CommOffsetHsOutBufOutPtr
 CommIOMAP, 718

NBCCCommon.h, 934
CommOffsetHsSpeed
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetHsState
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetPFunc
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetPFuncTwo
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbInBufBuf
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbInBufInPtr
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbInBufOutPtr
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbOutBufBuf
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbOutBufInPtr
 CommIOMAP, 718
 NBCCCommon.h, 934
CommOffsetUsbOutBufOutPtr
 CommIOMAP, 719
 NBCCCommon.h, 935
CommOffsetUsbPollBufBuf
 CommIOMAP, 719
 NBCCCommon.h, 935
CommOffsetUsbPollBufInPtr
 CommIOMAP, 719
 NBCCCommon.h, 935
CommOffsetUsbPollBufOutPtr
 CommIOMAP, 719
 NBCCCommon.h, 935
CommOffsetUsbState
 CommIOMAP, 719
 NBCCCommon.h, 935
CommStatusCodesConstants
 BT_CMD_READY, 712
 HS_CMD_READY, 712
LR_COULD_NOT_SAVE, 712
LR_ENTRY_REMOVED, 712
LR_STORE_IS_FULL, 713
LR_SUCCESS, 713
LR_UNKNOWN_ADDR, 713
USB_CMD_READY, 713
Communications specific errors, 579
ComputeCalibValue
 NBCCCommon.h, 935
 SysCallConstants, 556
ComputeCalibValueType, 817
 Name, 817
 RawVal, 817
 Result, 818
Connection
 CommBTCheckStatusType, 801
 CommBTWriteType, 804
ConnectionSlot
 CommBTConnectionType, 802
Copy
 cstringAPI, 532
 NXCDefs.h, 1224
Cos
 cmathAPI, 482
 NXCDefs.h, 1164
cos
 cmathAPI, 496
 NXCDefs.h, 1225
CosD
 cmathAPI, 483
 NXCDefs.h, 1164
cosd
 cmathAPI, 496
 NXCDefs.h, 1225
Cosh
 cmathAPI, 483
 NXCDefs.h, 1165
cosh
 cmathAPI, 497
 NXCDefs.h, 1226
CoshD
 cmathAPI, 484
 NXCDefs.h, 1165
coshd
 cmathAPI, 497
 NXCDefs.h, 1226

Count
 IOMapReadByIDType, [851](#)
 IOMapReadType, [853](#)
 ReadButtonType, [868](#)

CreateFile
 LoaderModuleFunctions, [455](#)
 NXCDefs.h, [1227](#)

CreateFileLinear
 LoaderModuleFunctions, [456](#)
 NXCDefs.h, [1227](#)

CreateFileNonLinear
 LoaderModuleFunctions, [457](#)
 NXCDefs.h, [1228](#)

cstdio API, [507](#)

cstdioAPI
 fclose, [510](#)
 feof, [510](#)
 fflush, [510](#)
 fgetc, [511](#)
 fgets, [511](#)
 fopen, [512](#)
 fprintf, [512](#)
 fputc, [513](#)
 fputs, [513](#)
 fseek, [514](#)
 ftell, [514](#)
 getc, [509](#)
 getchar, [515](#)
 printf, [515](#)
 putc, [509](#)
 remove, [515](#)
 rename, [516](#)
 rewind, [516](#)
 sprintf, [516](#)

cstdlib API, [518](#)

cstdlib API types, [527](#)

cstdlibAPI
 abort, [519](#)
 abs, [519](#)
 atof, [520](#)
 atoi, [521](#)
 atol, [521](#)
 div, [522](#)
 labs, [522](#)
 ldiv, [523](#)
 rand, [523](#)

Random, [524](#)
strtod, [524](#)
strtol, [525](#)
strtoul, [526](#)
SysRandomNumber, [526](#)

cstring API, [527](#)

cstringAPI
 addressOf, [530](#)
 addressOfEx, [530](#)
 ByteArrayToStr, [531](#)
 ByteArrayToStrEx, [531](#)
 Copy, [532](#)
 Flatten, [532](#)
 FlattenVar, [533](#)
 FormatNum, [533](#)
 LeftStr, [534](#)
 memcmp, [534](#)
 memcpy, [535](#)
 memmove, [535](#)
 MidStr, [535](#)
 NumToStr, [536](#)
 reladdressOf, [536](#)
 RightStr, [537](#)
 StrCat, [538](#)
 strcat, [537](#)
 strcmp, [538](#)
 strcpy, [539](#)
 StrIndex, [539](#)
 StrLen, [540](#)
 strlen, [540](#)
 strncat, [540](#)
 strncmp, [541](#)
 strncpy, [541](#)
 StrReplace, [542](#)
 StrToByteArray, [542](#)
 StrToNum, [543](#)
 SubStr, [543](#)
 UnflattenVar, [544](#)

ctype API, [545](#)

ctypeAPI
 isalnum, [546](#)
 isalpha, [546](#)
 iscntrl, [547](#)
 isdigit, [547](#)
 isgraph, [547](#)
 islower, [548](#)

isprint, 548
ispunct, 549
isspace, 549
isupper, 549
isxdigit, 550
tolower, 550
toupper, 551

CurrentTick
 CommandModuleFunctions, 339
 NXCDefs.h, 1228

CustomSensorActiveStatus
 InputModuleFunctions, 223
 NXCDefs.h, 1229

CustomSensorPercentFullScale
 InputModuleFunctions, 223
 NXCDefs.h, 1229

CustomSensorZeroOffset
 InputModuleFunctions, 224
 NXCDefs.h, 1229

Data
 DrawGraphicArrayType, 829

Data type limits, 791

DataAvailable
 CommHSCheckStatusType, 808

DatalogGetTimes
 NBCCCommon.h, 935
 SysCallConstants, 556

DatalogGetTimesType, 818
 SyncTick, 819
 SyncTime, 819

DatalogWrite
 NBCCCommon.h, 935
 SysCallConstants, 556

DatalogWriteType, 819
 Message, 820
 Result, 820

DEGREES_PER_RADIAN
 MiscConstants, 192
 NBCCCommon.h, 935

DeleteFile
 LoaderModuleFunctions, 457
 NXCDefs.h, 1230

Device status constants, 708

Display contrast constants, 688
Display flags, 687

Display module, 57
Display module constants, 674
Display module functions, 270
Display module IOMAP offsets, 690
Display module types, 269
DISPLAY_BUSY
 DisplayFlagsGroup, 687
 NBCCCommon.h, 935

DISPLAY_CHAR
 DisplayExecuteFunctionConstants, 680
 NBCCCommon.h, 936

DISPLAY_CONTRAST_DEFAULT
 DisplayContrastConstants, 688
 NBCCCommon.h, 936

DISPLAY_CONTRAST_MAX
 DisplayContrastConstants, 688
 NBCCCommon.h, 936

DISPLAY_ERASE_ALL
 DisplayExecuteFunctionConstants, 680
 NBCCCommon.h, 936

DISPLAY_ERASE_LINE
 DisplayExecuteFunctionConstants, 681
 NBCCCommon.h, 936

DISPLAY_FILL_REGION
 DisplayExecuteFunctionConstants, 681
 NBCCCommon.h, 936

DISPLAY_FRAME
 DisplayExecuteFunctionConstants, 681
 NBCCCommon.h, 936

DISPLAY_HEIGHT
 DisplayModuleConstants, 676
 NBCCCommon.h, 937

DISPLAY_HORIZONTAL_LINE
 DisplayExecuteFunctionConstants, 681
 NBCCCommon.h, 937

DISPLAY_MENUICONS_X_DIFF
 DisplayModuleConstants, 676
 NBCCCommon.h, 937

DISPLAY_MENUICONS_X_OFFSETS
 DisplayModuleConstants, 677

NBCCCommon.h, 937
DISPLAY_MENUICONS_Y
 DisplayModuleConstants, 677
 NBCCCommon.h, 937
DISPLAY_ON
 DisplayFlagsGroup, 687
 NBCCCommon.h, 937
DISPLAY_PIXEL
 DisplayExecuteFunctionConstants,
 681
 NBCCCommon.h, 937
DISPLAY_POPUP
 DisplayFlagsGroup, 687
 NBCCCommon.h, 937
DISPLAY_REFRESH
 DisplayFlagsGroup, 688
 NBCCCommon.h, 938
DISPLAY_REFRESH_DISABLED
 DisplayFlagsGroup, 688
 NBCCCommon.h, 938
DISPLAY_VERTICAL_LINE
 DisplayExecuteFunctionConstants,
 681
 NBCCCommon.h, 938
DISPLAY_WIDTH
 DisplayModuleConstants, 677
 NBCCCommon.h, 938
DisplayContrast
 DisplayModuleFunctions, 275
 NXCDefs.h, 1230
DisplayContrastConstants
 DISPLAY_CONTRAST_-
 DEFAULT, 688
 DISPLAY_CONTRAST_MAX, 688
DisplayDisplay
 DisplayModuleFunctions, 275
 NXCDefs.h, 1231
DisplayDrawOptionConstants
 DRAW_OPT_CLEAR, 683
 DRAW_OPT_CLEAR_EXCEPT_-
 STATUS_SCREEN, 683
 DRAW_OPT_CLEAR_PIXELS,
 683
 DRAW_OPT_CLEAR_SCREEN_-
 MODES, 683
DRAW_OPT_CLEAR_WHOLE_-
 SCREEN, 683
DRAW_OPT_FILL_SHAPE, 683
DRAW_OPT_INVERT, 683
DRAW_OPT_LOGICAL_AND,
 684
DRAW_OPT_LOGICAL_COPY,
 684
DRAW_OPT_LOGICAL_-
 OPERATIONS, 684
DRAW_OPT_LOGICAL_OR, 684
DRAW_OPT_LOGICAL_XOR,
 684
DRAW_OPT_NORMAL, 684
DRAW_OPT_POLYGON_-
 POLYLINE, 685
DisplayEraseMask
 DisplayModuleFunctions, 276
 NXCDefs.h, 1231
DisplayExecuteFunction
 NBCCCommon.h, 938
 SysCallConstants, 556
DisplayExecuteFunction constants, 680
DisplayExecuteFunctionConstants
 DISPLAY_CHAR, 680
 DISPLAY_ERASE_ALL, 680
 DISPLAY_ERASE_LINE, 681
 DISPLAY_FILL_REGION, 681
 DISPLAY_FRAME, 681
 DISPLAY_HORIZONTAL_LINE,
 681
 DISPLAY_PIXEL, 681
 DISPLAY_VERTICAL_LINE, 681
DisplayExecuteFunctionType, 820
 Cmd, 821
 On, 821
 Status, 822
 X1, 822
 X2, 822
 Y1, 822
 Y2, 822
DisplayFlags
 DisplayModuleFunctions, 276
 NXCDefs.h, 1231
DisplayFlagsGroup
 DISPLAY_BUSY, 687

DISPLAY_ON, [687](#)
DISPLAY_POPUP, [687](#)
DISPLAY_REFRESH, [688](#)
DISPLAY_REFRESH_DISABLED, [688](#)
DisplayFont
 DisplayModuleFunctions, [276](#)
 NXCDefs.h, [1232](#)
DisplayFontDrawOptionConstants
 DRAW_OPT_FONT_DIR_B2TL, [685](#)
 DRAW_OPT_FONT_DIR_B2TR, [685](#)
 DRAW_OPT_FONT_DIR_L2RB, [686](#)
 DRAW_OPT_FONT_DIR_L2RT, [686](#)
 DRAW_OPT_FONT_DIR_R2LB, [686](#)
 DRAW_OPT_FONT_DIR_R2LT, [686](#)
 DRAW_OPT_FONT_DIR_T2BL, [686](#)
 DRAW_OPT_FONT_DIR_T2BR, [686](#)
 DRAW_OPT_FONT_- DIRECTIONS, [686](#)
 DRAW_OPT_FONT_WRAP, [687](#)
DisplayIOMAP
 DisplayOffsetContrast, [691](#)
 DisplayOffsetDisplay, [691](#)
 DisplayOffsetEraseMask, [691](#)
 DisplayOffsetFlags, [691](#)
 DisplayOffsetNormal, [692](#)
 DisplayOffsetPBitmaps, [692](#)
 DisplayOffsetPFont, [692](#)
 DisplayOffsetPFunc, [692](#)
 DisplayOffsetPMenuIcons, [692](#)
 DisplayOffsetPMenuText, [692](#)
 DisplayOffsetPopUp, [692](#)
 DisplayOffsetPScreens, [692](#)
 DisplayOffsetPStatusIcons, [692](#)
 DisplayOffsetPStatusText, [692](#)
 DisplayOffsetPStepIcons, [693](#)
 DisplayOffsetPTextLines, [693](#)
 DisplayOffsetStatusIcons, [693](#)
DisplayOffsetStepIcons, [693](#)
DisplayOffsetTextLinesCenterFlags, [693](#)
DisplayOffsetUpdateMask, [693](#)
DisplayModuleConstants
 BITMAP_1, [676](#)
 BITMAP_2, [676](#)
 BITMAP_3, [676](#)
 BITMAP_4, [676](#)
 BITMAPS, [676](#)
 DISPLAY_HEIGHT, [676](#)
 DISPLAY_MENUICONS_X_DIFF, [676](#)
 DISPLAY_MENUICONS_X_- OFFS, [677](#)
 DISPLAY_MENUICONS_Y, [677](#)
 DISPLAY_WIDTH, [677](#)
 FRAME_SELECT, [677](#)
 MENUICON_CENTER, [677](#)
 MENUICON_LEFT, [677](#)
 MENUICON_RIGHT, [677](#)
 MENUICONS, [677](#)
 MENUTEXT, [677](#)
 SCREEN_BACKGROUND, [678](#)
 SCREEN_LARGE, [678](#)
 SCREEN_MODE_CLEAR, [678](#)
 SCREEN_MODE_RESTORE, [678](#)
 SCREEN_SMALL, [678](#)
 SCREENS, [678](#)
 SPECIALS, [678](#)
 STATUSICON_BATTERY, [678](#)
 STATUSICON_BLUETOOTH, [679](#)
 STATUSICON_USB, [679](#)
 STATUSICON_VM, [679](#)
 STATUSICONS, [679](#)
 STATUSTEXT, [679](#)
 STEPICON_1, [679](#)
 STEPICON_2, [679](#)
 STEPICON_3, [679](#)
 STEPICON_4, [679](#)
 STEPICON_5, [679](#)
 STEPICONS, [680](#)
 STEPLINE, [680](#)
 TOPLINE, [680](#)
DisplayModuleFunctions
 CircleOut, [274](#)

ClearLine, 275
ClearScreen, 275
DisplayContrast, 275
DisplayDisplay, 275
DisplayEraseMask, 276
DisplayFlags, 276
DisplayFont, 276
DisplayTextLinesCenterFlags, 277
DisplayUpdateMask, 277
EllipseOut, 277
FontNumOut, 278
FontTextOut, 279
GetDisplayNormal, 280
GetDisplayPopup, 280
GraphicArrayOut, 281
GraphicArrayOutEx, 281
GraphicOut, 282
GraphicOutEx, 283
LineOut, 283
NumOut, 284
PointOut, 285
PolyOut, 285
RectOut, 286
ResetScreen, 287
SetDisplayContrast, 287
SetDisplayDisplay, 287
SetDisplayEraseMask, 288
SetDisplayFlags, 288
SetDisplayFont, 288
SetDisplayNormal, 289
SetDisplayPopup, 289
SetDisplayTextLinesCenterFlags,
 290
SetDisplayUpdateMask, 290
SysDisplayExecuteFunction, 290
SysDrawCircle, 291
SysDrawEllipse, 291
SysDrawFont, 292
SysDrawGraphic, 292
SysDrawGraphicArray, 292
SysDrawLine, 293
SysDrawPoint, 293
SysDrawPolygon, 293
SysDrawRect, 294
SysDrawText, 294
SysSetScreenMode, 294
TextOut, 295
DisplayModuleID
 ModuleIDConstants, 190
 NBCCCommon.h, 938
DisplayModuleName
 ModuleNameConstants, 188
 NBCCCommon.h, 938
DisplayOffsetContrast
 DisplayIOMAP, 691
 NBCCCommon.h, 938
DisplayOffsetDisplay
 DisplayIOMAP, 691
 NBCCCommon.h, 939
DisplayOffsetEraseMask
 DisplayIOMAP, 691
 NBCCCommon.h, 939
DisplayOffsetFlags
 DisplayIOMAP, 691
 NBCCCommon.h, 939
DisplayOffsetNormal
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPBitmaps
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPFont
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPFunc
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPMenuIcons
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPMenuText
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPopup
 DisplayIOMAP, 692
 NBCCCommon.h, 939
DisplayOffsetPScreens
 DisplayIOMAP, 692
 NBCCCommon.h, 940
DisplayOffsetPStatusIcons
 DisplayIOMAP, 692
 NBCCCommon.h, 940

DisplayOffsetPStatusText
 DisplayIOMAP, 692
 NBCCCommon.h, 940

DisplayOffsetPStepIcons
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayOffsetPTextLines
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayOffsetStatusIcons
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayOffsetStepIcons
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayOffsetTextLinesCenterFlags
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayOffsetUpdateMask
 DisplayIOMAP, 693
 NBCCCommon.h, 940

DisplayTextLineConstants
 TEXTLINE_1, 689
 TEXTLINE_2, 689
 TEXTLINE_3, 690
 TEXTLINE_4, 690
 TEXTLINE_5, 690
 TEXTLINE_6, 690
 TEXTLINE_7, 690
 TEXTLINE_8, 690
 TEXTLINES, 690

DisplayTextLinesCenterFlags
 DisplayModuleFunctions, 277
 NXCDefs.h, 1232

DisplayUpdateMask
 DisplayModuleFunctions, 277
 NXCDefs.h, 1232

DIST_CMD_CUSTOM
 MSDistNX, 780
 NBCCCommon.h, 940

DIST_CMD_GP2D12
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_CMD_GP2D120
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_CMD_GP2YA02
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_CMD_GP2YA21
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_REG_DIST
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_REG_DIST1
 MSDistNX, 780
 NBCCCommon.h, 941

DIST_REG_DIST_MAX
 MSDistNX, 781
 NBCCCommon.h, 941

DIST_REG_DIST_MIN
 MSDistNX, 781
 NBCCCommon.h, 941

DIST_REG_MODULE_TYPE
 MSDistNX, 781
 NBCCCommon.h, 941

DIST_REG_NUM_POINTS
 MSDistNX, 781
 NBCCCommon.h, 941

DIST_REG_VOLT
 MSDistNX, 781
 NBCCCommon.h, 942

DIST_REG_VOLT1
 MSDistNX, 781
 NBCCCommon.h, 942

DISTNxDistance
 MindSensorsAPI, 125
 NXCDefs.h, 1233

DISTNxGP2D12
 MindSensorsAPI, 125
 NXCDefs.h, 1233

DISTNxGP2D120
 MindSensorsAPI, 125
 NXCDefs.h, 1233

DISTNxGP2YA02
 MindSensorsAPI, 126
 NXCDefs.h, 1234

DISTNxGP2YA21
 MindSensorsAPI, 126
 NXCDefs.h, 1234

DISTNxMaxDistance

MindSensorsAPI, 127
NXCDefs.h, 1235

DISTNxMinDistance
MindSensorsAPI, 127
NXCDefs.h, 1235

DISTNxModuleType
MindSensorsAPI, 128
NXCDefs.h, 1236

DISTNxNumPoints
MindSensorsAPI, 128
NXCDefs.h, 1236

DISTNxVoltage
MindSensorsAPI, 129
NXCDefs.h, 1237

div
cstdlibAPI, 522
NXCDefs.h, 1237

div_t, 823
quot, 823
rem, 823

DRAW_OPT_CLEAR
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_CLEAR_EXCEPT_-
STATUS_SCREEN
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_CLEAR_PIXELS
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_CLEAR_SCREEN_-
MODES
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_CLEAR_WHOLE_-
SCREEN
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_FILL_SHAPE
DisplayDrawOptionConstants, 683
NBCCCommon.h, 942

DRAW_OPT_FONT_DIR_B2TL
DisplayFontDrawOptionConstants,
685
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_B2TR
DisplayFontDrawOptionConstants,
685
NBCCCommon.h, 943

DisplayFontDrawOptionConstants,
685
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_L2RB
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_L2RT
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_R2LB
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_R2LT
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_T2BL
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 943

DRAW_OPT_FONT_DIR_T2BR
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 944

DRAW_OPT_FONT_DIRECTIONS
DisplayFontDrawOptionConstants,
686
NBCCCommon.h, 944

DRAW_OPT_FONT_WRAP
DisplayFontDrawOptionConstants,
687
NBCCCommon.h, 944

DRAW_OPT_INVERT
DisplayDrawOptionConstants, 683
NBCCCommon.h, 944

DRAW_OPT_LOGICAL_AND
DisplayDrawOptionConstants, 684
NBCCCommon.h, 944

DRAW_OPT_LOGICAL_COPY
DisplayDrawOptionConstants, 684
NBCCCommon.h, 944

DRAW_OPT_LOGICAL_-
OPERATIONS

DisplayDrawOptionConstants, 684
NBCCCommon.h, 945

DRAW_OPT_LOGICAL_OR
DisplayDrawOptionConstants, 684
NBCCCommon.h, 945

DRAW_OPT_LOGICAL_XOR
DisplayDrawOptionConstants, 684
NBCCCommon.h, 945

DRAW_OPT_NORMAL
DisplayDrawOptionConstants, 684
NBCCCommon.h, 945

DRAW_OPT_POLYGON_POLYLINE
DisplayDrawOptionConstants, 685
NBCCCommon.h, 945

DrawCircle
NBCCCommon.h, 945
SysCallConstants, 556

DrawCircleType, 824
Center, 824
Options, 824
Result, 825
Size, 825

DrawEllipse
NBCCCommon.h, 945
SysCallConstants, 556

DrawEllipseType, 825
Center, 826
Options, 826
Result, 826
SizeX, 826
SizeY, 826

DrawFont
NBCCCommon.h, 946
SysCallConstants, 556

DrawFontType, 827
Filename, 827
Location, 827
Options, 828
Result, 828
Text, 828

DrawGraphic
NBCCCommon.h, 946
SysCallConstants, 556

DrawGraphicArray
NBCCCommon.h, 946
SysCallConstants, 557

DrawGraphicArrayType, 828
Data, 829
Location, 829
Options, 829
Result, 829
Variables, 829

DrawGraphicType, 830
Filename, 830
Location, 831
Options, 831
Result, 831
Variables, 831

Drawing option constants, 681

DrawLine
NBCCCommon.h, 946
SysCallConstants, 557

DrawLineType, 831
EndLoc, 832
Options, 832
Result, 832
StartLoc, 832

DrawPoint
NBCCCommon.h, 946
SysCallConstants, 557

DrawPointType, 833
Location, 833
Options, 834
Result, 834

DrawPolygon
NBCCCommon.h, 946
SysCallConstants, 557

DrawPolygonType, 834
Options, 835
Points, 835
Result, 835

DrawRect
NBCCCommon.h, 946
SysCallConstants, 557

DrawRectType, 835
Location, 836
Options, 836
Result, 836
Size, 836

DrawText
NBCCCommon.h, 946
SysCallConstants, 557

DrawTextType, 837
 Location, 837
 Options, 837
 Result, 838
 Text, 838
Duration
 SoundPlayToneType, 875
 Tone, 878

E-Meter sensor constants, 673
EllipseOut
 DisplayModuleFunctions, 277
 NXCDefs.h, 1238
EMETER_REG_AIN
 EMeterI2CConstants, 673
 NBCCCommon.h, 947
EMETER_REG_AOUT
 EMeterI2CConstants, 673
 NBCCCommon.h, 947
EMETER_REG_JOULES
 EMeterI2CConstants, 673
 NBCCCommon.h, 947
EMETER_REG_VIN
 EMeterI2CConstants, 674
 NBCCCommon.h, 947
EMETER_REG_VOUT
 EMeterI2CConstants, 674
 NBCCCommon.h, 947
EMETER_REG_WIN
 EMeterI2CConstants, 674
 NBCCCommon.h, 947
EMETER_REG_WOUT
 EMeterI2CConstants, 674
 NBCCCommon.h, 947
EMeterI2CConstants
 EMETER_REG_AIN, 673
 EMETER_REG_AOUT, 673
 EMETER_REG_JOULES, 673
 EMETER_REG_VIN, 674
 EMETER_REG_VOUT, 674
 EMETER_REG_WIN, 674
 EMETER_REG_WOUT, 674
EndLoc
 DrawLineType, 832
EOF
 LoaderModuleConstants, 586

NBCCCommon.h, 947
ERR_ARG
 CommandFatalErrors, 576
 NBCCCommon.h, 947
ERR_BAD_POOL_SIZE
 CommandFatalErrors, 576
 NBCCCommon.h, 947
ERR_BAD_PTR
 CommandFatalErrors, 576
 NBCCCommon.h, 948
ERR_CLUMP_COUNT
 CommandFatalErrors, 576
 NBCCCommon.h, 948
ERR_COMM_BUFFER_FULL
 CommandCommErrors, 579
 NBCCCommon.h, 948
ERR_COMM_BUS_ERR
 CommandCommErrors, 579
 NBCCCommon.h, 948
ERR_COMM_CHAN_INVALID
 CommandCommErrors, 579
 NBCCCommon.h, 948
ERR_COMM_CHAN_NOT_READY
 CommandCommErrors, 579
 NBCCCommon.h, 948
ERR_DEFAULT_OFFSETS
 CommandFatalErrors, 576
 NBCCCommon.h, 948
ERR_FILE
 CommandFatalErrors, 576
 NBCCCommon.h, 948
ERR_INSANE_OFFSET
 CommandFatalErrors, 576
 NBCCCommon.h, 948
ERR_INSTR
 CommandFatalErrors, 577
 NBCCCommon.h, 948
ERR_INVALID_FIELD
 CommandGenErrors, 578
 NBCCCommon.h, 949
ERR_INVALID_PORT
 CommandGenErrors, 578
 NBCCCommon.h, 949
ERR_INVALID_QUEUE
 CommandGenErrors, 578
 NBCCCommon.h, 949

ERR_INVALID_SIZE
 CommandGenErrors, 578
 NBCCommon.h, 949

ERR_LOADER_ERR
 CommandFatalErrors, 577
 NBCCommon.h, 949

ERR_MEM
 CommandFatalErrors, 577
 NBCCommon.h, 949

ERR_MEMMGR_FAIL
 CommandFatalErrors, 577
 NBCCommon.h, 949

ERR_NO_ACTIVE_CLUMP
 CommandFatalErrors, 577
 NBCCommon.h, 949

ERR_NO_CODE
 CommandFatalErrors, 577
 NBCCommon.h, 949

ERR_NO_PROG
 CommandGenErrors, 578
 NBCCommon.h, 949

ERR_NON_FATAL
 CommandFatalErrors, 577
 NBCCommon.h, 950

ERR_RC_BAD_PACKET
 CommandRCErrors, 580
 NBCCommon.h, 950

ERR_RC_FAILED
 CommandRCErrors, 580
 NBCCommon.h, 950

ERR_RC_ILLEGAL_VAL
 CommandRCErrors, 580
 NBCCommon.h, 950

ERR_RC_UNKNOWN_CMD
 CommandRCErrors, 580
 NBCCommon.h, 950

ERR_SPOTCHECK_FAIL
 CommandFatalErrors, 577
 NBCCommon.h, 950

ERR_VER
 CommandFatalErrors, 577
 NBCCommon.h, 950

ExitTo
 CommandModuleFunctions, 340
 NXCDefs.h, 1238

Exp
 cmathAPI, 484
 NXCDefs.h, 1166

exp
 cmathAPI, 498
 NXCDefs.h, 1239

FALSE
 MiscConstants, 192
 NBCCommon.h, 950

Fatal errors, 575

fclose
 cstdioAPI, 510
 NXCDefs.h, 1239

feof
 cstdioAPI, 510
 NXCDefs.h, 1240

fflush
 cstdioAPI, 510
 NXCDefs.h, 1240

fgetc
 cstdioAPI, 511
 NXCDefs.h, 1240

fgets
 cstdioAPI, 511
 NXCDefs.h, 1241

FileClose
 NBCCommon.h, 950
 SysCallConstants, 557

FileCloseType, 838
 FileHandle, 839
 Result, 839

FileDelete
 NBCCommon.h, 950
 SysCallConstants, 557

FileDeleteType, 839
 Filename, 840
 Result, 840

FileFindFirst
 NBCCommon.h, 951
 SysCallConstants, 558

FileFindNext
 NBCCommon.h, 951
 SysCallConstants, 558

FileFindType, 840
 FileHandle, 841
 Filename, 841

Length, 841
Result, 841

FileHandle
 FileCloseType, 839
 FileFindType, 841
 FileOpenType, 842
 FileReadWriteType, 844
 FileResizeType, 846
 FileResolveHandleType, 847
 FileSeekType, 849

FileList
 ListFilesType, 859

Filename
 DrawFontType, 827
 DrawGraphicType, 830
 FileDeleteType, 840
 FileFindType, 841
 FileOpenType, 842
 FileResolveHandleType, 847
 LoaderExecuteFunctionType, 862
 SoundPlayFileType, 874

FileOpenAppend
 NBCCCommon.h, 951
 SysCallConstants, 558

FileOpenRead
 NBCCCommon.h, 951
 SysCallConstants, 558

FileOpenReadLinear
 NBCCCommon.h, 951
 SysCallConstants, 558

FileOpenType, 841
 FileHandle, 842
 Filename, 842
 Length, 842
 Result, 842

FileOpenWrite
 NBCCCommon.h, 951
 SysCallConstants, 558

FileOpenWriteLinear
 NBCCCommon.h, 951
 SysCallConstants, 558

FileOpenWriteNonLinear
 NBCCCommon.h, 951
 SysCallConstants, 558

FileRead
 NBCCCommon.h, 951

SysCallConstants, 558

FileReadWriteType, 844

FileRename
 NBCCCommon.h, 951
 SysCallConstants, 558

FileRenameType, 844
 NewFilename, 845
 OldFilename, 845
 Result, 845

FileResize
 NBCCCommon.h, 952
 SysCallConstants, 559

FileResizeType, 846
 FileHandle, 846
 NewSize, 846
 Result, 846

FileResolveHandle
 NBCCCommon.h, 952
 SysCallConstants, 559

FileResolveHandleType, 847
 FileHandle, 847
 Filename, 847
 Result, 848
 WriteHandle, 848

FileSeek
 NBCCCommon.h, 952
 SysCallConstants, 559

FileSeekType, 848
 FileHandle, 849
 Length, 849
 Origin, 849
 Result, 849

FileWrite
 NBCCCommon.h, 952
 SysCallConstants, 559

FindFirstFile
 LoaderModuleFunctions, 458
 NXCDefs.h, 1241

FindNextFile
 LoaderModuleFunctions, 458
 NXCDefs.h, 1242

FirstTick

CommandModuleFunctions, 340
NXCDefs.h, 1242

Flags
SoundGetStateType, 873
SoundSetStateType, 877

Flatten
cstringAPI, 532
NXCDefs.h, 1243

FlattenVar
cstringAPI, 533
NXCDefs.h, 1243

Float
NXCDefs.h, 1244
OutputModuleFunctions, 246

Floor
cmathAPI, 484
NXCDefs.h, 1166

floor
cmathAPI, 498
NXCDefs.h, 1244

Follows
CommandModuleFunctions, 340
NXCDefs.h, 1244

Font drawing option constants, 685

FontNumOut
DisplayModuleFunctions, 278
NXCDefs.h, 1245

FontTextOut
DisplayModuleFunctions, 279
NXCDefs.h, 1246

fopen
cstdioAPI, 512
NXCDefs.h, 1246

ForceOff
NXCDefs.h, 1247
UiModuleFunctions, 442

FormatNum
cstringAPI, 533
NXCDefs.h, 1247

fprintf
cstdioAPI, 512
NXCDefs.h, 1248

fputc
cstdioAPI, 513
NXCDefs.h, 1248

fputs

cstdioAPI, 513
NXCDefs.h, 1249

Frac
cmathAPI, 485
NXCDefs.h, 1166

frac
cmathAPI, 499
NXCDefs.h, 1249

FRAME_SELECT
DisplayModuleConstants, 677
NBCCCommon.h, 952

FreeMemory
LoaderModuleFunctions, 459
NXCDefs.h, 1250

Frequency
SoundPlayToneType, 876
Tone, 878

FREQUENCY_MAX
NBCCCommon.h, 952
SoundMisc, 601

FREQUENCY_MIN
NBCCCommon.h, 952
SoundMisc, 601

fseek
cstdioAPI, 514
NXCDefs.h, 1250

fseek origin constants, 517

fseekConstants
SEEK_CUR, 517
SEEK_END, 517
SEEK_SET, 518

ftell
cstdioAPI, 514
NXCDefs.h, 1251

General errors, 578

GenericI2CConstants
I2C_ADDR_DEFAULT, 666
I2C_REG_CMD, 667
I2C_REG_DEVICE_ID, 667
I2C_REG_VENDOR_ID, 667
I2C_REG_VERSION, 667

GetBrickDataAddress
CommModuleFunctions, 391
NXCDefs.h, 1251

GetBTConnectionAddress

CommModuleFunctions, 391
NXCDefs.h, 1251

GetBTDeviceAddress
CommModuleFunctions, 391
NXCDefs.h, 1252

GetBTInputBuffer
CommModuleFunctions, 392
NXCDefs.h, 1252

GetBTOutputBuffer
CommModuleFunctions, 392
NXCDefs.h, 1253

GetButtonModuleValue
CommandModuleFunctions, 341
NXCDefs.h, 1253

getc
cstdioAPI, 509
NXCDefs.h, 1167

getchar
cstdioAPI, 515
NXCDefs.h, 1253

GetCommandModuleBytes
CommandModuleFunctions, 341
NXCDefs.h, 1254

GetCommandModuleValue
CommandModuleFunctions, 342
NXCDefs.h, 1254

GetCommModuleBytes
CommandModuleFunctions, 342
NXCDefs.h, 1255

GetCommModuleValue
CommandModuleFunctions, 342
NXCDefs.h, 1255

GetDisplayModuleBytes
CommandModuleFunctions, 343
NXCDefs.h, 1255

GetDisplayModuleValue
CommandModuleFunctions, 343
NXCDefs.h, 1256

GetDisplayNormal
DisplayModuleFunctions, 280
NXCDefs.h, 1256

GetDisplayPopup
DisplayModuleFunctions, 280
NXCDefs.h, 1257

GetHSInputBuffer
CommModuleFunctions, 393

NXCDefs.h, 1257

GetHSOutputBuffer
CommModuleFunctions, 393
NXCDefs.h, 1258

GetInput
InputModuleFunctions, 224
NXCDefs.h, 1258

GetInputModuleValue
CommandModuleFunctions, 344
NXCDefs.h, 1258

GetIOMapBytes
CommandModuleFunctions, 344
NXCDefs.h, 1259

GetIOMapBytesByID
CommandModuleFunctions, 344
NXCDefs.h, 1259

GetIOMapValue
CommandModuleFunctions, 345
NXCDefs.h, 1260

GetIOMapValueByID
CommandModuleFunctions, 345
NXCDefs.h, 1260

GetLoaderModuleValue
CommandModuleFunctions, 346
NXCDefs.h, 1261

GetLowSpeedModuleBytes
CommandModuleFunctions, 346
NXCDefs.h, 1261

GetLowSpeedModuleValue
CommandModuleFunctions, 347
NXCDefs.h, 1261

GetLSInputBuffer
LowLevelLowSpeedModuleFunctions, 325
NXCDefs.h, 1262

GetLSSOutputBuffer
LowLevelLowSpeedModuleFunctions, 326
NXCDefs.h, 1262

GetOutput
NXCDefs.h, 1263

OutputModuleFunctions, 247

GetOutputModuleValue
CommandModuleFunctions, 347
NXCDefs.h, 1263

GetSoundModuleValue

CommandModuleFunctions, 347
 NXCDefs.h, 1263

GetStartTick
 NBCCCommon.h, 952
 SysCallConstants, 559

GetStartTickType, 850
 Result, 850

GetUIModuleValue
 CommandModuleFunctions, 348
 NXCDefs.h, 1264

GetUSBInputBuffer
 CommModuleFunctions, 394
 NXCDefs.h, 1264

GetUSBOutputBuffer
 CommModuleFunctions, 394
 NXCDefs.h, 1265

GetUSBPollBuffer
 CommModuleFunctions, 394
 NXCDefs.h, 1265

GL_CAMERA_DEPTH
 GLConstantsSettings, 797
 NBCCCommon.h, 952

GL_CIRCLE
 GLConstantsBeginModes, 794
 NBCCCommon.h, 952

GL_CIRCLE_SIZE
 GLConstantsSettings, 797
 NBCCCommon.h, 953

GL_CULL_BACK
 GLConstantsCullMode, 798
 NBCCCommon.h, 953

GL_CULL_FRONT
 GLConstantsCullMode, 798
 NBCCCommon.h, 953

GL_CULL_MODE
 GLConstantsSettings, 797
 NBCCCommon.h, 953

GL_CULL_NONE
 GLConstantsCullMode, 798
 NBCCCommon.h, 953

GL_LINE
 GLConstantsBeginModes, 794
 NBCCCommon.h, 953

GL_POINT
 GLConstantsBeginModes, 794
 NBCCCommon.h, 953

 GL_POLYGON
 GLConstantsBeginModes, 794
 NBCCCommon.h, 954

 GL_ROTATE_X
 GLConstantsActions, 795
 NBCCCommon.h, 954

 GL_ROTATE_Y
 GLConstantsActions, 795
 NBCCCommon.h, 954

 GL_ROTATE_Z
 GLConstantsActions, 795
 NBCCCommon.h, 954

 GL_SCALE_X
 GLConstantsActions, 795
 NBCCCommon.h, 954

 GL_SCALE_Y
 GLConstantsActions, 796
 NBCCCommon.h, 954

 GL_SCALE_Z
 GLConstantsActions, 796
 NBCCCommon.h, 955

 GL_TRANSLATE_X
 GLConstantsActions, 796
 NBCCCommon.h, 955

 GL_TRANSLATE_Y
 GLConstantsActions, 796
 NBCCCommon.h, 955

 GL_TRANSLATE_Z
 GLConstantsActions, 796
 NBCCCommon.h, 955

 GL_ZOOM_FACTOR
 GLConstantsSettings, 797
 NBCCCommon.h, 955

glAddToAngleX
 GraphicsLibrary, 196
 NXCDefs.h, 1265

glAddToAngleY
 GraphicsLibrary, 197
 NXCDefs.h, 1266

glAddToAngleZ
 GraphicsLibrary, 197
 NXCDefs.h, 1266

glAddVertex
 GraphicsLibrary, 197
 NXCDefs.h, 1266

glBegin

GraphicsLibrary, 197
 NXCDefs.h, 1267

glBeginObject
 GraphicsLibrary, 198
 NXCDefs.h, 1267

glBeginRender
 GraphicsLibrary, 198
 NXCDefs.h, 1267

glBox
 GraphicsLibrary, 198
 NXCDefs.h, 1267

glCallObject
 GraphicsLibrary, 199
 NXCDefs.h, 1268

GLConstantsActions
 GL_ROTATE_X, 795
 GL_ROTATE_Y, 795
 GL_ROTATE_Z, 795
 GL_SCALE_X, 795
 GL_SCALE_Y, 796
 GL_SCALE_Z, 796
 GL_TRANSLATE_X, 796
 GL_TRANSLATE_Y, 796
 GL_TRANSLATE_Z, 796

GLConstantsBeginModes
 GL_CIRCLE, 794
 GL_LINE, 794
 GL_POINT, 794
 GL_POLYGON, 794

GLConstantsCullMode
 GL_CULL_BACK, 798
 GL_CULL_FRONT, 798
 GL_CULL_NONE, 798

GLConstantsSettings
 GL_CAMERA_DEPTH, 797
 GL_CIRCLE_SIZE, 797
 GL_CULL_MODE, 797
 GL_ZOOM_FACTOR, 797

glCos32768
 GraphicsLibrary, 199
 NXCDefs.h, 1268

glCube
 GraphicsLibrary, 199
 NXCDefs.h, 1268

glEnd
 GraphicsLibrary, 200

 NXCDefs.h, 1269

glEndObject
 GraphicsLibrary, 200
 NXCDefs.h, 1269

glFinishRender
 GraphicsLibrary, 200
 NXCDefs.h, 1269

glInit
 GraphicsLibrary, 200
 NXCDefs.h, 1269

glObjectAction
 GraphicsLibrary, 201
 NXCDefs.h, 1270

glPyramid
 GraphicsLibrary, 201
 NXCDefs.h, 1270

glSet
 GraphicsLibrary, 201
 NXCDefs.h, 1270

glSetAngleX
 GraphicsLibrary, 202
 NXCDefs.h, 1271

glSetAngleY
 GraphicsLibrary, 202
 NXCDefs.h, 1271

glSetAngleZ
 GraphicsLibrary, 202
 NXCDefs.h, 1271

glSin32768
 GraphicsLibrary, 203
 NXCDefs.h, 1272

GraphicArrayOut
 DisplayModuleFunctions, 281
 NXCDefs.h, 1272

GraphicArrayOutEx
 DisplayModuleFunctions, 281
 NXCDefs.h, 1272

GraphicOut
 DisplayModuleFunctions, 282
 NXCDefs.h, 1273

GraphicOutEx
 DisplayModuleFunctions, 283
 NXCDefs.h, 1274

Graphics library actions, 794

Graphics library begin modes, 793

Graphics library cull mode, 797

Graphics library settings, 797
GraphicsLibrary
 glAddToAngleX, 196
 glAddToAngleY, 197
 glAddToAngleZ, 197
 glAddVertex, 197
 glBegin, 197
 glBeginObject, 198
 glBeginRender, 198
 glBox, 198
 glCallObject, 199
 glCos32768, 199
 glCube, 199
 glEnd, 200
 glEndObject, 200
 glFinishRender, 200
 glInit, 200
 glObjectAction, 201
 glPyramid, 201
 glSet, 201
 glSetAngleX, 202
 glSetAngleY, 202
 glSetAngleZ, 202
 glSin32768, 203

Height
 SizeType, 872

Hi-speed port baud rate constants, 702
Hi-speed port combined UART constants, 707
Hi-speed port constants, 699
Hi-speed port data bits constants, 705
Hi-speed port flags constants, 700
Hi-speed port parity constants, 706
Hi-speed port state constants, 700
Hi-speed port stop bits constants, 706
Hi-speed port SysCommHSControl constants, 701
HiTechnic API Functions, 58
HiTechnic Color2 constants, 774
HiTechnic device constants, 768
HiTechnic IRReceiver constants, 772
HiTechnic IRSeeker2 constants, 770
HiTechnic/mindsensors Power Function/IR Train constants, 756
HiTechnicAPI

HTIRTrain, 67
HTPFComboDirect, 67
HTPFComboPWM, 68
HTPFRawOutput, 68
HTPFRepeat, 69
HTPFSingleOutputCST, 69
HTPFSingleOutputPWM, 70
HTPFSinglePin, 71
HTPFTrain, 71
HTRCXAddToDatalog, 72
HTRCXBatteryLevel, 72
HTRCXClearAllEvents, 73
HTRCXClearCounter, 73
HTRCXClearMsg, 73
HTRCXClearSensor, 73
HTRCXClearSound, 74
HTRCXClearTimer, 74
HTRCXCreateDatalog, 74
HTRCXDecCounter, 75
HTRCXDeleteSub, 75
HTRCXDeleteSubs, 75
HTRCXDeleteTask, 75
HTRCXDeleteTasks, 76
HTRCXDisableOutput, 76
HTRCXEnableOutput, 76
HTRCXEvent, 77
HTRCXFloat, 77
HTRCXFwd, 77
HTRCXIncCounter, 77
HTRCXInvertOutput, 78
HTRCXMuteSound, 78
HTRCXObvertOutput, 78
HTRCXOff, 79
HTRCXOn, 79
HTRCXOnFor, 79
HTRCXOnFwd, 79
HTRCXOnRev, 80
HTRCXPBTurnOff, 80
HTRCXPing, 80
HTRCXPlaySound, 81
HTRCXPlayTone, 81
HTRCXPlayToneVar, 81
HTRCXPoll, 82
HTRCXPollMemory, 82
HTRCXRemote, 82
HTRCXRev, 83

HTRCXSelectDisplay, 83
HTRCXSelectProgram, 83
HTRCXSendSerial, 84
HTRCXSetDirection, 84
HTRCXSetEvent, 84
HTRCXSetGlobalDirection, 85
HTRCXSetGlobalOutput, 85
HTRCXSetIRLinkPort, 86
HTRCXSetMaxPower, 86
HTRCXSetMessage, 86
HTRCXSetOutput, 86
HTRCXSetPower, 87
HTRCXSetPriority, 87
HTRCXSetSensorMode, 87
HTRCXSetSensorType, 88
HTRCXSetSleepTime, 88
HTRCXSetTxPower, 88
HTRCXSetWatch, 89
HTRCXStartTask, 89
HTRCXStopAllTasks, 89
HTRCXStopTask, 90
HTRCXToggle, 90
HTRCXUnmuteSound, 90
HTScoutCalibrateSensor, 90
HTScoutMuteSound, 91
HTScoutSelectSounds, 91
HTScoutSendVLL, 91
HTScoutSetEventFeedback, 91
HTScoutSetLight, 92
HTScoutSetScoutMode, 92
HTScoutSetSensorClickTime, 92
HTScoutSetSensorHysteresis, 93
HTScoutSetSensorLowerLimit, 93
HTScoutSetSensorUpperLimit, 94
HTScoutUnmuteSound, 94
ReadSensorHTAccel, 94
ReadSensorHTColor, 95
ReadSensorHTColor2Active, 95
ReadSensorHTIRReceiver, 96
ReadSensorHTIRReceiverEx, 96
ReadSensorHTIRSeeker, 97
ReadSensorHTIRSeeker2AC, 97
ReadSensorHTIRSeeker2DC, 98
ReadSensorHTNormalizedColor, 99
ReadSensorHTNormalized-
Color2Active, 99
ReadSensorHTRawColor, 100
ReadSensorHTRawColor2, 101
ReadSensorHTTouchMultiplexer,
 101
SensorHTColorNum, 102
SensorHTCompass, 102
SensorHTEOPD, 102
SensorHTGyro, 103
SensorHTIRSeeker2ACDir, 103
SensorHTIRSeeker2Addr, 104
SensorHTIRSeeker2DCDir, 104
SensorHTIRSeekerDir, 105
SetHTColor2Mode, 105
SetHTIRSeeker2Mode, 105
SetSensorHTEOPD, 106
SetSensorHTGyro, 106
HiTechnicConstants
 HT_ADDR_ACCEL, 769
 HT_ADDR_COLOR, 769
 HT_ADDR_COLOR2, 769
 HT_ADDR_COMPASS, 769
 HT_ADDR_IRLINK, 769
 HT_ADDR_IRRECEIVER, 769
 HT_ADDR_IRSEEKER, 769
 HT_ADDR_IRSEEKER2, 769
HS_BAUD_115200
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 955
HS_BAUD_1200
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 955
HS_BAUD_14400
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 956
HS_BAUD_19200
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 956
HS_BAUD_230400
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 956
HS_BAUD_2400
 CommHiSpeedBaudConstants, 703
 NBCCommon.h, 956
HS_BAUD_28800
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_3600
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_38400
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_460800
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_4800
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_57600
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 956

HS_BAUD_7200
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 957

HS_BAUD_76800
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 957

HS_BAUD_921600
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 957

HS_BAUD_9600
 CommHiSpeedBaudConstants, 704
 NBCCommon.h, 957

HS_CMD_READY
 CommStatusCodesConstants, 712
 NBCCommon.h, 957

HS_CTRL_EXIT
 CommHiSpeedCtrlConstants, 702
 NBCCommon.h, 957

HS_CTRL_INIT
 CommHiSpeedCtrlConstants, 702
 NBCCommon.h, 957

HS_CTRL_UART
 CommHiSpeedCtrlConstants, 702
 NBCCommon.h, 957

HS_DISABLE
 CommHiSpeedStateConstants, 701
 NBCCommon.h, 958

HS_ENABLE
 CommHiSpeedStateConstants, 701
 NBCCommon.h, 958

HS_INIT_RECEIVER

 CommHiSpeedStateConstants, 701
 NBCCommon.h, 958

 NBCCommon.h, 958

HS_INITIALISE
 CommHiSpeedStateConstants, 701
 NBCCommon.h, 958

HS_MODE_10_STOP
 CommHiSpeedStopBitsConstants,
 706
 NBCCommon.h, 958

HS_MODE_15_STOP
 CommHiSpeedStopBitsConstants,
 706
 NBCCommon.h, 958

HS_MODE_20_STOP
 CommHiSpeedStopBitsConstants,
 706
 NBCCommon.h, 958

HS_MODE_5_DATA
 CommHiSpeedDataBitsConstants,
 705
 NBCCommon.h, 958

HS_MODE_6_DATA
 CommHiSpeedDataBitsConstants,
 705
 NBCCommon.h, 958

HS_MODE_7_DATA
 CommHiSpeedDataBitsConstants,
 705
 NBCCommon.h, 959

HS_MODE_7E1
 CommHiSpeedCombinedConstants,
 708
 NBCCommon.h, 959

HS_MODE_8_DATA
 CommHiSpeedDataBitsConstants,
 705
 NBCCommon.h, 959

HS_MODE_8N1
 CommHiSpeedCombinedConstants,
 708
 NBCCommon.h, 959

HS_MODE_E_PARITY
 CommHiSpeedParityConstants, 707
 NBCCommon.h, 959

HS_MODE_M_PARITY
 CommHiSpeedParityConstants, 707

NBCCCommon.h, 959
HS_MODE_N_PARITY
 CommHiSpeedParityConstants, 707
 NBCCCommon.h, 959
HS_MODE_O_PARITY
 CommHiSpeedParityConstants, 707
 NBCCCommon.h, 959
HS_MODE_S_PARITY
 CommHiSpeedParityConstants, 707
 NBCCCommon.h, 959
HS_SEND_DATA
 CommHiSpeedStateConstants, 701
 NBCCCommon.h, 960
HS_UPDATE
 CommHiSpeedFlagsConstants, 700
 NBCCCommon.h, 960
HSFlags
 CommModuleFunctions, 395
 NXCDefs.h, 1274
HSInputBufferInPtr
 CommModuleFunctions, 395
 NXCDefs.h, 1275
HSInputBufferOutPtr
 CommModuleFunctions, 395
 NXCDefs.h, 1275
HSMode
 CommModuleFunctions, 396
 NXCDefs.h, 1275
HSOutputBufferInPtr
 CommModuleFunctions, 396
 NXCDefs.h, 1276
HSOutputBufferOutPtr
 CommModuleFunctions, 396
 NXCDefs.h, 1276
HSSpeed
 CommModuleFunctions, 397
 NXCDefs.h, 1276
HSState
 CommModuleFunctions, 397
 NXCDefs.h, 1277
HT_ADDR_ACCEL
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_COLOR
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_COLOR2
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_COMPASS
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_IRLINK
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_IRRECEIVER
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_IRSEEKER
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_ADDR_IRSEEKER2
 HiTechnicConstants, 769
 NBCCCommon.h, 960
HT_CH1_A
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH1_B
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH2_A
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH2_B
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH3_A
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH3_B
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH4_A
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CH4_B
 HTIRReceiverConstants, 773
 NBCCCommon.h, 961
HT_CMD_COLOR2_50HZ
 HTColor2Constants, 774
 NBCCCommon.h, 961
HT_CMD_COLOR2_60HZ

HTColor2Constants, 774
NBCCCommon.h, 962

HT_CMD_COLOR2_ACTIVE
HTColor2Constants, 774
NBCCCommon.h, 962

HT_CMD_COLOR2_BLCAL
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_FAR
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_LED_HI
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_LED_LOW
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_NEAR
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_PASSIVE
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_RAW
HTColor2Constants, 775
NBCCCommon.h, 962

HT_CMD_COLOR2_WBCAL
HTColor2Constants, 775
NBCCCommon.h, 963

HTColor2Constants
HT_CMD_COLOR2_50HZ, 774
HT_CMD_COLOR2_60HZ, 774
HT_CMD_COLOR2_ACTIVE, 774
HT_CMD_COLOR2_BLCAL, 775
HT_CMD_COLOR2_FAR, 775
HT_CMD_COLOR2_LED_HI, 775
HT_CMD_COLOR2_LED_LOW,
 775
HT_CMD_COLOR2_NEAR, 775
HT_CMD_COLOR2_PASSIVE,
 775
HT_CMD_COLOR2_RAW, 775
HT_CMD_COLOR2_WBCAL, 775

HTIR2_MODE_1200
HTIRSeeker2Constants, 770
NBCCCommon.h, 963

HTIR2_MODE_600
HTIRSeeker2Constants, 770
NBCCCommon.h, 963

HTIR2_REG_AC01
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_AC02
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_AC03
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_AC04
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_AC05
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_ACDIR
HTIRSeeker2Constants, 771
NBCCCommon.h, 963

HTIR2_REG_DC01
HTIRSeeker2Constants, 771
NBCCCommon.h, 964

HTIR2_REG_DC02
HTIRSeeker2Constants, 771
NBCCCommon.h, 964

HTIR2_REG_DC03
HTIRSeeker2Constants, 771
NBCCCommon.h, 964

HTIR2_REG_DC04
HTIRSeeker2Constants, 771
NBCCCommon.h, 964

HTIR2_REG_DC05
HTIRSeeker2Constants, 772
NBCCCommon.h, 964

HTIR2_REG_DCAVG
HTIRSeeker2Constants, 772
NBCCCommon.h, 964

HTIR2_REG_DCDIR
HTIRSeeker2Constants, 772
NBCCCommon.h, 964

HTIR2_REG_MODE
HTIRSeeker2Constants, 772
NBCCCommon.h, 964

HTIRReceiverConstants

HT_CH1_A, [773](#)
HT_CH1_B, [773](#)
HT_CH2_A, [773](#)
HT_CH2_B, [773](#)
HT_CH3_A, [773](#)
HT_CH3_B, [773](#)
HT_CH4_A, [773](#)
HT_CH4_B, [773](#)
HTIRSeeker2Constants
 HTIR2_MODE_1200, [770](#)
 HTIR2_MODE_600, [770](#)
 HTIR2_REG_AC01, [771](#)
 HTIR2_REG_AC02, [771](#)
 HTIR2_REG_AC03, [771](#)
 HTIR2_REG_AC04, [771](#)
 HTIR2_REG_AC05, [771](#)
 HTIR2_REG_ACDIR, [771](#)
 HTIR2_REG_DC01, [771](#)
 HTIR2_REG_DC02, [771](#)
 HTIR2_REG_DC03, [771](#)
 HTIR2_REG_DC04, [771](#)
 HTIR2_REG_DC05, [772](#)
 HTIR2_REG_DCAVG, [772](#)
 HTIR2_REG_DCDIR, [772](#)
 HTIR2_REG_MODE, [772](#)
HTIIRTrain
 HiTechnicAPI, [67](#)
 NXCDefs.h, [1277](#)
HTPFCComboDirect
 HiTechnicAPI, [67](#)
 NXCDefs.h, [1278](#)
HTPFCComboPWM
 HiTechnicAPI, [68](#)
 NXCDefs.h, [1278](#)
HTPFRawOutput
 HiTechnicAPI, [68](#)
 NXCDefs.h, [1279](#)
HTPFRRepeat
 HiTechnicAPI, [69](#)
 NXCDefs.h, [1279](#)
HTPFSingleOutputCST
 HiTechnicAPI, [69](#)
 NXCDefs.h, [1280](#)
HTPFSingleOutputPWM
 HiTechnicAPI, [70](#)
 NXCDefs.h, [1281](#)
HTPFSinglePin
 HiTechnicAPI, [71](#)
 NXCDefs.h, [1281](#)
HTPPTrain
 HiTechnicAPI, [71](#)
 NXCDefs.h, [1282](#)
HTRCXAddToDatalog
 HiTechnicAPI, [72](#)
 NXCDefs.h, [1283](#)
HTRCXBatteryLevel
 HiTechnicAPI, [72](#)
 NXCDefs.h, [1283](#)
HTRCXClearAllEvents
 HiTechnicAPI, [73](#)
 NXCDefs.h, [1283](#)
HTRCXClearCounter
 HiTechnicAPI, [73](#)
 NXCDefs.h, [1283](#)
HTRCXClearMsg
 HiTechnicAPI, [73](#)
 NXCDefs.h, [1284](#)
HTRCXClearSensor
 HiTechnicAPI, [73](#)
 NXCDefs.h, [1284](#)
HTRCXClearSound
 HiTechnicAPI, [74](#)
 NXCDefs.h, [1284](#)
HTRCXClearTimer
 HiTechnicAPI, [74](#)
 NXCDefs.h, [1284](#)
HTRCXCreateDatalog
 HiTechnicAPI, [74](#)
 NXCDefs.h, [1285](#)
HTRCXDecCounter
 HiTechnicAPI, [75](#)
 NXCDefs.h, [1285](#)
HTRCXDeleteSub
 HiTechnicAPI, [75](#)
 NXCDefs.h, [1285](#)
HTRCXDeleteSubs
 HiTechnicAPI, [75](#)
 NXCDefs.h, [1286](#)
HTRCXDeleteTask
 HiTechnicAPI, [75](#)
 NXCDefs.h, [1286](#)
HTRCXDeleteTasks

HiTechnicAPI, 76
NXCDefs.h, 1286

HTRCXDisableOutput
 HiTechnicAPI, 76
 NXCDefs.h, 1286

HTRCXEnableOutput
 HiTechnicAPI, 76
 NXCDefs.h, 1287

HTRCXEvent
 HiTechnicAPI, 77
 NXCDefs.h, 1287

HTRCXFloat
 HiTechnicAPI, 77
 NXCDefs.h, 1287

HTRCXFwd
 HiTechnicAPI, 77
 NXCDefs.h, 1288

HTRCXIIncCounter
 HiTechnicAPI, 77
 NXCDefs.h, 1288

HTRCXIInvertOutput
 HiTechnicAPI, 78
 NXCDefs.h, 1288

HTRCXMuteSound
 HiTechnicAPI, 78
 NXCDefs.h, 1289

HTRCXObvertOutput
 HiTechnicAPI, 78
 NXCDefs.h, 1289

HTRCXOff
 HiTechnicAPI, 79
 NXCDefs.h, 1289

HTRCXOn
 HiTechnicAPI, 79
 NXCDefs.h, 1290

HTRCXOnFor
 HiTechnicAPI, 79
 NXCDefs.h, 1290

HTRCXOnFwd
 HiTechnicAPI, 79
 NXCDefs.h, 1290

HTRCXOnRev
 HiTechnicAPI, 80
 NXCDefs.h, 1291

HTRCXPBTurnOff
 HiTechnicAPI, 80

NXCDefs.h, 1291

HTRCXPing
 HiTechnicAPI, 80
 NXCDefs.h, 1291

HTRCXPlaySound
 HiTechnicAPI, 81
 NXCDefs.h, 1291

HTRCXPlayTone
 HiTechnicAPI, 81
 NXCDefs.h, 1292

HTRCXPlayToneVar
 HiTechnicAPI, 81
 NXCDefs.h, 1292

HTRCXPoll
 HiTechnicAPI, 82
 NXCDefs.h, 1292

HTRCXPollMemory
 HiTechnicAPI, 82
 NXCDefs.h, 1293

HTRCXRemote
 HiTechnicAPI, 82
 NXCDefs.h, 1293

HTRCXRev
 HiTechnicAPI, 83
 NXCDefs.h, 1293

HTRCXSelectDisplay
 HiTechnicAPI, 83
 NXCDefs.h, 1294

HTRCXSelectProgram
 HiTechnicAPI, 83
 NXCDefs.h, 1294

HTRCXSendSerial
 HiTechnicAPI, 84
 NXCDefs.h, 1294

HTRCXSetDirection
 HiTechnicAPI, 84
 NXCDefs.h, 1295

HTRCXSetEvent
 HiTechnicAPI, 84
 NXCDefs.h, 1295

HTRCXSetGlobalDirection
 HiTechnicAPI, 85
 NXCDefs.h, 1296

HTRCXSetGlobalOutput
 HiTechnicAPI, 85
 NXCDefs.h, 1296

HTRCXSetIRLinkPort
 HiTechnicAPI, 86
 NXCDefs.h, 1296

HTRCXSetMaxPower
 HiTechnicAPI, 86
 NXCDefs.h, 1297

HTRCXSetMessage
 HiTechnicAPI, 86
 NXCDefs.h, 1297

HTRCXSetOutput
 HiTechnicAPI, 86
 NXCDefs.h, 1297

HTRCXSetPower
 HiTechnicAPI, 87
 NXCDefs.h, 1298

HTRCXSetPriority
 HiTechnicAPI, 87
 NXCDefs.h, 1298

HTRCXSetSensorMode
 HiTechnicAPI, 87
 NXCDefs.h, 1298

HTRCXSetSensorType
 HiTechnicAPI, 88
 NXCDefs.h, 1299

HTRCXSetSleepTime
 HiTechnicAPI, 88
 NXCDefs.h, 1299

HTRCXSetTxPower
 HiTechnicAPI, 88
 NXCDefs.h, 1299

HTRCXSetWatch
 HiTechnicAPI, 89
 NXCDefs.h, 1300

HTRCXStartTask
 HiTechnicAPI, 89
 NXCDefs.h, 1300

HTRCXStopAllTasks
 HiTechnicAPI, 89
 NXCDefs.h, 1300

HTRCXStopTask
 HiTechnicAPI, 90
 NXCDefs.h, 1300

HTRCXToggle
 HiTechnicAPI, 90
 NXCDefs.h, 1301

HTRCXUnmuteSound
 HiTechnicAPI, 90
 NXCDefs.h, 1301

HTScoutCalibrateSensor
 HiTechnicAPI, 90
 NXCDefs.h, 1301

HTScoutMuteSound
 HiTechnicAPI, 91
 NXCDefs.h, 1301

HTScoutSelectSounds
 HiTechnicAPI, 91
 NXCDefs.h, 1302

HTScoutSendVLL
 HiTechnicAPI, 91
 NXCDefs.h, 1302

HTScoutSetEventFeedback
 HiTechnicAPI, 91
 NXCDefs.h, 1302

HTScoutSetLight
 HiTechnicAPI, 92
 NXCDefs.h, 1303

HTScoutSetScoutMode
 HiTechnicAPI, 92
 NXCDefs.h, 1303

HTScoutSetSensorClickTime
 HiTechnicAPI, 92
 NXCDefs.h, 1303

HTScoutSetSensorHysteresis
 HiTechnicAPI, 93
 NXCDefs.h, 1304

HTScoutSetSensorLowerLimit
 HiTechnicAPI, 93
 NXCDefs.h, 1304

HTScoutSetSensorUpperLimit
 HiTechnicAPI, 94
 NXCDefs.h, 1304

HTScoutUnmuteSound
 HiTechnicAPI, 94
 NXCDefs.h, 1305

I2C_ADDR_DEFAULT
 GenericI2CConstants, 666
 NBCCCommon.h, 964

I2C_REG_CMD
 GenericI2CConstants, 667
 NBCCCommon.h, 965

I2C_REG_DEVICE_ID

GenericI2CConstants, 667
 NBCCCommon.h, 965

I2C_REG_VENDOR_ID
 GenericI2CConstants, 667
 NBCCCommon.h, 965

I2C_REG_VERSION
 GenericI2CConstants, 667
 NBCCCommon.h, 965

I2CBytes
 LowSpeedModuleFunctions, 311
 NXCDefs.h, 1305

I2CBytesReady
 LowSpeedModuleFunctions, 312
 NXCDefs.h, 1306

I2CCheckStatus
 LowSpeedModuleFunctions, 312
 NXCDefs.h, 1307

I2DDeviceId
 LowSpeedModuleFunctions, 313
 NXCDefs.h, 1307

I2CDeviceInfo
 LowSpeedModuleFunctions, 314
 NXCDefs.h, 1308

I2CRead
 LowSpeedModuleFunctions, 314
 NXCDefs.h, 1308

I2CSendCommand
 LowSpeedModuleFunctions, 315
 NXCDefs.h, 1309

I2CStatus
 LowSpeedModuleFunctions, 315
 NXCDefs.h, 1310

I2CVendorId
 LowSpeedModuleFunctions, 316
 NXCDefs.h, 1310

I2CVersion
 LowSpeedModuleFunctions, 317
 NXCDefs.h, 1311

I2CWrite
 LowSpeedModuleFunctions, 317
 NXCDefs.h, 1311

IN_1
 NBCCCommon.h, 965
 NBCInputPortConstants, 625

IN_2
 NBCCCommon.h, 966

 NBCInputPortConstants, 625

IN_3
 NBCCCommon.h, 966
 NBCInputPortConstants, 626

IN_4
 NBCCCommon.h, 966
 NBCInputPortConstants, 626

IN_MODE_ANGLESTEP
 NBCCCommon.h, 966
 NBCSensorModeConstants, 629

IN_MODE_BOOLEAN
 NBCCCommon.h, 966
 NBCSensorModeConstants, 629

IN_MODE_CELSIUS
 NBCCCommon.h, 966
 NBCSensorModeConstants, 630

IN_MODE_FAHRENHEIT
 NBCCCommon.h, 966
 NBCSensorModeConstants, 630

IN_MODE_MODEMASK
 NBCCCommon.h, 966
 NBCSensorModeConstants, 630

IN_MODE_PCTFULLSCALE
 NBCCCommon.h, 966
 NBCSensorModeConstants, 630

IN_MODE_PERIODCOUNTER
 NBCCCommon.h, 966
 NBCSensorModeConstants, 630

IN_MODE_RAW
 NBCCCommon.h, 967
 NBCSensorModeConstants, 630

IN_MODE_SLOPEMASK
 NBCCCommon.h, 967
 NBCSensorModeConstants, 630

IN_MODE_TRANSITIONCNT
 NBCCCommon.h, 967
 NBCSensorModeConstants, 630

IN_TYPE_ANGLE
 NBCCCommon.h, 967
 NBCSensorTypeConstants, 627

IN_TYPE_COLORBLUE
 NBCCCommon.h, 967
 NBCSensorTypeConstants, 627

IN_TYPE_COLOREXIT
 NBCCCommon.h, 967
 NBCSensorTypeConstants, 627

IN_TYPE_COLORFULL
 NBCCCommon.h, [967](#)
 NBCSensorTypeConstants, [627](#)

IN_TYPE_COLORGREEN
 NBCCCommon.h, [967](#)
 NBCSensorTypeConstants, [627](#)

IN_TYPE_COLORNONE
 NBCCCommon.h, [967](#)
 NBCSensorTypeConstants, [627](#)

IN_TYPE_COLORRED
 NBCCCommon.h, [967](#)
 NBCSensorTypeConstants, [627](#)

IN_TYPE_CUSTOM
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [627](#)

IN_TYPE_HISPEED
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_LIGHT_ACTIVE
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_LIGHT_INACTIVE
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_LOWSPEED
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_LOWSPEED_9V
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_NO_SENSOR
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_REFLECTION
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_SOUND_DB
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_SOUND_DBA
 NBCCCommon.h, [968](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_SWITCH
 NBCCCommon.h, [969](#)
 NBCSensorTypeConstants, [628](#)

IN_TYPE_TEMPERATURE

NBCCCommon.h, [969](#)
NBCSensorTypeConstants, [629](#)

Index
 ReadButtonType, [868](#)

InPorts
 S1, [204](#)
 S2, [206](#)
 S3, [206](#)
 S4, [206](#)

Input field constants, [630](#)

Input module, [44](#)

Input module constants, [45](#)

Input module functions, [215](#)

Input module IOMAP offsets, [636](#)

Input module types, [215](#)

Input port constants, [204](#)

INPUT_BLACKCOLOR
 InputColorValueConstants, [633](#)
 NBCCCommon.h, [969](#)

INPUT_BLANK
 InputColorIdxConstants, [632](#)
 NBCCCommon.h, [969](#)

INPUT_BLUE
 InputColorIdxConstants, [632](#)
 NBCCCommon.h, [969](#)

INPUT_BLUECOLOR
 InputColorValueConstants, [633](#)
 NBCCCommon.h, [969](#)

INPUT_CAL_POINT_0
 InputColorCalibrationConstants,
 [635](#)
 NBCCCommon.h, [969](#)

INPUT_CAL_POINT_1
 InputColorCalibrationConstants,
 [636](#)
 NBCCCommon.h, [969](#)

INPUT_CAL_POINT_2
 InputColorCalibrationConstants,
 [636](#)
 NBCCCommon.h, [969](#)

INPUT_CUSTOM9V
 InputModuleConstants, [46](#)
 NBCCCommon.h, [970](#)

INPUT_CUSTOMACTIVE
 InputModuleConstants, [46](#)
 NBCCCommon.h, [970](#)

INPUT_CUSTOMINACTIVE
 InputModuleConstants, 46
 NBCCCommon.h, 970

INPUT_DIGI0
 InputModuleConstants, 46
 NBCCCommon.h, 970

INPUT_DIGI1
 InputModuleConstants, 46
 NBCCCommon.h, 970

INPUT_GREEN
 InputColorIdxConstants, 632
 NBCCCommon.h, 970

INPUT_GREENCOLOR
 InputColorValueConstants, 633
 NBCCCommon.h, 970

INPUT_INVALID_DATA
 InputModuleConstants, 46
 NBCCCommon.h, 970

INPUT_NO_OF_COLORS
 InputColorIdxConstants, 632
 NBCCCommon.h, 970

INPUT_NO_OF_POINTS
 InputColorCalibrationConstants,
 636
 NBCCCommon.h, 970

INPUT_RED
 InputColorIdxConstants, 633
 NBCCCommon.h, 971

INPUT_REDCOLOR
 InputColorValueConstants, 634
 NBCCCommon.h, 971

INPUT_RESETCAL
 InputColorCalibrationStateCon-
 stants, 634
 NBCCCommon.h, 971

INPUT_RUNNINGCAL
 InputColorCalibrationStateCon-
 stants, 634
 NBCCCommon.h, 971

INPUT_SENSORCAL
 InputColorCalibrationStateCon-
 stants, 635
 NBCCCommon.h, 971

INPUT_SENSOROFF
 InputColorCalibrationStateCon-
 stants, 635

 NBCCCommon.h, 971

 NBCCCommon.h, 971

INPUT_STARTCAL
 InputColorCalibrationStateCon-
 stants, 635
 NBCCCommon.h, 971

INPUT_WHITECOLOR
 InputColorValueConstants, 634
 NBCCCommon.h, 971

INPUT_YELLOWCOLOR
 InputColorValueConstants, 634
 NBCCCommon.h, 971

InputColorCalibrationConstants
 INPUT_CAL_POINT_0, 635
 INPUT_CAL_POINT_1, 636
 INPUT_CAL_POINT_2, 636
 INPUT_NO_OF_POINTS, 636

InputColorCalibrationStateConstants
 INPUT_RESETCAL, 634
 INPUT_RUNNINGCAL, 634
 INPUT_SENSORCAL, 635
 INPUT_SENSOROFF, 635
 INPUT_STARTCAL, 635

InputColorIdxConstants
 INPUT_BLANK, 632
 INPUT_BLUE, 632
 INPUT_GREEN, 632
 INPUT_NO_OF_COLORS, 632
 INPUT_RED, 633

InputColorValueConstants
 INPUT_BLACKCOLOR, 633
 INPUT_BLUECOLOR, 633
 INPUT_GREENCOLOR, 633
 INPUT_REDCOLOR, 634
 INPUT_WHITECOLOR, 634
 INPUT_YELLOWCOLOR, 634

InputFieldConstants
 InputMode, 631
 InvalidData, 631
 NormalizedValue, 631
 RawValue, 631
 ScaledValue, 631
 Type, 631

InputIOMAP
 InputOffsetADRaw, 637
 InputOffsetColorADRaw, 637
 InputOffsetColorCalibration, 637

InputOffsetColorCalibrationState, 637
InputOffsetColorCalLimits, 637
InputOffsetColorSensorBoolean, 637
InputOffsetColorSensorRaw, 637
InputOffsetColorSensorValue, 637
InputOffsetCustomActiveStatus, 637
InputOffsetCustomPctFullScale, 638
InputOffsetCustomZeroOffset, 638
InputOffsetDigiPinsDir, 638
InputOffsetDigiPinsIn, 638
InputOffsetDigiPinsOut, 638
InputOffsetInvalidData, 638
InputOffsetSensorBoolean, 638
InputOffsetSensorMode, 638
InputOffsetSensorRaw, 638
InputOffsetSensorType, 638
InputOffsetSensorValue, 639
InputMode
 InputFieldConstants, 631
 NBCCommon.h, 972
InputModuleConstants
 INPUT_CUSTOM9V, 46
 INPUT_CUSTOMACTIVE, 46
 INPUT_CUSTOMINACTIVE, 46
 INPUT_DIGI0, 46
 INPUT_DIGI1, 46
 INPUT_INVALID_DATA, 46
InputModuleFunctions
 ClearSensor, 219
 ColorADRaw, 219
 ColorBoolean, 220
 ColorCalibration, 220
 ColorCalibrationState, 221
 ColorCalLimits, 221
 ColorSensorRaw, 222
 ColorSensorValue, 222
 CustomSensorActiveStatus, 223
 CustomSensorPercentFullScale, 223
 CustomSensorZeroOffset, 224
 GetInput, 224
 ReadSensorColorEx, 225
 ReadSensorColorRaw, 225
 ResetSensor, 226
 Sensor, 226
 SensorBoolean, 227
 SensorDigiPinsDirection, 227
 SensorDigiPinsOutputLevel, 227
 SensorDigiPinsStatus, 228
 SensorInvalid, 228
 SensorMode, 229
 SensorNormalized, 229
 SensorRaw, 229
 SensorScaled, 230
 SensorType, 230
 SensorValue, 231
 SensorValueBool, 231
 SensorValueRaw, 232
 SetCustomSensorActiveStatus, 232
 SetCustomSensorPercentFullScale, 232
 SetCustomSensorZeroOffset, 233
 SetInput, 233
 SetSensor, 234
 SetSensorBoolean, 234
 SetSensorColorBlue, 234
 SetSensorColorFull, 235
 SetSensorColorGreen, 235
 SetSensorColorNone, 236
 SetSensorColorRed, 236
 SetSensorDigiPinsDirection, 236
 SetSensorDigiPinsOutputLevel, 237
 SetSensorDigiPinsStatus, 237
 SetSensorLight, 237
 SetSensorLowspeed, 238
 SetSensorMode, 238
 SetSensorSound, 239
 SetSensorTouch, 239
 SetSensorType, 239
 SysColorSensorRead, 240
InputModuleID
 ModuleIDConstants, 190
 NBCCommon.h, 972
InputModuleName
 ModuleNameConstants, 188
 NBCCommon.h, 972
InputOffsetADRaw
 InputIOMAP, 637
 NBCCommon.h, 972
InputOffsetColorADRaw
 InputIOMAP, 637

NBCCCommon.h, 972
InputOffsetColorCalibration
 InputIOMAP, 637
 NBCCCommon.h, 972
InputOffsetColorCalibrationState
 InputIOMAP, 637
 NBCCCommon.h, 972
InputOffsetColorCalLimits
 InputIOMAP, 637
 NBCCCommon.h, 972
InputOffsetColorSensorBoolean
 InputIOMAP, 637
 NBCCCommon.h, 972
InputOffsetColorSensorRaw
 InputIOMAP, 637
 NBCCCommon.h, 973
InputOffsetColorSensorValue
 InputIOMAP, 637
 NBCCCommon.h, 973
InputOffsetCustomActiveStatus
 InputIOMAP, 637
 NBCCCommon.h, 973
InputOffsetCustomPctFullScale
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetCustomZeroOffset
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetDigiPinsDir
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetDigiPinsIn
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetDigiPinsOut
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetInvalidData
 InputIOMAP, 638
 NBCCCommon.h, 973
InputOffsetSensorBoolean
 InputIOMAP, 638
 NBCCCommon.h, 974
InputOffsetSensorMode
 InputIOMAP, 638
 NBCCCommon.h, 974
InputOffsetSensorRaw
 InputIOMAP, 638
 NBCCCommon.h, 974
InputOffsetSensorType
 InputIOMAP, 638
 NBCCCommon.h, 974
InputOffsetSensorValue
 InputIOMAP, 639
 NBCCCommon.h, 974
INT_MAX
 NBCCCommon.h, 974
 NXTLimits, 792
INT_MIN
 NBCCCommon.h, 974
 NXTLimits, 792
INTF_BTOFF
 CommInterfaceConstants, 710
 NBCCCommon.h, 974
INTF_BTON
 CommInterfaceConstants, 710
 NBCCCommon.h, 974
INTF_CONNECT
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_CONNECTBYNAME
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_CONNECTREQ
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_DISCONNECT
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_DISCONNECTALL
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_EXTREAD
 CommInterfaceConstants, 710
 NBCCCommon.h, 975
INTF_FACTORYRESET
 CommInterfaceConstants, 711
 NBCCCommon.h, 975
INTF_OPENSTREAM
 CommInterfaceConstants, 711
 NBCCCommon.h, 975
INTF_PINREQ

CommInterfaceConstants, 711
NBCCCommon.h, 975

INTF_REMOVEDevice
 CommInterfaceConstants, 711
 NBCCCommon.h, 975

INTF_SEARCH
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_SENDDATA
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_SENDFILE
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_SETBTNAME
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_SETCMDMODE
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_STOPSEARCH
 CommInterfaceConstants, 711
 NBCCCommon.h, 976

INTF_VISIBILITY
 CommInterfaceConstants, 712
 NBCCCommon.h, 976

Invalid
 ColorSensorReadType, 799

InvalidData
 InputFieldConstants, 631
 NBCCCommon.h, 976

IOCtrl module, 53

IOCtrl module constants, 584

IOCtrl module functions, 372

IOCtrl module IOMAP offsets, 585

IOCtrl module types, 372

IOCTRL_BOOT
 IOCtrlPO, 584
 NBCCCommon.h, 976

IOCTRL_POWERDOWN
 IOCtrlPO, 584
 NBCCCommon.h, 976

IOCtrlIOMAP
 IOCtrlOffsetPowerOn, 585

IOCtrlModuleFunctions
 PowerDown, 372

 RebootInFirmwareMode, 372
 SleepNow, 373

IOCtrlModuleID
 ModuleIDConstants, 190
 NBCCCommon.h, 977

IOCtrlModuleName
 ModuleNameConstants, 189
 NBCCCommon.h, 977

IOCtrlOffsetPowerOn
 IOCtrlIOMAP, 585
 NBCCCommon.h, 977

IOCtrlPO
 IOCTRL_BOOT, 584
 IOCTRL_POWERDOWN, 584

IOMapRead
 NBCCCommon.h, 977
 SysCallConstants, 559

IOMapReadByID
 NBCCCommon.h, 977
 SysCallConstants, 559

IOMapReadByIDType, 851
 Buffer, 851
 Count, 851
 ModuleID, 851
 Offset, 852
 Result, 852

IOMapReadType, 852
 Buffer, 853
 Count, 853
 ModuleName, 853
 Offset, 853
 Result, 853

IOMapWrite
 NBCCCommon.h, 977
 SysCallConstants, 559

IOMapWriteByID
 NBCCCommon.h, 977
 SysCallConstants, 559

IOMapWriteByIDType, 854
 Buffer, 854
 ModuleID, 855
 Offset, 855
 Result, 855

IOMapWriteType, 855
 Buffer, 856
 ModuleName, 856

Offset, 856
Result, 856

IR Train channel constants, 761

IRTrainChannels

- TRAIN_CHANNEL_1, 761
- TRAIN_CHANNEL_2, 761
- TRAIN_CHANNEL_3, 762
- TRAIN_CHANNEL_ALL, 762

IRTrainFuncs

- TRAIN_FUNC_DECR_SPEED, 760
- TRAIN_FUNC_INCR_SPEED, 760
- TRAIN_FUNC_STOP, 761
- TRAIN_FUNC_TOGGLE_LIGHT, 761

isalnum

- ctypeAPI, 546
- NXCDefs.h, 1312

isalpha

- ctypeAPI, 546
- NXCDefs.h, 1312

iscntrl

- ctypeAPI, 547
- NXCDefs.h, 1313

isdigit

- ctypeAPI, 547
- NXCDefs.h, 1313

isgraph

- ctypeAPI, 547
- NXCDefs.h, 1314

islower

- ctypeAPI, 548
- NXCDefs.h, 1314

isNaN

- cmathAPI, 499
- NXCDefs.h, 1314

isprint

- ctypeAPI, 548
- NXCDefs.h, 1315

ispunct

- ctypeAPI, 549
- NXCDefs.h, 1315

isspace

- ctypeAPI, 549
- NXCDefs.h, 1316

isupper

ctypeAPI, 549
NXCDefs.h, 1316

ctypeAPI, 549
NXCDefs.h, 1316

isxdigit

- ctypeAPI, 550
- NXCDefs.h, 1316

KeepAlive

- NBCCCommon.h, 977
- SysCallConstants, 560

KeepAliveType, 857

Result, 857

labs

- cstdlibAPI, 522
- NXCDefs.h, 1317

LCD_LINE1

- LineConstants, 562
- NBCCCommon.h, 977

LCD_LINE2

- LineConstants, 562
- NBCCCommon.h, 978

LCD_LINE3

- LineConstants, 563
- NBCCCommon.h, 978

LCD_LINE4

- LineConstants, 563
- NBCCCommon.h, 979

LCD_LINE5

- LineConstants, 563
- NBCCCommon.h, 979

LCD_LINE6

- LineConstants, 564
- NBCCCommon.h, 979

LCD_LINE7

- LineConstants, 564
- NBCCCommon.h, 979

LCD_LINE8

- LineConstants, 564
- NBCCCommon.h, 980

ldiv

- cstdlibAPI, 523
- NXCDefs.h, 1317

ldiv_t, 857

- quot, 858
- rem, 858

LDR_APPENDNOTPOSSIBLE

- LoaderErrors, 588
 NBCCCommon.h, 980
- LDR_BTBUSY**
 - LoaderErrors, 588
 NBCCCommon.h, 980
- LDR_BTCONNECTFAIL**
 - LoaderErrors, 588
 NBCCCommon.h, 980
- LDR_BTTIMEOUT**
 - LoaderErrors, 588
 NBCCCommon.h, 980
- LDR_CMD_BOOTCMD**
 - LoaderFunctionConstants, 592
 NBCCCommon.h, 980
- LDR_CMD_BTFACTORYRESET**
 - LoaderFunctionConstants, 592
 NBCCCommon.h, 980
- LDR_CMD_BTGETADR**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 980
- LDR_CMD_CLOSE**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_CLOSEMODHANDLE**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_CROPDATAFILE**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_DELETE**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_DELETEUSERFLASH**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_DEVICEINFO**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_FINDFIRST**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_FINDFIRSTMODULE**
 - LoaderFunctionConstants, 593
 NBCCCommon.h, 981
- LDR_CMD_FINDNEXT**
 - LoaderFunctionConstants, 593
- NBCCCommon.h, 981**
- LDR_CMD_FINDNEXTMODULE**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 981
- LDR_CMD_IOMAPREAD**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_IOMAPWRITE**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_OPENAPPENDDATA**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_OPENREADLINEAR**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_OPENWRITE**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_OPENWRITEDATA**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_OPENWITELINEAR**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_POLLCMD**
 - LoaderFunctionConstants, 594
 NBCCCommon.h, 982
- LDR_CMD_POLLCMDLEN**
 - LoaderFunctionConstants, 595
 NBCCCommon.h, 982
- LDR_CMD_READ**
 - LoaderFunctionConstants, 595
 NBCCCommon.h, 983
- LDR_CMD_RENAMEFILE**
 - LoaderFunctionConstants, 595
 NBCCCommon.h, 983
- LDR_CMD_RESIZEDATAFILE**
 - LoaderFunctionConstants, 595
 NBCCCommon.h, 983
- LDR_CMD_SEEKFROMCURRENT**
 - LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_CMD_SEEKFROMEND
 LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_CMD_SEEKFROMSTART
 LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_CMD_SETBRICKNAME
 LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_CMD_VERSIONS
 LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_CMD_WRITE
 LoaderFunctionConstants, 595
 NBCCCommon.h, 983

LDR_ENDOFFILE
 LoaderErrors, 588
 NBCCCommon.h, 983

LDR_EOFEXPECTED
 LoaderErrors, 588
 NBCCCommon.h, 984

LDR_FILEEXISTS
 LoaderErrors, 588
 NBCCCommon.h, 984

LDR_FILEISBUSY
 LoaderErrors, 589
 NBCCCommon.h, 984

LDR_FILEISFULL
 LoaderErrors, 589
 NBCCCommon.h, 984

LDR_FILENOFOUND
 LoaderErrors, 589
 NBCCCommon.h, 984

LDR_FILETX_CLOSEERROR
 LoaderErrors, 589
 NBCCCommon.h, 984

LDR_FILETX_DSTEXISTS
 LoaderErrors, 589
 NBCCCommon.h, 985

LDR_FILETX_SRCMISSING
 LoaderErrors, 589
 NBCCCommon.h, 985

LDR_FILETX_STREAMERROR
 LoaderErrors, 589
 NBCCCommon.h, 985

LDR_FILETX_TIMEOUT
 LoaderErrors, 589

LoaderErrors, 589
NBCCCommon.h, 985

LDR_HANDLEALREADYCLOSED
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_ILLEGALFILENAME
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_ILLEGALHANDLE
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_INPROGRESS
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_INVALIDSEEK
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_MODULENOTFOUND
 LoaderErrors, 590
 NBCCCommon.h, 985

LDR_NOLINEARSPACE
 LoaderErrors, 590
 NBCCCommon.h, 986

LDR_NOMOREFILES
 LoaderErrors, 590
 NBCCCommon.h, 986

LDR_NOMOREHANDLES
 LoaderErrors, 590
 NBCCCommon.h, 986

LDR_NOSPACE
 LoaderErrors, 590
 NBCCCommon.h, 986

LDR_NOTLINEARFILE
 LoaderErrors, 591
 NBCCCommon.h, 986

LDR_NOWRITEBUFFERS
 LoaderErrors, 591
 NBCCCommon.h, 986

LDR_OUTOFBOUNDARY
 LoaderErrors, 591
 NBCCCommon.h, 986

LDR_REQPIN
 LoaderErrors, 591
 NBCCCommon.h, 986

LDR_SUCCESS
 LoaderErrors, 591

NBCCCommon.h, 986
LDR_UNDEFINEDERROR
 LoaderErrors, 591
 NBCCCommon.h, 987
LeftStr
 cstringAPI, 534
 NXCDefs.h, 1318
LEGO I2C address constants, 667
LEGO temperature sensor constants, 670
LEGO_ADDR_EMETER
 LEGOI2CAddressConstants, 668
 NBCCCommon.h, 987
LEGO_ADDR_TEMP
 LEGOI2CAddressConstants, 668
 NBCCCommon.h, 987
LEGO_ADDR_US
 LEGOI2CAddressConstants, 668
 NBCCCommon.h, 987
LEGOI2CAddressConstants
 LEGO_ADDR_EMETER, 668
 LEGO_ADDR_TEMP, 668
 LEGO_ADDR_US, 668
Length
 FileFindType, 841
 FileOpenType, 842
 FileReadWriteType, 844
 FileSeekType, 849
 LoaderExecuteFunctionType, 862
Line number constants, 561
LineConstants
 LCD_LINE1, 562
 LCD_LINE2, 562
 LCD_LINE3, 563
 LCD_LINE4, 563
 LCD_LINE5, 563
 LCD_LINE6, 564
 LCD_LINE7, 564
 LCD_LINE8, 564
LineOut
 DisplayModuleFunctions, 283
 NXCDefs.h, 1318
ListFiles
 NBCCCommon.h, 987
 SysCallConstants, 560
ListFileType, 858
 FileList, 859
Pattern, 859
Result, 859
Loader module, 54
Loader module constants, 585
Loader module error codes, 587
Loader module function constants, 591
Loader module functions, 452
Loader module IOMAP offsets, 586
Loader module types, 451
LoaderErrors
 LDR_APPENDNOTPOSSIBLE, 588
 LDR_BTBUSY, 588
 LDR_BTCONNECTFAIL, 588
 LDR_BTTIMEOUT, 588
 LDR_ENDOFFILE, 588
 LDR_EOFEXPECTED, 588
 LDR_FILEEXISTS, 588
 LDR_FILEISBUSY, 589
 LDR_FILEISFULL, 589
 LDR_FILENOFOUND, 589
 LDR_FILETX_CLOSEERROR, 589
 LDR_FILETX_DSTEXISTS, 589
 LDR_FILETX_SRCMISSING, 589
 LDR_FILETX_STREAMERROR, 589
 LDR_FILETX_TIMEOUT, 589
LDR_-
 HANDLEALREADYCLOSED, 590
 LDR_ILLEGALFILENAME, 590
 LDR_ILLEGALHANDLE, 590
 LDR_INPROGRESS, 590
 LDR_INVALIDSEEK, 590
 LDR_MODULENOTFOUND, 590
 LDR_NOLINEARSPACE, 590
 LDR_NOMOREFILES, 590
 LDR_NOMOREHANDLES, 590
 LDR_NOSPACE, 590
 LDR_NOTLINEARFILE, 591
 LDR_NOWRITEBUFFERS, 591
 LDR_OUTOFCBOUNDARY, 591
 LDR_REQPIN, 591
 LDR_SUCCESS, 591
 LDR_UNDEFINEDERROR, 591

LoaderExecuteFunction
 NBCCCommon.h, [987](#)
 SysCallConstants, [560](#)

LoaderExecuteFunctionType, [860](#)
 Buffer, [862](#)
 Cmd, [862](#)
 Filename, [862](#)
 Length, [862](#)
 Result, [862](#)

LoaderFunctionConstants
 LDR_CMD_BOOTCMD, [592](#)
 LDR_CMD_BTFACTORYRESET,
 [592](#)
 LDR_CMD_BTGETADR, [593](#)
 LDR_CMD_CLOSE, [593](#)
 LDR_CMD_-
 CLOSEMODHANDLE,
 [593](#)
 LDR_CMD_CROPDATAFILE, [593](#)
 LDR_CMD_DELETE, [593](#)
 LDR_CMD_-
 DELETEUSERFLASH, [593](#)
 LDR_CMD_DEVICEINFO, [593](#)
 LDR_CMD_FINDFIRST, [593](#)
 LDR_CMD_-
 FINDFIRSTMODULE, [593](#)
 LDR_CMD_FINDNEXT, [593](#)
 LDR_CMD_-
 FINDNEXTMODULE, [594](#)
 LDR_CMD_IOMAPREAD, [594](#)
 LDR_CMD_IOMAPWRITE, [594](#)
 LDR_CMD_OPENAPPENDDATA,
 [594](#)
 LDR_CMD_OPENREAD, [594](#)
 LDR_CMD_OPENREADLINEAR,
 [594](#)
 LDR_CMD_OPENWRITE, [594](#)
 LDR_CMD_OPENWRITEDATA,
 [594](#)
 LDR_CMD_-
 OPENWITELINEAR, [594](#)
 LDR_CMD_POLLCMD, [594](#)
 LDR_CMD_POLLCMDLEN, [595](#)
 LDR_CMD_READ, [595](#)
 LDR_CMD_RENAMEFILE, [595](#)

 LDR_CMD_RESIZEDATAFILE,
 [595](#)

 LDR_CMD_-
 SEEKFROMCURRENT,
 [595](#)

 LDR_CMD_SEEKFROMEND, [595](#)

 LDR_CMD_SEEKFROMSTART,
 [595](#)

 LDR_CMD_SETBRICKNAME,
 [595](#)

 LDR_CMD VERSIONS, [595](#)

 LDR_CMD_WRITE, [595](#)

LoaderIOMAP
 LoaderOffsetFreeUserFlash, [586](#)
 LoaderOffsetPFunc, [586](#)

LoaderModuleConstants
 EOF, [586](#)
 NULL, [586](#)

LoaderModuleFunctions
 CloseFile, [455](#)
 CreateFile, [455](#)
 CreateFileLinear, [456](#)
 CreateFileNonLinear, [457](#)
 DeleteFile, [457](#)
 FindFirstFile, [458](#)
 FindNextFile, [458](#)
 FreeMemory, [459](#)
 OpenFileAppend, [459](#)
 OpenFileRead, [460](#)
 OpenFileReadLinear, [460](#)
 Read, [461](#)
 ReadBytes, [461](#)
 ReadLn, [462](#)
 ReadLnString, [462](#)
 RenameFile, [463](#)
 ResizeFile, [463](#)
 ResolveHandle, [464](#)
 SysFileClose, [464](#)
 SysFileDelete, [464](#)
 SysFileFindFirst, [465](#)
 SysFileFindNext, [465](#)
 SysFileOpenAppend, [466](#)
 SysFileOpenRead, [466](#)
 SysFileOpenReadLinear, [466](#)
 SysFileOpenWrite, [467](#)
 SysFileOpenWriteLinear, [467](#)

SysFileOpenWriteNonLinear, [467](#)
SysFileRead, [468](#)
SysFileRename, [468](#)
SysFileResize, [469](#)
SysFileResolveHandle, [469](#)
SysFileSeek, [469](#)
SysFileWrite, [470](#)
SysListFiles, [470](#)
SysLoaderExecuteFunction, [470](#)
Write, [471](#)
WriteBytes, [471](#)
WriteBytesEx, [472](#)
WriteLn, [472](#)
WriteLnString, [473](#)
WriteString, [473](#)
LoaderModuleID
 ModuleIDConstants, [191](#)
 NBCCCommon.h, [987](#)
LoaderModuleName
 ModuleNameConstants, [189](#)
 NBCCCommon.h, [987](#)
LoaderOffsetFreeUserFlash
 LoaderIOMAP, [586](#)
 NBCCCommon.h, [987](#)
LoaderOffsetPFunc
 LoaderIOMAP, [586](#)
 NBCCCommon.h, [988](#)
Location
 DrawFontType, [827](#)
 DrawGraphicArrayType, [829](#)
 DrawGraphicType, [831](#)
 DrawPointType, [833](#)
 DrawRectType, [836](#)
 DrawTextType, [837](#)
LocationType, [862](#)
 X, [863](#)
 Y, [863](#)
Log
 cmathAPI, [485](#)
 NXCDefs.h, [1167](#)
log
 cmathAPI, [500](#)
 NXCDefs.h, [1319](#)
Log10
 cmathAPI, [486](#)
 NXCDefs.h, [1168](#)
log10
 cmathAPI, [500](#)
 NXCDefs.h, [1319](#)
LONG_MAX
 NBCCCommon.h, [988](#)
 NXTLimits, [792](#)
LONG_MIN
 NBCCCommon.h, [988](#)
 NXTLimits, [792](#)
LongAbort
 NXCDefs.h, [1320](#)
 UiModuleFunctions, [442](#)
Loop
 SoundPlayFileType, [874](#)
 SoundPlayToneType, [876](#)
Low level LowSpeed module functions, [324](#)
Low Speed module, [56](#)
Low speed module IOMAP offsets, [662](#)
LowLevelLowSpeedModuleFunctions
 GetLSInputBuffer, [325](#)
 GetLSOutputBuffer, [326](#)
 LSCchannelState, [326](#)
 LSErrorType, [327](#)
 LSInputBufferBytesToRx, [327](#)
 LSInputBufferInPtr, [327](#)
 LSInputBufferOutPtr, [328](#)
 LSMode, [328](#)
 LSNoRestartOnRead, [328](#)
 LSOutputBufferBytesToRx, [329](#)
 LSOutputBufferInPtr, [329](#)
 LSOutputBufferOutPtr, [330](#)
 LSSpeed, [330](#)
 LSSState, [330](#)
LowSpeed module constants, [657](#)
LowSpeed module functions, [309](#)
LowSpeed module system call functions, [331](#)
LowSpeed module types, [308](#)
LOWSPEED_CH_NOT_READY
 LowSpeedErrorTypeConstants, [662](#)
 NBCCCommon.h, [988](#)
LOWSPEED_COMMUNICATING
 LowSpeedChannelStateConstants, [660](#)
 NBCCCommon.h, [988](#)

LOWSPEED_DATA_RECEIVED
 LowSpeedModeConstants, [661](#)
 NBCCommon.h, [988](#)

LOWSPEED_DONE
 LowSpeedChannelStateConstants,
 [660](#)
 NBCCommon.h, [988](#)

LOWSPEED_ERROR
 LowSpeedChannelStateConstants,
 [660](#)
 NBCCommon.h, [988](#)

LOWSPEED_IDLE
 LowSpeedChannelStateConstants,
 [660](#)
 NBCCommon.h, [988](#)

LOWSPEED_INIT
 LowSpeedChannelStateConstants,
 [660](#)
 NBCCommon.h, [989](#)

LOWSPEED_LOAD_BUFFER
 LowSpeedChannelStateConstants,
 [660](#)
 NBCCommon.h, [989](#)

LOWSPEED_NO_ERROR
 LowSpeedErrorTypeConstants, [662](#)
 NBCCommon.h, [989](#)

LOWSPEED RECEIVING
 LowSpeedModeConstants, [661](#)
 NBCCommon.h, [989](#)

LOWSPEED_RX_ERROR
 LowSpeedErrorTypeConstants, [662](#)
 NBCCommon.h, [989](#)

LOWSPEED TRANSMITTING
 LowSpeedModeConstants, [661](#)
 NBCCommon.h, [989](#)

LOWSPEED_TX_ERROR
 LowSpeedErrorTypeConstants, [662](#)
 NBCCommon.h, [989](#)

LowspeedBytesReady
 LowSpeedModuleFunctions, [318](#)
 NXCDefs.h, [1320](#)

LowSpeedChannelStateConstants
 LOWSPEED_COMMUNICATING,
 [660](#)
 LOWSPEED_DONE, [660](#)
 LOWSPEED_ERROR, [660](#)

LOWSPEED_IDLE, [660](#)
LOWSPEED_INIT, [660](#)
LOWSPEED_LOAD_BUFFER, [660](#)

LowspeedCheckStatus
 LowSpeedModuleFunctions, [319](#)
 NXCDefs.h, [1321](#)

LowSpeedErrorTypeConstants
 LOWSPEED_CH_NOT_READY,
 [662](#)
 LOWSPEED_NO_ERROR, [662](#)
 LOWSPEED_RX_ERROR, [662](#)
 LOWSPEED_TX_ERROR, [662](#)

LowSpeedIOMAP
 LowSpeedOffsetChannelState, [663](#)
 LowSpeedOffsetErrorType, [663](#)
 LowSpeedOffsetInBufBuf, [663](#)
 LowSpeedOffsetInBufBytesToRx,
 [663](#)
 LowSpeedOffsetInBufInPtr, [663](#)
 LowSpeedOffsetInBufOutPtr, [663](#)
 LowSpeedOffsetMode, [664](#)
 LowSpeedOffsetNoRestartOnRead,
 [664](#)
 LowSpeedOffsetOutBufBuf, [664](#)
 LowSpeedOffsetOutBufBytesToRx,
 [664](#)
 LowSpeedOffsetOutBufInPtr, [664](#)
 LowSpeedOffsetOutBufOutPtr, [664](#)
 LowSpeedOffsetSpeed, [664](#)
 LowSpeedOffsetState, [664](#)

LowSpeedModeConstants
 LOWSPEED_DATA_RECEIVED,
 [661](#)
 LOWSPEED_RECEIVING, [661](#)
 LOWSPEED_TRANSMITTING,
 [661](#)

LowSpeedModuleFunctions
 I2CBytes, [311](#)
 I2CBytesReady, [312](#)
 I2CCheckStatus, [312](#)
 I2CDeviceId, [313](#)
 I2CDeviceInfo, [314](#)
 I2CRead, [314](#)
 I2CSendCommand, [315](#)
 I2CStatus, [315](#)
 I2CVendorId, [316](#)

I2CVersion, [317](#)
I2CWrite, [317](#)
LowspeedBytesReady, [318](#)
LowspeedCheckStatus, [319](#)
LowspeedRead, [319](#)
LowspeedStatus, [320](#)
LowspeedWrite, [321](#)
ReadI2CRegister, [321](#)
ReadSensorEMeter, [322](#)
ReadSensorUSEx, [323](#)
SensorUS, [323](#)
WriteI2CRegister, [323](#)
LowSpeedModuleID
 ModuleIDConstants, [191](#)
 NBCCCommon.h, [989](#)
LowSpeedModuleName
 ModuleNameConstants, [189](#)
 NBCCCommon.h, [989](#)
LowSpeedModuleSystemCallFunctions
 SysCommLSCheckStatus, [331](#)
 SysCommLSRead, [332](#)
 SysCommLSWrite, [332](#)
 SysCommLSWriteEx, [332](#)
LowSpeedNoRestartConstants
 LSREAD_NO_RESTART_1, [665](#)
 LSREAD_NO_RESTART_2, [665](#)
 LSREAD_NO_RESTART_3, [665](#)
 LSREAD_NO_RESTART_4, [665](#)
 LSREAD_NO_RESTART_MASK,
 [665](#)
 LSREAD_RESTART_ALL, [666](#)
 LSREAD_RESTART_NONE, [666](#)
LowSpeedOffsetChannelState
 LowSpeedIOMAP, [663](#)
 NBCCCommon.h, [989](#)
LowSpeedOffsetErrorType
 LowSpeedIOMAP, [663](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetInBufBuf
 LowSpeedIOMAP, [663](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetInBufBytesToRx
 LowSpeedIOMAP, [663](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetInBufInPtr
 LowSpeedIOMAP, [663](#)
NBCCCommon.h, [990](#)
LowSpeedOffsetInBufOutPtr
 LowSpeedIOMAP, [663](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetMode
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetNoRestartOnRead
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetOutBufBuf
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetOutBufBytesToRx
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetOutBufInPtr
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [990](#)
LowSpeedOffsetOutBufOutPtr
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [991](#)
LowSpeedOffsetSpeed
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [991](#)
LowSpeedOffsetState
 LowSpeedIOMAP, [664](#)
 NBCCCommon.h, [991](#)
LowspeedRead
 LowSpeedModuleFunctions, [319](#)
 NXCDefs.h, [1322](#)
LowSpeedStateConstants
 COM_CHANNEL_FOUR_-
 ACTIVE, [659](#)
 COM_CHANNEL_NONE_-
 ACTIVE, [659](#)
 COM_CHANNEL_ONE_ACTIVE,
 [659](#)
 COM_CHANNEL_THREE_-
 ACTIVE, [659](#)
 COM_CHANNEL_TWO_ACTIVE,
 [659](#)
LowspeedStatus
 LowSpeedModuleFunctions, [320](#)
 NXCDefs.h, [1322](#)
LowspeedWrite

LowSpeedModuleFunctions, 321
NXCDefs.h, 1323

LR_COULD_NOT_SAVE
CommStatusCodesConstants, 712
NBCCommon.h, 991

LR_ENTRY_REMOVED
CommStatusCodesConstants, 712
NBCCommon.h, 991

LR_STORE_IS_FULL
CommStatusCodesConstants, 713
NBCCommon.h, 991

LR_SUCCESS
CommStatusCodesConstants, 713
NBCCommon.h, 991

LR_UNKNOWN_ADDR
CommStatusCodesConstants, 713
NBCCommon.h, 991

LSChannelState
LowLevelLowSpeedModuleFunctions, 326
NXCDefs.h, 1324

LSChannelState constants, 659

LSErrorType
LowLevelLowSpeedModuleFunctions, 327
NXCDefs.h, 1324

LSErrorType constants, 661

LSInputBufferBytesToRx
LowLevelLowSpeedModuleFunctions, 327
NXCDefs.h, 1325

LSInputBufferInPtr
LowLevelLowSpeedModuleFunctions, 327
NXCDefs.h, 1325

LSInputBufferOutPtr
LowLevelLowSpeedModuleFunctions, 328
NXCDefs.h, 1325

LSMode
LowLevelLowSpeedModuleFunctions, 328
NXCDefs.h, 1326

LSMode constants, 661

LSNoRestartOnRead

LowLevelLowSpeedModuleFunctions, 328
NXCDefs.h, 1326

LSNoRestartOnRead constants, 665

LSOutputBufferBytesToRx
LowLevelLowSpeedModuleFunctions, 329
NXCDefs.h, 1326

LSOutputBufferInPtr
LowLevelLowSpeedModuleFunctions, 329
NXCDefs.h, 1327

LSOutputBufferOutPtr
LowLevelLowSpeedModuleFunctions, 330
NXCDefs.h, 1327

LSREAD_NO_RESTART_1
LowSpeedNoRestartConstants, 665
NBCCommon.h, 991

LSREAD_NO_RESTART_2
LowSpeedNoRestartConstants, 665
NBCCommon.h, 991

LSREAD_NO_RESTART_3
LowSpeedNoRestartConstants, 665
NBCCommon.h, 992

LSREAD_NO_RESTART_4
LowSpeedNoRestartConstants, 665
NBCCommon.h, 992

LSREAD_NO_RESTART_MASK
LowSpeedNoRestartConstants, 665
NBCCommon.h, 992

LSREAD_RESTART_ALL
LowSpeedNoRestartConstants, 666
NBCCommon.h, 992

LSREAD_RESTART_NONE
LowSpeedNoRestartConstants, 666
NBCCommon.h, 992

LSSpeed
LowLevelLowSpeedModuleFunctions, 330
NXCDefs.h, 1328

LSState
LowLevelLowSpeedModuleFunctions, 330
NXCDefs.h, 1328

LSState constants, 658

Mailbox constants, [572](#)
MAILBOX1
 MailboxConstants, [573](#)
 NBCCCommon.h, [992](#)
MAILBOX10
 MailboxConstants, [573](#)
 NBCCCommon.h, [992](#)
MAILBOX2
 MailboxConstants, [573](#)
 NBCCCommon.h, [992](#)
MAILBOX3
 MailboxConstants, [573](#)
 NBCCCommon.h, [993](#)
MAILBOX4
 MailboxConstants, [573](#)
 NBCCCommon.h, [993](#)
MAILBOX5
 MailboxConstants, [573](#)
 NBCCCommon.h, [993](#)
MAILBOX6
 MailboxConstants, [574](#)
 NBCCCommon.h, [993](#)
MAILBOX7
 MailboxConstants, [574](#)
 NBCCCommon.h, [993](#)
MAILBOX8
 MailboxConstants, [574](#)
 NBCCCommon.h, [993](#)
MAILBOX9
 MailboxConstants, [574](#)
 NBCCCommon.h, [993](#)
MailboxConstants
 MAILBOX1, [573](#)
 MAILBOX10, [573](#)
 MAILBOX2, [573](#)
 MAILBOX3, [573](#)
 MAILBOX4, [573](#)
 MAILBOX5, [573](#)
 MAILBOX6, [574](#)
 MAILBOX7, [574](#)
 MAILBOX8, [574](#)
 MAILBOX9, [574](#)
MAX_BT_MSG_SIZE
 CommMiscConstants, [695](#)
 NBCCCommon.h, [993](#)
MaxVal

 UpdateCalibCacheInfoType, [879](#)
memcmp
 cstringAPI, [534](#)
 NXCDefs.h, [1328](#)
memcpy
 cstringAPI, [535](#)
 NXCDefs.h, [1329](#)
memmove
 cstringAPI, [535](#)
 NXCDefs.h, [1329](#)
MENUICON_CENTER
 DisplayModuleConstants, [677](#)
 NBCCCommon.h, [993](#)
MENUICON_LEFT
 DisplayModuleConstants, [677](#)
 NBCCCommon.h, [993](#)
MENUICON_RIGHT
 DisplayModuleConstants, [677](#)
 NBCCCommon.h, [994](#)
MENUICONS
 DisplayModuleConstants, [677](#)
 NBCCCommon.h, [994](#)
MENUTEXT
 DisplayModuleConstants, [677](#)
 NBCCCommon.h, [994](#)
Message
 DatalogWriteType, [820](#)
 MessageReadType, [864](#)
 MessageWriteType, [866](#)
MessageRead
 NBCCCommon.h, [994](#)
 SysCallConstants, [560](#)
MessageReadType, [864](#)
 Message, [864](#)
 QueueID, [864](#)
 Remove, [864](#)
 Result, [865](#)
MessageWrite
 NBCCCommon.h, [994](#)
 SysCallConstants, [560](#)
MessageWriteType, [865](#)
 Message, [866](#)
 QueueID, [866](#)
 Result, [866](#)
MidStr
 cstringAPI, [535](#)

NXCDefs.h, 1329
 MIN_1
 NBCCCommon.h, 994
 TimeConstants, 566
 MindSensors ACCL-Nx constants, 787
 MindSensors ACCL-Nx sensitivity level
 constants, 790
 MindSensors API Functions, 107
 MindSensors device constants, 775
 MindSensors DIST-Nx constants, 779
 MindSensors nRLink constants, 785
 MindSensors PSP-Nx button set 1 con-
 stants, 783
 MindSensors PSP-Nx button set 2 con-
 stants, 784
 MindSensors PSP-Nx constants, 781
 MindSensorsAPI
 ACCLNxCalibrateX, 119
 ACCLNxCalibrateXEnd, 119
 ACCLNxCalibrateY, 119
 ACCLNxCalibrateYEnd, 120
 ACCLNxCalibrateZ, 120
 ACCLNxCalibrateZEnd, 121
 ACCLNxResetCalibration, 121
 ACCLNxSensitivity, 122
 ACCLNxXOffset, 122
 ACCLNxXRange, 122
 ACCLNxYOffset, 123
 ACCLNxYRange, 123
 ACCLNxZOffset, 124
 ACCLNxZRange, 124
 DISTNxDistance, 125
 DISTNxGP2D12, 125
 DISTNxGP2D120, 125
 DISTNxGP2YA02, 126
 DISTNxGP2YA21, 126
 DISTNxMaxDistance, 127
 DISTNxMinDistance, 127
 DISTNxModuleType, 128
 DISTNxNumPoints, 128
 DISTNxVoltage, 129
 MSADPAOff, 129
 MSADPAOn, 130
 MSDeenergize, 130
 MSEnergize, 130
 MSIRTrain, 131
 MSPFComboDirect, 131
 MSPFComboPWM, 132
 MSPFRawOutput, 133
 MSPFRepeat, 133
 MSPFSingleOutputCST, 134
 MSPFSingleOutputPWM, 135
 MSPFSinglePin, 135
 MSPFTrain, 136
 MSRCXAbsVar, 137
 MSRCXAddToDatalog, 137
 MSRCXAndVar, 137
 MSRCXBatteryLevel, 138
 MSRCXBoot, 138
 MSRCXCalibrateEvent, 138
 MSRCXClearAllEvents, 139
 MSRCXClearCounter, 139
 MSRCXClearMsg, 139
 MSRCXClearSensor, 140
 MSRCXClearSound, 140
 MSRCXClearTimer, 140
 MSRCXCreateDatalog, 140
 MSRCXDecCounter, 141
 MSRCXDeleteSub, 141
 MSRCXDeleteSubs, 141
 MSRCXDeleteTask, 142
 MSRCXDeleteTasks, 142
 MSRCXDisableOutput, 142
 MSRCXDivVar, 142
 MSRCXEnableOutput, 143
 MSRCXEvent, 143
 MSRCXFloat, 143
 MSRCXFwd, 144
 MSRCXIIncCounter, 144
 MSRCXIvertOutput, 144
 MSRCXMulVar, 145
 MSRCXMuteSound, 145
 MSRCXObvertOutput, 145
 MSRCXOff, 146
 MSRCXOn, 146
 MSRCXOnFor, 146
 MSRCXOnFwd, 147
 MSRCXOnRev, 147
 MSRCXOrVar, 147
 MSRCXPBTurnOff, 148
 MSRCXPing, 148
 MSRCXPlaySound, 148

MSRCXPlayTone, 148
MSRCXPlayToneVar, 149
MSRCXPoll, 149
MSRCXPollMemory, 150
MSRCXRemote, 150
MSRCXReset, 150
MSRCXRev, 150
MSRCXSelectDisplay, 151
MSRCXSelectProgram, 151
MSRCXSendSerial, 151
MSRCXSet, 152
MSRCXSetDirection, 152
MSRCXSetEvent, 153
MSRCXSetGlobalDirection, 153
MSRCXSetGlobalOutput, 153
MSRCXSetMaxPower, 154
MSRCXSetMessage, 154
MSRCXSetNRLLinkPort, 154
MSRCXSetOutput, 155
MSRCXSetPower, 155
MSRCXSetPriority, 156
MSRCXSetSensorMode, 156
MSRCXSetSensorType, 156
MSRCXSetSleepTime, 157
MSRCXSetTxPower, 157
MSRCXSetUserDisplay, 157
MSRCXSetVar, 158
MSRCXSetWatch, 158
MSRCXSgnVar, 158
MSRCXStartTask, 159
MSRCXStopAllTasks, 159
MSRCXStopTask, 159
MSRCXSubVar, 160
MSRCXSumVar, 160
MSRCXToggle, 160
MSRCXUnlock, 161
MSRCXUnmuteSound, 161
MSReadValue, 161
MSScoutCalibrateSensor, 162
MSScoutMuteSound, 162
MSScoutSelectSounds, 162
MSScoutSendVLL, 162
MSScoutSetCounterLimit, 163
MSScoutSetEventFeedback, 163
MSScoutSetLight, 163
MSScoutSetScoutMode, 164
MSScoutSetScoutRules, 164
MSScoutSetSensorClickTime, 164
MSScoutSetSensorHysteresis, 165
MSScoutSetSensorLowerLimit, 165
MSScoutSetSensorUpperLimit, 166
MSScoutSetTimerLimit, 166
MSScoutUnmuteSound, 166
NRLLink2400, 166
NRLLink4800, 167
NRLLinkFlush, 167
NRLLinkIRLong, 168
NRLLinkIRShort, 168
NRLLinkSetPF, 169
NRLLinkSetRCX, 169
NRLLinkSetTrain, 169
NRLLinkStatus, 170
NRLLinkTxRaw, 170
PSPNxAnalog, 171
PSPNxDigital, 171
ReadNRLLinkBytes, 172
ReadSensorMSAccel, 172
ReadSensorMSPlayStation, 173
ReadSensorMSRTClock, 173
ReadSensorMSTilt, 174
RunNRLLinkMacro, 175
SensorMSCompass, 175
SensorMSDROD, 176
SensorMSPressure, 176
SensorMSPressureRaw, 176
SetACCLNxSensitivity, 177
SetSensorMSDROD, 177
SetSensorMSPressure, 178
WriteNRLLinkBytes, 178
MindSensorsConstants
 MS_ADDR_ACCLNX, 777
 MS_ADDR_CMPSNX, 777
 MS_ADDR_DISTNX, 777
 MS_ADDR_IVSENS, 777
 MS_ADDR_LINELDR, 777
 MS_ADDR_MTRMUX, 777
 MS_ADDR_NRLINK, 778
 MS_ADDR_NXTCAM, 778
 MS_ADDR_NXTHID, 778
 MS_ADDR_NXTMMX, 778
 MS_ADDR_NXTSERVO, 778
 MS_ADDR_PFMATE, 778

MS_ADDR_PSPNX, [778](#)
MS_ADDR_RTCLOCK, [779](#)
MS_ADDR_RXMUX, [779](#)
MS_CMD_ADPA_OFF, [779](#)
MS_CMD_ADPA_ON, [779](#)
MS_CMD_DEENERGIZED, [779](#)
MS_CMD_ENERGIZED, [779](#)

MinVal
 UpdateCalibCacheInfoType, [879](#)

MiscConstants
 DEGREES_PER_RADIAN, [192](#)
 FALSE, [192](#)
 NA, [192](#)
 PI, [192](#)
 RADIANSPERDEGREE, [192](#)
 TRUE, [193](#)

Miscellaneous Comm module constants, [694](#)

Miscellaneous NBC/NXC constants, [191](#)

Mode
 CommHSControlType, [809](#)

ModuleID
 IOMapReadByIDType, [851](#)
 IOMapWriteByIDType, [855](#)

ModuleIDConstants
 ButtonModuleID, [190](#)
 CommandModuleID, [190](#)
 CommModuleID, [190](#)
 DisplayModuleID, [190](#)
 InputModuleID, [190](#)
 IOCtrlModuleID, [190](#)
 LoaderModuleID, [191](#)
 LowSpeedModuleID, [191](#)
 OutputModuleID, [191](#)
 SoundModuleID, [191](#)
 UIModuleID, [191](#)

ModuleName
 IOMapReadType, [853](#)
 IOMapWriteType, [856](#)

ModuleNameConstants
 ButtonModuleName, [188](#)
 CommandModuleName, [188](#)
 CommModuleName, [188](#)
 DisplayModuleName, [188](#)
 InputModuleName, [188](#)
 IOCtrlModuleName, [189](#)

LoaderModuleName, [189](#)
LowSpeedModuleName, [189](#)
OutputModuleName, [189](#)
SoundModuleName, [189](#)
UIModuleName, [189](#)

MotorActualSpeed
 NXCDefs.h, [1330](#)
 OutputModuleFunctions, [247](#)

MotorBlockTachoCount
 NXCDefs.h, [1330](#)
 OutputModuleFunctions, [247](#)

MotorMode
 NXCDefs.h, [1331](#)
 OutputModuleFunctions, [248](#)

MotorOverload
 NXCDefs.h, [1331](#)
 OutputModuleFunctions, [248](#)

MotorPower
 NXCDefs.h, [1331](#)
 OutputModuleFunctions, [248](#)

MotorPwnFreq
 NXCDefs.h, [1331](#)
 OutputModuleFunctions, [249](#)

MotorRegDValue
 NXCDefs.h, [1332](#)
 OutputModuleFunctions, [249](#)

MotorRegIValue
 NXCDefs.h, [1332](#)
 OutputModuleFunctions, [249](#)

MotorRegPValue
 NXCDefs.h, [1332](#)
 OutputModuleFunctions, [250](#)

MotorRegulation
 NXCDefs.h, [1333](#)
 OutputModuleFunctions, [250](#)

MotorRotationCount
 NXCDefs.h, [1333](#)
 OutputModuleFunctions, [250](#)

MotorRunState
 NXCDefs.h, [1333](#)
 OutputModuleFunctions, [251](#)

MotorTachoCount
 NXCDefs.h, [1334](#)
 OutputModuleFunctions, [251](#)

MotorTachoLimit
 NXCDefs.h, [1334](#)

OutputModuleFunctions, 251
MotorTurnRatio
 NXCDefs.h, 1334
 OutputModuleFunctions, 252
MS_1
 NBCCCommon.h, 994
 TimeConstants, 566
MS_10
 NBCCCommon.h, 994
 TimeConstants, 566
MS_100
 NBCCCommon.h, 994
 TimeConstants, 566
MS_150
 NBCCCommon.h, 995
 TimeConstants, 566
MS_2
 NBCCCommon.h, 995
 TimeConstants, 566
MS_20
 NBCCCommon.h, 995
 TimeConstants, 566
MS_200
 NBCCCommon.h, 995
 TimeConstants, 567
MS_250
 NBCCCommon.h, 995
 TimeConstants, 567
MS_3
 NBCCCommon.h, 995
 TimeConstants, 567
MS_30
 NBCCCommon.h, 995
 TimeConstants, 567
MS_300
 NBCCCommon.h, 996
 TimeConstants, 567
MS_350
 NBCCCommon.h, 996
 TimeConstants, 567
MS_4
 NBCCCommon.h, 996
 TimeConstants, 567
MS_40
 NBCCCommon.h, 996
 TimeConstants, 567
MS_400
 NBCCCommon.h, 996
 TimeConstants, 568
MS_450
 NBCCCommon.h, 996
 TimeConstants, 568
MS_5
 NBCCCommon.h, 996
 TimeConstants, 568
MS_50
 NBCCCommon.h, 996
 TimeConstants, 568
MS_500
 NBCCCommon.h, 997
 TimeConstants, 568
MS_6
 NBCCCommon.h, 997
 TimeConstants, 568
MS_60
 NBCCCommon.h, 997
 TimeConstants, 568
MS_600
 NBCCCommon.h, 997
 TimeConstants, 569
MS_7
 NBCCCommon.h, 997
 TimeConstants, 569
MS_70
 NBCCCommon.h, 997
 TimeConstants, 569
MS_700
 NBCCCommon.h, 997
 TimeConstants, 569
MS_8
 NBCCCommon.h, 997
 TimeConstants, 569
MS_80
 NBCCCommon.h, 997
 TimeConstants, 569
MS_800
 NBCCCommon.h, 998
 TimeConstants, 569
MS_9
 NBCCCommon.h, 998
 TimeConstants, 569
MS_90

NBCCCommon.h, 998
TimeConstants, 569

MS_900
 NBCCCommon.h, 998
 TimeConstants, 569

MS_ADDR_ACCLNX
 MindSensorsConstants, 777
 NBCCCommon.h, 998

MS_ADDR_CMPSNX
 MindSensorsConstants, 777
 NBCCCommon.h, 998

MS_ADDR_DISTNX
 MindSensorsConstants, 777
 NBCCCommon.h, 998

MS_ADDR_IVSENS
 MindSensorsConstants, 777
 NBCCCommon.h, 999

MS_ADDR_LINELDR
 MindSensorsConstants, 777
 NBCCCommon.h, 999

MS_ADDR_MTRMUX
 MindSensorsConstants, 777
 NBCCCommon.h, 999

MS_ADDR_NRLINK
 MindSensorsConstants, 778
 NBCCCommon.h, 999

MS_ADDR_NXTCAM
 MindSensorsConstants, 778
 NBCCCommon.h, 999

MS_ADDR_NXTHID
 MindSensorsConstants, 778
 NBCCCommon.h, 999

MS_ADDR_NXTMMX
 MindSensorsConstants, 778
 NBCCCommon.h, 999

MS_ADDR_NXTSERVO
 MindSensorsConstants, 778
 NBCCCommon.h, 1000

MS_ADDR_PFMATE
 MindSensorsConstants, 778
 NBCCCommon.h, 1000

MS_ADDR_PSPNX
 MindSensorsConstants, 778
 NBCCCommon.h, 1000

MS_ADDR_RTCLOCK
 MindSensorsConstants, 779

NBCCCommon.h, 1000
MindSensorsConstants, 779

MS_CMD_ADPA_OFF
 MindSensorsConstants, 779
 NBCCCommon.h, 1000

MS_CMD_ADPA_ON
 MindSensorsConstants, 779
 NBCCCommon.h, 1000

MS_CMD_DEENERGIZED
 MindSensorsConstants, 779
 NBCCCommon.h, 1000

MS_CMD_ENERGIZED
 MindSensorsConstants, 779
 NBCCCommon.h, 1000

MSACCLNx
 ACCL_CMD_RESET_CAL, 788
 ACCL_CMD_X_CAL, 788
 ACCL_CMD_X_CAL_END, 788
 ACCL_CMD_Y_CAL, 788
 ACCL_CMD_Y_CAL_END, 788
 ACCL_CMD_Z_CAL, 789
 ACCL_CMD_Z_CAL_END, 789
 ACCL_REG_SENS_LVL, 789
 ACCL_REG_X_ACCEL, 789
 ACCL_REG_X_OFFSET, 789
 ACCL_REG_X_RANGE, 789
 ACCL_REG_X_TILT, 789
 ACCL_REG_Y_ACCEL, 789
 ACCL_REG_Y_OFFSET, 789
 ACCL_REG_Y_RANGE, 789
 ACCL_REG_Y_TILT, 790
 ACCL_REG_Z_ACCEL, 790
 ACCL_REG_Z_OFFSET, 790
 ACCL_REG_Z_RANGE, 790
 ACCL_REG_Z_TILT, 790

MSACCLNxSLevel
 ACCL_SENSITIVITY_LEVEL_1,
 791
 ACCL_SENSITIVITY_LEVEL_2,
 791
 ACCL_SENSITIVITY_LEVEL_3,
 791
 ACCL_SENSITIVITY_LEVEL_4,
 791

MSADPAOff
 MindSensorsAPI, 129
 NXCDefs.h, 1335

MSADPAOn
 MindSensorsAPI, 130
 NXCDefs.h, 1335

MSDeenergize
 MindSensorsAPI, 130
 NXCDefs.h, 1336

MSDistNX
 DIST_CMD_CUSTOM, 780
 DIST_CMD_GP2D12, 780
 DIST_CMD_GP2D120, 780
 DIST_CMD_GP2YA02, 780
 DIST_CMD_GP2YA21, 780
 DIST_REG_DIST, 780
 DIST_REG_DIST1, 780
 DIST_REG_DIST_MAX, 781
 DIST_REG_DIST_MIN, 781
 DIST_REG_MODULE_TYPE, 781
 DIST_REG_NUM_POINTS, 781
 DIST_REG_VOLT, 781
 DIST_REG_VOLT1, 781

MSEnergize
 MindSensorsAPI, 130
 NXCDefs.h, 1336

MSIRTrain
 MindSensorsAPI, 131
 NXCDefs.h, 1336

MSNRLink
 NRLINK_CMD_2400, 786
 NRLINK_CMD_4800, 786
 NRLINK_CMD_FLUSH, 786
 NRLINK_CMD_IR_LONG, 786
 NRLINK_CMD_IR_SHORT, 786
 NRLINK_CMD_RUN_MACRO,
 786
 NRLINK_CMD_SET_PF, 786
 NRLINK_CMD_SET_RCX, 786
 NRLINK_CMD_SET_TRAIN, 787
 NRLINK_CMD_TX_RAW, 787
 NRLINK_REG_BYTES, 787
 NRLINK_REG_DATA, 787
 NRLINK_REG_EEPROM, 787

MSPFComboDirect
 MindSensorsAPI, 131

NXCDefs.h, 1337

MSPFComboPWM
 MindSensorsAPI, 132
 NXCDefs.h, 1338

MSPFRawOutput
 MindSensorsAPI, 133
 NXCDefs.h, 1338

MSPFRepeat
 MindSensorsAPI, 133
 NXCDefs.h, 1339

MSPFSingleOutputCST
 MindSensorsAPI, 134
 NXCDefs.h, 1340

MSPFSingleOutputPWM
 MindSensorsAPI, 135
 NXCDefs.h, 1340

MSPFSinglePin
 MindSensorsAPI, 135
 NXCDefs.h, 1341

MSPFTrain
 MindSensorsAPI, 136
 NXCDefs.h, 1342

MSPSPNX
 PSP_CMD_ANALOG, 782
 PSP_CMD_DIGITAL, 782
 PSP_REG_BTNSET1, 782
 PSP_REG_BTNSET2, 782
 PSP_REG_XLEFT, 782
 PSP_REG_XRIGHT, 782
 PSP_REG_YLEFT, 782
 PSP_REG_YRIGHT, 783

MSPSPNxBtnSet1
 PSP_BTNSET1_DOWN, 783
 PSP_BTNSET1_L3, 783
 PSP_BTNSET1_LEFT, 783
 PSP_BTNSET1_R3, 783
 PSP_BTNSET1_RIGHT, 784
 PSP_BTNSET1_UP, 784

MSPSPNxBtnSet2
 PSP_BTNSET2_CIRCLE, 784
 PSP_BTNSET2_CROSS, 784
 PSP_BTNSET2_L1, 784
 PSP_BTNSET2_L2, 785
 PSP_BTNSET2_R1, 785
 PSP_BTNSET2_R2, 785
 PSP_BTNSET2_SQUARE, 785

PSP_BTNSET2_TRIANGLE, 785
MSRCXAbsVar
 MindSensorsAPI, 137
 NXCDefs.h, 1342
MSRCXAddToDatalog
 MindSensorsAPI, 137
 NXCDefs.h, 1343
MSRCXAndVar
 MindSensorsAPI, 137
 NXCDefs.h, 1343
MSRCXBatteryLevel
 MindSensorsAPI, 138
 NXCDefs.h, 1344
MSRCXBoot
 MindSensorsAPI, 138
 NXCDefs.h, 1344
MSRCXCalibrateEvent
 MindSensorsAPI, 138
 NXCDefs.h, 1344
MSRCXClearAllEvents
 MindSensorsAPI, 139
 NXCDefs.h, 1344
MSRCXClearCounter
 MindSensorsAPI, 139
 NXCDefs.h, 1345
MSRCXClearMsg
 MindSensorsAPI, 139
 NXCDefs.h, 1345
MSRCXClearSensor
 MindSensorsAPI, 140
 NXCDefs.h, 1345
MSRCXClearSound
 MindSensorsAPI, 140
 NXCDefs.h, 1345
MSRCXClearTimer
 MindSensorsAPI, 140
 NXCDefs.h, 1346
MSRCXCreateDatalog
 MindSensorsAPI, 140
 NXCDefs.h, 1346
MSRCXDecCounter
 MindSensorsAPI, 141
 NXCDefs.h, 1346
MSRCXDeleteSub
 MindSensorsAPI, 141
 NXCDefs.h, 1347
MSRCXDeleteSubs
 MindSensorsAPI, 141
 NXCDefs.h, 1347
MSRCXDeleteTask
 MindSensorsAPI, 142
 NXCDefs.h, 1347
MSRCXDeleteTasks
 MindSensorsAPI, 142
 NXCDefs.h, 1347
MSRCXDisableOutput
 MindSensorsAPI, 142
 NXCDefs.h, 1348
MSRCXDivVar
 MindSensorsAPI, 142
 NXCDefs.h, 1348
MSRCXEnableOutput
 MindSensorsAPI, 143
 NXCDefs.h, 1348
MSRCXEvent
 MindSensorsAPI, 143
 NXCDefs.h, 1349
MSRCXFloat
 MindSensorsAPI, 143
 NXCDefs.h, 1349
MSRCXFwd
 MindSensorsAPI, 144
 NXCDefs.h, 1349
MSRCXIncCounter
 MindSensorsAPI, 144
 NXCDefs.h, 1350
MSRCXIvertOutput
 MindSensorsAPI, 144
 NXCDefs.h, 1350
MSRCXMulVar
 MindSensorsAPI, 145
 NXCDefs.h, 1350
MSRCXMuteSound
 MindSensorsAPI, 145
 NXCDefs.h, 1351
MSRCXObvertOutput
 MindSensorsAPI, 145
 NXCDefs.h, 1351
MSRCXOff
 MindSensorsAPI, 146
 NXCDefs.h, 1351
MSRCXOn

MindSensorsAPI, 146
NXCDefs.h, 1351

MSRCXOnFor
MindSensorsAPI, 146
NXCDefs.h, 1352

MSRCXOnFwd
MindSensorsAPI, 147
NXCDefs.h, 1352

MSRCXOnRev
MindSensorsAPI, 147
NXCDefs.h, 1352

MSRCXOrVar
MindSensorsAPI, 147
NXCDefs.h, 1353

MSRCXPBTurnOff
MindSensorsAPI, 148
NXCDefs.h, 1353

MSRCXPing
MindSensorsAPI, 148
NXCDefs.h, 1353

MSRCXPlaySound
MindSensorsAPI, 148
NXCDefs.h, 1354

MSRCXPlayTone
MindSensorsAPI, 148
NXCDefs.h, 1354

MSRCXPlayToneVar
MindSensorsAPI, 149
NXCDefs.h, 1354

MSRCXPoll
MindSensorsAPI, 149
NXCDefs.h, 1355

MSRCXPollMemory
MindSensorsAPI, 150
NXCDefs.h, 1355

MSRCXRemote
MindSensorsAPI, 150
NXCDefs.h, 1355

MSRCXReset
MindSensorsAPI, 150
NXCDefs.h, 1356

MSRCXRev
MindSensorsAPI, 150
NXCDefs.h, 1356

MSRCXSelectDisplay
MindSensorsAPI, 151
NXCDefs.h, 1356

MSRCXSelectProgram
MindSensorsAPI, 151
NXCDefs.h, 1357

MSRCXSendSerial
MindSensorsAPI, 151
NXCDefs.h, 1357

MSRCXSet
MindSensorsAPI, 152
NXCDefs.h, 1357

MSRCXSetDirection
MindSensorsAPI, 152
NXCDefs.h, 1358

MSRCXSetEvent
MindSensorsAPI, 153
NXCDefs.h, 1358

MSRCXSetGlobalDirection
MindSensorsAPI, 153
NXCDefs.h, 1358

MSRCXSetGlobalOutput
MindSensorsAPI, 153
NXCDefs.h, 1359

MSRCXSetMaxPower
MindSensorsAPI, 154
NXCDefs.h, 1359

MSRCXSetMessage
MindSensorsAPI, 154
NXCDefs.h, 1360

MSRCXSetNRLinkPort
MindSensorsAPI, 154
NXCDefs.h, 1360

MSRCXSetOutput
MindSensorsAPI, 155
NXCDefs.h, 1360

MSRCXSetPower
MindSensorsAPI, 155
NXCDefs.h, 1361

MSRCXSetPriority
MindSensorsAPI, 156
NXCDefs.h, 1361

MSRCXSetSensorMode
MindSensorsAPI, 156
NXCDefs.h, 1361

MSRCXSetSensorType
MindSensorsAPI, 156
NXCDefs.h, 1362

MSRCXSetSleepTime
 MindSensorsAPI, 157
 NXCDefs.h, 1362

MSRCXSetTxPower
 MindSensorsAPI, 157
 NXCDefs.h, 1362

MSRCXSetUserDisplay
 MindSensorsAPI, 157
 NXCDefs.h, 1363

MSRCXSetVar
 MindSensorsAPI, 158
 NXCDefs.h, 1363

MSRCXSetWatch
 MindSensorsAPI, 158
 NXCDefs.h, 1363

MSRCXSgnVar
 MindSensorsAPI, 158
 NXCDefs.h, 1364

MSRCXStartTask
 MindSensorsAPI, 159
 NXCDefs.h, 1364

MSRCXStopAllTasks
 MindSensorsAPI, 159
 NXCDefs.h, 1364

MSRCXStopTask
 MindSensorsAPI, 159
 NXCDefs.h, 1365

MSRCXSubVar
 MindSensorsAPI, 160
 NXCDefs.h, 1365

MSRCXSumVar
 MindSensorsAPI, 160
 NXCDefs.h, 1365

MSRCXToggle
 MindSensorsAPI, 160
 NXCDefs.h, 1366

MSRCXUnlock
 MindSensorsAPI, 161
 NXCDefs.h, 1366

MSRCXUnmuteSound
 MindSensorsAPI, 161
 NXCDefs.h, 1366

MSReadValue
 MindSensorsAPI, 161
 NXCDefs.h, 1366

MSScoutCalibrateSensor

 MindSensorsAPI, 162
 NXCDefs.h, 1367

MSScoutMuteSound
 MindSensorsAPI, 162
 NXCDefs.h, 1367

MSScoutSelectSounds
 MindSensorsAPI, 162
 NXCDefs.h, 1367

MSScoutSendVLL
 MindSensorsAPI, 162
 NXCDefs.h, 1368

MSScoutSetCounterLimit
 MindSensorsAPI, 163
 NXCDefs.h, 1368

MSScoutSetEventFeedback
 MindSensorsAPI, 163
 NXCDefs.h, 1368

MSScoutSetLight
 MindSensorsAPI, 163
 NXCDefs.h, 1369

MSScoutSetScoutMode
 MindSensorsAPI, 164
 NXCDefs.h, 1369

MSScoutSetScoutRules
 MindSensorsAPI, 164
 NXCDefs.h, 1369

MSScoutSetSensorClickTime
 MindSensorsAPI, 164
 NXCDefs.h, 1370

MSScoutSetSensorHysteresis
 MindSensorsAPI, 165
 NXCDefs.h, 1370

MSScoutSetSensorLowerLimit
 MindSensorsAPI, 165
 NXCDefs.h, 1371

MSScoutSetSensorUpperLimit
 MindSensorsAPI, 166
 NXCDefs.h, 1371

MSScoutSetTimerLimit
 MindSensorsAPI, 166
 NXCDefs.h, 1371

MSScoutUnmuteSound
 MindSensorsAPI, 166
 NXCDefs.h, 1372

MulDiv32
 cmathAPI, 486

NXCDefs.h, 1168
muldiv32
 cmathAPI, 501
 NXCDefs.h, 1372

NA
 MiscConstants, 192
 NBCCCommon.h, 1001

Name
 CommBTConnectionType, 802
 CommExecuteFunctionType, 807
 ComputeCalibValueType, 817
 UpdateCalibCacheInfoType, 879

NBC Input port constants, 625
NBC sensor mode constants, 629
NBC sensor type constants, 626
NBCCCommon.h, 881
 ACCL_CMD_RESET_CAL, 917
 ACCL_CMD_X_CAL, 917
 ACCL_CMD_X_CAL_END, 917
 ACCL_CMD_Y_CAL, 917
 ACCL_CMD_Y_CAL_END, 918
 ACCL_CMD_Z_CAL, 918
 ACCL_CMD_Z_CAL_END, 918
 ACCL_REG_SENS_LVL, 918
 ACCL_REG_X_ACCEL, 918
 ACCL_REG_X_OFFSET, 918
 ACCL_REG_X_RANGE, 918
 ACCL_REG_X_TILT, 918
 ACCL_REG_Y_ACCEL, 918
 ACCL_REG_Y_OFFSET, 918
 ACCL_REG_Y_RANGE, 919
 ACCL_REG_Y_TILT, 919
 ACCL_REG_Z_ACCEL, 919
 ACCL_REG_Z_OFFSET, 919
 ACCL_REG_Z_RANGE, 919
 ACCL_REG_Z_TILT, 919
 ACCL_SENSITIVITY_LEVEL_1, 919
 ACCL_SENSITIVITY_LEVEL_2, 919
 ACCL_SENSITIVITY_LEVEL_3, 919
 ACCL_SENSITIVITY_LEVEL_4, 920
 ActualSpeed, 920

BITMAP_1, 920
BITMAP_2, 920
BITMAP_3, 920
BITMAP_4, 920
BITMAPS, 920
BlockTachoCount, 920
BREAKOUT_REQ, 921
BT_ARM_CMD_MODE, 921
BT_ARM_DATA_MODE, 921
BT_ARM_OFF, 921
BT_BRICK_PORT_OPEN, 921
BT_BRICK_VISIBILITY, 921
BT_CMD_BYTE, 921
BT_CMD_READY, 921
BT_CONNECTION_0_ENABLE, 921
BT_CONNECTION_1_ENABLE, 922
BT_CONNECTION_2_ENABLE, 922
BT_CONNECTION_3_ENABLE, 922
BT_DEFAULT_INQUIRY_MAX, 922
BT_DEFAULT_INQUIRY_TIMEOUT_LO, 922
BT_DEVICE_AWAY, 922
BT_DEVICE_EMPTY, 922
BT_DEVICE_KNOWN, 922
BT_DEVICE_NAME, 922
BT_DEVICE_UNKNOWN, 922
BT_DISABLE, 923
BT_ENABLE, 923
BTN1, 923
BTN2, 923
BTN3, 923
BTN4, 923
BTNCENTER, 923
BTNEXTIT, 924
BTNLEFT, 924
BTNRIGHT, 924
BTNSTATE_LONG_PRESSED_EV, 924
BTNSTATE_LONG_RELEASED_EV, 924
BTNSTATE_NONE, 924

BTNSTATE_PRESSED_EV, 925
BTNSTATE_PRESSED_STATE,
 925
BTNSTATE_SHORT_-
 RELEASED_EV, 925
ButtonModuleID, 925
ButtonModuleName, 925
ButtonOffsetLongPressCnt, 925
ButtonOffsetLongRelCnt, 925
ButtonOffsetPressedCnt, 925
ButtonOffsetRelCnt, 925
ButtonOffsetShortRelCnt, 926
ButtonOffsetState, 926
CHAR_BIT, 926
CHAR_MAX, 926
CHAR_MIN, 926
CLUMP_DONE, 926
CLUMP_SUSPEND, 926
ColorSensorRead, 926
COM_CHANNEL_FOUR_-
 ACTIVE, 926
COM_CHANNEL_NONE_-
 ACTIVE, 926
COM_CHANNEL_ONE_ACTIVE,
 927
COM_CHANNEL_THREE_-
 ACTIVE, 927
COM_CHANNEL_TWO_ACTIVE,
 927
CommandModuleID, 927
CommandModuleName, 927
CommandOffsetActivateFlag, 927
CommandOffsetAwake, 927
CommandOffsetDeactivateFlag, 927
CommandOffsetFileName, 928
CommandOffsetFormatString, 928
CommandOffsetMemoryPool, 928
CommandOffsetOffsetDS, 928
CommandOffsetOffsetDVA, 928
CommandOffsetPRCHandler, 928
CommandOffsetProgStatus, 928
CommandOffsetSyncTick, 928
CommandOffsetSyncTime, 928
CommandOffsetTick, 929
CommBTCheckStatus, 929
CommBTConnection, 929
CommBTOnOff, 929
CommBTRead, 929
CommBTWrite, 929
CommExecuteFunction, 929
CommHSCheckStatus, 929
CommHSControl, 929
CommHSRead, 930
CommHSWrite, 930
CommLSCheckStatus, 930
CommLSRead, 930
CommLSWrite, 930
CommLSWriteEx, 930
CommModuleID, 930
CommModuleName, 930
CommOffsetBrickDataBdAddr, 930
CommOffsetBrickDataBluecoreVer-
 sion, 930
CommOffsetBrickDataBtHwStatus,
 931
CommOffsetBrickDataBtStateSta-
 tus, 931
CommOffsetBrickDataName, 931
CommOffsetBrickDataTimeOut-
 Value, 931
CommOffsetBtConnectTableB-
 dAddr, 931
CommOffsetBtConnectTableClas-
 sOfDevice, 931
CommOffsetBtConnectTableHan-
 dleNr, 931
CommOffsetBtConnect-
 TableLinkQuality, 931
CommOffsetBtConnectTableName,
 931
CommOffsetBtConnectTablePin-
 Code, 931
CommOffsetBtConnectTa-
 bleStreamStatus, 932
CommOffsetBtDeviceCnt, 932
CommOffsetBtDeviceNameCnt, 932
CommOffsetBtDeviceTableBdAddr,
 932
CommOffsetBtDeviceTableClas-
 sOfDevice, 932
CommOffsetBtDeviceTableDe-
 viceStatus, 932

CommOffsetBtDeviceTableName, 932
CommOffsetBtInBufBuf, 932
CommOffsetBtInBufInPtr, 932
CommOffsetBtInBufOutPtr, 932
CommOffsetBtOutBufBuf, 933
CommOffsetBtOutBufInPtr, 933
CommOffsetBtOutBufOutPtr, 933
CommOffsetHsFlags, 933
CommOffsetHsInBufBuf, 933
CommOffsetHsInBufInPtr, 933
CommOffsetHsInBufOutPtr, 933
CommOffsetHsMode, 933
CommOffsetHsOutBufBuf, 933
CommOffsetHsOutBufInPtr, 933
CommOffsetHsOutBufOutPtr, 934
CommOffsetHsSpeed, 934
CommOffsetHsState, 934
CommOffsetPFunc, 934
CommOffsetPFuncTwo, 934
CommOffsetUsbInBufBuf, 934
CommOffsetUsbInBufInPtr, 934
CommOffsetUsbInBufOutPtr, 934
CommOffsetUsbOutBufBuf, 934
CommOffsetUsbOutBuInPtr, 934
CommOffsetUsbOutBufOutPtr, 935
CommOffsetUsbPollBufBuf, 935
CommOffsetUsbPollBufInPtr, 935
CommOffsetUsbPollBufOutPtr, 935
CommOffsetUsbState, 935
ComputeCalibValue, 935
DatalogGetTimes, 935
DatalogWrite, 935
DEGREES_PER_RADIAN, 935
DISPLAY_BUSY, 935
DISPLAY_CHAR, 936
DISPLAY_CONTRAST_-
 DEFAULT, 936
DISPLAY_CONTRAST_MAX, 936
DISPLAY_ERASE_ALL, 936
DISPLAY_ERASE_LINE, 936
DISPLAY_FILL_REGION, 936
DISPLAY_FRAME, 936
DISPLAY_HEIGHT, 937
DISPLAY_HORIZONTAL_LINE,
 937
DISPLAY_MENUICONS_X_DIFF,
 937
DISPLAY_MENUICONS_X_-
 OFFS, 937
DISPLAY_MENUICONS_Y, 937
DISPLAY_ON, 937
DISPLAY_PIXEL, 937
DISPLAY_POPUP, 937
DISPLAY_REFRESH, 938
DISPLAY_REFRESH_DISABLED,
 938
DISPLAY_VERTICAL_LINE, 938
DISPLAY_WIDTH, 938
DisplayExecuteFunction, 938
DisplayModuleID, 938
DisplayModuleName, 938
DisplayOffsetContrast, 938
DisplayOffsetDisplay, 939
DisplayOffsetEraseMask, 939
DisplayOffsetFlags, 939
DisplayOffsetNormal, 939
DisplayOffsetPBitmaps, 939
DisplayOffsetPFont, 939
DisplayOffsetPFunc, 939
DisplayOffsetPMenuIcons, 939
DisplayOffsetPMenuText, 939
DisplayOffsetPopup, 939
DisplayOffsetPScreens, 940
DisplayOffsetPStatusIcons, 940
DisplayOffsetPStatusText, 940
DisplayOffsetPStepIcons, 940
DisplayOffsetPTextLines, 940
DisplayOffsetStatusIcons, 940
DisplayOffsetStepIcons, 940
DisplayOffsetTextLinesCenterFlags,
 940
DisplayOffsetUpdateMask, 940
DIST_CMD_CUSTOM, 940
DIST_CMD_GP2D12, 941
DIST_CMD_GP2D120, 941
DIST_CMD_GP2YA02, 941
DIST_CMD_GP2YA21, 941
DIST_REG_DIST, 941
DIST_REG_DIST1, 941
DIST_REG_DIST_MAX, 941
DIST_REG_DIST_MIN, 941

DIST_REG_MODULE_TYPE, 941
DIST_REG_NUM_POINTS, 941
DIST_REG_VOLT, 942
DIST_REG_VOLT1, 942
DRAW_OPT_CLEAR, 942
DRAW_OPT_CLEAR_EXCEPT_-
 STATUS_SCREEN, 942
DRAW_OPT_CLEAR_PIXELS,
 942
DRAW_OPT_CLEAR_SCREEN_-
 MODES, 942
DRAW_OPT_CLEAR_WHOLE_-
 SCREEN, 942
DRAW_OPT_FILL_SHAPE, 942
DRAW_OPT_FONT_DIR_B2TL,
 943
DRAW_OPT_FONT_DIR_B2TR,
 943
DRAW_OPT_FONT_DIR_L2RB,
 943
DRAW_OPT_FONT_DIR_L2RT,
 943
DRAW_OPT_FONT_DIR_R2LB,
 943
DRAW_OPT_FONT_DIR_R2LT,
 943
DRAW_OPT_FONT_DIR_T2BL,
 943
DRAW_OPT_FONT_DIR_T2BR,
 944
DRAW_OPT_FONT_-
 DIRECTIONS, 944
DRAW_OPT_FONT_WRAP, 944
DRAW_OPT_INVERT, 944
DRAW_OPT_LOGICAL_AND,
 944
DRAW_OPT_LOGICAL_COPY,
 944
DRAW_OPT_LOGICAL_-
 OPERATIONS, 945
DRAW_OPT_LOGICAL_OR, 945
DRAW_OPT_LOGICAL_XOR,
 945
DRAW_OPT_NORMAL, 945
DRAW_OPT_POLYGON_-
 POLYLINE, 945
DrawCircle, 945
DrawEllipse, 945
DrawFont, 946
DrawGraphic, 946
DrawGraphicArray, 946
DrawLine, 946
DrawPoint, 946
DrawPolygon, 946
DrawRect, 946
DrawText, 946
EMETER_REG_AIN, 947
EMETER_REG_AOUT, 947
EMETER_REG_JOULES, 947
EMETER_REG_VIN, 947
EMETER_REG_VOUT, 947
EMETER_REG_WIN, 947
EMETER_REG_WOUT, 947
EOF, 947
ERR_ARG, 947
ERR_BAD_POOL_SIZE, 947
ERR_BAD_PTR, 948
ERR_CLUMP_COUNT, 948
ERR_COMM_BUFFER_FULL,
 948
ERR_COMM_BUS_ERR, 948
ERR_COMM_CHAN_INVALID,
 948
ERR_COMM_CHAN_NOT_-
 READY, 948
ERR_DEFAULT_OFFSETS, 948
ERR_FILE, 948
ERR_INSANE_OFFSET, 948
ERR_INSTR, 948
ERR_INVALID_FIELD, 949
ERR_INVALID_PORT, 949
ERR_INVALID_QUEUE, 949
ERR_INVALID_SIZE, 949
ERR_LOADER_ERR, 949
ERR_MEM, 949
ERR_MEMMGR_FAIL, 949
ERR_NO_ACTIVE_CLUMP, 949
ERR_NO_CODE, 949
ERR_NO_PROG, 949
ERR_NON_FATAL, 950
ERR_RC_BAD_PACKET, 950
ERR_RC_FAILED, 950

ERR_RC_ILLEGAL_VAL, 950	HS_BAUD_1200, 955
ERR_RC_UNKNOWN_CMD, 950	HS_BAUD_14400, 956
ERR_SPOTCHECK_FAIL, 950	HS_BAUD_19200, 956
ERR_VER, 950	HS_BAUD_230400, 956
FALSE, 950	HS_BAUD_2400, 956
FileClose, 950	HS_BAUD_28800, 956
FileDelete, 950	HS_BAUD_3600, 956
FileFindFirst, 951	HS_BAUD_38400, 956
FileFindNext, 951	HS_BAUD_460800, 956
FileOpenAppend, 951	HS_BAUD_4800, 956
FileOpenRead, 951	HS_BAUD_57600, 956
FileOpenReadLinear, 951	HS_BAUD_7200, 957
FileOpenWrite, 951	HS_BAUD_76800, 957
FileOpenWriteLinear, 951	HS_BAUD_921600, 957
FileOpenWriteNonLinear, 951	HS_BAUD_9600, 957
FileRead, 951	HS_CMD_READY, 957
FileRename, 951	HS_CTRL_EXIT, 957
FileResize, 952	HS_CTRL_INIT, 957
FileResolveHandle, 952	HS_CTRL_UART, 957
FileSeek, 952	HS_DISABLE, 958
FileWrite, 952	HS_ENABLE, 958
FRAME_SELECT, 952	HS_INIT_RECEIVER, 958
FREQUENCY_MAX, 952	HS_INITIALISE, 958
FREQUENCY_MIN, 952	HS_MODE_10_STOP, 958
GetStartTick, 952	HS_MODE_15_STOP, 958
GL_CAMERA_DEPTH, 952	HS_MODE_20_STOP, 958
GL_CIRCLE, 952	HS_MODE_5_DATA, 958
GL_CIRCLE_SIZE, 953	HS_MODE_6_DATA, 958
GL_CULL_BACK, 953	HS_MODE_7_DATA, 959
GL_CULL_FRONT, 953	HS_MODE_7E1, 959
GL_CULL_MODE, 953	HS_MODE_8_DATA, 959
GL_CULL_NONE, 953	HS_MODE_8N1, 959
GL_LINE, 953	HS_MODE_E_PARITY, 959
GL_POINT, 953	HS_MODE_M_PARITY, 959
GL_POLYGON, 954	HS_MODE_N_PARITY, 959
GL_ROTATE_X, 954	HS_MODE_O_PARITY, 959
GL_ROTATE_Y, 954	HS_MODE_S_PARITY, 959
GL_ROTATE_Z, 954	HS_SEND_DATA, 960
GL_SCALE_X, 954	HS_UPDATE, 960
GL_SCALE_Y, 954	HT_ADDR_ACCEL, 960
GL_SCALE_Z, 955	HT_ADDR_COLOR, 960
GL_TRANSLATE_X, 955	HT_ADDR_COLOR2, 960
GL_TRANSLATE_Y, 955	HT_ADDR_COMPASS, 960
GL_TRANSLATE_Z, 955	HT_ADDR_IRLINK, 960
GL_ZOOM_FACTOR, 955	HT_ADDR_IRRECEIVER, 960
HS_BAUD_115200, 955	HT_ADDR_IRSEEKER, 960

HT_ADDR_IRSEEKER2, 960
HT_CH1_A, 961
HT_CH1_B, 961
HT_CH2_A, 961
HT_CH2_B, 961
HT_CH3_A, 961
HT_CH3_B, 961
HT_CH4_A, 961
HT_CH4_B, 961
HT_CMD_COLOR2_50HZ, 961
HT_CMD_COLOR2_60HZ, 962
HT_CMD_COLOR2_ACTIVE, 962
HT_CMD_COLOR2_BLCAL, 962
HT_CMD_COLOR2_FAR, 962
HT_CMD_COLOR2_LED_HI, 962
HT_CMD_COLOR2_LED_LOW,
 962
HT_CMD_COLOR2_NEAR, 962
HT_CMD_COLOR2_PASSIVE,
 962
HT_CMD_COLOR2_RAW, 962
HT_CMD_COLOR2_WBCAL, 963
HTIR2_MODE_1200, 963
HTIR2_MODE_600, 963
HTIR2_REG_AC01, 963
HTIR2_REG_AC02, 963
HTIR2_REG_AC03, 963
HTIR2_REG_AC04, 963
HTIR2_REG_AC05, 963
HTIR2_REG_ACDIR, 963
HTIR2_REG_DC01, 964
HTIR2_REG_DC02, 964
HTIR2_REG_DC03, 964
HTIR2_REG_DC04, 964
HTIR2_REG_DC05, 964
HTIR2_REG_DCAVG, 964
HTIR2_REG_DCDIR, 964
HTIR2_REG_MODE, 964
I2C_ADDR_DEFAULT, 964
I2C_REG_CMD, 965
I2C_REG_DEVICE_ID, 965
I2C_REG_VENDOR_ID, 965
I2C_REG_VERSION, 965
IN_1, 965
IN_2, 966
IN_3, 966
IN_4, 966
IN_MODE_ANGLESTEP, 966
IN_MODE_BOOLEAN, 966
IN_MODE_CELSIUS, 966
IN_MODE_FAHRENHEIT, 966
IN_MODE_MODEMASK, 966
IN_MODE_PCTFULLSCALE, 966
IN_MODE_PERIODCOUNTER,
 966
IN_MODE_RAW, 967
IN_MODE_SLOPEMASK, 967
IN_MODE_TRANSITIONCNT,
 967
IN_TYPE_ANGLE, 967
IN_TYPE_COLORBLUE, 967
IN_TYPE_COLOREXIT, 967
IN_TYPE_COLORFULL, 967
IN_TYPE_COLORGREEN, 967
IN_TYPE_COLORNONE, 967
IN_TYPE_COLORRED, 967
IN_TYPE_CUSTOM, 968
IN_TYPE_HISPEED, 968
IN_TYPE_LIGHT_ACTIVE, 968
IN_TYPE_LIGHT_INACTIVE, 968
IN_TYPE_LOWSPEED, 968
IN_TYPE_LOWSPEED_9V, 968
IN_TYPE_NO_SENSOR, 968
IN_TYPE_REFLECTION, 968
IN_TYPE_SOUND_DB, 968
IN_TYPE_SOUND_DBA, 968
IN_TYPE_SWITCH, 969
IN_TYPE_TEMPERATURE, 969
INPUT_BLACKCOLOR, 969
INPUT_BLANK, 969
INPUT_BLUE, 969
INPUT_BLUECOLOR, 969
INPUT_CAL_POINT_0, 969
INPUT_CAL_POINT_1, 969
INPUT_CAL_POINT_2, 969
INPUT_CUSTOM9V, 970
INPUT_CUSTOMACTIVE, 970
INPUT_CUSTOMINACTIVE, 970
INPUT_DIGI0, 970
INPUT_DIGI1, 970
INPUT_GREEN, 970
INPUT_GREENCOLOR, 970

INPUT_INVALID_DATA, 970
INPUT_NO_OF_COLORS, 970
INPUT_NO_OF_POINTS, 970
INPUT_RED, 971
INPUT_REDCOLOR, 971
INPUT_RESETCAL, 971
INPUT_RUNNINGCAL, 971
INPUT_SENSORCAL, 971
INPUT_SENSOROFF, 971
INPUT_STARTCAL, 971
INPUT_WHITECOLOR, 971
INPUT_YELLOWCOLOR, 971
InputMode, 972
InputModuleID, 972
InputModuleName, 972
InputOffsetADRaw, 972
InputOffsetColorADRaw, 972
InputOffsetColorCalibration, 972
InputOffsetColorCalibrationState,
 972
InputOffsetColorCalLimits, 972
InputOffsetColorSensorBoolean,
 972
InputOffsetColorSensorRaw, 973
InputOffsetColorSensorValue, 973
InputOffsetCustomActiveStatus, 973
InputOffsetCustomPctFullScale, 973
InputOffsetCustomZeroOffset, 973
InputOffsetDigiPinsDir, 973
InputOffsetDigiPinsIn, 973
InputOffsetDigiPinsOut, 973
InputOffsetInvalidData, 973
InputOffsetSensorBoolean, 974
InputOffsetSensorMode, 974
InputOffsetSensorRaw, 974
InputOffsetSensorType, 974
InputOffsetSensorValue, 974
INT_MAX, 974
INT_MIN, 974
INTF_BTOFF, 974
INTF_BTTON, 974
INTF_CONNECT, 975
INTF_CONNECTBYNAME, 975
INTF_CONNECTREQ, 975
INTF_DISCONNECT, 975
INTF_DISCONNECTALL, 975
INTF_EXTREAD, 975
INTF_FACTORYRESET, 975
INTF_OPENSTREAM, 975
INTF_PINREQ, 975
INTF_REMOVEDEVICE, 975
INTF_SEARCH, 976
INTF_SENDDATA, 976
INTF_SENDFILE, 976
INTF_SETBTNAME, 976
INTF_SETCMDMODE, 976
INTF_STOPSEARCH, 976
INTF_VISIBILITY, 976
InvalidData, 976
IOCTRL_BOOT, 976
IOCTRL_POWERDOWN, 976
IOCtrlModuleID, 977
IOCtrlModuleName, 977
IOCtrlOffsetPowerOn, 977
IOMapRead, 977
IOMapReadByID, 977
IOMapWrite, 977
IOMapWriteByID, 977
KeepAlive, 977
LCD_LINE1, 977
LCD_LINE2, 978
LCD_LINE3, 978
LCD_LINE4, 979
LCD_LINE5, 979
LCD_LINE6, 979
LCD_LINE7, 979
LCD_LINE8, 980
LDR_APPENDNOTPOSSIBLE,
 980
LDR_BTBUSY, 980
LDR_BTCONNECTFAIL, 980
LDR_BTTIMEOUT, 980
LDR_CMD_BOOTCMD, 980
LDR_CMD_BTFACTORYRESET,
 980
LDR_CMD_BTGETADR, 980
LDR_CMD_CLOSE, 981
LDR_CMD_-
 CLOSEMODHANDLE,
 981
LDR_CMD_CROPDATAFILE, 981
LDR_CMD_DELETE, 981

LDR_CMD_-
 DELETEUSERFLASH, 981
LDR_CMD_DEVICEINFO, 981
LDR_CMD_FINDFIRST, 981
LDR_CMD_-
 FINDFIRSTMODULE, 981
LDR_CMD_FINDNEXT, 981
LDR_CMD_-
 FINDNEXTMODULE, 981
LDR_CMD_IOMAPREAD, 982
LDR_CMD_IOMAPWRITE, 982
LDR_CMD_OPENAPPENDDATA,
 982
LDR_CMD_OPENREAD, 982
LDR_CMD_OPENREADLINEAR,
 982
LDR_CMD_OPENWRITE, 982
LDR_CMD_OPENWRITEDATA,
 982
LDR_CMD_-
 OPENWRITELINEAR, 982
LDR_CMD_POLLCMD, 982
LDR_CMD_POLLCMDLEN, 982
LDR_CMD_READ, 983
LDR_CMD_RENAMEFILE, 983
LDR_CMD_RESIZEDATAFILE,
 983
LDR_CMD_-
 SEEKFROMCURRENT,
 983
LDR_CMD_SEEKFROMEND, 983
LDR_CMD_SEEKFROMSTART,
 983
LDR_CMD_SETBRICKNAME,
 983
LDR_CMD_VERSIONS, 983
LDR_CMD_WRITE, 983
LDR_ENDOFFILE, 983
LDR_EOFEXPECTED, 984
LDR_FILEEXISTS, 984
LDR_FILEISBUSY, 984
LDR_FILEISFULL, 984
LDR_FILENOFOUND, 984
LDR_FILETX_CLOSEERROR,
 984
LDR_FILETX_DSTEXISTS, 985
LDR_FILETX_SRCMISSING, 985
LDR_FILETX_STREAMERROR,
 985
LDR_FILETX_TIMEOUT, 985
LDR_-
 HANDLEALREADYCLOSED,
 985
LDR_ILLEGALFILENAME, 985
LDR_ILLEGALHANDLE, 985
LDR_INPROGRESS, 985
LDR_INVALIDSEEK, 985
LDR_MODULENOTFOUND, 985
LDR_NOLINEARSPACE, 986
LDR_NMOREFILES, 986
LDR_NMOREHANDLES, 986
LDR_NOSPACE, 986
LDR_NOTLINEARFILE, 986
LDR_NOWRITEBUFFERS, 986
LDR_OUTOFCBOUNDARY, 986
LDR_REQPIN, 986
LDR_SUCCESS, 986
LDR_UNDEFINEDERROR, 987
LEGO_ADDR_EMETER, 987
LEGO_ADDR_TEMP, 987
LEGO_ADDR_US, 987
ListFiles, 987
LoaderExecuteFunction, 987
LoaderModuleID, 987
LoaderModuleName, 987
LoaderOffsetFreeUserFlash, 987
LoaderOffsetPFunc, 988
LONG_MAX, 988
LONG_MIN, 988
LOWSPEED_CH_NOT_READY,
 988
LOWSPEED_COMMUNICATING,
 988
LOWSPEED_DATA_RECEIVED,
 988
LOWSPEED_DONE, 988
LOWSPEED_ERROR, 988
LOWSPEED_IDLE, 988
LOWSPEED_INIT, 989
LOWSPEED_LOAD_BUFFER, 989
LOWSPEED_NO_ERROR, 989
LOWSPEED RECEIVING, 989

LOWSPEED_RX_ERROR, 989
LOWSPEED_TRANSMITTING,
 989
LOWSPEED_TX_ERROR, 989
LowSpeedModuleID, 989
LowSpeedModuleName, 989
LowSpeedOffsetChannelState, 989
LowSpeedOffsetErrorType, 990
LowSpeedOffsetInBufBuf, 990
LowSpeedOffsetInBufBytesToRx,
 990
LowSpeedOffsetInBufInPtr, 990
LowSpeedOffsetInBufOutPtr, 990
LowSpeedOffsetMode, 990
LowSpeedOffsetNoRestartOnRead,
 990
LowSpeedOffsetOutBufBuf, 990
LowSpeedOffsetOutBufBytesToRx,
 990
LowSpeedOffsetOutBufInPtr, 990
LowSpeedOffsetOutBufOutPtr, 991
LowSpeedOffsetSpeed, 991
LowSpeedOffsetState, 991
LR_COULD_NOT_SAVE, 991
LR_ENTRY_REMOVED, 991
LR_STORE_IS_FULL, 991
LR_SUCCESS, 991
LR_UNKNOWN_ADDR, 991
LSREAD_NO_RESTART_1, 991
LSREAD_NO_RESTART_2, 991
LSREAD_NO_RESTART_3, 992
LSREAD_NO_RESTART_4, 992
LSREAD_NO_RESTART_MASK,
 992
LSREAD_RESTART_ALL, 992
LSREAD_RESTART_NONE, 992
MAILBOX1, 992
MAILBOX10, 992
MAILBOX2, 992
MAILBOX3, 993
MAILBOX4, 993
MAILBOX5, 993
MAILBOX6, 993
MAILBOX7, 993
MAILBOX8, 993
MAILBOX9, 993
MAX_BT_MSG_SIZE, 993
MENUICON_CENTER, 993
MENUICON_LEFT, 993
MENUICON_RIGHT, 994
MENUICONS, 994
MENUTEXT, 994
MessageRead, 994
MessageWrite, 994
MIN_1, 994
MS_1, 994
MS_10, 994
MS_100, 994
MS_150, 995
MS_2, 995
MS_20, 995
MS_200, 995
MS_250, 995
MS_3, 995
MS_30, 995
MS_300, 996
MS_350, 996
MS_4, 996
MS_40, 996
MS_400, 996
MS_450, 996
MS_5, 996
MS_50, 996
MS_500, 997
MS_6, 997
MS_60, 997
MS_600, 997
MS_7, 997
MS_70, 997
MS_700, 997
MS_8, 997
MS_80, 997
MS_800, 998
MS_9, 998
MS_90, 998
MS_900, 998
MS_ADDR_ACCLNX, 998
MS_ADDR_CMPSNX, 998
MS_ADDR_DISTNX, 998
MS_ADDR_IVSENS, 999
MS_ADDR_LINELDR, 999
MS_ADDR_MTRMUX, 999

MS_ADDR_NRLINK, 999
MS_ADDR_NXTCAM, 999
MS_ADDR_NXTHID, 999
MS_ADDR_NXTMMX, 999
MS_ADDR_NXTSERVO, 1000
MS_ADDR_PFMATE, 1000
MS_ADDR_PSPNX, 1000
MS_ADDR_RTCLOCK, 1000
MS_ADDR_RXMUX, 1000
MS_CMD_ADPA_OFF, 1000
MS_CMD_ADPA_ON, 1000
MS_CMD_DEENERGIZED, 1000
MS_CMD_ENERGIZED, 1000
NA, 1001
NO_ERR, 1001
NO_OF_BTNS, 1001
NormalizedValue, 1001
NRLINK_CMD_2400, 1001
NRLINK_CMD_4800, 1001
NRLINK_CMD_FLUSH, 1002
NRLINK_CMD_IR_LONG, 1002
NRLINK_CMD_IR_SHORT, 1002
NRLINK_CMD_RUN_MACRO,
 1002
NRLINK_CMD_SET_PF, 1002
NRLINK_CMD_SET_RCX, 1002
NRLINK_CMD_SET_TRAIN,
 1002
NRLINK_CMD_TX_RAW, 1002
NRLINK_REG_BYTES, 1002
NRLINK_REG_DATA, 1002
NRLINK_REG_EEPROM, 1003
NULL, 1003
OPARR_MAX, 1003
OPARR_MEAN, 1003
OPARR_MIN, 1003
OPARR_SORT, 1003
OPARR_STD, 1003
OPARR_SUM, 1003
OPARR_SUMSQR, 1003
OUT_A, 1004
OUT_AB, 1004
OUT_ABC, 1004
OUT_AC, 1004
OUT_B, 1004
OUT_BC, 1005
OUT_C, 1005
OUT_MODE BRAKE, 1005
OUT_MODE_COAST, 1005
OUT_MODE_MOTORON, 1005
OUT_MODE_REGMETHOD, 1005
OUT_MODE_REGULATED, 1005
OUT_OPTION_HOLDATLIMIT,
 1005
OUT_OPTION_-
 RAMPDOWNLIMIT,
 1005
OUT_REGMODE_IDLE, 1006
OUT_REGMODE_SPEED, 1006
OUT_REGMODE_SYNC, 1006
OUT_RUNSTATE_HOLD, 1006
OUT_RUNSTATE_IDLE, 1006
OUT_RUNSTATE_RAMPDOWN,
 1006
OUT_RUNSTATE_RAMPUP, 1006
OUT_RUNSTATE_RUNNING,
 1007
OutputMode, 1007
OutputModuleID, 1007
OutputModuleName, 1007
OutputOffsetActualSpeed, 1007
OutputOffsetBlockTachoCount,
 1007
OutputOffsetFlags, 1007
OutputOffsetMode, 1008
OutputOffsetMotorRPM, 1008
OutputOffsetOptions, 1008
OutputOffsetOverloaded, 1008
OutputOffsetPwnFreq, 1008
OutputOffsetRegDParameter, 1008
OutputOffsetRegIParameter, 1008
OutputOffsetRegMode, 1008
OutputOffsetRegPParameter, 1008
OutputOffsetRotationCount, 1009
OutputOffsetRunState, 1009
OutputOffsetSpeed, 1009
OutputOffsetSyncTurnParameter,
 1009
OutputOffsetTachoCount, 1009
OutputOffsetTachoLimit, 1009
OutputOptions, 1009
Overload, 1009

PF_CHANNEL_1, 1010
PF_CHANNEL_2, 1010
PF_CHANNEL_3, 1010
PF_CHANNEL_4, 1010
PF_CMD_BRAKE, 1010
PF_CMD_FWD, 1010
PF_CMD_REV, 1011
PF_CMD_STOP, 1011
PF_CST_CLEAR1_CLEAR2, 1011
PF_CST_CLEAR1_SET2, 1011
PF_CST_DECREMENT_PWM,
 1011
PF_CST_FULL_FWD, 1011
PF_CST_FULL_REV, 1011
PF_CST_INCREMENT_PWM,
 1011
PF_CST_SET1_CLEAR2, 1012
PF_CST_SET1_SET2, 1012
PF_CST_TOGGLE_DIR, 1012
PF_FUNC_CLEAR, 1012
PF_FUNC_NOCHANGE, 1012
PF_FUNC_SET, 1012
PF_FUNC_TOGGLE, 1012
PF_MODE_COMBO_DIRECT,
 1012
PF_MODE_COMBO_PWM, 1013
PF_MODE_SINGLE_OUTPUT_-
 CST, 1013
PF_MODE_SINGLE_OUTPUT_-
 PWM, 1013
PF_MODE_SINGLE_PIN_CONT,
 1013
PF_MODE_SINGLE_PIN_TIME,
 1013
PF_MODE_TRAIN, 1013
PF_OUT_A, 1013
PF_OUT_B, 1013
PF_PIN_C1, 1013
PF_PIN_C2, 1014
PF_PWM_BRAKE, 1014
PF_PWM_FLOAT, 1014
PF_PWM_FWD1, 1014
PF_PWM_FWD2, 1014
PF_PWM_FWD3, 1014
PF_PWM_FWD4, 1014
PF_PWM_FWD5, 1014
PF_PWM_FWD6, 1015
PF_PWM_FWD7, 1015
PF_PWM_REV1, 1015
PF_PWM_REV2, 1015
PF_PWM_REV3, 1015
PF_PWM_REV4, 1015
PF_PWM_REV5, 1015
PF_PWM_REV6, 1015
PF_PWM_REV7, 1015
PI, 1016
PID_0, 1016
PID_1, 1016
PID_2, 1016
PID_3, 1016
PID_4, 1016
PID_5, 1016
PID_6, 1016
PID_7, 1016
Power, 1017
PROG_ABORT, 1017
PROG_ERROR, 1017
PROG_IDLE, 1017
PROG_OK, 1017
PROG_RESET, 1017
PROG_RUNNING, 1017
PSP_BTNSET1_DOWN, 1017
PSP_BTNSET1_L3, 1017
PSP_BTNSET1_LEFT, 1018
PSP_BTNSET1_R3, 1018
PSP_BTNSET1_RIGHT, 1018
PSP_BTNSET1_UP, 1018
PSP_BTNSET2_CIRCLE, 1018
PSP_BTNSET2_CROSS, 1018
PSP_BTNSET2_L1, 1018
PSP_BTNSET2_L2, 1018
PSP_BTNSET2_R1, 1018
PSP_BTNSET2_R2, 1018
PSP_BTNSET2_SQUARE, 1019
PSP_BTNSET2_TRIANGLE, 1019
PSP_CMD_ANALOG, 1019
PSP_CMD_DIGITAL, 1019
PSP_REG_BTNSET1, 1019
PSP_REG_BTNSET2, 1019
PSP_REG_XLEFT, 1019
PSP_REG_XRIGHT, 1019
PSP_REG_YLEFT, 1019

PSP_REG_YRIGHT, 1019
RADIANs_PER_DEGREE, 1020
RAND_MAX, 1020
RandomNumber, 1020
RawValue, 1020
RCX_AbsVarOp, 1020
RCX_AndVarOp, 1020
RCX_AutoOffOp, 1020
RCX_BatteryLevelOp, 1020
RCX_BatteryLevelSrc, 1020
RCX_BootModeOp, 1021
RCX_CalibrateEventOp, 1021
RCX_ClearAllEventsOp, 1021
RCX_ClearCounterOp, 1021
RCX_ClearMsgOp, 1021
RCX_ClearSensorOp, 1021
RCX_ClearSoundOp, 1021
RCX_ClearTimerOp, 1021
RCX_ClickCounterSrc, 1021
RCX_ConstantSrc, 1021
RCX_CounterSrc, 1022
RCX_DatalogOp, 1022
RCX_DatalogRawDirectSrc, 1022
RCX_DatalogRawIndirectSrc, 1022
RCX_DatalogSrcDirectSrc, 1022
RCX_DatalogSrcIndirectSrc, 1022
RCX_DatalogValueDirectSrc, 1022
RCX_DatalogValueIndirectSrc,
 1023
RCX_DecCounterOp, 1023
RCX_DeleteSubOp, 1023
RCX_DeleteSubsOp, 1023
RCX_DeleteTaskOp, 1023
RCX_DeleteTasksOp, 1023
RCX_DirectEventOp, 1023
RCX_DisplayOp, 1023
RCX_DivVarOp, 1023
RCX_DurationSrc, 1023
RCX_EventStateSrc, 1024
RCX_FirmwareVersionSrc, 1024
RCX_GlobalMotorStatusSrc, 1024
RCX_GOutputDirOp, 1024
RCX_GOutputModeOp, 1024
RCX_GOutputPowerOp, 1024
RCX_HysteresisSrc, 1024
RCX_IncCounterOp, 1024
RCX_IndirectVarSrc, 1024
RCX_InputBooleanSrc, 1024
RCX_InputModeOp, 1025
RCX_InputModeSrc, 1025
RCX_InputRawSrc, 1025
RCX_InputTypeOp, 1025
RCX_InputTypeSrc, 1025
RCX_InputValueSrc, 1025
RCX_IRModeOp, 1025
RCX_LightOp, 1025
RCX_LowerThresholdSrc, 1025
RCX_LSBlinkTimeOp, 1026
RCX_LSCalibrateOp, 1026
RCX_LSHysteresisOp, 1026
RCX_LSLowerThreshOp, 1026
RCX_LSupperThreshOp, 1026
RCX_MessageOp, 1026
RCX_MessageSrc, 1026
RCX_MulVarOp, 1026
RCX_MuteSoundOp, 1026
RCX_OnOffFloatOp, 1026
RCX_OrVarOp, 1027
RCX_OUT_A, 1027
RCX_OUT_AB, 1027
RCX_OUT_ABC, 1027
RCX_OUT_AC, 1027
RCX_OUT_B, 1027
RCX_OUT_BC, 1028
RCX_OUT_C, 1028
RCX_OUT_FLOAT, 1028
RCX_OUT_FULL, 1028
RCX_OUT_FWD, 1028
RCX_OUT_HALF, 1028
RCX_OUT_LOW, 1028
RCX_OUT_OFF, 1028
RCX_OUT_ON, 1029
RCX_OUT_REV, 1029
RCX_OUT_TOGGLE, 1029
RCX_OutputDirOp, 1029
RCX_OutputPowerOp, 1029
RCX_OutputStatusSrc, 1029
RCX_PBTurnOffOp, 1029
RCX_PingOp, 1029
RCX_PlaySoundOp, 1029
RCX_PlayToneOp, 1030
RCX_PlayToneVarOp, 1030

RCX_PollMemoryOp, 1030
RCX_PollOp, 1030
RCX_ProgramSlotSrc, 1030
RCX_RandomSrc, 1030
RCX_RemoteKeysReleased, 1030
RCX_RemoteOp, 1030
RCX_RemoteOutABackward, 1030
RCX_RemoteOutAForward, 1031
RCX_RemoteOutBBackward, 1031
RCX_RemoteOutBForward, 1031
RCX_RemoteOutCBackward, 1031
RCX_RemoteOutCForward, 1031
RCX_RemotePBMessagel, 1031
RCX_RemotePBMessagel2, 1031
RCX_RemotePBMessagel3, 1031
RCX_RemotePlayASound, 1031
RCX_RemoteSelProgram1, 1032
RCX_RemoteSelProgram2, 1032
RCX_RemoteSelProgram3, 1032
RCX_RemoteSelProgram4, 1032
RCX_RemoteSelProgram5, 1032
RCX_RemoteStopOutOff, 1032
RCX_ScoutCounterLimitSrc, 1032
RCX_ScoutEventFBSrc, 1032
RCX_ScoutLightParamsSrc, 1032
RCX_ScoutOp, 1032
RCX_ScoutRulesOp, 1033
RCX_ScoutRulesSrc, 1033
RCX_ScoutTimerLimitSrc, 1033
RCX_SelectProgramOp, 1033
RCX_SendUARTDataOp, 1033
RCX_SetCounterOp, 1033
RCX_SetDatalogOp, 1033
RCX_SetEventOp, 1033
RCX_SetFeedbackOp, 1033
RCX_SetPriorityOp, 1033
RCX_SetSourceValueOp, 1034
RCX_SetTimerLimitOp, 1034
RCX_SetVarOp, 1034
RCX_SetWatchOp, 1034
RCX_SgnVarOp, 1034
RCX_SoundOp, 1034
RCX_StartTaskOp, 1034
RCX_StopAllTasksOp, 1034
RCX_StopTaskOp, 1034
RCX_SubVarOp, 1034
RCX_SumVarOp, 1035
RCX_TaskEventsSrc, 1035
RCX_TenMSTimerSrc, 1035
RCX_TimerSrc, 1035
RCX_UARTSetupSrc, 1035
RCX_UnlockFirmOp, 1035
RCX_UnlockOp, 1035
RCX_UnmuteSoundOp, 1035
RCX_UploadDatalogOp, 1035
RCX_UpperThresholdSrc, 1035
RCX_VariableSrc, 1036
RCX_ViewSourceValOp, 1036
RCX_VLLOp, 1036
RCX_WatchSrc, 1036
ReadButton, 1036
ReadSemData, 1036
RegDValue, 1036
RegIValue, 1037
RegMode, 1037
RegPValue, 1037
RESET_ALL, 1037
RESET_BLOCK_COUNT, 1038
RESET_BLOCKANDTACHO,
 1038
RESET_COUNT, 1038
RESET_NONE, 1038
RESET_ROTATION_COUNT,
 1038
RICArg, 1038
RICImgPoint, 1039
RICImgRect, 1039
RICMapArg, 1039
RICMapElement, 1040
RICMapFunction, 1040
RICOpCircle, 1040
RICOpCopyBits, 1040
RICOpDescription, 1041
RICOpEllipse, 1041
RICOpLine, 1042
RICOpNumBox, 1042
RICOpPixel, 1042
RICOpPolygon, 1043
RICOpRect, 1043
RICOpSprite, 1043
RICOpVarMap, 1044
RICPolygonPoints, 1044

RICSpriteData, 1044
ROTATE_QUEUE, 1045
RotationCount, 1045
RunState, 1045
SAMPLERATE_DEFAULT, 1045
SAMPLERATE_MAX, 1045
SAMPLERATE_MIN, 1045
ScaledValue, 1046
SCHAR_MAX, 1046
SCHAR_MIN, 1046
SCOUT_FXR_ALARM, 1046
SCOUT_FXR_BUG, 1046
SCOUT_FXR_NONE, 1046
SCOUT_FXR_RANDOM, 1046
SCOUT_FXR_SCIENCE, 1046
SCOUT_LIGHT_OFF, 1046
SCOUT_LIGHT_ON, 1047
SCOUT_LR_AVOID, 1047
SCOUT_LR_IGNORE, 1047
SCOUT_LR_OFF_WHEN, 1047
SCOUT_LR_SEEK_DARK, 1047
SCOUT_LR_SEEK_LIGHT, 1047
SCOUT_LR_WAIT_FOR, 1047
SCOUT_MODE_POWER, 1047
SCOUT_MODE_STANDALONE,
 1048
SCOUT_MR_CIRCLE_LEFT, 1048
SCOUT_MR_CIRCLE_RIGHT,
 1048
SCOUT_MR_FORWARD, 1048
SCOUT_MR_LOOP_A, 1048
SCOUT_MR_LOOP_AB, 1048
SCOUT_MR_LOOP_B, 1048
SCOUT_MR_NO_MOTION, 1048
SCOUT_MR_ZIGZAG, 1049
SCOUT SNDSET_ALARM, 1049
SCOUT SNDSET_BASIC, 1049
SCOUT SNDSET_BUG, 1049
SCOUT SNDSET_NONE, 1049
SCOUT SNDSET_RANDOM,
 1049
SCOUT SNDSET_SCIENCE, 1049
SCOUT_SOUND_1_BLINK, 1049
SCOUT_SOUND_2_BLINK, 1049
SCOUT_SOUND_COUNTER1,
 1049
SCOUT_SOUND_COUNTER2,
 1050
SCOUT_SOUND_ENTER_-
 BRIGHT, 1050
SCOUT_SOUND_ENTER_DARK,
 1050
SCOUT_SOUND_ENTER_-
 NORMAL, 1050
SCOUT_SOUND_ENTERSA, 1050
SCOUT_SOUND_KEYERROR,
 1050
SCOUT_SOUND_MAIL_-
 RECEIVED, 1050
SCOUT_SOUND_NONE, 1050
SCOUT_SOUND_REMOTE, 1050
SCOUT_SOUND_SPECIAL1, 1050
SCOUT_SOUND_SPECIAL2, 1051
SCOUT_SOUND_SPECIAL3, 1051
SCOUT_SOUND_TIMER1, 1051
SCOUT_SOUND_TIMER2, 1051
SCOUT_SOUND_TIMER3, 1051
SCOUT_SOUND_TOUCH1_-
 PRES, 1051
SCOUT_SOUND_TOUCH1_REL,
 1051
SCOUT_SOUND_TOUCH2_-
 PRES, 1051
SCOUT_SOUND_TOUCH2_REL,
 1051
SCOUT_TGS_LONG, 1051
SCOUT_TGS_MEDIUM, 1052
SCOUT_TGS_SHORT, 1052
SCOUT_TR_AVOID, 1052
SCOUT_TR_IGNORE, 1052
SCOUT_TR_OFF_WHEN, 1052
SCOUT_TR_REVERSE, 1052
SCOUT_TR_WAIT_FOR, 1052
SCREEN_BACKGROUND, 1052
SCREEN_LARGE, 1053
SCREEN_MODE_CLEAR, 1053
SCREEN_MODE_RESTORE, 1053
SCREEN_SMALL, 1053
SCREENS, 1053
SEC_1, 1053
SEC_10, 1053
SEC_15, 1054

SEC_2, 1054
SEC_20, 1054
SEC_3, 1054
SEC_30, 1054
SEC_4, 1055
SEC_5, 1055
SEC_6, 1055
SEC_7, 1055
SEC_8, 1055
SEC_9, 1056
SetScreenMode, 1056
SetSleepTimeoutVal, 1056
SHRT_MAX, 1056
SHRT_MIN, 1056
SIZE_OF_BDADDR, 1056
SIZE_OF_BRICK_NAME, 1056
SIZE_OF_BT_CONNECT_-
TABLE, 1056
SIZE_OF_BT_DEVICE_TABLE,
1056
SIZE_OF_BT_NAME, 1057
SIZE_OF_BT_PINCODE, 1057
SIZE_OF_BTBUF, 1057
SIZE_OF_CLASS_OF_DEVICE,
1057
SIZE_OF_HSBUF, 1057
SIZE_OF_USBBUF, 1057
SIZE_OF_USBDATA, 1057
SOUND_CLICK, 1057
SOUND_DOUBLE_BEEP, 1057
SOUND_DOWN, 1057
SOUND_FAST_UP, 1058
SOUND_FLAGS_IDLE, 1058
SOUND_FLAGS_RUNNING, 1058
SOUND_FLAGS_UPDATE, 1058
SOUND_LOW_BEEP, 1058
SOUND_MODE_LOOP, 1058
SOUND_MODE_ONCE, 1059
SOUND_MODE_TONE, 1059
SOUND_STATE_FILE, 1059
SOUND_STATE_IDLE, 1059
SOUND_STATE_STOP, 1059
SOUND_STATE_TONE, 1059
SOUND_UP, 1059
SoundGetState, 1060
SoundModuleID, 1060
SoundModuleName, 1060
SoundOffsetDuration, 1060
SoundOffsetFlags, 1060
SoundOffsetFreq, 1060
SoundOffsetMode, 1060
SoundOffsetSampleRate, 1061
SoundOffsetSoundFilename, 1061
SoundOffsetState, 1061
SoundOffsetVolume, 1061
SoundPlayFile, 1061
SoundPlayTone, 1061
SoundSetState, 1061
SPECIALS, 1061
STAT_COMM_PENDING, 1061
STAT_MSG_EMPTY_MAILBOX,
1062
STATUSICON_BATTERY, 1062
STATUSICON_BLUETOOTH,
1062
STATUSICON_USB, 1062
STATUSICON_VM, 1062
STATUSICONS, 1062
STATUSTEXT, 1062
STEPICON_1, 1062
STEPICON_2, 1062
STEPICON_3, 1062
STEPICON_4, 1063
STEPICON_5, 1063
STEPICONS, 1063
STEPLINE, 1063
STOP_REQ, 1063
TachoCount, 1063
TachoLimit, 1063
TEMP_FQ_1, 1064
TEMP_FQ_2, 1064
TEMP_FQ_4, 1064
TEMP_FQ_6, 1064
TEMP_OS_ONESHOT, 1064
TEMP_POL_HIGH, 1064
TEMP_POL_LOW, 1064
TEMP_REG_CONFIG, 1064
TEMP_REG_TEMP, 1064
TEMP_REG_THIGH, 1065
TEMP_REG_TLOW, 1065
TEMP_RES_10BIT, 1065
TEMP_RES_11BIT, 1065

TEMP_RES_12BIT, 1065
TEMP_RES_9BIT, 1065
TEMP_SD_CONTINUOUS, 1065
TEMP_SD_SHUTDOWN, 1065
TEMP_TM_COMPARATOR, 1065
TEMP_TM_INTERRUPT, 1065
TEXTLINE_1, 1066
TEXTLINE_2, 1066
TEXTLINE_3, 1066
TEXTLINE_4, 1066
TEXTLINE_5, 1066
TEXTLINE_6, 1066
TEXTLINE_7, 1066
TEXTLINE_8, 1066
TEXTLINES, 1066
TIMES_UP, 1067
TONE_A3, 1067
TONE_A4, 1067
TONE_A5, 1067
TONE_A6, 1067
TONE_A7, 1067
TONE_AS3, 1067
TONE_AS4, 1067
TONE_AS5, 1067
TONE_AS6, 1068
TONE_AS7, 1068
TONE_B3, 1068
TONE_B4, 1068
TONE_B5, 1068
TONE_B6, 1068
TONE_B7, 1068
TONE_C4, 1068
TONE_C5, 1068
TONE_C6, 1069
TONE_C7, 1069
TONE_CS4, 1069
TONE_CS5, 1069
TONE_CS6, 1069
TONE_CS7, 1069
TONE_D4, 1069
TONE_D5, 1069
TONE_D6, 1070
TONE_D7, 1070
TONE_DS4, 1070
TONE_DS5, 1070
TONE_DS6, 1070
TONE_DS7, 1070
TONE_E4, 1070
TONE_E5, 1070
TONE_E6, 1071
TONE_E7, 1071
TONE_F4, 1071
TONE_F5, 1071
TONE_F6, 1071
TONE_F7, 1071
TONE_FS4, 1071
TONE_FS5, 1071
TONE_FS6, 1071
TONE_FS7, 1071
TONE_G4, 1072
TONE_G5, 1072
TONE_G6, 1072
TONE_G7, 1072
TONE_GS4, 1072
TONE_GS5, 1072
TONE_GS6, 1072
TONE_GS7, 1072
TOPLINE, 1073
TRAIN_CHANNEL_1, 1073
TRAIN_CHANNEL_2, 1073
TRAIN_CHANNEL_3, 1073
TRAIN_CHANNEL_ALL, 1073
TRAIN_FUNC_DECR_SPEED,
 1073
TRAIN_FUNC_INCR_SPEED,
 1073
TRAIN_FUNC_STOP, 1073
TRAIN_FUNC_TOGGLE_LIGHT,
 1074
TRUE, 1074
TurnRatio, 1074
Type, 1074
UCHAR_MAX, 1074
UF_PENDING_UPDATES, 1075
UF_UPDATE_MODE, 1075
UF_UPDATE_PID_VALUES, 1075
UF_UPDATE_RESET_BLOCK_-
 COUNT, 1075
UF_UPDATE_RESET_COUNT,
 1075
UF_UPDATE_RESET_-
 ROTATION_COUNT, 1075

UF_UPDATE_SPEED, 1075
UF_UPDATE_TACHO_LIMIT,
 1075
UI_BT_CONNECT_REQUEST,
 1075
UI_BT_ERROR_ATTENTION,
 1075
UI_BT_PIN_REQUEST, 1076
UI_BT_STATE_CONNECTED,
 1076
UI_BT_STATE_OFF, 1076
UI_BT_STATE_VISIBLE, 1076
UI_BUTTON_ENTER, 1076
UI_BUTTON_EXIT, 1076
UI_BUTTON_LEFT, 1076
UI_BUTTON_NONE, 1076
UI_BUTTON_RIGHT, 1077
UI_FLAGS_BUSY, 1077
UI_FLAGS_DISABLE_EXIT, 1077
UI_FLAGS_DISABLE_LEFT_-
 RIGHT_ENTER, 1077
UI_FLAGS_ENABLE_STATUS_-
 UPDATE, 1077
UI_FLAGS_EXECUTE_LMS_-
 FILE, 1077
UI_FLAGS_REDRAW_STATUS,
 1077
UI_FLAGS_RESET_SLEEP_-
 TIMER, 1077
UI_FLAGS_UPDATE, 1077
UI_STATE_BT_ERROR, 1078
UI_STATE_CONNECT_-
 REQUEST, 1078
UI_STATE_DRAW_MENU, 1078
UI_STATE_ENTER_PRESSED,
 1078
UI_STATE_EXECUTE_FILE, 1078
UI_STATE_EXECUTING_FILE,
 1078
UI_STATE_EXIT_PRESSED, 1078
UI_STATE_INIT_DISPLAY, 1078
UI_STATE_INIT_INTRO, 1078
UI_STATE_INIT_LOW_-
 BATTERY, 1078
UI_STATE_INIT_MENU, 1079
UI_STATE_INIT_WAIT, 1079
UI_STATE_LEFT_PRESSED, 1079
UI_STATE_LOW_BATTERY, 1079
UI_STATE_NEXT_MENU, 1079
UI_STATE_RIGHT_PRESSED,
 1079
UI_STATE_TEST_BUTTONS,
 1079
UI_VM_IDLE, 1079
UI_VM_RESET1, 1079
UI_VM_RESET2, 1080
UI_VM_RUN_FREE, 1080
UI_VM_RUN_PAUSE, 1080
UI_VM_RUN_SINGLE, 1080
UIModuleID, 1080
UIModuleName, 1080
UINT_MAX, 1080
UIOffsetAbortFlag, 1080
UIOffsetBatteryState, 1080
UIOffsetBatteryVoltage, 1081
UIOffsetBluetoothState, 1081
UIOffsetButton, 1081
UIOffsetError, 1081
UIOffsetFlags, 1081
UIOffsetForceOff, 1081
UIOffsetLMSfilename, 1081
UIOffsetOBPPointer, 1081
UIOffsetPMenu, 1081
UIOffsetRechargeable, 1081
UIOffsetRunState, 1082
UIOffsetSleepTimeout, 1082
UIOffsetSleepTimer, 1082
UIOffsetState, 1082
UIOffsetUsbState, 1082
UIOffsetVolume, 1082
ULONG_MAX, 1082
UpdateCalibCacheInfo, 1082
UpdateFlags, 1082
US_CMD_CONTINUOUS, 1083
US_CMD_EVENTCAPTURE,
 1083
US_CMD_OFF, 1083
US_CMD_SINGLESHT, 1083
US_CMD_WARMRESET, 1083
US_REG_ACTUAL_ZERO, 1083
US_REG_CM_INTERVAL, 1083

US_REG_FACTORY_ACTUAL_-
ZERO, 1083
US_REG_FACTORY_SCALE_-
DIVISOR, 1084
US_REG_FACTORY_SCALE_-
FACTOR, 1084
US_REG_MEASUREMENT_-
UNITS, 1084
US_REG_SCALE_DIVISOR, 1084
US_REG_SCALE_FACTOR, 1084
USB_CMD_READY, 1084
USB_PROTOCOL_OVERHEAD,
1084
USHRT_MAX, 1084
WriteSemData, 1084

NBCInputPortConstants
IN_1, 625
IN_2, 625
IN_3, 626
IN_4, 626

NBCSensorModeConstants
IN_MODE_ANGLESTEP, 629
IN_MODE_BOOLEAN, 629
IN_MODE_CELSIUS, 630
IN_MODE_FAHRENHEIT, 630
IN_MODE_MODEMASK, 630
IN_MODE_PCTFULLSCALE, 630
IN_MODE_PERIODCOUNTER,
630
IN_MODE_RAW, 630
IN_MODE_SLOPEMASK, 630
IN_MODE_TRANSITIONCNT,
630

NBCSensorTypeConstants
IN_TYPE_ANGLE, 627
IN_TYPE_COLORBLUE, 627
IN_TYPE_COLOREXIT, 627
IN_TYPE_COLORFULL, 627
IN_TYPE_COLORGREEN, 627
IN_TYPE_COLORNONE, 627
IN_TYPE_COLORRED, 627
IN_TYPE_CUSTOM, 627
IN_TYPE_HISPEED, 628
IN_TYPE_LIGHT_ACTIVE, 628
IN_TYPE_LIGHT_INACTIVE, 628
IN_TYPE_LOWSPEED, 628

IN_TYPE_LOWSPEED_9V, 628
IN_TYPE_NO_SENSOR, 628
IN_TYPE_REFLECTION, 628
IN_TYPE_SOUND_DB, 628
IN_TYPE_SOUND_DBA, 628
IN_TYPE_SWITCH, 628
IN_TYPE_TEMPERATURE, 629

NewFilename
FileRenameType, 845
NewSize
FileResizeType, 846
NewVal
WriteSemDataType, 881

NO_ERR
CommandModuleConstants, 51
NBCCommon.h, 1001

NO_OF_BTNS
ButtonNameConstants, 611
NBCCommon.h, 1001

NoRestartOnRead
CommLSWriteExType, 814

NormalizedArray
ColorSensorReadType, 799

NormalizedValue
InputFieldConstants, 631
NBCCommon.h, 1001

NRLLink2400
MindSensorsAPI, 166
NXCDefs.h, 1373

NRLLink4800
MindSensorsAPI, 167
NXCDefs.h, 1373

NRLINK_CMD_2400
MSNRLLink, 786
NBCCommon.h, 1001

NRLINK_CMD_4800
MSNRLLink, 786
NBCCommon.h, 1001

NRLINK_CMD_FLUSH
MSNRLLink, 786
NBCCommon.h, 1002

NRLINK_CMD_IR_LONG
MSNRLLink, 786
NBCCommon.h, 1002

NRLINK_CMD_IR_SHORT
MSNRLLink, 786

NBCCCommon.h, 1002
NRLINK_CMD_RUN_MACRO
 MSNRLink, 786
 NBCCCommon.h, 1002
NRLINK_CMD_SET_PF
 MSNRLink, 786
 NBCCCommon.h, 1002
NRLINK_CMD_SET_RCX
 MSNRLink, 786
 NBCCCommon.h, 1002
NRLINK_CMD_SET_TRAIN
 MSNRLink, 787
 NBCCCommon.h, 1002
NRLINK_CMD_TX_RAW
 MSNRLink, 787
 NBCCCommon.h, 1002
NRLINK_REG_BYTES
 MSNRLink, 787
 NBCCCommon.h, 1002
NRLINK_REG_DATA
 MSNRLink, 787
 NBCCCommon.h, 1002
NRLINK_REG_EEPROM
 MSNRLink, 787
 NBCCCommon.h, 1003
NRLinkFlush
 MindSensorsAPI, 167
 NXCDefs.h, 1373
NRLinkIRLong
 MindSensorsAPI, 168
 NXCDefs.h, 1374
NRLinkIRShort
 MindSensorsAPI, 168
 NXCDefs.h, 1374
NRLinkSetPF
 MindSensorsAPI, 169
 NXCDefs.h, 1375
NRLinkSetRCX
 MindSensorsAPI, 169
 NXCDefs.h, 1375
NRLinkSetTrain
 MindSensorsAPI, 169
 NXCDefs.h, 1376
NRLinkStatus
 MindSensorsAPI, 170
 NXCDefs.h, 1376
NRLinkTxRaw
 MindSensorsAPI, 170
 NXCDefs.h, 1376
NULL
 LoaderModuleConstants, 586
 NBCCCommon.h, 1003
NumOut
 DisplayModuleFunctions, 284
 NXCDefs.h, 1377
NumToStr
 cstringAPI, 536
 NXCDefs.h, 1378
NXCAPIDocs.h, 1085
NXCDefs.h, 1085
 _SENSOR_CFG, 1160
 abort, 1183
 AbortFlag, 1183
 abs, 1183
 ACCLNxCalibrateX, 1184
 ACCLNxCalibrateXEnd, 1184
 ACCLNxCalibrateY, 1185
 ACCLNxCalibrateYEnd, 1185
 ACCLNxCalibrateZ, 1186
 ACCLNxCalibrateZEnd, 1186
 ACCLNxResetCalibration, 1186
 ACCLNxSensitivity, 1187
 ACCLNxXOffset, 1187
 ACCLNxXRange, 1188
 ACCLNxYOffset, 1188
 ACCLNxYRange, 1189
 ACCLNxZOffset, 1189
 ACCLNxZRange, 1189
 Acos, 1160
 acos, 1190
 AcosD, 1161
 acosd, 1190
 Acquire, 1191
 addressOf, 1191
 addressOfEx, 1192
 ArrayBuild, 1192
 ArrayInit, 1193
 ArrayLen, 1193
 ArrayMax, 1194
 ArrayMean, 1194
 ArrayMin, 1195
 ArrayOp, 1195

ArraySort, 1196
ArrayStd, 1196
ArraySubset, 1197
ArraySum, 1197
ArraySumSqr, 1198
Asin, 1161
asin, 1198
AsinD, 1162
asind, 1199
Atan, 1162
atan, 1199
Atan2, 1162
atan2, 1200
Atan2D, 1163
atan2d, 1200
AtanD, 1163
atand, 1201
atof, 1201
atoi, 1202
atol, 1203
BatteryLevel, 1203
BatteryState, 1204
bcd2dec, 1204
BluetoothState, 1204
BluetoothStatus, 1205
BluetoothWrite, 1205
BrickDataBluecoreVersion, 1206
BrickDataBtHardwareStatus, 1206
BrickDataBtStateStatus, 1206
BrickDataName, 1207
BrickDataTimeoutValue, 1207
BTConnectionClass, 1207
BTConnectionHandleNum, 1208
BTConnectionLinkQuality, 1208
BTConnectionName, 1208
BTConnectionPinCode, 1209
BTConnectionStreamStatus, 1209
BTDeviceClass, 1210
BTDeviceCount, 1210
BTDeviceName, 1210
BTDeviceNameCount, 1211
BTDeviceStatus, 1211
BTInputBufferInPtr, 1211
BTInputBufferOutPtr, 1212
BTOutputBufferInPtr, 1212
BTOutputBufferOutPtr, 1212
ButtonCount, 1213
ButtonLongPressCount, 1213
ButtonLongReleaseCount, 1214
ButtonPressCount, 1214
ButtonPressed, 1214
ButtonReleaseCount, 1215
ButtonShortReleaseCount, 1215
ButtonState, 1216
ByteArrayToStr, 1216
ByteArrayToStrEx, 1216
Ceil, 1164
ceil, 1217
CircleOut, 1217
ClearLine, 1218
ClearScreen, 1218
ClearSensor, 1219
CloseFile, 1219
Coast, 1219
CoastEx, 1220
ColorADRaw, 1220
ColorBoolean, 1221
ColorCalibration, 1221
ColorCalibrationState, 1222
ColorCallLimits, 1222
ColorSensorRaw, 1223
ColorSensorValue, 1223
CommandFlags, 1224
Copy, 1224
Cos, 1164
cos, 1225
CosD, 1164
cosd, 1225
Cosh, 1165
cosh, 1226
CoshD, 1165
coshd, 1226
CreateFile, 1227
CreateFileLinear, 1227
CreateFileNonLinear, 1228
CurrentTick, 1228
CustomSensorActiveStatus, 1229
CustomSensorPercentFullScale, 1229
CustomSensorZeroOffset, 1229
DeleteFile, 1230
DisplayContrast, 1230

DisplayDisplay, 1231
DisplayEraseMask, 1231
DisplayFlags, 1231
DisplayFont, 1232
DisplayTextLinesCenterFlags, 1232
DisplayUpdateMask, 1232
DISTNxDistance, 1233
DISTNxGP2D12, 1233
DISTNxGP2D120, 1233
DISTNxGP2YA02, 1234
DISTNxGP2YA21, 1234
DISTNxMaxDistance, 1235
DISTNxMinDistance, 1235
DISTNxModuleType, 1236
DISTNxNumPoints, 1236
DISTNxVoltage, 1237
div, 1237
EllipseOut, 1238
ExitTo, 1238
Exp, 1166
exp, 1239
fclose, 1239
feof, 1240
fflush, 1240
fgetc, 1240
fgets, 1241
FindFirstFile, 1241
FindNextFile, 1242
FirstTick, 1242
Flatten, 1243
FlattenVar, 1243
Float, 1244
Floor, 1166
floor, 1244
Follows, 1244
FontNumOut, 1245
FontTextOut, 1246
fopen, 1246
ForceOff, 1247
FormatNum, 1247
fprintf, 1248
fputc, 1248
fputs, 1249
Frac, 1166
frac, 1249
FreeMemory, 1250
fseek, 1250
ftell, 1251
GetBrickDataAddress, 1251
GetBTConnectionAddress, 1251
GetBTDeviceAddress, 1252
GetBTInputBuffer, 1252
GetBTOutputBuffer, 1253
GetButtonModuleValue, 1253
getc, 1167
getchar, 1253
GetCommandModuleBytes, 1254
GetCommandModuleValue, 1254
GetCommModuleBytes, 1255
GetCommModuleValue, 1255
GetDisplayModuleBytes, 1255
GetDisplayModuleValue, 1256
GetDisplayNormal, 1256
GetDisplayPopup, 1257
GetHSInputBuffer, 1257
GetHSOutputBuffer, 1258
GetInput, 1258
GetInputModuleValue, 1258
GetIOMapBytes, 1259
GetIOMapBytesByID, 1259
GetIOMapValue, 1260
GetIOMapValueByID, 1260
GetLoaderModuleValue, 1261
GetLowSpeedModuleBytes, 1261
GetLowSpeedModuleValue, 1261
GetLSInputBuffer, 1262
GetLSOutputBuffer, 1262
GetOutput, 1263
GetOutputModuleValue, 1263
GetSoundModuleValue, 1263
GetUIModuleValue, 1264
GetUSBInputBuffer, 1264
GetUSBOutputBuffer, 1265
GetUSBPollBuffer, 1265
glAddToAngleX, 1265
glAddToAngleY, 1266
glAddToAngleZ, 1266
glAddVertex, 1266
glBegin, 1267
glBeginObject, 1267
glBeginRender, 1267
glBox, 1267

glCallObject, 1268
glCos32768, 1268
glCube, 1268
glEnd, 1269
glEndObject, 1269
glFinishRender, 1269
glInit, 1269
glObjectAction, 1270
glPyramid, 1270
glSet, 1270
glSetAngleX, 1271
glSetAngleY, 1271
glSetAngleZ, 1271
glSin32768, 1272
GraphicArrayOut, 1272
GraphicArrayOutEx, 1272
GraphicOut, 1273
GraphicOutEx, 1274
HSFlags, 1274
HSInputBufferInPtr, 1275
HSInputBufferOutPtr, 1275
HSMode, 1275
HSOutputBufferInPtr, 1276
HSOutputBufferOutPtr, 1276
HSSpeed, 1276
HSState, 1277
HTIRTrain, 1277
HTPFComboDirect, 1278
HTPFComboPWM, 1278
HTPFRawOutput, 1279
HTPFRepeat, 1279
HTPFSingleOutputCST, 1280
HTPFSingleOutputPWM, 1281
HTPFSinglePin, 1281
HTPFTTrain, 1282
HTRCXAddToDatalog, 1283
HTRCXBatteryLevel, 1283
HTRCXClearAllEvents, 1283
HTRCXClearCounter, 1283
HTRCXClearMsg, 1284
HTRCXClearSensor, 1284
HTRCXClearSound, 1284
HTRCXClearTimer, 1284
HTRCXCreateDatalog, 1285
HTRCXDecCounter, 1285
HTRCXDeleteSub, 1285
HTRCXDeleteSubs, 1286
HTRCXDeleteTask, 1286
HTRCXDeleteTasks, 1286
HTRCXDisableOutput, 1286
HTRCXEnableOutput, 1287
HTRCXEvent, 1287
HTRCXFloat, 1287
HTRCXFwd, 1288
HTRCXIncCounter, 1288
HTRCXInvertOutput, 1288
HTRCXMuteSound, 1289
HTRCXObvertOutput, 1289
HTRCXOff, 1289
HTRCXOn, 1290
HTRCXOnFor, 1290
HTRCXOnFwd, 1290
HTRCXOnRev, 1291
HTRCXPBTurnOff, 1291
HTRCXPing, 1291
HTRCXPlaySound, 1291
HTRCXPlayTone, 1292
HTRCXPlayToneVar, 1292
HTRCXPoll, 1292
HTRCXPollMemory, 1293
HTRCXRemote, 1293
HTRCXRev, 1293
HTRCXSelectDisplay, 1294
HTRCXSelectProgram, 1294
HTRCXSendSerial, 1294
HTRCXSetDirection, 1295
HTRCXSetEvent, 1295
HTRCXSetGlobalDirection, 1296
HTRCXSetGlobalOutput, 1296
HTRCXSetIRLinkPort, 1296
HTRCXSetMaxPower, 1297
HTRCXSetMessage, 1297
HTRCXSetOutput, 1297
HTRCXSetPower, 1298
HTRCXSetPriority, 1298
HTRCXSetSensorMode, 1298
HTRCXSetSensorType, 1299
HTRCXSetSleepTime, 1299
HTRCXSetTxPower, 1299
HTRCXSetWatch, 1300
HTRCXStartTask, 1300
HTRCXStopAllTasks, 1300

HTRCXStopTask, 1300
HTRCXToggle, 1301
HTRCXUnmuteSound, 1301
HTScoutCalibrateSensor, 1301
HTScoutMuteSound, 1301
HTScoutSelectSounds, 1302
HTScoutSendVLL, 1302
HTScoutSetEventFeedback, 1302
HTScoutSetLight, 1303
HTScoutSetScoutMode, 1303
HTScoutSetSensorClickTime, 1303
HTScoutSetSensorHysteresis, 1304
HTScoutSetSensorLowerLimit,
 1304
HTScoutSetSensorUpperLimit,
 1304
HTScoutUnmuteSound, 1305
I2CBytes, 1305
I2CBytesReady, 1306
I2CCheckStatus, 1307
I2DDeviceId, 1307
I2CDeviceInfo, 1308
I2CRead, 1308
I2CSendCommand, 1309
I2CStatus, 1310
I2CVendorId, 1310
I2CVersion, 1311
I2CWrite, 1311
isalnum, 1312
isalpha, 1312
iscntrl, 1313
isdigit, 1313
isgraph, 1314
islower, 1314
isNaN, 1314
isprint, 1315
ispunct, 1315
isspace, 1316
isupper, 1316
isxdigit, 1316
labs, 1317
ldiv, 1317
LeftStr, 1318
LineOut, 1318
Log, 1167
log, 1319
Log10, 1168
log10, 1319
LongAbort, 1320
LowspeedBytesReady, 1320
LowspeedCheckStatus, 1321
LowspeedRead, 1322
LowspeedStatus, 1322
LowspeedWrite, 1323
LSChannelState, 1324
LSErrorType, 1324
LSInputBufferBytesToRx, 1325
LSInputBufferInPtr, 1325
LSInputBufferOutPtr, 1325
LSMode, 1326
LSNoRestartOnRead, 1326
LSOutputBufferBytesToRx, 1326
LSOutputBufferInPtr, 1327
LSOutputBufferOutPtr, 1327
LSSpeed, 1328
LSState, 1328
memcmp, 1328
memcpy, 1329
memmove, 1329
MidStr, 1329
MotorActualSpeed, 1330
MotorBlockTachoCount, 1330
MotorMode, 1331
MotorOverload, 1331
MotorPower, 1331
MotorPwnFreq, 1331
MotorRegDValue, 1332
MotorRegIValue, 1332
MotorRegPValue, 1332
MotorRegulation, 1333
MotorRotationCount, 1333
MotorRunState, 1333
MotorTachoCount, 1334
MotorTachoLimit, 1334
MotorTurnRatio, 1334
MSADPAOff, 1335
MSADPAOn, 1335
MSDeenergize, 1336
MSEnergize, 1336
MSIRTrain, 1336
MSPFComboDirect, 1337
MSPFComboPWM, 1338

MSPFRawOutput, 1338
MSPFRepeat, 1339
MSPFSingleOutputCST, 1340
MSPFSingleOutputPWM, 1340
MSPFSinglePin, 1341
MSPFTrain, 1342
MSRCXAbsVar, 1342
MSRCXAddToDatalog, 1343
MSRCXAndVar, 1343
MSRCXBatteryLevel, 1344
MSRCXBoot, 1344
MSRCXCalibrateEvent, 1344
MSRCXClearAllEvents, 1344
MSRCXClearCounter, 1345
MSRCXClearMsg, 1345
MSRCXClearSensor, 1345
MSRCXClearSound, 1345
MSRCXClearTimer, 1346
MSRCXCreateDatalog, 1346
MSRCXDecCounter, 1346
MSRCXDeleteSub, 1347
MSRCXDeleteSubs, 1347
MSRCXDeleteTask, 1347
MSRCXDeleteTasks, 1347
MSRCXDisableOutput, 1348
MSRCXDivVar, 1348
MSRCXEnableOutput, 1348
MSRCXEvent, 1349
MSRCXFloat, 1349
MSRCXFwd, 1349
MSRCXIncCounter, 1350
MSRCXIvertOutput, 1350
MSRCXMulVar, 1350
MSRCXMuteSound, 1351
MSRCXObvertOutput, 1351
MSRCXOff, 1351
MSRCXOn, 1351
MSRCXOnFor, 1352
MSRCXOnFwd, 1352
MSRCXOnRev, 1352
MSRCXOrVar, 1353
MSRCXPBTurnOff, 1353
MSRCXPing, 1353
MSRCXPlaySound, 1354
MSRCXPlayTone, 1354
MSRCXPlayToneVar, 1354
MSRCXPoll, 1355
MSRCXPollMemory, 1355
MSRCXRemote, 1355
MSRCXReset, 1356
MSRCXRev, 1356
MSRCXSelectDisplay, 1356
MSRCXSelectProgram, 1357
MSRCXSendSerial, 1357
MSRCXSet, 1357
MSRCXSetDirection, 1358
MSRCXSetEvent, 1358
MSRCXSetGlobalDirection, 1358
MSRCXSetGlobalOutput, 1359
MSRCXSetMaxPower, 1359
MSRCXSetMessage, 1360
MSRCXSetNRLinkPort, 1360
MSRCXSetOutput, 1360
MSRCXSetPower, 1361
MSRCXSetPriority, 1361
MSRCXSetSensorMode, 1361
MSRCXSetSensorType, 1362
MSRCXSetSleepTime, 1362
MSRCXSetTxPower, 1362
MSRCXSetUserDisplay, 1363
MSRCXSetVar, 1363
MSRCXSetWatch, 1363
MSRCXSgnVar, 1364
MSRCXStartTask, 1364
MSRCXStopAllTasks, 1364
MSRCXStopTask, 1365
MSRCXSubVar, 1365
MSRCXSumVar, 1365
MSRCXToggle, 1366
MSRCXUnlock, 1366
MSRCXUnmuteSound, 1366
MSReadValue, 1366
MSScoutCalibrateSensor, 1367
MSScoutMuteSound, 1367
MSScoutSelectSounds, 1367
MSScoutSendVLL, 1368
MSScoutSetCounterLimit, 1368
MSScoutSetEventFeedback, 1368
MSScoutSetLight, 1369
MSScoutSetScoutMode, 1369
MSScoutSetScoutRules, 1369
MSScoutSetSensorClickTime, 1370

MSScoutSetSensorHysteresis, 1370
MSScoutSetSensorLowerLimit,
 1371
MSScoutSetSensorUpperLimit,
 1371
MSScoutSetTimerLimit, 1371
MSScoutUnmuteSound, 1372
MulDiv32, 1168
muldiv32, 1372
NRLink2400, 1373
NRLink4800, 1373
NRLinkFlush, 1373
NRLinkIRLong, 1374
NRLinkIRShort, 1374
NRLinkSetPF, 1375
NRLinkSetRCX, 1375
NRLinkSetTrain, 1376
NRLinkStatus, 1376
NRLinkTxRaw, 1376
NumOut, 1377
NumToStr, 1378
Off, 1378
OffEx, 1379
OnBrickProgramPointer, 1379
OnFwd, 1379
OnFwdEx, 1380
OnFwdReg, 1380
OnFwdRegEx, 1381
OnFwdRegExPID, 1381
OnFwdRegPID, 1382
OnFwdSync, 1382
OnFwdSyncEx, 1383
OnFwdSyncExPID, 1383
OnFwdSyncPID, 1384
OnRev, 1385
OnRevEx, 1385
OnRevReg, 1386
OnRevRegEx, 1386
OnRevRegExPID, 1387
OnRevRegPID, 1387
OnRevSync, 1388
OnRevSyncEx, 1388
OnRevSyncExPID, 1389
OnRevSyncPID, 1390
OpenFileAppend, 1390
OpenFileRead, 1391
OpenFileReadLinear, 1391
PlayFile, 1392
PlayFileEx, 1392
PlaySound, 1393
PlayTone, 1393
PlayToneEx, 1394
PlayTones, 1394
PointOut, 1395
PolyOut, 1395
Pow, 1169
pow, 1396
PowerDown, 1396
Precedes, 1397
printf, 1397
PSPNxAnalog, 1398
PSPNxDigital, 1398
putc, 1169
rand, 1398
Random, 1399
Read, 1399
ReadButtonEx, 1400
ReadBytes, 1400
ReadI2CRegister, 1401
ReadLn, 1401
ReadLnString, 1402
ReadNRLinkBytes, 1402
ReadSensorColorEx, 1403
ReadSensorColorRaw, 1403
ReadSensorEMeter, 1404
ReadSensorHTAccel, 1404
ReadSensorHTColor, 1405
ReadSensorHTColor2Active, 1406
ReadSensorHTIRReceiver, 1406
ReadSensorHTIRReceiverEx, 1407
ReadSensorHTIRSeeker, 1407
ReadSensorHTIRSeeker2AC, 1408
ReadSensorHTIRSeeker2DC, 1408
ReadSensorHTNormalizedColor,
 1409
ReadSensorHTNormalized-
 Color2Active, 1410
ReadSensorHTRawColor, 1410
ReadSensorHTRawColor2, 1411
ReadSensorHTTouchMultiplexer,
 1411
ReadSensorMSAccel, 1412

ReadSensorMSPlayStation, 1412
ReadSensorMSRTClock, 1413
ReadSensorMSTilt, 1414
ReadSensorUSEx, 1414
RebootInFirmwareMode, 1415
ReceiveMessage, 1415
ReceiveRemoteBool, 1416
ReceiveRemoteMessageEx, 1416
ReceiveRemoteNumber, 1417
ReceiveRemoteString, 1417
RechargeableBattery, 1418
RectOut, 1418
reladdressOf, 1419
Release, 1419
RemoteKeepAlive, 1420
RemoteMessageRead, 1420
RemoteMessageWrite, 1420
RemotePlaySoundFile, 1421
RemotePlayTone, 1421
RemoteResetMotorPosition, 1422
RemoteResetScaledValue, 1422
RemoteSetInputModule, 1423
RemoteSetOutputState, 1423
RemoteStartProgram, 1424
RemoteStopProgram, 1425
RemoteStopSound, 1425
remove, 1425
rename, 1426
RenameFile, 1426
ResetAllTachoCounts, 1427
ResetBlockTachoCount, 1427
ResetRotationCount, 1427
ResetScreen, 1428
ResetSensor, 1428
ResetSleepTimer, 1428
ResetTachoCount, 1429
ResizeFile, 1429
ResolveHandle, 1430
rewind, 1430
RICSetValue, 1169
RightStr, 1430
RotateMotor, 1431
RotateMotorEx, 1431
RotateMotorExPID, 1432
RotateMotorPID, 1433
RS485Control, 1433
RS485DataAvailable, 1434
RS485Exit, 1434
RS485Init, 1435
RS485Read, 1435
RS485SendingData, 1435
RS485Status, 1436
RS485Uart, 1436
RS485Write, 1437
RunNRLinkMacro, 1437
S1, 1170
s16, 1172
S2, 1172
S3, 1172
s32, 1172
S4, 1172
s8, 1172
SEEK_CUR, 1172
SEEK_END, 1173
SEEK_SET, 1173
SendMessage, 1438
SendRemoteBool, 1438
SendRemoteNumber, 1439
SendRemoteString, 1439
SendResponseBool, 1440
SendResponseNumber, 1440
SendResponseString, 1440
SendRS485Bool, 1441
SendRS485Number, 1441
SendRS485String, 1442
Sensor, 1442
SENSOR_1, 1173
SENSOR_2, 1173
SENSOR_3, 1173
SENSOR_4, 1173
SENSOR_CELSIUS, 1173
SENSOR_COLORBLUE, 1173
SENSOR_COLORFULL, 1173
SENSOR_COLORGREEN, 1174
SENSOR_COLORNONE, 1174
SENSOR_COLORRED, 1174
SENSOR_EDGE, 1174
SENSOR_FAHRENHEIT, 1174
SENSOR_LIGHT, 1174
SENSOR_LOWSPEED, 1174
SENSOR_LOWSPEED_9V, 1174
SENSOR_MODE_BOOL, 1175

SENSOR_MODE_CELSIUS, 1175
SENSOR_MODE_EDGE, 1175
SENSOR_MODE_FAHRENHEIT, 1175
SENSOR_MODE_PERCENT, 1175
SENSOR_MODE_PULSE, 1175
SENSOR_MODE_RAW, 1175
SENSOR_MODE_ROTATION, 1175
SENSOR_NXTLIGHT, 1176
SENSOR_PULSE, 1176
SENSOR_ROTATION, 1176
SENSOR_SOUND, 1176
SENSOR_TOUCH, 1176
SENSOR_TYPE_COLORBLUE, 1176
SENSOR_TYPE_COLORFULL, 1176
SENSOR_TYPE_COLORGREEN, 1176
SENSOR_TYPE_COLORNONE, 1177
SENSOR_TYPE_COLORRED, 1177
SENSOR_TYPE_CUSTOM, 1177
SENSOR_TYPE_HIGHSPEED, 1177
SENSOR_TYPE_LIGHT, 1177
SENSOR_TYPE_LIGHT_ACTIVE, 1177
SENSOR_TYPE_LIGHT_INACTIVE, 1177
SENSOR_TYPE_LOWSPEED, 1177
SENSOR_TYPE_LOWSPEED_9V, 1178
SENSOR_TYPE_NONE, 1178
SENSOR_TYPE_ROTATION, 1178
SENSOR_TYPE_SOUND_DB, 1178
SENSOR_TYPE_SOUND_DBA, 1178
SENSOR_TYPE_TEMPERATURE, 1178
SENSOR_TYPE_TOUCH, 1178
SensorBoolean, 1443
SensorDigiPinsDirection, 1443
SensorDigiPinsOutputLevel, 1443
SensorDigiPinsStatus, 1444
SensorHTColorNum, 1444
SensorHTCompass, 1445
SensorHTEOPD, 1445
SensorHTGyro, 1445
SensorHTIRSeeker2ACDir, 1446
SensorHTIRSeeker2Addr, 1446
SensorHTIRSeeker2DCDir, 1447
SensorHTIRSeekerDir, 1447
SensorInvalid, 1448
SensorMode, 1448
SensorMSCompass, 1448
SensorMSDROD, 1449
SensorMSPressure, 1449
SensorMSPressureRaw, 1450
SensorNormalized, 1450
SensorRaw, 1450
SensorScaled, 1451
SensorType, 1451
SensorUS, 1452
SensorValue, 1452
SensorValueBool, 1453
SensorValueRaw, 1453
SetAbortFlag, 1453
SetACCLNxSensitivity, 1454
SetBatteryState, 1454
SetBluetoothState, 1455
SetBTInputBuffer, 1455
SetBTInputBufferInPtr, 1455
SetBTInputBufferOutPtr, 1456
SetBTOutputBuffer, 1456
SetBTOutputBufferInPtr, 1456
SetBTOutputBufferOutPtr, 1457
SetButtonLongPressCount, 1457
SetButtonLongReleaseCount, 1457
SetButtonModuleValue, 1458
SetButtonPressCount, 1458
SetButtonReleaseCount, 1458
SetButtonShortReleaseCount, 1459
SetButtonState, 1459
SetCommandFlags, 1459
SetCommandModuleBytes, 1460
SetCommandModuleValue, 1460
SetCommModuleBytes, 1461

SetCommModuleValue, 1461
SetCustomSensorActiveStatus, 1461
SetCustomSensorPercentFullScale,
 1462
SetCustomSensorZeroOffset, 1462
SetDisplayContrast, 1462
SetDisplayDisplay, 1463
SetDisplayEraseMask, 1463
SetDisplayFlags, 1463
SetDisplayFont, 1464
SetDisplayModuleBytes, 1464
SetDisplayModuleValue, 1464
SetDisplayNormal, 1465
SetDisplayPopup, 1465
SetDisplayTextLinesCenterFlags,
 1466
SetDisplayUpdateMask, 1466
SetHSFlags, 1466
SetHSInputBuffer, 1467
SetHSInputBufferInPtr, 1467
SetHSInputBufferOutPtr, 1467
SetHSMode, 1468
SetHSOutputBuffer, 1468
SetHSOutputBufferInPtr, 1468
SetHSOutputBufferOutPtr, 1469
SetHSSpeed, 1469
SetHSState, 1469
SetHTColor2Mode, 1470
SetHTIRSeeker2Mode, 1470
SetInput, 1471
SetInputModuleValue, 1471
SetIOCctrlModuleValue, 1471
SetIOMapBytes, 1472
SetIOMapBytesByID, 1472
SetIOMapValue, 1473
SetIOMapValueByID, 1473
SetLoaderModuleValue, 1474
SetLongAbort, 1474
SetLowSpeedModuleBytes, 1474
SetLowSpeedModuleValue, 1475
SetMotorPwnFreq, 1475
SetOnBrickProgramPointer, 1476
SetOutput, 1476
SetOutputModuleValue, 1476
SetSensor, 1477
SetSensorBoolean, 1477
SetSensorColorBlue, 1478
SetSensorColorFull, 1478
SetSensorColorGreen, 1478
SetSensorColorNone, 1479
SetSensorColorRed, 1479
SetSensorDigiPinsDirection, 1480
SetSensorDigiPinsOutputLevel,
 1480
SetSensorDigiPinsStatus, 1480
SetSensorHTEOPD, 1481
SetSensorHTGyro, 1481
SetSensorLight, 1481
SetSensorLowspeed, 1482
SetSensorMode, 1482
SetSensorMSDROD, 1483
SetSensorMSPressure, 1483
SetSensorSound, 1483
SetSensorTouch, 1484
SetSensorType, 1484
SetSleepTime, 1485
SetSleepTimeout, 1485
SetSleepTimer, 1485
SetSoundDuration, 1486
SetSoundFlags, 1486
SetSoundFrequency, 1486
SetSoundMode, 1487
SetSoundModuleState, 1487
SetSoundModuleValue, 1488
SetSoundSampleRate, 1488
SetSoundVolume, 1488
SetUIButton, 1489
SetUIModuleValue, 1489
SetUIState, 1489
SetUSBInputBuffer, 1490
SetUSBInputBufferInPtr, 1490
SetUSBInputBufferOutPtr, 1490
SetUSBOutputBuffer, 1491
SetUSBOutputBufferInPtr, 1491
SetUSBOutputBufferOutPtr, 1491
SetUSBPollBuffer, 1492
SetUSBPollBufferInPtr, 1492
SetUSBPollBufferOutPtr, 1492
SetUSBState, 1493
SetVMRunState, 1493
SetVolume, 1493
sign, 1493

Sin, 1178
sin, 1494
SinD, 1179
sind, 1494
Sinh, 1179
sinh, 1495
SinhD, 1180
sinhd, 1495
SleepNow, 1496
SleepTime, 1496
SleepTimeout, 1496
SleepTimer, 1497
SoundDuration, 1497
SoundFlags, 1497
SoundFrequency, 1498
SoundMode, 1498
SoundSampleRate, 1498
SoundState, 1499
SoundVolume, 1499
sprintf, 1500
Sqrt, 1180
sqrt, 1500
StartTask, 1500
Stop, 1501
StopAllTasks, 1501
StopSound, 1501
StopTask, 1502
StrCat, 1503
strcat, 1502
strcmp, 1503
strcpy, 1503
StrIndex, 1504
StrLen, 1505
strlen, 1504
strncat, 1505
strncmp, 1506
strncpy, 1506
StrReplace, 1507
StrToByteArray, 1507
strtod, 1508
strtol, 1508
StrToNum, 1509
strtoul, 1510
SubStr, 1510
SysCall, 1511
SysColorSensorRead, 1511
SysCommBTCheckStatus, 1512
SysCommBTConnection, 1512
SysCommBTOnOff, 1513
SysCommBTWrite, 1513
SysCommExecuteFunction, 1513
SysCommHSCheckStatus, 1514
SysCommHSControl, 1514
SysCommHSRead, 1515
SysCommHSWrite, 1515
SysCommLSCheckStatus, 1516
SysCommLSRead, 1516
SysCommLSWrite, 1516
SysCommLSWriteEx, 1517
SysComputeCalibValue, 1517
SysDatalogGetTimes, 1518
SysDatalogWrite, 1518
SysDisplayExecuteFunction, 1518
SysDrawCircle, 1519
SysDrawEllipse, 1519
SysDrawFont, 1520
SysDrawGraphic, 1520
SysDrawGraphicArray, 1520
SysDrawLine, 1521
SysDrawPoint, 1521
SysDrawPolygon, 1521
SysDrawRect, 1522
SysDrawText, 1522
SysFileClose, 1522
SysFileDelete, 1523
SysFileFindFirst, 1523
SysFileFindNext, 1523
SysFileOpenAppend, 1524
SysFileOpenRead, 1524
SysFileOpenReadLinear, 1525
SysFileOpenWrite, 1525
SysFileOpenWriteLinear, 1525
SysFileOpenWriteNonLinear, 1526
SysFileRead, 1526
SysFileRename, 1526
SysFileResize, 1527
SysFileResolveHandle, 1527
SysFileSeek, 1528
SysFileWrite, 1528
SysGetStartTick, 1528
SysIOMapRead, 1529
SysIOMapReadByID, 1529

SysIOMapWrite, 1529
SysIOMapWriteByID, 1530
SysKeepAlive, 1530
SysListFiles, 1531
SysLoaderExecuteFunction, 1531
SysMessageRead, 1531
SysMessageWrite, 1532
SysRandomNumber, 1532
SysReadButton, 1532
SysReadSemData, 1533
SysSetScreenMode, 1533
SysSetSleepTimeout, 1533
SysSoundGetState, 1534
SysSoundPlayFile, 1534
SysSoundPlayTone, 1535
SysSoundSetState, 1535
SysUpdateCalibCacheInfo, 1535
SysWriteSemData, 1536
Tan, 1180
tan, 1536
TanD, 1181
tand, 1537
Tanh, 1181
tanh, 1537
TanhD, 1182
tanhd, 1537
TextOut, 1538
tolower, 1539
toupper, 1539
Trunc, 1182
trunc, 1540
u16, 1182
u32, 1183
u8, 1183
UIButton, 1540
UIState, 1540
UnflattenVar, 1541
USBInputBufferInPtr, 1541
USBInputBufferOutPtr, 1542
USBOutputBufferInPtr, 1542
USBOutputBufferOutPtr, 1542
USBPollBufferInPtr, 1542
USBPollBufferOutPtr, 1543
USBState, 1543
UsbState, 1543
VMRunState, 1544
Volume, 1544
Wait, 1544
Write, 1545
WriteBytes, 1545
WriteBytesEx, 1546
WriteI2CRegister, 1546
WriteLn, 1547
WriteLnString, 1547
WriteNRLLinkBytes, 1548
WriteString, 1548
Yield, 1549
NXT firmware module IDs, 189
NXT firmware module names, 187
NXT Firmware Modules, 43
NXTLimits
 CHAR_BIT, 792
 CHAR_MAX, 792
 CHAR_MIN, 792
 INT_MAX, 792
 INT_MIN, 792
 LONG_MAX, 792
 LONG_MIN, 792
 RAND_MAX, 792
 SCHAR_MAX, 793
 SCHAR_MIN, 793
 SHRT_MAX, 793
 SHRT_MIN, 793
 UCHAR_MAX, 793
 UINT_MAX, 793
 ULONG_MAX, 793
 USHRT_MAX, 793
Off
 NXCDefs.h, 1378
 OutputModuleFunctions, 252
OffEx
 NXCDefs.h, 1379
 OutputModuleFunctions, 252
Offset
 IOMapReadByIDType, 852
 IOMapReadType, 853
 IOMapWriteByIDType, 855
 IOMapWriteType, 856
OldFilename
 FileRenameType, 845
On

DisplayExecuteFunctionType, 821
OnBrickProgramPointer
 NXCDefs.h, 1379
 UiModuleFunctions, 443
OnFwd
 NXCDefs.h, 1379
 OutputModuleFunctions, 253
OnFwdEx
 NXCDefs.h, 1380
 OutputModuleFunctions, 253
OnFwdReg
 NXCDefs.h, 1380
 OutputModuleFunctions, 254
OnFwdRegEx
 NXCDefs.h, 1381
 OutputModuleFunctions, 254
OnFwdRegExPID
 NXCDefs.h, 1381
 OutputModuleFunctions, 255
OnFwdRegPID
 NXCDefs.h, 1382
 OutputModuleFunctions, 255
OnFwdSync
 NXCDefs.h, 1382
 OutputModuleFunctions, 256
OnFwdSyncEx
 NXCDefs.h, 1383
 OutputModuleFunctions, 257
OnFwdSyncExPID
 NXCDefs.h, 1383
 OutputModuleFunctions, 257
OnFwdSyncPID
 NXCDefs.h, 1384
 OutputModuleFunctions, 258
OnRev
 NXCDefs.h, 1385
 OutputModuleFunctions, 258
OnRevEx
 NXCDefs.h, 1385
 OutputModuleFunctions, 259
OnRevReg
 NXCDefs.h, 1386
 OutputModuleFunctions, 259
OnRevRegEx
 NXCDefs.h, 1386
 OutputModuleFunctions, 260
OnRevRegExPID
 NXCDefs.h, 1387
 OutputModuleFunctions, 260
OnRevRegPID
 NXCDefs.h, 1387
 OutputModuleFunctions, 261
OnRevSync
 NXCDefs.h, 1388
 OutputModuleFunctions, 262
OnRevSyncEx
 NXCDefs.h, 1388
 OutputModuleFunctions, 262
OnRevSyncExPID
 NXCDefs.h, 1389
 OutputModuleFunctions, 263
OnRevSyncPID
 NXCDefs.h, 1390
 OutputModuleFunctions, 263
OPARR_MAX
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_MEAN
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_MIN
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_SORT
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_STD
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_SUM
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OPARR_SUMSQR
 ArrayOpConstants, 552
 NBCCCommon.h, 1003
OpenFileAppend
 LoaderModuleFunctions, 459
 NXCDefs.h, 1390
OpenFileRead
 LoaderModuleFunctions, 460
 NXCDefs.h, 1391
OpenFileReadLinear

LoaderModuleFunctions, 460
 NXCDefs.h, 1391

Options
 DrawCircleType, 824
 DrawEllipseType, 826
 DrawFontType, 828
 DrawGraphicArrayType, 829
 DrawGraphicType, 831
 DrawLineType, 832
 DrawPointType, 834
 DrawPolygonType, 835
 DrawRectType, 836
 DrawTextType, 837

Origin
 FileSeekType, 849

OUT_A
 NBCCCommon.h, 1004
 OutputPortConstants, 639

OUT_AB
 NBCCCommon.h, 1004
 OutputPortConstants, 640

OUT_ABC
 NBCCCommon.h, 1004
 OutputPortConstants, 640

OUT_AC
 NBCCCommon.h, 1004
 OutputPortConstants, 640

OUT_B
 NBCCCommon.h, 1004
 OutputPortConstants, 640

OUT_BC
 NBCCCommon.h, 1005
 OutputPortConstants, 640

OUT_C
 NBCCCommon.h, 1005
 OutputPortConstants, 640

OUT_MODE BRAKE
 NBCCCommon.h, 1005
 OutModeConstants, 645

OUT_MODE COAST
 NBCCCommon.h, 1005
 OutModeConstants, 645

OUT_MODE MOTORON
 NBCCCommon.h, 1005
 OutModeConstants, 645

OUT_MODE REGMETHOD
 NBCCCommon.h, 1005
 OutModeConstants, 645

NBCCCommon.h, 1005
 OutModeConstants, 646

OutModeConstants
 OUT_MODE BRAKE, 645
 OUT_MODE COAST, 645
 OUT_MODE MOTORON, 645
 OUT_MODE REGMETHOD, 646
 OUT_MODE REGULATED, 646

OutOptionConstants
 OUT_OPTION_HOLDATLIMIT,
 646
 OUT_OPTION_-
 RAMPDOWNTOLIMIT,

646
Output field constants, 649
Output module, 48
Output module constants, 48
Output module functions, 241
Output module IOMAP offsets, 655
Output port constants, 639
Output port mode constants, 645
Output port option constants, 646
Output port regulation mode constants, 648
Output port run state constants, 647
Output port update flag constants, 642
OutputFieldConstants
 ActualSpeed, 650
 BlockTachoCount, 650
 OutputMode, 651
 OutputOptions, 651
 Overload, 651
 Power, 652
 RegDValue, 652
 RegIValue, 652
 RegMode, 652
 RegPValue, 653
 RotationCount, 653
 RunState, 653
 TachoCount, 653
 TachoLimit, 654
 TurnRatio, 654
 UpdateFlags, 654
OutputIOMAP
 OutputOffsetActualSpeed, 655
 OutputOffsetBlockTachoCount, 655
 OutputOffsetFlags, 655
 OutputOffsetMode, 656
 OutputOffsetMotorRPM, 656
 OutputOffsetOptions, 656
 OutputOffsetOverloaded, 656
 OutputOffsetPwnFreq, 656
 OutputOffsetRegDParameter, 656
 OutputOffsetRegIParameter, 656
 OutputOffsetRegMode, 656
 OutputOffsetRegPParameter, 656
 OutputOffsetRotationCount, 657
 OutputOffsetRunState, 657
 OutputOffsetSpeed, 657
OutputOffsetSyncTurnParameter, 657
OutputOffsetTachoCount, 657
OutputOffsetTachoLimit, 657
OutputMode
 NBCCCommon.h, 1007
 OutputFieldConstants, 651
OutputModuleFunctions
 Coast, 246
 CoastEx, 246
 Float, 246
 GetOutput, 247
 MotorActualSpeed, 247
 MotorBlockTachoCount, 247
 MotorMode, 248
 MotorOverload, 248
 MotorPower, 248
 MotorPwnFreq, 249
 MotorRegDValue, 249
 MotorRegIValue, 249
 MotorRegPValue, 250
 MotorRegulation, 250
 MotorRotationCount, 250
 MotorRunState, 251
 MotorTachoCount, 251
 MotorTachoLimit, 251
 MotorTurnRatio, 252
 Off, 252
 OffEx, 252
 OnFwd, 253
 OnFwdEx, 253
 OnFwdReg, 254
 OnFwdRegEx, 254
 OnFwdRegExPID, 255
 OnFwdRegPID, 255
 OnFwdSync, 256
 OnFwdSyncEx, 257
 OnFwdSyncExPID, 257
 OnFwdSyncPID, 258
 OnRev, 258
 OnRevEx, 259
 OnRevReg, 259
 OnRevRegEx, 260
 OnRevRegExPID, 260
 OnRevRegPID, 261
 OnRevSync, 262

OnRevSyncEx, [262](#)
OnRevSyncExPID, [263](#)
OnRevSyncPID, [263](#)
ResetAllTachoCounts, [264](#)
ResetBlockTachoCount, [264](#)
ResetRotationCount, [265](#)
ResetTachoCount, [265](#)
RotateMotor, [265](#)
RotateMotorEx, [266](#)
RotateMotorExPID, [266](#)
RotateMotorPID, [267](#)
SetMotorPwnFreq, [268](#)
SetOutput, [268](#)

OutputModuleID
 ModuleIDConstants, [191](#)
 NBCCommon.h, [1007](#)

OutputModuleName
 ModuleNameConstants, [189](#)
 NBCCommon.h, [1007](#)

OutputOffsetActualSpeed
 NBCCommon.h, [1007](#)
 OutputIOMAP, [655](#)

OutputOffsetBlockTachoCount
 NBCCommon.h, [1007](#)
 OutputIOMAP, [655](#)

OutputOffsetFlags
 NBCCommon.h, [1007](#)
 OutputIOMAP, [655](#)

OutputOffsetMode
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetMotorRPM
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetOptions
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetOverloaded
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetPwnFreq
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetRegDParameter
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetRegIParameter
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetRegMode
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetRegPParameter
 NBCCommon.h, [1008](#)
 OutputIOMAP, [656](#)

OutputOffsetRotationCount
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOffsetRunState
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOffsetSpeed
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOffsetSyncTurnParameter
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOffsetTachoCount
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOffsetTachoLimit
 NBCCommon.h, [1009](#)
 OutputIOMAP, [657](#)

OutputOptions
 NBCCommon.h, [1009](#)
 OutputFieldConstants, [651](#)

OutputPortConstants
 OUT_A, [639](#)
 OUT_AB, [640](#)
 OUT_ABC, [640](#)
 OUT_AC, [640](#)
 OUT_B, [640](#)
 OUT_BC, [640](#)
 OUT_C, [640](#)

OutRegModeConstants
 OUT_REGMODE_IDLE, [648](#)
 OUT_REGMODE_SPEED, [648](#)
 OUT_REGMODE_SYNC, [649](#)

OutRunStateConstants
 OUT_RUNSTATE_HOLD, [647](#)
 OUT_RUNSTATE_IDLE, [647](#)

OUT_RUNSTATE_RAMPDOWN,
 647

OUT_RUNSTATE_RAMPUP, **647**

OUT_RUNSTATE_RUNNING, **648**

OutUFConstants

- UF_PENDING_UPDATES, **643**
- UF_UPDATE_MODE, **643**
- UF_UPDATE_PID_VALUES, **643**
- UF_UPDATE_RESET_BLOCK_-
 COUNT, **643**
- UF_UPDATE_RESET_COUNT,
 643
- UF_UPDATE_RESET_-
 ROTATION_COUNT, **643**
- UF_UPDATE_SPEED, **643**
- UF_UPDATE_TACHO_LIMIT, **643**

Overload

- NBCCCommon.h, **1009**
- OutputFieldConstants, **651**

Param1

- CommExecuteFunctionType, **807**

Param2

- CommExecuteFunctionType, **807**

Param3

- CommExecuteFunctionType, **807**

Pattern

- ListFilesType, **859**

PF/IR Train function constants, **760**

PF_CHANNEL_1

- NBCCCommon.h, **1010**
- PFChannelConstants, **758**

PF_CHANNEL_2

- NBCCCommon.h, **1010**
- PFChannelConstants, **758**

PF_CHANNEL_3

- NBCCCommon.h, **1010**
- PFChannelConstants, **759**

PF_CHANNEL_4

- NBCCCommon.h, **1010**
- PFChannelConstants, **759**

PF_CMD_BRAKE

- NBCCCommon.h, **1010**
- PFCmdConstants, **757**

PF_CMD_FWD

- NBCCCommon.h, **1010**

PFCmdConstants, **757**

PF_CMD_REV

- NBCCCommon.h, **1011**

PFCmdConstants, **757**

PF_CMD_STOP

- NBCCCommon.h, **1011**
- PFCmdConstants, **758**

PF_CST_CLEAR1_CLEAR2

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_CLEAR1_SET2

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_DECREMENT_PWM

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_FULL_FWD

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_FULL_REV

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_INCREMENT_PWM

- NBCCCommon.h, **1011**
- PFCSTOptions, **765**

PF_CST_SET1_CLEAR2

- NBCCCommon.h, **1012**
- PFCSTOptions, **765**

PF_CST_SET1_SET2

- NBCCCommon.h, **1012**
- PFCSTOptions, **765**

PF_CST_TOGGLE_DIR

- NBCCCommon.h, **1012**
- PFCSTOptions, **765**

PF_FUNC_CLEAR

- NBCCCommon.h, **1012**
- PFPinFuncs, **764**

PF_FUNC_NOCHANGE

- NBCCCommon.h, **1012**
- PFPinFuncs, **764**

PF_FUNC_SET

- NBCCCommon.h, **1012**
- PFPinFuncs, **764**

PF_FUNC_TOGGLE

- NBCCCommon.h, **1012**
- PFPinFuncs, **764**

PF_MODE_COMBO_DIRECT
 NBCCCommon.h, 1012
 PFFModeConstants, 759

PF_MODE_COMBO_PWM
 NBCCCommon.h, 1013
 PFFModeConstants, 759

PF_MODE_SINGLE_OUTPUT_CST
 NBCCCommon.h, 1013
 PFFModeConstants, 759

PF_MODE_SINGLE_OUTPUT_PWM
 NBCCCommon.h, 1013
 PFFModeConstants, 760

PF_MODE_SINGLE_PIN_CONT
 NBCCCommon.h, 1013
 PFFModeConstants, 760

PF_MODE_SINGLE_PIN_TIME
 NBCCCommon.h, 1013
 PFFModeConstants, 760

PF_MODE_TRAIN
 NBCCCommon.h, 1013
 PFFModeConstants, 760

PF_OUT_A
 NBCCCommon.h, 1013
 PFOOutputs, 762

PF_OUT_B
 NBCCCommon.h, 1013
 PFOOutputs, 762

PF_PIN_C1
 NBCCCommon.h, 1013
 PFPinConstants, 763

PF_PIN_C2
 NBCCCommon.h, 1014
 PFPinConstants, 763

PF_PWM BRAKE
 NBCCCommon.h, 1014
 PFPWMOptions, 766

PF_PWM_FLOAT
 NBCCCommon.h, 1014
 PFPWMOptions, 766

PF_PWM_FWD1
 NBCCCommon.h, 1014
 PFPWMOptions, 766

PF_PWM_FWD2
 NBCCCommon.h, 1014
 PFPWMOptions, 766

PF_PWM_FWD3
 NBCCCommon.h, 1014
 PFPWMOptions, 767

 NBCCCommon.h, 1014
 PFPWMOptions, 767

PF_PWM_FWD4
 NBCCCommon.h, 1014
 PFPWMOptions, 767

PF_PWM_FWD5
 NBCCCommon.h, 1014
 PFPWMOptions, 767

PF_PWM_FWD6
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_FWD7
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_REV1
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_REV2
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_REV3
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_REV4
 NBCCCommon.h, 1015
 PFPWMOptions, 767

PF_PWM_REV5
 NBCCCommon.h, 1015
 PFPWMOptions, 768

PF_PWM_REV6
 NBCCCommon.h, 1015
 PFPWMOptions, 768

PF_PWM_REV7
 NBCCCommon.h, 1015
 PFPWMOptions, 768

PFChannelConstants
 PF_CHANNEL_1, 758
 PF_CHANNEL_2, 758
 PF_CHANNEL_3, 759
 PF_CHANNEL_4, 759

PFCmdConstants
 PF_CMD BRAKE, 757
 PF_CMD_FWD, 757
 PF_CMD_REV, 757
 PF_CMD_STOP, 758

PFCSTOptions

PF_CST_CLEAR1_CLEAR2, [765](#)
PF_CST_CLEAR1_SET2, [765](#)
PF_CST_DECREMENT_PWM,
 [765](#)
PF_CST_FULL_FWD, [765](#)
PF_CST_FULL_REV, [765](#)
PF_CST_INCREMENT_PWM, [765](#)
PF_CST_SET1_CLEAR2, [765](#)
PF_CST_SET1_SET2, [765](#)
PF_CST_TOGGLE_DIR, [765](#)

PFModeConstants
 PF_MODE_COMBO_DIRECT,
 [759](#)
 PF_MODE_COMBO_PWM, [759](#)
 PF_MODE_SINGLE_OUTPUT_-
 CST, [759](#)
 PF_MODE_SINGLE_OUTPUT_-
 PWM, [760](#)
 PF_MODE_SINGLE_PIN_CONT,
 [760](#)
 PF_MODE_SINGLE_PIN_TIME,
 [760](#)
 PF_MODE_TRAIN, [760](#)

PFOoutputs
 PF_OUT_A, [762](#)
 PF_OUT_B, [762](#)

PFPinConstants
 PF_PIN_C1, [763](#)
 PF_PIN_C2, [763](#)

PFPinFuncs
 PF_FUNC_CLEAR, [764](#)
 PF_FUNC_NOCHANGE, [764](#)
 PF_FUNC_SET, [764](#)
 PF_FUNC_TOGGLE, [764](#)

PFPWMOptions
 PF_PWM BRAKE, [766](#)
 PF_PWM_FLOAT, [766](#)
 PF_PWM_FWD1, [766](#)
 PF_PWM_FWD2, [766](#)
 PF_PWM_FWD3, [767](#)
 PF_PWM_FWD4, [767](#)
 PF_PWM_FWD5, [767](#)
 PF_PWM_FWD6, [767](#)
 PF_PWM_FWD7, [767](#)
 PF_PWM_REV1, [767](#)
 PF_PWM_REV2, [767](#)

PF_PWM_REV3, [767](#)
PF_PWM_REV4, [767](#)
PF_PWM_REV5, [768](#)
PF_PWM_REV6, [768](#)
PF_PWM_REV7, [768](#)

PI
 MiscConstants, [192](#)
 NBCCCommon.h, [1016](#)

PID constants, [640](#)

PID_0
 NBCCCommon.h, [1016](#)
 PIDConstants, [641](#)

PID_1
 NBCCCommon.h, [1016](#)
 PIDConstants, [641](#)

PID_2
 NBCCCommon.h, [1016](#)
 PIDConstants, [641](#)

PID_3
 NBCCCommon.h, [1016](#)
 PIDConstants, [641](#)

PID_4
 NBCCCommon.h, [1016](#)
 PIDConstants, [641](#)

PID_5
 NBCCCommon.h, [1016](#)
 PIDConstants, [642](#)

PID_6
 NBCCCommon.h, [1016](#)
 PIDConstants, [642](#)

PID_7
 NBCCCommon.h, [1016](#)
 PIDConstants, [642](#)

PIDConstants
 PID_0, [641](#)
 PID_1, [641](#)
 PID_2, [641](#)
 PID_3, [641](#)
 PID_4, [641](#)
 PID_5, [642](#)
 PID_6, [642](#)
 PID_7, [642](#)

PlayFile
 NXCDefs.h, [1392](#)
 SoundModuleFunctions, [299](#)

PlayFileEx

NXCDefs.h, 1392
SoundModuleFunctions, 299

PlaySound
 NXCDefs.h, 1393
 SoundModuleFunctions, 299

PlayTone
 NXCDefs.h, 1393
 SoundModuleFunctions, 300

PlayToneEx
 NXCDefs.h, 1394
 SoundModuleFunctions, 300

PlayTones
 NXCDefs.h, 1394
 SoundModuleFunctions, 301

PointOut
 DisplayModuleFunctions, 285
 NXCDefs.h, 1395

Points
 DrawPolygonType, 835

PolyOut
 DisplayModuleFunctions, 285
 NXCDefs.h, 1395

Port
 ColorSensorReadType, 799
 CommLSCheckStatusType, 811
 CommLSReadType, 813
 CommLSSwriteExType, 814
 CommLSSwriteType, 816

Pow
 cmathAPI, 487
 NXCDefs.h, 1169

pow
 cmathAPI, 501
 NXCDefs.h, 1396

Power
 NBCCCommon.h, 1017
 OutputFieldConstants, 652

Power Function channel constants, 758

Power Function command constants, 757

Power Function CST options constants, 764

Power Function mode constants, 759

Power Function output constants, 762

Power Function pin constants, 763

Power Function PWM option constants, 766

Power Function single pin function constants, 763

PowerDown
 IOCtrlModuleFunctions, 372
 NXCDefs.h, 1396

PowerOn constants, 584

PowerState
 CommBTOnOffType, 803

Precedes
 CommandModuleFunctions, 348
 NXCDefs.h, 1397

Pressed
 ReadButtonType, 868

printf
 cstdioAPI, 515
 NXCDefs.h, 1397

PROG_ABORT
 CommandProgStatus, 581
 NBCCCommon.h, 1017

PROG_ERROR
 CommandProgStatus, 581
 NBCCCommon.h, 1017

PROG_IDLE
 CommandProgStatus, 581
 NBCCCommon.h, 1017

PROG_OK
 CommandProgStatus, 581
 NBCCCommon.h, 1017

PROG_RESET
 CommandProgStatus, 581
 NBCCCommon.h, 1017

PROG_RUNNING
 CommandProgStatus, 581
 NBCCCommon.h, 1017

Program status constants, 580

PSP_BTNSET1_DOWN
 MSPSPNXBtnSet1, 783
 NBCCCommon.h, 1017

PSP_BTNSET1_L3
 MSPSPNXBtnSet1, 783
 NBCCCommon.h, 1017

PSP_BTNSET1_LEFT
 MSPSPNXBtnSet1, 783
 NBCCCommon.h, 1018

PSP_BTNSET1_R3
 MSPSPNXBtnSet1, 783

NBCCCommon.h, 1018
PSP_BTNSET1_RIGHT
 MSPSPNXBtnSet1, 784
 NBCCCommon.h, 1018
PSP_BTNSET1_UP
 MSPSPNXBtnSet1, 784
 NBCCCommon.h, 1018
PSP_BTNSET2_CIRCLE
 MSPSPNXBtnSet2, 784
 NBCCCommon.h, 1018
PSP_BTNSET2_CROSS
 MSPSPNXBtnSet2, 784
 NBCCCommon.h, 1018
PSP_BTNSET2_L1
 MSPSPNXBtnSet2, 784
 NBCCCommon.h, 1018
PSP_BTNSET2_L2
 MSPSPNXBtnSet2, 785
 NBCCCommon.h, 1018
PSP_BTNSET2_R1
 MSPSPNXBtnSet2, 785
 NBCCCommon.h, 1018
PSP_BTNSET2_R2
 MSPSPNXBtnSet2, 785
 NBCCCommon.h, 1018
PSP_BTNSET2_SQUARE
 MSPSPNXBtnSet2, 785
 NBCCCommon.h, 1019
PSP_BTNSET2_TRIANGLE
 MSPSPNXBtnSet2, 785
 NBCCCommon.h, 1019
PSP_CMD_ANALOG
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_CMD_DIGITAL
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_BTNSET1
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_BTNSET2
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_XLEFT
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_XRIGHT
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_YLEFT
 MSPSPNX, 782
 NBCCCommon.h, 1019
PSP_REG_YRIGHT
 MSPSPNX, 783
 NBCCCommon.h, 1019
PSPNxAnalog
 MindSensorsAPI, 171
 NXCDefs.h, 1398
PSPNxDigital
 MindSensorsAPI, 171
 NXCDefs.h, 1398
putc
 cstdioAPI, 509
 NXCDefs.h, 1169
QueueID
 MessageReadType, 864
 MessageWriteType, 866
quot
 div_t, 823
 ldiv_t, 858
RADIANS_PER_DEGREE
 MiscConstants, 192
 NBCCCommon.h, 1020
rand
 cstdlibAPI, 523
 NXCDefs.h, 1398
RAND_MAX
 NBCCCommon.h, 1020
 NXTLimits, 792
Random
 cstdlibAPI, 524
 NXCDefs.h, 1399
RandomNumber
 NBCCCommon.h, 1020
 SysCallConstants, 560
RandomNumberType, 866
 Result, 867
RawArray
 ColorSensorReadType, 800
RawVal

ComputeCalibValueType, 817
RawValue
 InputFieldConstants, 631
 NBCCCommon.h, 1020
RCX and Scout opcode constants, 746
RCX and Scout sound constants, 727
RCX and Scout source constants, 740
RCX constants, 719
RCX IR remote constants, 724
RCX output constants, 720
RCX output direction constants, 722
RCX output mode constants, 722
RCX output power constants, 723
RCX_AbsVarOp
 NBCCCommon.h, 1020
 RCXOpcodeConstants, 748
RCX_AndVarOp
 NBCCCommon.h, 1020
 RCXOpcodeConstants, 748
RCX_AutoOffOp
 NBCCCommon.h, 1020
 RCXOpcodeConstants, 748
RCX_BatteryLevelOp
 NBCCCommon.h, 1020
 RCXOpcodeConstants, 748
RCX_BatteryLevelSrc
 NBCCCommon.h, 1020
 RCXSourceConstants, 742
RCX_BootModeOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_CalibrateEventOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearAllEventsOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearCounterOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearMsgOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearSensorOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearSoundOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClearTimerOp
 NBCCCommon.h, 1021
 RCXOpcodeConstants, 749
RCX_ClickCounterSrc
 NBCCCommon.h, 1021
 RCXSourceConstants, 742
RCX_ConstantSrc
 NBCCCommon.h, 1021
 RCXSourceConstants, 742
RCX_CounterSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 742
RCX_DatalogOp
 NBCCCommon.h, 1022
 RCXOpcodeConstants, 749
RCX_DatalogRawDirectSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 742
RCX_DatalogRawIndirectSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 742
RCX_DatalogSrcDirectSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 742
RCX_DatalogSrcIndirectSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 742
RCX_DatalogValueDirectSrc
 NBCCCommon.h, 1022
 RCXSourceConstants, 743
RCX_DatalogValueIndirectSrc
 NBCCCommon.h, 1023
 RCXSourceConstants, 743
RCX_DecCounterOp
 NBCCCommon.h, 1023
 RCXOpcodeConstants, 749
RCX_DeleteSubOp
 NBCCCommon.h, 1023
 RCXOpcodeConstants, 750
RCX_DeleteSubsOp
 NBCCCommon.h, 1023
 RCXOpcodeConstants, 750
RCX_DeleteTaskOp

NBCCCommon.h, 1023
RCXOpcodeConstants, 750

RCX_DeleteTasksOp
NBCCCommon.h, 1023
RCXOpcodeConstants, 750

RCX_DirectEventOp
NBCCCommon.h, 1023
RCXOpcodeConstants, 750

RCX_DisplayOp
NBCCCommon.h, 1023
RCXOpcodeConstants, 750

RCX_DivVarOp
NBCCCommon.h, 1023
RCXOpcodeConstants, 750

RCX_DurationSrc
NBCCCommon.h, 1023
RCXSourceConstants, 743

RCX_EventStateSrc
NBCCCommon.h, 1024
RCXSourceConstants, 743

RCX_FirmwareVersionSrc
NBCCCommon.h, 1024
RCXSourceConstants, 743

RCX_GlobalMotorStatusSrc
NBCCCommon.h, 1024
RCXSourceConstants, 743

RCX_GOutputDirOp
NBCCCommon.h, 1024
RCXOpcodeConstants, 750

RCX_GOutputModeOp
NBCCCommon.h, 1024
RCXOpcodeConstants, 750

RCX_GOutputPowerOp
NBCCCommon.h, 1024
RCXOpcodeConstants, 750

RCX_HysteresisSrc
NBCCCommon.h, 1024
RCXSourceConstants, 743

RCX_IncCounterOp
NBCCCommon.h, 1024
RCXOpcodeConstants, 751

RCX_IndirectVarSrc
NBCCCommon.h, 1024
RCXSourceConstants, 743

RCX_InputBooleanSrc
NBCCCommon.h, 1024

RCXSourceConstants, 743

RCXSourceConstants, 743

RCX_InputModeOp
NBCCCommon.h, 1025
RCXOpcodeConstants, 751

RCX_InputModeSrc
NBCCCommon.h, 1025
RCXSourceConstants, 743

RCX_InputRawSrc
NBCCCommon.h, 1025
RCXSourceConstants, 744

RCX_InputTypeOp
NBCCCommon.h, 1025
RCXOpcodeConstants, 751

RCX_InputTypeSrc
NBCCCommon.h, 1025
RCXSourceConstants, 744

RCX_InputValueSrc
NBCCCommon.h, 1025
RCXSourceConstants, 744

RCX_IRModeOp
NBCCCommon.h, 1025
RCXOpcodeConstants, 751

RCX_LightOp
NBCCCommon.h, 1025
RCXOpcodeConstants, 751

RCX_LowerThresholdSrc
NBCCCommon.h, 1025
RCXSourceConstants, 744

RCX_LSBlinkTimeOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 751

RCX_LSCalibrateOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 751

RCX_LSHysteresisOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 751

RCX_LSLowerThreshOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 751

RCX_LSSupperThreshOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 751

RCX_MessageOp
NBCCCommon.h, 1026
RCXOpcodeConstants, 752

RCX_MessageSrc
 NBCCCommon.h, 1026
 RCXSourceConstants, 744

RCX_MulVarOp
 NBCCCommon.h, 1026
 RCXOpcodeConstants, 752

RCX_MuteSoundOp
 NBCCCommon.h, 1026
 RCXOpcodeConstants, 752

RCX_OnOffFloatOp
 NBCCCommon.h, 1026
 RCXOpcodeConstants, 752

RCX_OrVarOp
 NBCCCommon.h, 1027
 RCXOpcodeConstants, 752

RCX_OUT_A
 NBCCCommon.h, 1027
 RCXOutputConstants, 721

RCX_OUT_AB
 NBCCCommon.h, 1027
 RCXOutputConstants, 721

RCX_OUT_ABC
 NBCCCommon.h, 1027
 RCXOutputConstants, 721

RCX_OUT_AC
 NBCCCommon.h, 1027
 RCXOutputConstants, 721

RCX_OUT_B
 NBCCCommon.h, 1027
 RCXOutputConstants, 721

RCX_OUT_BC
 NBCCCommon.h, 1028
 RCXOutputConstants, 721

RCX_OUT_C
 NBCCCommon.h, 1028
 RCXOutputConstants, 721

RCX_OUT_FLOAT
 NBCCCommon.h, 1028
 RCXOutputMode, 722

RCX_OUT_FULL
 NBCCCommon.h, 1028
 RCXOutputPower, 724

RCX_OUT_FWD
 NBCCCommon.h, 1028
 RCXOutputDirection, 723

RCX_OUT_HALF
 NBCCCommon.h, 1028
 RCXOutputPower, 724

 NBCCCommon.h, 1028
 RCXOutputPower, 724

RCX_OUT_LOW
 NBCCCommon.h, 1028
 RCXOutputPower, 724

RCX_OUT_OFF
 NBCCCommon.h, 1028
 RCXOutputMode, 722

RCX_OUT_ON
 NBCCCommon.h, 1029
 RCXOutputMode, 722

RCX_OUT_REV
 NBCCCommon.h, 1029
 RCXOutputDirection, 723

RCX_OUT_TOGGLE
 NBCCCommon.h, 1029
 RCXOutputDirection, 723

RCX_OutputDirOp
 NBCCCommon.h, 1029
 RCXOpcodeConstants, 752

RCX_OutputPowerOp
 NBCCCommon.h, 1029
 RCXOpcodeConstants, 752

RCX_OutputStatusSrc
 NBCCCommon.h, 1029
 RCXSourceConstants, 744

RCX_PBTurnOffOp
 NBCCCommon.h, 1029
 RCXOpcodeConstants, 752

RCX_PingOp
 NBCCCommon.h, 1029
 RCXOpcodeConstants, 752

RCX_PlaySoundOp
 NBCCCommon.h, 1029
 RCXOpcodeConstants, 752

RCX_PlayToneOp
 NBCCCommon.h, 1030
 RCXOpcodeConstants, 753

RCX_PlayToneVarOp
 NBCCCommon.h, 1030
 RCXOpcodeConstants, 753

RCX_PollMemoryOp
 NBCCCommon.h, 1030
 RCXOpcodeConstants, 753

RCX_PollOp
 NBCCCommon.h, 1030

RCXOpcodeConstants, 753
RCX_ProgramSlotSrc
 NBCCCommon.h, 1030
 RCXSourceConstants, 744
RCX_RandomSrc
 NBCCCommon.h, 1030
 RCXSourceConstants, 744
RCX_RemoteKeysReleased
 NBCCCommon.h, 1030
 RCXRemoteConstants, 725
RCX_RemoteOp
 NBCCCommon.h, 1030
 RCXOpcodeConstants, 753
RCX_RemoteOutABackward
 NBCCCommon.h, 1030
 RCXRemoteConstants, 725
RCX_RemoteOutAForward
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemoteOutBBackward
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemoteOutBForward
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemoteOutCBackward
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemoteOutCForward
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemotePBMessagel
 NBCCCommon.h, 1031
 RCXRemoteConstants, 725
RCX_RemotePBMessage2
 NBCCCommon.h, 1031
 RCXRemoteConstants, 726
RCX_RemotePBMessage3
 NBCCCommon.h, 1031
 RCXRemoteConstants, 726
RCX_RemotePlayASound
 NBCCCommon.h, 1031
 RCXRemoteConstants, 726
RCX_RemoteSelProgram1
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_RemoteSelProgram2
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_RemoteSelProgram3
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_RemoteSelProgram4
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_RemoteSelProgram5
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_RemoteStopOutOff
 NBCCCommon.h, 1032
 RCXRemoteConstants, 726
RCX_ScoutCounterLimitSrc
 NBCCCommon.h, 1032
 RCXSourceConstants, 745
RCX_ScoutEventFBSrc
 NBCCCommon.h, 1032
 RCXSourceConstants, 745
RCX_ScoutLightParamsSrc
 NBCCCommon.h, 1032
 RCXSourceConstants, 745
RCX_ScoutOp
 NBCCCommon.h, 1032
 RCXOpcodeConstants, 753
RCX_ScoutRulesOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 753
RCX_ScoutRulesSrc
 NBCCCommon.h, 1033
 RCXSourceConstants, 745
RCX_ScoutTimerLimitSrc
 NBCCCommon.h, 1033
 RCXSourceConstants, 745
RCX_SelectProgramOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 753
RCX_SendUARTDataOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 753
RCX_SetCounterOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 753
RCX_SetDatalogOp

NBCCCommon.h, 1033
RCXOpcodeConstants, 754

RCX_SetEventOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 754

RCX_SetFeedbackOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 754

RCX_SetPriorityOp
 NBCCCommon.h, 1033
 RCXOpcodeConstants, 754

RCX_SetSourceValueOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_SetTimerLimitOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_SetVarOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_SetWatchOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_SgnVarOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_SoundOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 754

RCX_StartTaskOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 755

RCX_StopAllTasksOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 755

RCX_StopTaskOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 755

RCX_SubVarOp
 NBCCCommon.h, 1034
 RCXOpcodeConstants, 755

RCX_SumVarOp
 NBCCCommon.h, 1035
 RCXOpcodeConstants, 755

RCX_TaskEventsSrc
 NBCCCommon.h, 1035

RCXSourceConstants, 745

RCX_TenMSTimerSrc
 NBCCCommon.h, 1035
 RCXSourceConstants, 745

RCX_TimerSrc
 NBCCCommon.h, 1035
 RCXSourceConstants, 745

RCX_UARTSetupSrc
 NBCCCommon.h, 1035
 RCXSourceConstants, 745

RCX_UnlockFirmOp
 NBCCCommon.h, 1035
 RCXOpcodeConstants, 755

RCX_UnlockOp
 NBCCCommon.h, 1035
 RCXOpcodeConstants, 755

RCX_UnmuteSoundOp
 NBCCCommon.h, 1035
 RCXOpcodeConstants, 755

RCX_UploadDatalogOp
 NBCCCommon.h, 1035
 RCXOpcodeConstants, 755

RCX_UpperThresholdSrc
 NBCCCommon.h, 1035
 RCXSourceConstants, 746

RCX_VariableSrc
 NBCCCommon.h, 1036
 RCXSourceConstants, 746

RCX_ViewSourceValOp
 NBCCCommon.h, 1036
 RCXOpcodeConstants, 755

RCX_VLLOp
 NBCCCommon.h, 1036
 RCXOpcodeConstants, 756

RCX_WatchSrc
 NBCCCommon.h, 1036
 RCXSourceConstants, 746

RCXOpcodeConstants
 RCX_AbsVarOp, 748
 RCX_AndVarOp, 748
 RCX_AutoOffOp, 748
 RCX_BatteryLevelOp, 748
 RCX_BootModeOp, 749
 RCX_CalibrateEventOp, 749
 RCX_ClearAllEventsOp, 749
 RCX_ClearCounterOp, 749

RCX_ClearMsgOp, 749
RCX_ClearSensorOp, 749
RCX_ClearSoundOp, 749
RCX_ClearTimerOp, 749
RCX_DatalogOp, 749
RCX_DecCounterOp, 749
RCX_DeleteSubOp, 750
RCX_DeleteSubsOp, 750
RCX_DeleteTaskOp, 750
RCX_DeleteTasksOp, 750
RCX_DirectEventOp, 750
RCX_DisplayOp, 750
RCX_DivVarOp, 750
RCX_GOutputDirOp, 750
RCX_GOutputModeOp, 750
RCX_GOutputPowerOp, 750
RCX_IncCounterOp, 751
RCX_InputModeOp, 751
RCX_InputTypeOp, 751
RCX_IRModeOp, 751
RCX_LightOp, 751
RCX_LSBlinkTimeOp, 751
RCX_LSCalibrateOp, 751
RCX_LSHysteresisOp, 751
RCX_LSLowerThreshOp, 751
RCX_LSupperThreshOp, 751
RCX_MessageOp, 752
RCX_MulVarOp, 752
RCX_MuteSoundOp, 752
RCX_OnOffFloatOp, 752
RCX_OrVarOp, 752
RCX_OutputDirOp, 752
RCX_OutputPowerOp, 752
RCX_PBTurnOffOp, 752
RCX_PingOp, 752
RCX_PlaySoundOp, 752
RCX_PlayToneOp, 753
RCX_PlayToneVarOp, 753
RCX_PollMemoryOp, 753
RCX_PollOp, 753
RCX_RemoteOp, 753
RCX_ScoutOp, 753
RCX_ScoutRulesOp, 753
RCX_SelectProgramOp, 753
RCX_SendUARTDataOp, 753
RCX_SetCounterOp, 753
RCX_SetDatalogOp, 754
RCX_SetEventOp, 754
RCX_SetFeedbackOp, 754
RCX_SetPriorityOp, 754
RCX_SetSourceValueOp, 754
RCX_SetTimerLimitOp, 754
RCX_SetVarOp, 754
RCX_SetWatchOp, 754
RCX_SgnVarOp, 754
RCX_SoundOp, 754
RCX_StartTaskOp, 755
RCX_StopAllTasksOp, 755
RCX_StopTaskOp, 755
RCX_SubVarOp, 755
RCX_SumVarOp, 755
RCX_UnlockFirmOp, 755
RCX_UnlockOp, 755
RCX_UnmuteSoundOp, 755
RCX_UploadDatalogOp, 755
RCX_ViewSourceValOp, 755
RCX_VLLOp, 756

RCXOutputConstants
 RCX_OUT_A, 721
 RCX_OUT_AB, 721
 RCX_OUT_ABC, 721
 RCX_OUT_AC, 721
 RCX_OUT_B, 721
 RCX_OUT_BC, 721
 RCX_OUT_C, 721

RCXOutputDirection
 RCX_OUT_FWD, 723
 RCX_OUT_REV, 723
 RCX_OUT_TOGGLE, 723

RCXOutputMode
 RCX_OUT_FLOAT, 722
 RCX_OUT_OFF, 722
 RCX_OUT_ON, 722

RCXOutputPower
 RCX_OUT_FULL, 724
 RCX_OUT_HALF, 724
 RCX_OUT_LOW, 724

RCXRemoteConstants
 RCX_RemoteKeysReleased, 725
 RCX_RemoteOutABackward, 725
 RCX_RemoteOutAForward, 725
 RCX_RemoteOutBBackward, 725

RCX_RemoteOutBForward, 725
RCX_RemoteOutCBackward, 725
RCX_RemoteOutCForward, 725
RCX_RemotePBMessagel, 725
RCX_RemotePBMessagel, 726
RCX_RemotePBMessagel, 726
RCX_RemotePlayASound, 726
RCX_RemoteSelProgram1, 726
RCX_RemoteSelProgram2, 726
RCX_RemoteSelProgram3, 726
RCX_RemoteSelProgram4, 726
RCX_RemoteSelProgram5, 726
RCX_RemoteStopOutOff, 726

RCXSoundConstants
 SOUND_CLICK, 727
 SOUND_DOUBLE_BEEP, 727
 SOUND_DOWN, 727
 SOUND_FAST_UP, 727
 SOUND_LOW_BEEP, 728
 SOUND_UP, 728

RCXSourceConstants
 RCX_BatteryLevelSrc, 742
 RCX_ClickCounterSrc, 742
 RCX_ConstantSrc, 742
 RCX_CounterSrc, 742
 RCX_DatalogRawDirectSrc, 742
 RCX_DatalogRawIndirectSrc, 742
 RCX_DatalogSrcDirectSrc, 742
 RCX_DatalogSrcIndirectSrc, 742
 RCX_DatalogValueDirectSrc, 743
 RCX_DatalogValueIndirectSrc, 743
 RCX_DurationSrc, 743
 RCX_EventStateSrc, 743
 RCX_FirmwareVersionSrc, 743
 RCX_GlobalMotorStatusSrc, 743
 RCX_HysteresisSrc, 743
 RCX_IndirectVarSrc, 743
 RCX_InputBooleanSrc, 743
 RCX_InputModeSrc, 743
 RCX_InputRawSrc, 744
 RCX_InputTypeSrc, 744
 RCX_InputValueSrc, 744
 RCX_LowerThresholdSrc, 744
 RCX_MessageSrc, 744
 RCX_OutputStatusSrc, 744
 RCX_ProgramSlotSrc, 744

RCX_RandomSrc, 744
RCX_ScoutCounterLimitSrc, 745
RCX_ScoutEventFBSrc, 745
RCX_ScoutLightParamsSrc, 745
RCX_ScoutRulesSrc, 745
RCX_ScoutTimerLimitSrc, 745
RCX_TaskEventsSrc, 745
RCX_TenMSTimerSrc, 745
RCX_TimerSrc, 745
RCX_UARTSetupSrc, 745
RCX_UpperThresholdSrc, 746
RCX_VariableSrc, 746
RCX_WatchSrc, 746

Read
 LoaderModuleFunctions, 461
 NXCDefs.h, 1399

ReadButton
 NBCCCommon.h, 1036
 SysCallConstants, 560

ReadButtonEx
 ButtonModuleFunctions, 435
 NXCDefs.h, 1400

ReadButtonType, 867
 Count, 868
 Index, 868
 Pressed, 868
 Reset, 868
 Result, 868

ReadBytes
 LoaderModuleFunctions, 461
 NXCDefs.h, 1400

ReadI2CRegister
 LowSpeedModuleFunctions, 321
 NXCDefs.h, 1401

ReadLn
 LoaderModuleFunctions, 462
 NXCDefs.h, 1401

ReadLnString
 LoaderModuleFunctions, 462
 NXCDefs.h, 1402

ReadNRLLinkBytes
 MindSensorsAPI, 172
 NXCDefs.h, 1402

ReadSemData
 NBCCCommon.h, 1036
 SysCallConstants, 560

ReadSemDataType, 868
 Request, 869
 SemData, 869

ReadSensorColorEx
 InputModuleFunctions, 225
 NXCDefs.h, 1403

ReadSensorColorRaw
 InputModuleFunctions, 225
 NXCDefs.h, 1403

ReadSensorEMeter
 LowSpeedModuleFunctions, 322
 NXCDefs.h, 1404

ReadSensorHTAccel
 HiTechnicAPI, 94
 NXCDefs.h, 1404

ReadSensorHTColor
 HiTechnicAPI, 95
 NXCDefs.h, 1405

ReadSensorHTColor2Active
 HiTechnicAPI, 95
 NXCDefs.h, 1406

ReadSensorHTIRReceiver
 HiTechnicAPI, 96
 NXCDefs.h, 1406

ReadSensorHTIRReceiverEx
 HiTechnicAPI, 96
 NXCDefs.h, 1407

ReadSensorHTIRSeeker
 HiTechnicAPI, 97
 NXCDefs.h, 1407

ReadSensorHTIRSeeker2AC
 HiTechnicAPI, 97
 NXCDefs.h, 1408

ReadSensorHTIRSeeker2DC
 HiTechnicAPI, 98
 NXCDefs.h, 1408

ReadSensorHTNormalizedColor
 HiTechnicAPI, 99
 NXCDefs.h, 1409

ReadSensorHTNormalizedColor2Active
 HiTechnicAPI, 99
 NXCDefs.h, 1410

ReadSensorHTRawColor
 HiTechnicAPI, 100
 NXCDefs.h, 1410

ReadSensorHTRawColor2

HiTechnicAPI, 101
NXCDefs.h, 1411

ReadSensorHTTouchMultiplexer
 HiTechnicAPI, 101
 NXCDefs.h, 1411

ReadSensorMSAccel
 MindSensorsAPI, 172
 NXCDefs.h, 1412

ReadSensorMSPlayStation
 MindSensorsAPI, 173
 NXCDefs.h, 1412

ReadSensorMSRTClock
 MindSensorsAPI, 173
 NXCDefs.h, 1413

ReadSensorMSTilt
 MindSensorsAPI, 174
 NXCDefs.h, 1414

ReadSensorUSEx
 LowSpeedModuleFunctions, 323
 NXCDefs.h, 1414

RebootInFirmwareMode
 IOCtrlModuleFunctions, 372
 NXCDefs.h, 1415

ReceiveMessage
 CommModuleFunctions, 397
 NXCDefs.h, 1415

ReceiveRemoteBool
 CommModuleFunctions, 398
 NXCDefs.h, 1416

ReceiveRemoteMessageEx
 CommModuleFunctions, 398
 NXCDefs.h, 1416

ReceiveRemoteNumber
 CommModuleFunctions, 399
 NXCDefs.h, 1417

ReceiveRemoteString
 CommModuleFunctions, 399
 NXCDefs.h, 1417

RechargeableBattery
 NXCDefs.h, 1418
 UiModuleFunctions, 443

RectOut
 DisplayModuleFunctions, 286
 NXCDefs.h, 1418

RegDValue
 NBCCCommon.h, 1036

OutputFieldConstants, 652
RegIValue
 NBCCommon.h, 1037
 OutputFieldConstants, 652
RegMode
 NBCCommon.h, 1037
 OutputFieldConstants, 652
RegPValue
 NBCCommon.h, 1037
 OutputFieldConstants, 653
reladdressOf
 cstringAPI, 536
 NXCDefs.h, 1419
Release
 CommandModuleFunctions, 348
 NXCDefs.h, 1419
rem
 div_t, 823
 ldiv_t, 858
Remote control (direct commands) errors, 579
RemoteKeepAlive
 CommModuleFunctions, 400
 NXCDefs.h, 1420
RemoteMessageRead
 CommModuleFunctions, 400
 NXCDefs.h, 1420
RemoteMessageWrite
 CommModuleFunctions, 401
 NXCDefs.h, 1420
RemotePlaySoundFile
 CommModuleFunctions, 401
 NXCDefs.h, 1421
RemotePlayTone
 CommModuleFunctions, 402
 NXCDefs.h, 1421
RemoteResetMotorPosition
 CommModuleFunctions, 402
 NXCDefs.h, 1422
RemoteResetScaledValue
 CommModuleFunctions, 403
 NXCDefs.h, 1422
RemoteSetInputMode
 CommModuleFunctions, 403
 NXCDefs.h, 1423
RemoteSetOutputState
 CommModuleFunctions, 404
 NXCDefs.h, 1423
 RemoteStartProgram
 CommModuleFunctions, 405
 NXCDefs.h, 1424
 RemoteStopProgram
 CommModuleFunctions, 405
 NXCDefs.h, 1425
 RemoteStopSound
 CommModuleFunctions, 405
 NXCDefs.h, 1425
 Remove
 MessageReadType, 864
remove
 cstdioAPI, 515
 NXCDefs.h, 1425
rename
 cstdioAPI, 516
 NXCDefs.h, 1426
RenameFile
 LoaderModuleFunctions, 463
 NXCDefs.h, 1426
Request
 ReadSemDataType, 869
 WriteSemDataType, 881
Reset
 ReadButtonType, 868
RESET_ALL
 NBCCommon.h, 1037
 TachoResetConstants, 644
RESET_BLOCK_COUNT
 NBCCommon.h, 1038
 TachoResetConstants, 644
RESET_BLOCKANDTACHO
 NBCCommon.h, 1038
 TachoResetConstants, 644
RESET_COUNT
 NBCCommon.h, 1038
 TachoResetConstants, 644
RESET_NONE
 NBCCommon.h, 1038
 TachoResetConstants, 644
RESET_ROTATION_COUNT
 NBCCommon.h, 1038
 TachoResetConstants, 645
ResetAllTachoCounts

NXCDefs.h, 1427
OutputModuleFunctions, 264
ResetBlockTachoCount
 NXCDefs.h, 1427
 OutputModuleFunctions, 264
ResetRotationCount
 NXCDefs.h, 1427
 OutputModuleFunctions, 265
ResetScreen
 DisplayModuleFunctions, 287
 NXCDefs.h, 1428
ResetSensor
 InputModuleFunctions, 226
 NXCDefs.h, 1428
ResetSleepTimer
 CommandModuleFunctions, 349
 NXCDefs.h, 1428
ResetTachoCount
 NXCDefs.h, 1429
 OutputModuleFunctions, 265
ResizeFile
 LoaderModuleFunctions, 463
 NXCDefs.h, 1429
ResolveHandle
 LoaderModuleFunctions, 464
 NXCDefs.h, 1430
Result
 ColorSensorReadType, 800
 CommBTCheckStatusType, 801
 CommBTConnectionType, 802
 CommBTOffOnType, 803
 CommBTWriteType, 805
 CommExecuteFunctionType, 807
 CommHSControlType, 809
 CommLSCheckStatusType, 812
 CommLSReadType, 813
 CommLSWriteExType, 815
 CommLSWriteType, 816
 ComputeCalibValueType, 818
 DatalogWriteType, 820
 DrawCircleType, 825
 DrawEllipseType, 826
 DrawFontType, 828
 DrawGraphicArrayType, 829
 DrawGraphicType, 831
 DrawLineType, 832
 DrawPointType, 834
 DrawPolygonType, 835
 DrawRectType, 836
 DrawTextType, 838
 FileCloseType, 839
 FileDeleteType, 840
 FileFindType, 841
 FileOpenType, 842
 FileReadWriteType, 844
 FileRenameType, 845
 FileResizeType, 846
 FileResolveHandleType, 848
 FileSeekType, 849
 GetStartTickType, 850
 IOMapReadByIDType, 852
 IOMapReadType, 853
 IOMapWriteByIDType, 855
 IOMapWriteType, 856
 KeepAliveType, 857
 ListFilesType, 859
 LoaderExecuteFunctionType, 862
 MessageReadType, 865
 MessageWriteType, 866
 RandomNumberType, 867
 ReadButtonType, 868
 SetScreenModeType, 870
 SetSleepTimeoutType, 871
 SoundPlayFileType, 874
 SoundPlayToneType, 876
 SoundSetStateType, 877
 UpdateCalibCacheInfoType, 880
ReturnLen
 CommLSWriteExType, 815
 CommLSWriteType, 816
RetVal
 CommExecuteFunctionType, 807
rewind
 cstdioAPI, 516
 NXCDefs.h, 1430
RIC Macro Wrappers, 179
RICArg
 NBCCCommon.h, 1038
 RICMacros, 181
RICImgPoint
 NBCCCommon.h, 1039
 RICMacros, 181

RICImgRect
 NBCCCommon.h, 1039
 RICMacros, 181

RICMacros
 RICArg, 181
 RICImgPoint, 181
 RICImgRect, 181
 RICMapArg, 182
 RICMapElement, 182
 RICMapFunction, 182
 RICOOpCircle, 183
 RICOOpCopyBits, 183
 RICOOpDescription, 183
 RICOOpEllipse, 184
 RICOOpLine, 184
 RICOOpNumBox, 184
 RICOOpPixel, 185
 RICOOpPolygon, 185
 RICOOpRect, 185
 RICOOpSprite, 186
 RICOOpVarMap, 186
 RICPolygonPoints, 187
 RICSetValue, 187
 RICSpriteData, 187

RICMapArg
 NBCCCommon.h, 1039
 RICMacros, 182

RICMapElement
 NBCCCommon.h, 1040
 RICMacros, 182

RICMapFunction
 NBCCCommon.h, 1040
 RICMacros, 182

RICOOpCircle
 NBCCCommon.h, 1040
 RICMacros, 183

RICOOpCopyBits
 NBCCCommon.h, 1040
 RICMacros, 183

RICOOpDescription
 NBCCCommon.h, 1041
 RICMacros, 183

RICOOpEllipse
 NBCCCommon.h, 1041
 RICMacros, 184

RICOOpLine

NBCCCommon.h, 1042
 RICMacros, 184

RICOOpNumBox
 NBCCCommon.h, 1042
 RICMacros, 184

RICOOpPixel
 NBCCCommon.h, 1042
 RICMacros, 185

RICOOpPolygon
 NBCCCommon.h, 1043
 RICMacros, 185

RICOOpRect
 NBCCCommon.h, 1043
 RICMacros, 185

RICOOpSprite
 NBCCCommon.h, 1043
 RICMacros, 186

RICOOpVarMap
 NBCCCommon.h, 1044
 RICMacros, 186

RICPolygonPoints
 NBCCCommon.h, 1044
 RICMacros, 187

RICSetValue
 NXCDefs.h, 1169
 RICMacros, 187

RICSpriteData
 NBCCCommon.h, 1044
 RICMacros, 187

RightStr
 cstringAPI, 537
 NXCDefs.h, 1430

ROTATE_QUEUE
 CommandVMState, 575
 NBCCCommon.h, 1045

RotateMotor
 NXCDefs.h, 1431
 OutputModuleFunctions, 265

RotateMotorEx
 NXCDefs.h, 1431
 OutputModuleFunctions, 266

RotateMotorExPID
 NXCDefs.h, 1432
 OutputModuleFunctions, 266

RotateMotorPID
 NXCDefs.h, 1433

OutputModuleFunctions, 267
RotationCount
 NBCCCommon.h, 1045
 OutputFieldConstants, 653
RS485Control
 CommModuleFunctions, 406
 NXCDefs.h, 1433
RS485DataAvailable
 CommModuleFunctions, 406
 NXCDefs.h, 1434
RS485Exit
 CommModuleFunctions, 407
 NXCDefs.h, 1434
RS485Init
 CommModuleFunctions, 407
 NXCDefs.h, 1435
RS485Read
 CommModuleFunctions, 408
 NXCDefs.h, 1435
RS485SendingData
 CommModuleFunctions, 408
 NXCDefs.h, 1435
RS485Status
 CommModuleFunctions, 408
 NXCDefs.h, 1436
RS485Uart
 CommModuleFunctions, 409
 NXCDefs.h, 1436
RS485Write
 CommModuleFunctions, 409
 NXCDefs.h, 1437
RunNRLLinkMacro
 MindSensorsAPI, 175
 NXCDefs.h, 1437
RunState
 NBCCCommon.h, 1045
 OutputFieldConstants, 653

S1
 InPorts, 204
 NXCDefs.h, 1170

s16
 NXCDefs.h, 1172
 TypeAliases, 203

S2
 InPorts, 206

 NXCDefs.h, 1172
S3
 InPorts, 206
 NXCDefs.h, 1172

s32
 NXCDefs.h, 1172
 TypeAliases, 203

S4
 InPorts, 206
 NXCDefs.h, 1172

s8
 NXCDefs.h, 1172
 TypeAliases, 204

SAMPLERATE_DEFAULT
 NBCCCommon.h, 1045
 SoundMisc, 601

SAMPLERATE_MAX
 NBCCCommon.h, 1045
 SoundMisc, 601

SAMPLERATE_MIN
 NBCCCommon.h, 1045
 SoundMisc, 601

ScaledArray
 ColorSensorReadType, 800

ScaledValue
 InputFieldConstants, 631
 NBCCCommon.h, 1046

SCHAR_MAX
 NBCCCommon.h, 1046
 NXTLimits, 793

SCHAR_MIN
 NBCCCommon.h, 1046
 NXTLimits, 793

Scout constants, 728
Scout light constants, 729
Scout light rule constants, 737
Scout mode constants, 734
Scout motion rule constants, 735
Scout sound constants, 730
Scout sound set constants, 733
Scout special effect constants, 739
Scout touch rule constants, 736
Scout transmit rule constants, 738

SCOUT_FXR_ALARM
 NBCCCommon.h, 1046
 ScoutSpecialEffectConstants, 740

SCOUT_FXR_BUG
 NBCCCommon.h, 1046
 ScoutSpecialEffectConstants, 740

SCOUT_FXR_NONE
 NBCCCommon.h, 1046
 ScoutSpecialEffectConstants, 740

SCOUT_FXR_RANDOM
 NBCCCommon.h, 1046
 ScoutSpecialEffectConstants, 740

SCOUT_FXR_SCIENCE
 NBCCCommon.h, 1046
 ScoutSpecialEffectConstants, 740

SCOUT_LIGHT_OFF
 NBCCCommon.h, 1046
 ScoutLightConstants, 729

SCOUT_LIGHT_ON
 NBCCCommon.h, 1047
 ScoutLightConstants, 729

SCOUT_LR_AVOID
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_LR_IGNORE
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_LR_OFF_WHEN
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_LR_SEEK_DARK
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_LR_SEEK_LIGHT
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_LR_WAIT_FOR
 NBCCCommon.h, 1047
 ScoutLightRuleConstants, 738

SCOUT_MODE_POWER
 NBCCCommon.h, 1047
 ScoutModeConstants, 734

SCOUT_MODE_STANDALONE
 NBCCCommon.h, 1048
 ScoutModeConstants, 734

SCOUT_MR_CIRCLE_LEFT
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 735

SCOUT_MR_CIRCLE_RIGHT
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 735

SCOUT_MR_FORWARD
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 735

SCOUT_MR_LOOP_A
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 735

SCOUT_MR_LOOP_AB
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 736

SCOUT_MR_LOOP_B
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 736

SCOUT_MR_NO_MOTION
 NBCCCommon.h, 1048
 ScoutMotionRuleConstants, 736

SCOUT_MR_ZIGZAG
 NBCCCommon.h, 1049
 ScoutMotionRuleConstants, 736

SCOUT SNDSET_ALARM
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 733

SCOUT SNDSET_BASIC
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 733

SCOUT SNDSET_BUG
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 733

SCOUT SNDSET_NONE
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 734

SCOUT SNDSET_RANDOM
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 734

SCOUT SNDSET_SCIENCE
 NBCCCommon.h, 1049
 ScoutSndSetConstants, 734

SCOUT_SOUND_1_BLINK
 NBCCCommon.h, 1049
 ScoutSoundConstants, 731

SCOUT_SOUND_2_BLINK
 NBCCCommon.h, 1049
 ScoutSoundConstants, 731

SCOUT_SOUND_COUNTER1
 NBCCCommon.h, 1049

ScoutSoundConstants, 731
SCOUT_SOUND_COUNTER2
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_ENTER_BRIGHT
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_ENTER_DARK
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_ENTER_NORMAL
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_ENTERSA
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_KEYERROR
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_MAIL_RECEIVED
 NBCCommon.h, 1050
 ScoutSoundConstants, 731
SCOUT_SOUND_NONE
 NBCCommon.h, 1050
 ScoutSoundConstants, 732
SCOUT_SOUND_REMOTE
 NBCCommon.h, 1050
 ScoutSoundConstants, 732
SCOUT_SOUND_SPECIAL1
 NBCCommon.h, 1050
 ScoutSoundConstants, 732
SCOUT_SOUND_SPECIAL2
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_SPECIAL3
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_TIMER1
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_TIMER2
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_TIMER3
 NBCCommon.h, 1051
 ScoutSoundConstants, 732

SCOUT_SOUND_TOUCH1_PRES
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_TOUCH1_REL
 NBCCommon.h, 1051
 ScoutSoundConstants, 732
SCOUT_SOUND_TOUCH2_PRES
 NBCCommon.h, 1051
 ScoutSoundConstants, 733
SCOUT_SOUND_TOUCH2_REL
 NBCCommon.h, 1051
 ScoutSoundConstants, 733
SCOUT_TGS_LONG
 NBCCommon.h, 1051
 ScoutTransmitRuleConstants, 739
SCOUT_TGS_MEDIUM
 NBCCommon.h, 1052
 ScoutTransmitRuleConstants, 739
SCOUT_TGS_SHORT
 NBCCommon.h, 1052
 ScoutTransmitRuleConstants, 739
SCOUT_TR_AVOID
 NBCCommon.h, 1052
 ScoutTouchRuleConstants, 737
SCOUT_TR_IGNORE
 NBCCommon.h, 1052
 ScoutTouchRuleConstants, 737
SCOUT_TR_OFF_WHEN
 NBCCommon.h, 1052
 ScoutTouchRuleConstants, 737
SCOUT_TR_REVERSE
 NBCCommon.h, 1052
 ScoutTouchRuleConstants, 737
SCOUT_TR_WAIT_FOR
 NBCCommon.h, 1052
 ScoutTouchRuleConstants, 737
ScoutLightConstants
 SCOUT_LIGHT_OFF, 729
 SCOUT_LIGHT_ON, 729
ScoutLightRuleConstants
 SCOUT_LR_AVOID, 738
 SCOUT_LR_IGNORE, 738
 SCOUT_LR_OFF_WHEN, 738
 SCOUT_LR_SEEK_DARK, 738
 SCOUT_LR_SEEK_LIGHT, 738
 SCOUT_LR_WAIT_FOR, 738

ScoutModeConstants
 SCOUT_MODE_POWER, [734](#)
 SCOUT_MODE_STANDALONE,
 [734](#)

ScoutMotionRuleConstants
 SCOUT_MR_CIRCLE_LEFT, [735](#)
 SCOUT_MR_CIRCLE_RIGHT,
 [735](#)
 SCOUT_MR_FORWARD, [735](#)
 SCOUT_MR_LOOP_A, [735](#)
 SCOUT_MR_LOOP_AB, [736](#)
 SCOUT_MR_LOOP_B, [736](#)
 SCOUT_MR_NO_MOTION, [736](#)
 SCOUT_MR_ZIGZAG, [736](#)

ScoutSndSetConstants
 SCOUT SNDSET_ALARM, [733](#)
 SCOUT SNDSET_BASIC, [733](#)
 SCOUT SNDSET_BUG, [733](#)
 SCOUT SNDSET_NONE, [734](#)
 SCOUT SNDSET_RANDOM, [734](#)
 SCOUT SNDSET_SCIENCE, [734](#)

ScoutSoundConstants
 SCOUT_SOUND_1_BLINK, [731](#)
 SCOUT_SOUND_2_BLINK, [731](#)
 SCOUT_SOUND_COUNTER1,
 [731](#)
 SCOUT_SOUND_COUNTER2,
 [731](#)
 SCOUT_SOUND_ENTER_-
 BRIGHT, [731](#)
 SCOUT_SOUND_ENTER_DARK,
 [731](#)
 SCOUT_SOUND_ENTER_-
 NORMAL, [731](#)
 SCOUT_SOUND_ENTERSA, [731](#)
 SCOUT_SOUND_KEYERROR,
 [731](#)
 SCOUT_SOUND_MAIL_-
 RECEIVED, [731](#)
 SCOUT_SOUND_NONE, [732](#)
 SCOUT_SOUND_REMOTE, [732](#)
 SCOUT_SOUND_SPECIAL1, [732](#)
 SCOUT_SOUND_SPECIAL2, [732](#)
 SCOUT_SOUND_SPECIAL3, [732](#)
 SCOUT_SOUND_TIMER1, [732](#)
 SCOUT_SOUND_TIMER2, [732](#)

SCOUT_SOUND_TIMER3, [732](#)
SCOUT_SOUND_TOUCH1_-
 PRES, [732](#)
SCOUT_SOUND_TOUCH1_REL,
 [732](#)
SCOUT_SOUND_TOUCH2_-
 PRES, [733](#)
SCOUT_SOUND_TOUCH2_REL,
 [733](#)

ScoutSpecialEffectConstants
 SCOUT_FXR_ALARM, [740](#)
 SCOUT_FXR_BUG, [740](#)
 SCOUT_FXR_NONE, [740](#)
 SCOUT_FXR_RANDOM, [740](#)
 SCOUT_FXR_SCIENCE, [740](#)

ScoutTouchRuleConstants
 SCOUT_TR_AVOID, [737](#)
 SCOUT_TR_IGNORE, [737](#)
 SCOUT_TR_OFF_WHEN, [737](#)
 SCOUT_TR_REVERSE, [737](#)
 SCOUT_TR_WAIT_FOR, [737](#)

ScoutTransmitRuleConstants
 SCOUT_TGS_LONG, [739](#)
 SCOUT_TGS_MEDIUM, [739](#)
 SCOUT_TGS_SHORT, [739](#)

SCREEN_BACKGROUND
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1052](#)

SCREEN_LARGE
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1053](#)

SCREEN_MODE_CLEAR
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1053](#)

SCREEN_MODE_RESTORE
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1053](#)

SCREEN_SMALL
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1053](#)

ScreenMode
 SetScreenModeType, [870](#)

SCREENS
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1053](#)

SEC_1

NBCCCommon.h, 1053
TimeConstants, 570

SEC_10
NBCCCommon.h, 1053
TimeConstants, 570

SEC_15
NBCCCommon.h, 1054
TimeConstants, 570

SEC_2
NBCCCommon.h, 1054
TimeConstants, 570

SEC_20
NBCCCommon.h, 1054
TimeConstants, 570

SEC_3
NBCCCommon.h, 1054
TimeConstants, 571

SEC_30
NBCCCommon.h, 1054
TimeConstants, 571

SEC_4
NBCCCommon.h, 1055
TimeConstants, 571

SEC_5
NBCCCommon.h, 1055
TimeConstants, 571

SEC_6
NBCCCommon.h, 1055
TimeConstants, 571

SEC_7
NBCCCommon.h, 1055
TimeConstants, 572

SEC_8
NBCCCommon.h, 1055
TimeConstants, 572

SEC_9
NBCCCommon.h, 1056
TimeConstants, 572

SEEK_CUR
fseekConstants, 517
NXCDefs.h, 1172

SEEK_END
fseekConstants, 517
NXCDefs.h, 1173

SEEK_SET
fseekConstants, 518

NXCDefs.h, 1173

SemData
ReadSemDataType, 869
WriteSemDataType, 881

SendingData
CommHSCheckStatusType, 808

SendMessage
CommModuleFunctions, 410
NXCDefs.h, 1438

SendRemoteBool
CommModuleFunctions, 410
NXCDefs.h, 1438

SendRemoteNumber
CommModuleFunctions, 411
NXCDefs.h, 1439

SendRemoteString
CommModuleFunctions, 411
NXCDefs.h, 1439

SendResponseBool
CommModuleFunctions, 412
NXCDefs.h, 1440

SendResponseNumber
CommModuleFunctions, 412
NXCDefs.h, 1440

SendResponseString
CommModuleFunctions, 413
NXCDefs.h, 1440

SendRS485Bool
CommModuleFunctions, 413
NXCDefs.h, 1441

SendRS485Number
CommModuleFunctions, 413
NXCDefs.h, 1441

SendRS485String
CommModuleFunctions, 414
NXCDefs.h, 1442

Sensor
InputModuleFunctions, 226
NXCDefs.h, 1442

Sensor mode constants, 210

Sensor type constants, 207

Sensor types and modes, 46

SENSOR_1
BasicSensorValues, 241
NXCDefs.h, 1173

SENSOR_2

BasicSensorValues, 241
 NXCDefs.h, 1173

SENSOR_3
 BasicSensorValues, 241
 NXCDefs.h, 1173

SENSOR_4
 BasicSensorValues, 241
 NXCDefs.h, 1173

SENSOR_CELSIUS
 NXCDefs.h, 1173
 SensorTypeModes, 213

SENSOR_COLORBLUE
 NXCDefs.h, 1173
 SensorTypeModes, 213

SENSOR_COLORMFULL
 NXCDefs.h, 1173
 SensorTypeModes, 213

SENSOR_COLORGREEN
 NXCDefs.h, 1174
 SensorTypeModes, 213

SENSOR_COLORNONE
 NXCDefs.h, 1174
 SensorTypeModes, 213

SENSOR_COLORED
 NXCDefs.h, 1174
 SensorTypeModes, 213

SENSOR_EDGE
 NXCDefs.h, 1174
 SensorTypeModes, 213

SENSOR_FAHRENHEIT
 NXCDefs.h, 1174
 SensorTypeModes, 213

SENSOR_LIGHT
 NXCDefs.h, 1174
 SensorTypeModes, 214

SENSOR_LOWSPEED
 NXCDefs.h, 1174
 SensorTypeModes, 214

SENSOR_LOWSPEED_9V
 NXCDefs.h, 1174
 SensorTypeModes, 214

SENSOR_MODE_BOOL
 NXCDefs.h, 1175
 SensorModes, 210

SENSOR_MODE_CELSIUS
 NXCDefs.h, 1175

 SensorModes, 210

SENSOR_MODE_EDGE
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_MODE_FAHRENHEIT
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_MODE_PERCENT
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_MODE_PULSE
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_MODE_RAW
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_MODE_ROTATION
 NXCDefs.h, 1175

 SensorModes, 211

SENSOR_NXTLIGHT
 NXCDefs.h, 1176

 SensorTypeModes, 214

SENSOR_PULSE
 NXCDefs.h, 1176

 SensorTypeModes, 214

SENSOR_ROTATION
 NXCDefs.h, 1176

 SensorTypeModes, 214

SENSOR_SOUND
 NXCDefs.h, 1176

 SensorTypeModes, 214

SENSOR_TOUCH
 NXCDefs.h, 1176

 SensorTypeModes, 214

SENSOR_TYPE_COLORBLUE
 NXCDefs.h, 1176

 SensorTypes, 208

SENSOR_TYPE_COLORMFULL
 NXCDefs.h, 1176

 SensorTypes, 208

SENSOR_TYPE_COLORGREEN
 NXCDefs.h, 1176

 SensorTypes, 208

SENSOR_TYPE_COLORNONE
 NXCDefs.h, 1177

 SensorTypes, 208

SENSOR_TYPE_COLORRED
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_CUSTOM
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_HIGHSPEED
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_LIGHT
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_LIGHT_ACTIVE
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_LIGHT_INACTIVE
 NXCDefs.h, 1177
 SensorTypes, 208

SENSOR_TYPE_LOWSPEED
 NXCDefs.h, 1177
 SensorTypes, 209

SENSOR_TYPE_LOWSPEED_9V
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_NONE
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_ROTATION
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_SOUND_DB
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_SOUND_DBA
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_TEMPERATURE
 NXCDefs.h, 1178
 SensorTypes, 209

SENSOR_TYPE_TOUCH
 NXCDefs.h, 1178
 SensorTypes, 209

SensorBoolean
 InputModuleFunctions, 227
 NXCDefs.h, 1443

SensorDigiPinsDirection
 InputModuleFunctions, 227
 NXCDefs.h, 1443

SensorDigiPinsOutputLevel
 InputModuleFunctions, 227
 NXCDefs.h, 1443

SensorDigiPinsStatus
 InputModuleFunctions, 228
 NXCDefs.h, 1444

SensorHTColorNum
 HiTechnicAPI, 102
 NXCDefs.h, 1444

SensorHTCompass
 HiTechnicAPI, 102
 NXCDefs.h, 1445

SensorHTEOPD
 HiTechnicAPI, 102
 NXCDefs.h, 1445

SensorHTGyro
 HiTechnicAPI, 103
 NXCDefs.h, 1445

SensorHTIRSeeker2ACDir
 HiTechnicAPI, 103
 NXCDefs.h, 1446

SensorHTIRSeeker2Addr
 HiTechnicAPI, 104
 NXCDefs.h, 1446

SensorHTIRSeeker2DCDir
 HiTechnicAPI, 104
 NXCDefs.h, 1447

SensorHTIRSeekerDir
 HiTechnicAPI, 105
 NXCDefs.h, 1447

SensorInvalid
 InputModuleFunctions, 228
 NXCDefs.h, 1448

SensorMode
 InputModuleFunctions, 229
 NXCDefs.h, 1448

SensorModes
 SENSOR_MODE_BOOL, 210
 SENSOR_MODE_CELSIUS, 210
 SENSOR_MODE_EDGE, 211
 SENSOR_MODE_FAHRENHEIT,
 211
 SENSOR_MODE_PERCENT, 211
 SENSOR_MODE_PULSE, 211

SENSOR_MODE_RAW, 211
SENSOR_MODE_ROTATION, 211
SensorMSCompass
 MindSensorsAPI, 175
 NXCDefs.h, 1448
SensorMSDROD
 MindSensorsAPI, 176
 NXCDefs.h, 1449
SensorMSPressure
 MindSensorsAPI, 176
 NXCDefs.h, 1449
SensorMSPressureRaw
 MindSensorsAPI, 176
 NXCDefs.h, 1450
SensorNormalized
 InputModuleFunctions, 229
 NXCDefs.h, 1450
SensorRaw
 InputModuleFunctions, 229
 NXCDefs.h, 1450
SensorScaled
 InputModuleFunctions, 230
 NXCDefs.h, 1451
SensorType
 InputModuleFunctions, 230
 NXCDefs.h, 1451
SensorTypeModes
 _SENSOR_CFG, 213
 SENSOR_CELSIUS, 213
 SENSOR_COLORBLUE, 213
 SENSOR_COLORFULL, 213
 SENSOR_COLORGREEN, 213
 SENSOR_COLORNONE, 213
 SENSOR_COLORRED, 213
 SENSOR_EDGE, 213
 SENSOR_FAHRENHEIT, 213
 SENSOR_LIGHT, 214
 SENSOR_LOWSPEED, 214
 SENSOR_LOWSPEED_9V, 214
 SENSOR_NXTLIGHT, 214
 SENSOR_PULSE, 214
 SENSOR_ROTATION, 214
 SENSOR_SOUND, 214
 SENSOR_TOUCH, 214
SensorTypes
 SENSOR_TYPE_COLORBLUE,
 208
 SENSOR_TYPE_COLORFULL,
 208
 SENSOR_TYPE_COLORGREEN,
 208
 SENSOR_TYPE_COLORNONE,
 208
 SENSOR_TYPE_COLORRED, 208
 SENSOR_TYPE_CUSTOM, 208
 SENSOR_TYPE_HIGHSPEED,
 208
 SENSOR_TYPE_LIGHT, 208
 SENSOR_TYPE_LIGHT_ACTIVE,
 208
 SENSOR_TYPE_LIGHT_-
 INACTIVE, 208
 SENSOR_TYPE_LOWSPEED, 209
 SENSOR_TYPE_LOWSPEED_9V,
 209
 SENSOR_TYPE_NONE, 209
 SENSOR_TYPE_ROTATION, 209
 SENSOR_TYPE_SOUND_DB, 209
 SENSOR_TYPE_SOUND_DBA,
 209
 SENSOR_TYPE_-
 TEMPERATURE, 209
 SENSOR_TYPE_TOUCH, 209
SensorUS
 LowSpeedModuleFunctions, 323
 NXCDefs.h, 1452
SensorValue
 InputModuleFunctions, 231
 NXCDefs.h, 1452
SensorValueBool
 InputModuleFunctions, 231
 NXCDefs.h, 1453
SensorValueRaw
 InputModuleFunctions, 232
 NXCDefs.h, 1453
SetAbortFlag
 NXCDefs.h, 1453
 UiModuleFunctions, 443
SetACCLNxSensitivity
 MindSensorsAPI, 177
 NXCDefs.h, 1454

SetBatteryState
 NXCDefs.h, 1454
 UiModuleFunctions, 444

SetBluetoothState
 NXCDefs.h, 1455
 UiModuleFunctions, 444

SetBTInputBuffer
 CommModuleFunctions, 414
 NXCDefs.h, 1455

SetBTInputBufferInPtr
 CommModuleFunctions, 415
 NXCDefs.h, 1455

SetBTInputBufferOutPtr
 CommModuleFunctions, 415
 NXCDefs.h, 1456

SetBTOutputBuffer
 CommModuleFunctions, 415
 NXCDefs.h, 1456

SetBTOutputBufferInPtr
 CommModuleFunctions, 416
 NXCDefs.h, 1456

SetBTOutputBufferOutPtr
 CommModuleFunctions, 416
 NXCDefs.h, 1457

SetButtonLongPressCount
 ButtonModuleFunctions, 435
 NXCDefs.h, 1457

SetButtonLongReleaseCount
 ButtonModuleFunctions, 435
 NXCDefs.h, 1457

SetButtonModuleValue
 CommandModuleFunctions, 349
 NXCDefs.h, 1458

SetButtonPressCount
 ButtonModuleFunctions, 436
 NXCDefs.h, 1458

SetButtonReleaseCount
 ButtonModuleFunctions, 436
 NXCDefs.h, 1458

SetButtonShortReleaseCount
 ButtonModuleFunctions, 436
 NXCDefs.h, 1459

SetButtonState
 ButtonModuleFunctions, 437
 NXCDefs.h, 1459

SetCommandFlags
 NXCDefs.h, 1459
 UiModuleFunctions, 444

SetCommandModuleBytes
 CommandModuleFunctions, 349
 NXCDefs.h, 1460

SetCommandModuleValue
 CommandModuleFunctions, 350
 NXCDefs.h, 1460

SetCommModuleBytes
 CommandModuleFunctions, 350
 NXCDefs.h, 1461

SetCommModuleValue
 CommandModuleFunctions, 351
 NXCDefs.h, 1461

SetCustomSensorActiveStatus
 InputModuleFunctions, 232
 NXCDefs.h, 1461

SetCustomSensorPercentFullScale
 InputModuleFunctions, 232
 NXCDefs.h, 1462

SetCustomSensorZeroOffset
 InputModuleFunctions, 233
 NXCDefs.h, 1462

SetDisplayContrast
 DisplayModuleFunctions, 287
 NXCDefs.h, 1462

SetDisplayDisplay
 DisplayModuleFunctions, 287
 NXCDefs.h, 1463

SetDisplayEraseMask
 DisplayModuleFunctions, 288
 NXCDefs.h, 1463

SetDisplayFlags
 DisplayModuleFunctions, 288
 NXCDefs.h, 1463

SetDisplayFont
 DisplayModuleFunctions, 288
 NXCDefs.h, 1464

SetDisplayModuleBytes
 CommandModuleFunctions, 351
 NXCDefs.h, 1464

SetDisplayModuleValue
 CommandModuleFunctions, 351
 NXCDefs.h, 1464

SetDisplayNormal
 DisplayModuleFunctions, 289

NXCDefs.h, 1465
SetDisplayPopup
 DisplayModuleFunctions, 289
 NXCDefs.h, 1465
SetDisplayTextLinesCenterFlags
 DisplayModuleFunctions, 290
 NXCDefs.h, 1466
SetDisplayUpdateMask
 DisplayModuleFunctions, 290
 NXCDefs.h, 1466
SetHSFlags
 CommModuleFunctions, 416
 NXCDefs.h, 1466
SetHSInputBuffer
 CommModuleFunctions, 417
 NXCDefs.h, 1467
SetHSInputBufferInPtr
 CommModuleFunctions, 417
 NXCDefs.h, 1467
SetHSInputBufferOutPtr
 CommModuleFunctions, 417
 NXCDefs.h, 1467
SetHSMode
 CommModuleFunctions, 418
 NXCDefs.h, 1468
SetHSOutputBuffer
 CommModuleFunctions, 418
 NXCDefs.h, 1468
SetHSOutputBufferInPtr
 CommModuleFunctions, 418
 NXCDefs.h, 1468
SetHSOutputBufferOutPtr
 CommModuleFunctions, 419
 NXCDefs.h, 1469
SetHSSpeed
 CommModuleFunctions, 419
 NXCDefs.h, 1469
SetHSState
 CommModuleFunctions, 419
 NXCDefs.h, 1469
SetHTColor2Mode
 HiTechnicAPI, 105
 NXCDefs.h, 1470
SetHTIRSeeker2Mode
 HiTechnicAPI, 105
 NXCDefs.h, 1470

SetInputModule
 InputModuleFunctions, 233
 NXCDefs.h, 1471
SetInputModuleValue
 CommandModuleFunctions, 352
 NXCDefs.h, 1471
SetIOCctrlModuleValue
 CommandModuleFunctions, 352
 NXCDefs.h, 1471
SetIOMapBytes
 CommandModuleFunctions, 352
 NXCDefs.h, 1472
SetIOMapBytesByID
 CommandModuleFunctions, 353
 NXCDefs.h, 1472
SetIOMapValue
 CommandModuleFunctions, 353
 NXCDefs.h, 1473
SetIOMapValueByID
 CommandModuleFunctions, 354
 NXCDefs.h, 1473
SetLoaderModuleValue
 CommandModuleFunctions, 354
 NXCDefs.h, 1474
SetLongAbort
 NXCDefs.h, 1474
 UiModuleFunctions, 445
SetLowSpeedModuleBytes
 CommandModuleFunctions, 355
 NXCDefs.h, 1474
SetLowSpeedModuleValue
 CommandModuleFunctions, 355
 NXCDefs.h, 1475
SetMotorPwnFreq
 NXCDefs.h, 1475
 OutputModuleFunctions, 268
SetOnBrickProgramPointer
 NXCDefs.h, 1476
 UiModuleFunctions, 445
SetOutput
 NXCDefs.h, 1476
 OutputModuleFunctions, 268
SetOutputModuleValue
 CommandModuleFunctions, 356
 NXCDefs.h, 1476
SetScreenMode

NBCCCommon.h, 1056
SysCallConstants, 560
SetScreenModeType, 869
 Result, 870
 ScreenMode, 870
SetSensor
 InputModuleFunctions, 234
 NXCDefs.h, 1477
SetSensorBoolean
 InputModuleFunctions, 234
 NXCDefs.h, 1477
SetSensorColorBlue
 InputModuleFunctions, 234
 NXCDefs.h, 1478
SetSensorColorFull
 InputModuleFunctions, 235
 NXCDefs.h, 1478
SetSensorColorGreen
 InputModuleFunctions, 235
 NXCDefs.h, 1478
SetSensorColorNone
 InputModuleFunctions, 236
 NXCDefs.h, 1479
SetSensorColorRed
 InputModuleFunctions, 236
 NXCDefs.h, 1479
SetSensorDigiPinsDirection
 InputModuleFunctions, 236
 NXCDefs.h, 1480
SetSensorDigiPinsOutputLevel
 InputModuleFunctions, 237
 NXCDefs.h, 1480
SetSensorDigiPinsStatus
 InputModuleFunctions, 237
 NXCDefs.h, 1480
SetSensorHTEOPD
 HiTechnicAPI, 106
 NXCDefs.h, 1481
SetSensorHTGyro
 HiTechnicAPI, 106
 NXCDefs.h, 1481
SetSensorLight
 InputModuleFunctions, 237
 NXCDefs.h, 1481
SetSensorLowspeed
 InputModuleFunctions, 238
NXCDefs.h, 1482
SetSensorMode
 InputModuleFunctions, 238
 NXCDefs.h, 1482
SetSensorMSDROD
 MindSensorsAPI, 177
 NXCDefs.h, 1483
SetSensorMSPressure
 MindSensorsAPI, 178
 NXCDefs.h, 1483
SetSensorSound
 InputModuleFunctions, 239
 NXCDefs.h, 1483
SetSensorTouch
 InputModuleFunctions, 239
 NXCDefs.h, 1484
SetSensorType
 InputModuleFunctions, 239
 NXCDefs.h, 1484
SetSleepTime
 NXCDefs.h, 1485
 UiModuleFunctions, 445
SetSleepTimeout
 NXCDefs.h, 1485
 UiModuleFunctions, 446
SetSleepTimeoutType, 870
 Result, 871
 TheSleepTimeoutMS, 871
SetSleepTimeoutVal
 NBCCCommon.h, 1056
 SysCallConstants, 560
SetSleepTimer
 NXCDefs.h, 1485
 UiModuleFunctions, 446
SetSoundDuration
 NXCDefs.h, 1486
 SoundModuleFunctions, 301
SetSoundFlags
 NXCDefs.h, 1486
 SoundModuleFunctions, 301
SetSoundFrequency
 NXCDefs.h, 1486
 SoundModuleFunctions, 302
SetSoundMode
 NXCDefs.h, 1487
 SoundModuleFunctions, 302

SetSoundModuleState
 [NXCDefs.h, 1487](#)
 [SoundModuleFunctions, 303](#)

SetSoundModuleValue
 [CommandModuleFunctions, 356](#)
 [NXCDefs.h, 1488](#)

SetSoundSampleRate
 [NXCDefs.h, 1488](#)
 [SoundModuleFunctions, 303](#)

SetSoundVolume
 [NXCDefs.h, 1488](#)
 [SoundModuleFunctions, 303](#)

SetUIButton
 [NXCDefs.h, 1489](#)
 [UiModuleFunctions, 446](#)

SetUIModuleValue
 [CommandModuleFunctions, 356](#)
 [NXCDefs.h, 1489](#)

SetUIState
 [NXCDefs.h, 1489](#)
 [UiModuleFunctions, 447](#)

SetUSBInputBuffer
 [CommModuleFunctions, 420](#)
 [NXCDefs.h, 1490](#)

SetUSBInputBufferInPtr
 [CommModuleFunctions, 420](#)
 [NXCDefs.h, 1490](#)

SetUSBInputBufferOutPtr
 [CommModuleFunctions, 420](#)
 [NXCDefs.h, 1490](#)

SetUSBOutputBuffer
 [CommModuleFunctions, 421](#)
 [NXCDefs.h, 1491](#)

SetUSBOutputBufferInPtr
 [CommModuleFunctions, 421](#)
 [NXCDefs.h, 1491](#)

SetUSBOutputBufferOutPtr
 [CommModuleFunctions, 421](#)
 [NXCDefs.h, 1491](#)

SetUSBPollBuffer
 [CommModuleFunctions, 422](#)
 [NXCDefs.h, 1492](#)

SetUSBPollBufferInPtr
 [CommModuleFunctions, 422](#)
 [NXCDefs.h, 1492](#)

SetUSBPollBufferOutPtr
 [DrawCircleType, 825](#)

[CommModuleFunctions, 422](#)
 [NXCDefs.h, 1492](#)

SetUSBState
 [CommModuleFunctions, 423](#)
 [NXCDefs.h, 1493](#)

SetVMRunState
 [NXCDefs.h, 1493](#)
 [UiModuleFunctions, 447](#)

SetVolume
 [NXCDefs.h, 1493](#)
 [UiModuleFunctions, 447](#)

SHRT_MAX
 [NBCCommon.h, 1056](#)
 [NXTLimits, 793](#)

SHRT_MIN
 [NBCCommon.h, 1056](#)
 [NXTLimits, 793](#)

sign
 [cmathAPI, 502](#)
 [NXCDefs.h, 1493](#)

Sin
 [cmathAPI, 487](#)
 [NXCDefs.h, 1178](#)

sin
 [cmathAPI, 502](#)
 [NXCDefs.h, 1494](#)

SinD
 [cmathAPI, 487](#)
 [NXCDefs.h, 1179](#)

sind
 [cmathAPI, 503](#)
 [NXCDefs.h, 1494](#)

Sinh
 [cmathAPI, 488](#)
 [NXCDefs.h, 1179](#)

sinh
 [cmathAPI, 503](#)
 [NXCDefs.h, 1495](#)

SinhD
 [cmathAPI, 488](#)
 [NXCDefs.h, 1180](#)

sinhd
 [cmathAPI, 504](#)
 [NXCDefs.h, 1495](#)

Size

DrawRectType, 836
SIZE_OF_BDADDR
 CommMiscConstants, 695
 NBCCommon.h, 1056
SIZE_OF_BRICK_NAME
 CommMiscConstants, 695
 NBCCommon.h, 1056
SIZE_OF_BT_CONNECT_TABLE
 CommMiscConstants, 696
 NBCCommon.h, 1056
SIZE_OF_BT_DEVICE_TABLE
 CommMiscConstants, 696
 NBCCommon.h, 1056
SIZE_OF_BT_NAME
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_BT_PINCODE
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_BTBUF
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_CLASS_OF_DEVICE
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_HSBUF
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_USBBUF
 CommMiscConstants, 696
 NBCCommon.h, 1057
SIZE_OF_USBDATA
 CommMiscConstants, 696
 NBCCommon.h, 1057
SizeType, 871
 Height, 872
 Width, 872
SizeX
 DrawEllipseType, 826
SizeY
 DrawEllipseType, 826
SleepNow
 IOCtrlModuleFunctions, 373
 NXCDefs.h, 1496
SleepTime
 NXCDefs.h, 1496
UiModuleFunctions, 448
SleepTimeout
 NXCDefs.h, 1496
 UiModuleFunctions, 448
SleepTimer
 NXCDefs.h, 1497
 UiModuleFunctions, 448
Sound module, 55
Sound module constants, 596
Sound module functions, 297
Sound module IOMAP offsets, 599
Sound module miscellaneous constants, 600
Sound module types, 296
SOUND_CLICK
 NBCCommon.h, 1057
 RCXSoundConstants, 727
SOUND_DOUBLE_BEEP
 NBCCommon.h, 1057
 RCXSoundConstants, 727
SOUND_DOWN
 NBCCommon.h, 1057
 RCXSoundConstants, 727
SOUND_FAST_UP
 NBCCommon.h, 1058
 RCXSoundConstants, 727
SOUND_FLAGS_IDLE
 NBCCommon.h, 1058
 SoundFlagsConstants, 597
SOUND_FLAGS_RUNNING
 NBCCommon.h, 1058
 SoundFlagsConstants, 597
SOUND_FLAGS_UPDATE
 NBCCommon.h, 1058
 SoundFlagsConstants, 597
SOUND_LOW_BEEP
 NBCCommon.h, 1058
 RCXSoundConstants, 728
SOUND_MODE_LOOP
 NBCCommon.h, 1058
 SoundModeConstants, 599
SOUND_MODE_ONCE
 NBCCommon.h, 1059
 SoundModeConstants, 599
SOUND_MODE_TONE
 NBCCommon.h, 1059

SoundModeConstants, 599
SOUND_STATE_FILE
 NBCCommon.h, 1059
 SoundStateConstants, 598
SOUND_STATE_IDLE
 NBCCommon.h, 1059
 SoundStateConstants, 598
SOUND_STATE_STOP
 NBCCommon.h, 1059
 SoundStateConstants, 598
SOUND_STATE_TONE
 NBCCommon.h, 1059
 SoundStateConstants, 598
SOUND_UP
 NBCCommon.h, 1059
 RCXSoundConstants, 728
SoundDuration
 NXCDefs.h, 1497
 SoundModuleFunctions, 304
SoundFlags
 NXCDefs.h, 1497
 SoundModuleFunctions, 304
SoundFlags constants, 596
SoundFlagsConstants
 SOUND_FLAGS_IDLE, 597
 SOUND_FLAGS_RUNNING, 597
 SOUND_FLAGS_UPDATE, 597
SoundFrequency
 NXCDefs.h, 1498
 SoundModuleFunctions, 305
SoundGetState
 NBCCommon.h, 1060
 SysCallConstants, 561
SoundGetType, 872
 Flags, 873
 State, 873
SoundIOMAP
 SoundOffsetDuration, 599
 SoundOffsetFlags, 599
 SoundOffsetFreq, 600
 SoundOffsetMode, 600
 SoundOffsetSampleRate, 600
 SoundOffsetSoundFilename, 600
 SoundOffsetState, 600
 SoundOffsetVolume, 600
SoundLevel
 SoundPlayFileType, 874
 SoundPlayToneType, 876
SoundMisc
 FREQUENCY_MAX, 601
 FREQUENCY_MIN, 601
 SAMPLERATE_DEFAULT, 601
 SAMPLERATE_MAX, 601
 SAMPLERATE_MIN, 601
SoundMode
 NXCDefs.h, 1498
 SoundModuleFunctions, 305
SoundMode constants, 598
SoundModeConstants
 SOUND_MODE_LOOP, 599
 SOUND_MODE_ONCE, 599
 SOUND_MODE_TONE, 599
SoundModuleFunctions
 PlayFile, 299
 PlayFileEx, 299
 PlaySound, 299
 PlayTone, 300
 PlayToneEx, 300
 PlayTones, 301
 SetSoundDuration, 301
 SetSoundFlags, 301
 SetSoundFrequency, 302
 SetSoundMode, 302
 SetSoundModuleState, 303
 SetSoundSampleRate, 303
 SetSoundVolume, 303
 SoundDuration, 304
 SoundFlags, 304
 SoundFrequency, 305
 SoundMode, 305
 SoundSampleRate, 305
 SoundState, 306
 SoundVolume, 306
 StopSound, 307
 SysSoundGetState, 307
 SysSoundPlayFile, 307
 SysSoundPlayTone, 308
 SysSoundSetState, 308
SoundModuleID
 ModuleIDConstants, 191
 NBCCommon.h, 1060
SoundModuleName

ModuleNameConstants, 189
NBCCCommon.h, 1060

SoundOffsetDuration
NBCCCommon.h, 1060
SoundIOMAP, 599

SoundOffsetFlags
NBCCCommon.h, 1060
SoundIOMAP, 599

SoundOffsetFreq
NBCCCommon.h, 1060
SoundIOMAP, 600

SoundOffsetMode
NBCCCommon.h, 1060
SoundIOMAP, 600

SoundOffsetSampleRate
NBCCCommon.h, 1061
SoundIOMAP, 600

SoundOffsetSoundFilename
NBCCCommon.h, 1061
SoundIOMAP, 600

SoundOffsetState
NBCCCommon.h, 1061
SoundIOMAP, 600

SoundOffsetVolume
NBCCCommon.h, 1061
SoundIOMAP, 600

SoundPlayFile
NBCCCommon.h, 1061
SysCallConstants, 561

SoundPlayFileType, 873
Filename, 874
Loop, 874
Result, 874
SoundLevel, 874

SoundPlayTone
NBCCCommon.h, 1061
SysCallConstants, 561

SoundPlayToneType, 875
Duration, 875
Frequency, 876
Loop, 876
Result, 876
SoundLevel, 876

SoundSampleRate
NXCDefs.h, 1498
SoundModuleFunctions, 305

SoundSetState
NBCCCommon.h, 1061
SysCallConstants, 561

SoundSetStateType, 876
Flags, 877
Result, 877
State, 877

SoundState
NXCDdefs.h, 1499
SoundModuleFunctions, 306

SoundState constants, 597

SoundStateConstants
SOUND_STATE_FILE, 598
SOUND_STATE_IDLE, 598
SOUND_STATE_STOP, 598
SOUND_STATE_TONE, 598

SoundVolume
NXCDdefs.h, 1499
SoundModuleFunctions, 306

SPECIALS
DisplayModuleConstants, 678
NBCCCommon.h, 1061

sprintf
cstdioAPI, 516
NXCDdefs.h, 1500

Sqrt
cmathAPI, 489
NXCDdefs.h, 1180

sqrt
cmathAPI, 504
NXCDdefs.h, 1500

Standard I2C constants, 666

Standard-C API functions, 194

StartLoc
DrawLineType, 832

StartTask
CommandModuleFunctions, 357
NXCDdefs.h, 1500

STAT_COMM_PENDING
CommandModuleConstants, 52
NBCCCommon.h, 1061

STAT_MSG_EMPTY_MAILBOX
CommandModuleConstants, 52
NBCCCommon.h, 1062

State
SoundGetStateType, 873

SoundSetStateType, [877](#)
Status
 CommHSReadWriteType, [810](#)
 DisplayExecuteFunctionType, [822](#)
STATUSICON_BATTERY
 DisplayModuleConstants, [678](#)
 NBCCommon.h, [1062](#)
STATUSICON_BLUETOOTH
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STATUSICON_USB
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STATUSICON_VM
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STATUSICONS
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STATUSTEXT
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STEPICON_1
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STEPICON_2
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STEPICON_3
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1062](#)
STEPICON_4
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1063](#)
STEPICON_5
 DisplayModuleConstants, [679](#)
 NBCCommon.h, [1063](#)
STEPICONS
 DisplayModuleConstants, [680](#)
 NBCCommon.h, [1063](#)
STEPLINE
 DisplayModuleConstants, [680](#)
 NBCCommon.h, [1063](#)
Stop
 CommandModuleFunctions, [357](#)
 NXCDefs.h, [1501](#)
STOP_REQ
 CommandVMState, [575](#)
 NBCCommon.h, [1063](#)
StopAllTasks
 CommandModuleFunctions, [357](#)
 NXCDefs.h, [1501](#)
StopSound
 NXCDefs.h, [1501](#)
 SoundModuleFunctions, [307](#)
StopTask
 CommandModuleFunctions, [357](#)
 NXCDefs.h, [1502](#)
StrCat
 cstringAPI, [538](#)
 NXCDefs.h, [1503](#)
strcat
 cstringAPI, [537](#)
 NXCDefs.h, [1502](#)
strcmp
 cstringAPI, [538](#)
 NXCDefs.h, [1503](#)
strcpy
 cstringAPI, [539](#)
 NXCDefs.h, [1503](#)
StrIndex
 cstringAPI, [539](#)
 NXCDefs.h, [1504](#)
StrLen
 cstringAPI, [540](#)
 NXCDefs.h, [1505](#)
strlen
 cstringAPI, [540](#)
 NXCDefs.h, [1504](#)
strncat
 cstringAPI, [540](#)
 NXCDefs.h, [1505](#)
strncmp
 cstringAPI, [541](#)
 NXCDefs.h, [1506](#)
strncpy
 cstringAPI, [541](#)
 NXCDefs.h, [1506](#)
StrReplace
 cstringAPI, [542](#)
 NXCDefs.h, [1507](#)
StrToByteArray

cstringAPI, 542
 NXCDefs.h, 1507

strtod
 cstdlibAPI, 524
 NXCDefs.h, 1508

strtol
 cstdlibAPI, 525
 NXCDefs.h, 1508

StrToNum
 cstringAPI, 543
 NXCDefs.h, 1509

strtoul
 cstdlibAPI, 526
 NXCDefs.h, 1510

SubStr
 cstringAPI, 543
 NXCDefs.h, 1510

SyncTick
 DatalogGetTimesType, 819

SyncTime
 DatalogGetTimesType, 819

SysCall
 CommandModuleFunctions, 358
 NXCDefs.h, 1511

SysCallConstants
 ColorSensorRead, 554
 CommBTCheckStatus, 554
 CommBTConnection, 554
 CommBTOnOff, 555
 CommBTRead, 555
 CommBTWrite, 555
 CommExecuteFunction, 555
 CommHSCheckStatus, 555
 CommHSControl, 555
 CommHSRead, 555
 CommHSWrite, 555
 CommLSCheckStatus, 555
 CommLSRead, 555
 CommLSWrite, 556
 CommLSWriteEx, 556
 ComputeCalibValue, 556
 DatalogGetTimes, 556
 DatalogWrite, 556
 DisplayExecuteFunction, 556
 DrawCircle, 556
 DrawEllipse, 556

 DrawFont, 556
 DrawGraphic, 556
 DrawGraphicArray, 557
 DrawLine, 557
 DrawPoint, 557
 DrawPolygon, 557
 DrawRect, 557
 DrawText, 557
 FileClose, 557
 FileDelete, 557
 FileFindFirst, 558
 FileFindNext, 558
 FileOpenAppend, 558
 FileOpenRead, 558
 FileOpenReadLinear, 558
 FileOpenWrite, 558
 FileOpenWriteLinear, 558
 FileOpenWriteNonLinear, 558
 FileRead, 558
 FileRename, 558
 FileResize, 559
 FileResolveHandle, 559
 FileSeek, 559
 FileWrite, 559
 GetStartTick, 559
 IOMapRead, 559
 IOMapReadByID, 559
 IOMapWrite, 559
 IOMapWriteByID, 559
 KeepAlive, 560
 ListFiles, 560
 LoaderExecuteFunction, 560
 MessageRead, 560
 MessageWrite, 560
 RandomNumber, 560
 ReadButton, 560
 ReadSemData, 560
 SetScreenMode, 560
 SetSleepTimeoutVal, 560
 SoundGetState, 561
 SoundPlayFile, 561
 SoundPlayTone, 561
 SoundSetState, 561
 UpdateCalibCacheInfo, 561
 WriteSemData, 561

SysColorSensorRead

- InputModuleFunctions, 240
 NXCDefs.h, 1511
- SysCommBTCheckStatus
 - CommModuleFunctions, 423
 NXCDefs.h, 1512
- SysCommBTConnection
 - CommModuleFunctions, 423
 NXCDefs.h, 1512
- SysCommBTOnOff
 - CommModuleFunctions, 424
 NXCDefs.h, 1513
- SysCommBTWrite
 - CommModuleFunctions, 424
 NXCDefs.h, 1513
- SysCommExecuteFunction
 - CommModuleFunctions, 425
 NXCDefs.h, 1513
- SysCommHSCheckStatus
 - CommModuleFunctions, 425
 NXCDefs.h, 1514
- SysCommHSControl
 - CommModuleFunctions, 425
 NXCDefs.h, 1514
- SysCommHSRead
 - CommModuleFunctions, 426
 NXCDefs.h, 1515
- SysCommHSWrite
 - CommModuleFunctions, 426
 NXCDefs.h, 1515
- SysCommLSCheckStatus
 - LowSpeedModuleSystemCallFunctions, 331
 NXCDefs.h, 1516
- SysCommLSRead
 - LowSpeedModuleSystemCallFunctions, 332
 NXCDefs.h, 1516
- SysCommLSWrite
 - LowSpeedModuleSystemCallFunctions, 332
 NXCDefs.h, 1516
- SysCommLSWriteEx
 - LowSpeedModuleSystemCallFunctions, 332
 NXCDefs.h, 1517
- SysComputeCalibValue
- CommandModuleFunctions, 358
 NXCDefs.h, 1517
- SysDatalogGetTimes
 - CommandModuleFunctions, 359
 NXCDefs.h, 1518
- SysDatalogWrite
 - CommandModuleFunctions, 359
 NXCDefs.h, 1518
- SysDisplayExecuteFunction
 - DisplayModuleFunctions, 290
 NXCDefs.h, 1518
- SysDrawCircle
 - DisplayModuleFunctions, 291
 NXCDefs.h, 1519
- SysDrawEllipse
 - DisplayModuleFunctions, 291
 NXCDefs.h, 1519
- SysDrawFont
 - DisplayModuleFunctions, 292
 NXCDefs.h, 1520
- SysDrawGraphic
 - DisplayModuleFunctions, 292
 NXCDefs.h, 1520
- SysDrawGraphicArray
 - DisplayModuleFunctions, 292
 NXCDefs.h, 1520
- SysDrawLine
 - DisplayModuleFunctions, 293
 NXCDefs.h, 1521
- SysDrawPoint
 - DisplayModuleFunctions, 293
 NXCDefs.h, 1521
- SysDrawPolygon
 - DisplayModuleFunctions, 293
 NXCDefs.h, 1521
- SysDrawRect
 - DisplayModuleFunctions, 294
 NXCDefs.h, 1522
- SysDrawText
 - DisplayModuleFunctions, 294
 NXCDefs.h, 1522
- SysFileClose
 - LoaderModuleFunctions, 464
 NXCDefs.h, 1522
- SysFileDelete
 - LoaderModuleFunctions, 464

NXCDefs.h, 1523
SysFileFindFirst
 LoaderModuleFunctions, 465
 NXCDefs.h, 1523
SysFileFindNext
 LoaderModuleFunctions, 465
 NXCDefs.h, 1523
SysFileOpenAppend
 LoaderModuleFunctions, 466
 NXCDefs.h, 1524
SysFileOpenRead
 LoaderModuleFunctions, 466
 NXCDefs.h, 1524
SysFileOpenReadLinear
 LoaderModuleFunctions, 466
 NXCDefs.h, 1525
SysFileOpenWrite
 LoaderModuleFunctions, 467
 NXCDefs.h, 1525
SysFileOpenWriteLinear
 LoaderModuleFunctions, 467
 NXCDefs.h, 1525
SysFileOpenWriteNonLinear
 LoaderModuleFunctions, 467
 NXCDefs.h, 1526
SysFileRead
 LoaderModuleFunctions, 468
 NXCDefs.h, 1526
SysFileRename
 LoaderModuleFunctions, 468
 NXCDefs.h, 1526
SysFileResize
 LoaderModuleFunctions, 469
 NXCDefs.h, 1527
SysFileResolveHandle
 LoaderModuleFunctions, 469
 NXCDefs.h, 1527
SysFileSeek
 LoaderModuleFunctions, 469
 NXCDefs.h, 1528
SysFileWrite
 LoaderModuleFunctions, 470
 NXCDefs.h, 1528
SysGetStartTick
 CommandModuleFunctions, 360
 NXCDefs.h, 1528
SysIOMapRead
 CommandModuleFunctions, 360
 NXCDefs.h, 1529
SysIOMapReadByID
 CommandModuleFunctions, 360
 NXCDefs.h, 1529
SysIOMapWrite
 CommandModuleFunctions, 361
 NXCDefs.h, 1529
SysIOMapWriteByID
 CommandModuleFunctions, 361
 NXCDefs.h, 1530
SysKeepAlive
 CommandModuleFunctions, 362
 NXCDefs.h, 1530
SysListFiles
 LoaderModuleFunctions, 470
 NXCDefs.h, 1531
SysLoaderExecuteFunction
 LoaderModuleFunctions, 470
 NXCDefs.h, 1531
SysMessageRead
 CommModuleFunctions, 427
 NXCDefs.h, 1531
SysMessageWrite
 CommModuleFunctions, 427
 NXCDefs.h, 1532
SysRandomNumber
 cstdlibAPI, 526
 NXCDefs.h, 1532
SysReadButton
 ButtonModuleFunctions, 437
 NXCDefs.h, 1532
SysReadSemData
 CommandModuleFunctions, 362
 NXCDefs.h, 1533
SysSetScreenMode
 DisplayModuleFunctions, 294
 NXCDefs.h, 1533
SysSetSleepTimeout
 NXCDefs.h, 1533
 UiModuleFunctions, 449
SysSoundGetState
 NXCDefs.h, 1534
 SoundModuleFunctions, 307
SysSoundPlayFile

NXCDefs.h, 1534
SoundModuleFunctions, 307

SysSoundPlayTone
 NXCDefs.h, 1535
 SoundModuleFunctions, 308

SysSoundSetState
 NXCDefs.h, 1535
 SoundModuleFunctions, 308

System Call function constants, 552

SysUpdateCalibCacheInfo
 CommandModuleFunctions, 362
 NXCDefs.h, 1535

SysWriteSemData
 CommandModuleFunctions, 363
 NXCDefs.h, 1536

TachoCount
 NBCCCommon.h, 1063
 OutputFieldConstants, 653

TachoLimit
 NBCCCommon.h, 1063
 OutputFieldConstants, 654

Tachometer counter reset flags, 643

TachoResetConstants
 RESET_ALL, 644
 RESET_BLOCK_COUNT, 644
 RESET_BLOCKANDTACHO, 644
 RESET_COUNT, 644
 RESET_NONE, 644
 RESET_ROTATION_COUNT, 645

Tan
 cmathAPI, 489
 NXCDefs.h, 1180

tan
 cmathAPI, 504
 NXCDefs.h, 1536

TanD
 cmathAPI, 489
 NXCDefs.h, 1181

tand
 cmathAPI, 505
 NXCDefs.h, 1537

Tanh
 cmathAPI, 490
 NXCDefs.h, 1181

tanh
 cmathAPI, 505
 NXCDefs.h, 1537

cmathAPI, 505
NXCDefs.h, 1537

TanhD
 cmathAPI, 490
 NXCDefs.h, 1182

tanhd
 cmathAPI, 506
 NXCDefs.h, 1537

TEMP_FQ_1
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_FQ_2
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_FQ_4
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_FQ_6
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_OS_ONESHOT
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_POL_HIGH
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_POL_LOW
 NBCCCommon.h, 1064
 TempI2CConstants, 671

TEMP_REG_CONFIG
 NBCCCommon.h, 1064
 TempI2CConstants, 672

TEMP_REG_TEMP
 NBCCCommon.h, 1064
 TempI2CConstants, 672

TEMP_REG_THIGH
 NBCCCommon.h, 1065
 TempI2CConstants, 672

TEMP_REG_TLOW
 NBCCCommon.h, 1065
 TempI2CConstants, 672

TEMP_RES_10BIT
 NBCCCommon.h, 1065
 TempI2CConstants, 672

TEMP_RES_11BIT
 NBCCCommon.h, 1065

TempI2CConstants, 672
TEMP_RES_12BIT
 NBCCCommon.h, 1065
 TempI2CConstants, 672
TEMP_RES_9BIT
 NBCCCommon.h, 1065
 TempI2CConstants, 672
TEMP_SD_CONTINUOUS
 NBCCCommon.h, 1065
 TempI2CConstants, 672
TEMP_SD_SHUTDOWN
 NBCCCommon.h, 1065
 TempI2CConstants, 672
TEMP_TM_COMPARATOR
 NBCCCommon.h, 1065
 TempI2CConstants, 673
TEMP_TM_INTERRUPT
 NBCCCommon.h, 1065
 TempI2CConstants, 673
TempI2CConstants
 TEMP_FQ_1, 671
 TEMP_FQ_2, 671
 TEMP_FQ_4, 671
 TEMP_FQ_6, 671
 TEMP_OS_ONESHOT, 671
 TEMP_POL_HIGH, 671
 TEMP_POL_LOW, 671
 TEMP_REG_CONFIG, 672
 TEMP_REG_TEMP, 672
 TEMP_REG_THIGH, 672
 TEMP_REG_TLOW, 672
 TEMP_RES_10BIT, 672
 TEMP_RES_11BIT, 672
 TEMP_RES_12BIT, 672
 TEMP_RES_9BIT, 672
 TEMP_SD_CONTINUOUS, 672
 TEMP_SD_SHUTDOWN, 672
 TEMP_TM_COMPARATOR, 673
 TEMP_TM_INTERRUPT, 673
Text
 DrawFontType, 828
 DrawTextType, 838
Text line constants, 689
TEXTLINE_1
 DisplayTextLineConstants, 689
 NBCCCommon.h, 1066
TEXTLINE_2
 DisplayTextLineConstants, 689
 NBCCCommon.h, 1066
TEXTLINE_3
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINE_4
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINE_5
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINE_6
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINE_7
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINE_8
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TEXTLINES
 DisplayTextLineConstants, 690
 NBCCCommon.h, 1066
TextOut
 DisplayModuleFunctions, 295
 NXCDefs.h, 1538
TheSleepTimeoutMS
 SetSleepTimeoutType, 871
Third-party NXT devices, 193
Time constants, 564
TimeConstants
 MIN_1, 566
 MS_1, 566
 MS_10, 566
 MS_100, 566
 MS_150, 566
 MS_2, 566
 MS_20, 566
 MS_200, 567
 MS_250, 567
 MS_3, 567
 MS_30, 567
 MS_300, 567
 MS_350, 567
 MS_4, 567

MS_40, 567
MS_400, 568
MS_450, 568
MS_5, 568
MS_50, 568
MS_500, 568
MS_6, 568
MS_60, 568
MS_600, 569
MS_7, 569
MS_70, 569
MS_700, 569
MS_8, 569
MS_80, 569
MS_800, 569
MS_9, 569
MS_90, 569
MS_900, 569
SEC_1, 570
SEC_10, 570
SEC_15, 570
SEC_2, 570
SEC_20, 570
SEC_3, 571
SEC_30, 571
SEC_4, 571
SEC_5, 571
SEC_6, 571
SEC_7, 572
SEC_8, 572
SEC_9, 572
TIMES_UP
 CommandVMState, 575
 NBCCCommon.h, 1067
tolower
 ctypeAPI, 550
 NXCDefs.h, 1539
Tone, 878
 Duration, 878
 Frequency, 878
Tone constants, 601
TONE_A3
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_A4
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_A5
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_A6
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_A7
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_AS3
 NBCCCommon.h, 1067
 ToneConstants, 603
TONE_AS4
 NBCCCommon.h, 1067
 ToneConstants, 604
TONE_AS5
 NBCCCommon.h, 1067
 ToneConstants, 604
TONE_AS6
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_AS7
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_B3
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_B4
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_B5
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_B6
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_B7
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_C4
 NBCCCommon.h, 1068
 ToneConstants, 604
TONE_C5
 NBCCCommon.h, 1068
 ToneConstants, 605

TONE_C6
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_C7
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_CS4
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_CS5
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_CS6
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_CS7
 NBCCCommon.h, 1069
 ToneConstants, 605

TONE_D4
 NBCCCommon.h, 1069
 ToneConstants, 606

TONE_D5
 NBCCCommon.h, 1069
 ToneConstants, 606

TONE_D6
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_D7
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_DS4
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_DS5
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_DS6
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_DS7
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_E4
 NBCCCommon.h, 1070
 ToneConstants, 606

TONE_E5
 NBCCCommon.h, 1070
 ToneConstants, 607

TONE_E6
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_E7
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_F4
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_F5
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_F6
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_F7
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_FS4
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_FS5
 NBCCCommon.h, 1071
 ToneConstants, 607

TONE_FS6
 NBCCCommon.h, 1071
 ToneConstants, 608

TONE_FS7
 NBCCCommon.h, 1071
 ToneConstants, 608

TONE_G4
 NBCCCommon.h, 1072
 ToneConstants, 608

TONE_G5
 NBCCCommon.h, 1072
 ToneConstants, 608

TONE_G6
 NBCCCommon.h, 1072
 ToneConstants, 608

TONE_G7
 NBCCCommon.h, 1072
 ToneConstants, 608

TONE_GS4
 NBCCCommon.h, 1072

ToneConstants, 608
TONE_GS5
 NBCCCommon.h, 1072
 ToneConstants, 608
TONE_GS6
 NBCCCommon.h, 1072
 ToneConstants, 609
TONE_GS7
 NBCCCommon.h, 1072
 ToneConstants, 609
ToneConstants
 TONE_A3, 603
 TONE_A4, 603
 TONE_A5, 603
 TONE_A6, 603
 TONE_A7, 603
 TONE_AS3, 603
 TONE_AS4, 604
 TONE_AS5, 604
 TONE_AS6, 604
 TONE_AS7, 604
 TONE_B3, 604
 TONE_B4, 604
 TONE_B5, 604
 TONE_B6, 604
 TONE_B7, 604
 TONE_C4, 604
 TONE_C5, 605
 TONE_C6, 605
 TONE_C7, 605
 TONE_CS4, 605
 TONE_CS5, 605
 TONE_CS6, 605
 TONE_CS7, 605
 TONE_D4, 606
 TONE_D5, 606
 TONE_D6, 606
 TONE_D7, 606
 TONE_DS4, 606
 TONE_DS5, 606
 TONE_DS6, 606
 TONE_DS7, 606
 TONE_E4, 606
 TONE_E5, 607
 TONE_E6, 607
 TONE_E7, 607
 TONE_F4, 607
 TONE_F5, 607
 TONE_F6, 607
 TONE_F7, 607
 TONE_FS4, 607
 TONE_FS5, 607
 TONE_FS6, 608
 TONE_FS7, 608
 TONE_G4, 608
 TONE_G5, 608
 TONE_G6, 608
 TONE_G7, 608
 TONE_GS4, 608
 TONE_GS5, 608
 TONE_GS6, 609
 TONE_GS7, 609
TOPLINE
 DisplayModuleConstants, 680
 NBCCCommon.h, 1073
toupper
 ctypeAPI, 551
 NXCDefs.h, 1539
TRAIN_CHANNEL_1
 IRTrainChannels, 761
 NBCCCommon.h, 1073
TRAIN_CHANNEL_2
 IRTrainChannels, 761
 NBCCCommon.h, 1073
TRAIN_CHANNEL_3
 IRTrainChannels, 762
 NBCCCommon.h, 1073
TRAIN_CHANNEL_ALL
 IRTrainChannels, 762
 NBCCCommon.h, 1073
TRAIN_FUNC_DECR_SPEED
 IRTrainFuncs, 760
 NBCCCommon.h, 1073
TRAIN_FUNC_INCR_SPEED
 IRTrainFuncs, 760
 NBCCCommon.h, 1073
TRAIN_FUNC_STOP
 IRTrainFuncs, 761
 NBCCCommon.h, 1073
TRAIN_FUNC_TOGGLE_LIGHT
 IRTrainFuncs, 761
 NBCCCommon.h, 1074

TRUE
 MiscConstants, 193
 NBCCCommon.h, 1074

Trunc
 cmathAPI, 491
 NXCDefs.h, 1182

trunc
 cmathAPI, 506
 NXCDefs.h, 1540

TurnRatio
 NBCCCommon.h, 1074
 OutputFieldConstants, 654

Type
 InputFieldConstants, 631
 NBCCCommon.h, 1074

Type aliases, 203

TypeAliases
 s16, 203
 s32, 203
 s8, 204
 u16, 204
 u32, 204
 u8, 204

u16
 NXCDefs.h, 1182
 TypeAliases, 204

u32
 NXCDefs.h, 1183
 TypeAliases, 204

u8
 NXCDefs.h, 1183
 TypeAliases, 204

UCHAR_MAX
 NBCCCommon.h, 1074
 NXTLimits, 793

UF_PENDING_UPDATES
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_MODE
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_PID_VALUES
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_RESET_BLOCK_-
 COUNT
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_RESET_COUNT
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_RESET_ROTATION_-
 COUNT
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_SPEED
 NBCCCommon.h, 1075
 OutUFConstants, 643

UF_UPDATE_TACHO_LIMIT
 NBCCCommon.h, 1075
 OutUFConstants, 643

Ui module, 56

Ui module constants, 614

Ui module functions, 438

Ui module IOMAP offsets, 622

Ui module types, 438

UI_BT_CONNECT_REQUEST
 NBCCCommon.h, 1075
 UiBluetoothStateConstants, 620

UI_BT_ERROR_ATTENTION
 NBCCCommon.h, 1075
 UiBluetoothStateConstants, 620

UI_BT_PIN_REQUEST
 NBCCCommon.h, 1076
 UiBluetoothStateConstants, 621

UI_BT_STATE_CONNECTED
 NBCCCommon.h, 1076
 UiBluetoothStateConstants, 621

UI_BT_STATE_OFF
 NBCCCommon.h, 1076
 UiBluetoothStateConstants, 621

UI_BT_STATE_VISIBLE
 NBCCCommon.h, 1076
 UiBluetoothStateConstants, 621

UI_BUTTON_ENTER
 NBCCCommon.h, 1076
 UiButtonConstants, 619

UI_BUTTON_EXIT
 NBCCCommon.h, 1076
 UiButtonConstants, 619

UI_BUTTON_LEFT
 NBCCCommon.h, 1076
 UiButtonConstants, 620

UI_BUTTON_NONE
 NBCCCommon.h, 1076
 UiButtonConstants, 620

UI_BUTTON_RIGHT
 NBCCCommon.h, 1077
 UiButtonConstants, 620

UI_FLAGS_BUSY
 NBCCCommon.h, 1077
 UiFlagsConstants, 615

UI_FLAGS_DISABLE_EXIT
 NBCCCommon.h, 1077
 UiFlagsConstants, 615

UI_FLAGS_DISABLE_LEFT_RIGHT_-
 ENTER
 NBCCCommon.h, 1077
 UiFlagsConstants, 615

UI_FLAGS_ENABLE_STATUS_-
 UPDATE
 NBCCCommon.h, 1077
 UiFlagsConstants, 616

UI_FLAGS_EXECUTE_LMS_FILE
 NBCCCommon.h, 1077
 UiFlagsConstants, 616

UI_FLAGS_REDRAW_STATUS
 NBCCCommon.h, 1077
 UiFlagsConstants, 616

UI_FLAGS_RESET_SLEEP_TIMER
 NBCCCommon.h, 1077
 UiFlagsConstants, 616

UI_FLAGS_UPDATE
 NBCCCommon.h, 1077
 UiFlagsConstants, 616

UI_STATE_BT_ERROR
 NBCCCommon.h, 1078
 UiStateConstants, 617

UI_STATE_CONNECT_REQUEST
 NBCCCommon.h, 1078
 UiStateConstants, 617

UI_STATE_DRAW_MENU
 NBCCCommon.h, 1078
 UiStateConstants, 617

UI_STATE_ENTER_PRESSED
 NBCCCommon.h, 1078

 UiStateConstants, 617
 UiVMRunStateConstants, 622

UI_STATE_EXECUTE_FILE
 NBCCCommon.h, 1078
 UiStateConstants, 617

UI_STATE_EXECUTING_FILE
 NBCCCommon.h, 1078
 UiStateConstants, 617

UI_STATE_EXIT_PRESSED
 NBCCCommon.h, 1078
 UiStateConstants, 618

UI_STATE_INIT_DISPLAY
 NBCCCommon.h, 1078
 UiStateConstants, 618

UI_STATE_INIT_INTRO
 NBCCCommon.h, 1078
 UiStateConstants, 618

UI_STATE_INIT_LOW_BATTERY
 NBCCCommon.h, 1078
 UiStateConstants, 618

UI_STATE_INIT_MENU
 NBCCCommon.h, 1079
 UiStateConstants, 618

UI_STATE_INIT_WAIT
 NBCCCommon.h, 1079
 UiStateConstants, 618

UI_STATE_LEFT_PRESSED
 NBCCCommon.h, 1079
 UiStateConstants, 618

UI_STATE_LOW_BATTERY
 NBCCCommon.h, 1079
 UiStateConstants, 618

UI_STATE_NEXT_MENU
 NBCCCommon.h, 1079
 UiStateConstants, 618

UI_STATE_RIGHT_PRESSED
 NBCCCommon.h, 1079
 UiStateConstants, 619

UI_STATE_TEST_BUTTONS
 NBCCCommon.h, 1079
 UiStateConstants, 619

UI_VM_IDLE
 NBCCCommon.h, 1079
 UiVMRunStateConstants, 622

UI_VM_RESET1
 NBCCCommon.h, 1079
 UiVMRunStateConstants, 622

UI_VM_RESET2	UIOffsetBatteryState, 623
NBCCCommon.h, 1080	UIOffsetBatteryVoltage, 623
UiVMRunStateConstants, 622	UIOffsetBluetoothState, 623
UI_VM_RUN_FREE	UIOffsetButton, 623
NBCCCommon.h, 1080	UIOffsetError, 624
UiVMRunStateConstants, 622	UIOffsetFlags, 624
UI_VM_RUN_PAUSE	UIOffsetForceOff, 624
NBCCCommon.h, 1080	UIOffsetLMSfilename, 624
UiVMRunStateConstants, 622	UIOffsetOBPPointer, 624
UI_VM_RUN_SINGLE	UIOffsetPMenu, 624
NBCCCommon.h, 1080	UIOffsetRechargeable, 624
UiVMRunStateConstants, 622	UIOffsetRunState, 624
UiBluetoothStateConstants	UIOffsetSleepTimeout, 624
UI_BT_CONNECT_REQUEST,	UIOffsetSleepTimer, 624
620	UIOffsetState, 625
UI_BT_ERROR_ATTENTION, 620	UIOffsetUsbState, 625
UI_BT_PIN_REQUEST, 621	UIOffsetVolume, 625
UI_BT_STATE_CONNECTED, 621	UiModuleFunctions
UI_BT_STATE_OFF, 621	AbortFlag, 440
UI_BT_STATE_VISIBLE, 621	BatteryLevel, 441
UIButton	BatteryState, 441
NXCDefs.h, 1540	BluetoothState, 441
UiModuleFunctions, 449	CommandFlags, 442
UIButton constants, 619	ForceOff, 442
UiButtonConstants	LongAbort, 442
UI_BUTTON_ENTER, 619	OnBrickProgramPointer, 443
UI_BUTTON_EXIT, 619	RechargeableBattery, 443
UI_BUTTON_LEFT, 620	SetAbortFlag, 443
UI_BUTTON_NONE, 620	SetBatteryState, 444
UI_BUTTON_RIGHT, 620	SetBluetoothState, 444
UiFlagsConstants	SetCommandFlags, 444
UI_FLAGS_BUSY, 615	SetLongAbort, 445
UI_FLAGS_DISABLE_EXIT, 615	SetOnBrickProgramPointer, 445
UI_FLAGS_DISABLE_LEFT_-	SetSleepTime, 445
RIGHT_ENTER, 615	SetSleepTimeout, 446
UI_FLAGS_ENABLE_STATUS_-	SetSleepTimer, 446
UPDATE, 616	SetUIButton, 446
UI_FLAGS_EXECUTE_LMS_-	SetUIState, 447
FILE, 616	SetVMRunState, 447
UI_FLAGS_REDRAW_STATUS,	SetVolume, 447
616	SleepTime, 448
UI_FLAGS_RESET_SLEEP_-	SleepTimeout, 448
TIMER, 616	SleepTimer, 448
UI_FLAGS_UPDATE, 616	SysSetSleepTimeout, 449
UiIOMAP	UIButton, 449
UIOffsetAbortFlag, 623	UIState, 449

UsbState, 450
 VMRunState, 450
 Volume, 450

 UIModuleID
 ModuleIDConstants, 191
 NBCCCommon.h, 1080

 UIModuleName
 ModuleNameConstants, 189
 NBCCCommon.h, 1080

 UINT_MAX
 NBCCCommon.h, 1080
 NXTLimits, 793

 UIOffsetAbortFlag
 NBCCCommon.h, 1080
 UiIOMAP, 623

 UIOffsetBatteryState
 NBCCCommon.h, 1080
 UiIOMAP, 623

 UIOffsetBatteryVoltage
 NBCCCommon.h, 1081
 UiIOMAP, 623

 UIOffsetBluetoothState
 NBCCCommon.h, 1081
 UiIOMAP, 623

 UIOffsetButton
 NBCCCommon.h, 1081
 UiIOMAP, 623

 UIOffsetError
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetFlags
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetForceOff
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetLMSfilename
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetOBPPointer
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetPMenu
 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetRechargeable

 NBCCCommon.h, 1081
 UiIOMAP, 624

 UIOffsetRunState
 NBCCCommon.h, 1082
 UiIOMAP, 624

 UIOffsetSleepTimeout
 NBCCCommon.h, 1082
 UiIOMAP, 624

 UIOffsetSleepTimer
 NBCCCommon.h, 1082
 UiIOMAP, 624

 UIOffsetState
 NBCCCommon.h, 1082
 UiIOMAP, 625

 UIOffsetUsbState
 NBCCCommon.h, 1082
 UiIOMAP, 625

 UIOffsetVolume
 NBCCCommon.h, 1082
 UiIOMAP, 625

 UIState
 NXCDefs.h, 1540
 UiModuleFunctions, 449

 UIState constants, 616

 UiStateConstants
 UI_STATE_BT_ERROR, 617
 UI_STATE_CONNECT_-
 REQUEST, 617
 UI_STATE_DRAW_MENU, 617
 UI_STATE_ENTER_PRESSED,
 617
 UI_STATE_EXECUTE_FILE, 617
 UI_STATE_EXECUTING_FILE,
 617
 UI_STATE_EXIT_PRESSED, 618
 UI_STATE_INIT_DISPLAY, 618
 UI_STATE_INIT_INTRO, 618
 UI_STATE_INIT_LOW_-
 BATTERY, 618
 UI_STATE_INIT_MENU, 618
 UI_STATE_INIT_WAIT, 618
 UI_STATE_LEFT_PRESSED, 618
 UI_STATE_LOW_BATTERY, 618
 UI_STATE_NEXT_MENU, 618
 UI_STATE_RIGHT_PRESSED, 619
 UI_STATE_TEST_BUTTONS, 619

UiVMRunStateConstants
 UI_VM_IDLE, 622
 UI_VM_RESET1, 622
 UI_VM_RESET2, 622
 UI_VM_RUN_FREE, 622
 UI_VM_RUN_PAUSE, 622
 UI_VM_RUN_SINGLE, 622

ULONG_MAX
 NBCCCommon.h, 1082
 NXTLimits, 793

Ultrasonic sensor constants, 668

UnflattenVar
 cstringAPI, 544
 NXCDefs.h, 1541

UpdateCalibCacheInfo
 NBCCCommon.h, 1082
 SysCallConstants, 561

UpdateCalibCacheInfoType, 878
 MaxVal, 879
 MinVal, 879
 Name, 879
 Result, 880

UpdateFlags
 NBCCCommon.h, 1082
 OutputFieldConstants, 654

US_CMD_CONTINUOUS
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_CMD_EVENTCAPTURE
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_CMD_OFF
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_CMD_SINGLESHTOT
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_CMD_WARMRESET
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_REG_ACTUAL_ZERO
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_REG_CM_INTERVAL
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_REG_FACTORY_ACTUAL_ZERO
 NBCCCommon.h, 1083
 USI2CConstants, 669

US_REG_FACTORY_SCALE_-
 DIVISOR
 NBCCCommon.h, 1084
 USI2CConstants, 670

US_REG_FACTORY_SCALE_FACTOR
 NBCCCommon.h, 1084
 USI2CConstants, 670

US_REG_MEASUREMENT_UNITS
 NBCCCommon.h, 1084
 USI2CConstants, 670

US_REG_SCALE_DIVISOR
 NBCCCommon.h, 1084
 USI2CConstants, 670

US_REG_SCALE_FACTOR
 NBCCCommon.h, 1084
 USI2CConstants, 670

USB_CMD_READY
 CommStatusCodesConstants, 713
 NBCCCommon.h, 1084

USB_PROTOCOL_OVERHEAD
 CommMiscConstants, 696
 NBCCCommon.h, 1084

USBInputBufferInPtr
 CommModuleFunctions, 427
 NXCDefs.h, 1541

USBInputBufferOutPtr
 CommModuleFunctions, 428
 NXCDefs.h, 1542

USBOutputBufferInPtr
 CommModuleFunctions, 428
 NXCDefs.h, 1542

USBOutputBufferOutPtr
 CommModuleFunctions, 428
 NXCDefs.h, 1542

USBPollBufferInPtr
 CommModuleFunctions, 429
 NXCDefs.h, 1542

USBPollBufferOutPtr
 CommModuleFunctions, 429
 NXCDefs.h, 1543

USBState
 CommModuleFunctions, 429
 NXCDefs.h, 1543

UsbState
 NXCDefs.h, 1543
 UiModuleFunctions, 450

USHRT_MAX
 NBCCCommon.h, 1084
 NXTLimits, 793

USI2CConstants
 US_CMD_CONTINUOUS, 669
 US_CMD_EVENTCAPTURE, 669
 US_CMD_OFF, 669
 US_CMD_SINGLESHT, 669
 US_CMD_WARMRESET, 669
 US_REG_ACTUAL_ZERO, 669
 US_REG_CM_INTERVAL, 669
 US_REG_FACTORY_ACTUAL_ZERO, 669
 US_REG_FACTORY_SCALE_DIVISOR, 670
 US_REG_FACTORY_SCALE_FACTOR, 670
 US_REG_MEASUREMENT_UNITS, 670
 US_REG_SCALE_DIVISOR, 670
 US_REG_SCALE_FACTOR, 670

Variables
 DrawGraphicArrayType, 829
 DrawGraphicType, 831

VM run state constants, 621

VM state constants, 574

VMRunState
 NXCDefs.h, 1544
 UiModuleFunctions, 450

Volume
 NXCDefs.h, 1544
 UiModuleFunctions, 450

Wait
 CommandModuleFunctions, 363
 NXCDefs.h, 1544

Width
 SizeType, 872

Write
 LoaderModuleFunctions, 471
 NXCDefs.h, 1545

WriteBytes
 LoaderModuleFunctions, 471
 NXCDefs.h, 1545

WriteBytesEx
 LoaderModuleFunctions, 472
 NXCDefs.h, 1546

WriteHandle
 FileResolveHandleType, 848

WriteI2CRegister
 LowSpeedModuleFunctions, 323
 NXCDefs.h, 1546

WriteLn
 LoaderModuleFunctions, 472
 NXCDefs.h, 1547

WriteLnString
 LoaderModuleFunctions, 473
 NXCDefs.h, 1547

WriteNRLLinkBytes
 MindSensorsAPI, 178
 NXCDefs.h, 1548

WriteSemData
 NBCCCommon.h, 1084
 SysCallConstants, 561

WriteSemDataType, 880
 ClearBits, 881
 NewVal, 881
 Request, 881
 SemData, 881

WriteString
 LoaderModuleFunctions, 473
 NXCDefs.h, 1548

X
 LocationType, 863

X1
 DisplayExecuteFunctionType, 822

X2
 DisplayExecuteFunctionType, 822

Y
 LocationType, 863

Y1
 DisplayExecuteFunctionType, 822

Y2
 DisplayExecuteFunctionType, 822

Yield
 CommandModuleFunctions, 364

NXCDefs.h, [1549](#)