

Mark3 Realtime Kernel

Generated by Doxygen 1.8.4

Sat Jan 25 2014 18:56:55

Contents

1	The Mark3 Realtime Kernel	1
2	Preface	3
2.1	Who should read this	3
2.2	Why Mark3?	3
3	Can you Afford an RTOS?	5
3.1	Intro	5
3.2	Memory overhead:	6
3.3	Code Space Overhead:	7
3.4	Runtime Overhead	7
4	Superloops	9
4.1	Intro to Superloops	9
4.2	The simplest loop	9
4.3	Interrupt-Driven Super-loop	10
4.4	Cooperative multi-tasking	11
4.5	Hybrid cooperative/preemptive multi-tasking	12
4.6	Problems with superloops	13
5	Mark3 Overview	15
5.1	Intro	15
5.2	Features	15
5.3	Design Goals	16
6	Getting Started	17
6.1	Kernel Setup	17
6.2	Threads	18
6.2.1	Thread Setup	18
6.2.2	Entry Functions	19
6.3	Timers	19
6.4	Semaphores	20
6.5	Mutexes	21

6.6	Event Flags	21
6.7	Messages	22
6.7.1	Message Objects	22
6.7.2	Global Message Pool	23
6.7.3	Message Queues	23
6.7.4	Messaging Example	23
6.8	Sleep	24
6.9	Round-Robin Quantum	24
7	Inside The Scheduler	25
7.1	A Bit About Threads	25
7.2	Thread States and Thread Lists	25
7.3	About Blocking and Unblocking	26
7.4	The Scheduling Algorithm	26
7.5	Considerations For Round-Robin Scheduling	28
7.6	Context Switching	28
7.7	Putting It All Together	28
8	Porting Mark3 - An Example Using ARM Cortex-M0	31
8.1	Thread Stack Initialization	32
8.2	Kernel Startup	33
8.3	First Thread Entry	34
8.4	Context Switching	34
8.5	Kernel Timers	35
8.6	Critical Sections	36
8.7	Conclusion	37
9	Build System	39
9.1	Source Layout	39
9.2	Building the kernel	39
9.3	Building on Windows	40
9.4	Exporting the Source	41
9.4.1	Supported targets	41
10	License	43
10.1	License	43
11	Profiling Results	45
11.1	Date Performed	45
11.2	Compiler Information	45
11.3	Profiling Results	45

12 Code Size Profiling	47
12.1 Information	47
12.2 Compiler Version	47
12.3 Profiling Results	47
13 Hierarchical Index	49
13.1 Class Hierarchy	49
14 Class Index	53
14.1 Class List	53
15 File Index	57
15.1 File List	57
16 Class Documentation	63
16.1 Atomic Class Reference	63
16.1.1 Detailed Description	63
16.1.2 Member Function Documentation	63
16.1.2.1 Add	63
16.1.2.2 Set	64
16.1.2.3 Sub	64
16.1.2.4 TestAndSet	64
16.2 BitStreamer Class Reference	65
16.2.1 Detailed Description	65
16.2.2 Member Function Documentation	65
16.2.2.1 AdvanceByte	65
16.2.2.2 Init	65
16.2.2.3 ReadBits	66
16.3 BlockHeap Class Reference	66
16.3.1 Detailed Description	67
16.3.2 Member Function Documentation	67
16.3.2.1 Alloc	67
16.3.2.2 Create	67
16.3.2.3 Free	67
16.3.2.4 IsFree	67
16.4 BlockingObject Class Reference	68
16.4.1 Detailed Description	68
16.4.2 Member Function Documentation	68
16.4.2.1 Block	68
16.4.2.2 LockAndQueue	69
16.4.2.3 Unblock	69

16.4.2.4	UnLock	69
16.5	ButtonControl Class Reference	70
16.5.1	Detailed Description	70
16.5.2	Member Function Documentation	71
16.5.2.1	Activate	71
16.5.2.2	Draw	72
16.5.2.3	Init	72
16.5.2.4	ProcessEvent	72
16.6	CheckBoxControl Class Reference	72
16.6.1	Detailed Description	73
16.6.2	Member Function Documentation	73
16.6.2.1	Activate	73
16.6.2.2	Draw	73
16.6.2.3	Init	74
16.6.2.4	ProcessEvent	74
16.7	CircularLinkedList Class Reference	74
16.7.1	Detailed Description	75
16.7.2	Member Function Documentation	75
16.7.2.1	Add	75
16.7.2.2	Remove	75
16.8	CommandLine_t Struct Reference	75
16.8.1	Detailed Description	76
16.9	DCPU Class Reference	76
16.9.1	Detailed Description	78
16.9.2	Member Function Documentation	78
16.9.2.1	AddPlugin	78
16.9.2.2	GetOperand	78
16.9.2.3	GetRegisters	78
16.9.2.4	HWN	78
16.9.2.5	IAQ	78
16.9.2.6	Init	78
16.9.2.7	RFI	80
16.9.2.8	SendInterrupt	80
16.9.3	Member Data Documentation	80
16.9.3.1	m_clPluginList	80
16.10	DCPU_Registers Struct Reference	80
16.10.1	Detailed Description	81
16.11	DCPUPlugin Class Reference	81
16.11.1	Detailed Description	82
16.11.2	Member Function Documentation	82

16.11.2.1 Enumerate	82
16.11.2.2 GetDeviceNumber	82
16.11.2.3 Init	82
16.11.2.4 Interrupt	83
16.12DevNull Class Reference	83
16.12.1 Detailed Description	84
16.12.2 Member Function Documentation	84
16.12.2.1 Close	84
16.12.2.2 Control	84
16.12.2.3 Open	84
16.12.2.4 Read	85
16.12.2.5 Write	85
16.13DoubleLinkedList Class Reference	85
16.13.1 Detailed Description	86
16.13.2 Member Function Documentation	86
16.13.2.1 Add	86
16.13.2.2 Remove	86
16.14DrawBitmap_t Struct Reference	86
16.14.1 Detailed Description	87
16.15DrawCircle_t Struct Reference	87
16.15.1 Detailed Description	87
16.16DrawEllipse_t Struct Reference	88
16.16.1 Detailed Description	88
16.17DrawLine_t Struct Reference	88
16.17.1 Detailed Description	89
16.18DrawMove_t Struct Reference	89
16.18.1 Detailed Description	89
16.19DrawPoint_t Struct Reference	89
16.19.1 Detailed Description	90
16.20DrawPoly_t Struct Reference	90
16.20.1 Detailed Description	90
16.21DrawRectangle_t Struct Reference	90
16.21.1 Detailed Description	91
16.22DrawStamp_t Struct Reference	91
16.22.1 Detailed Description	91
16.23DrawText_t Struct Reference	92
16.23.1 Detailed Description	92
16.24DrawVector_t Struct Reference	92
16.24.1 Detailed Description	92
16.25DrawWindow_t Struct Reference	93

16.25.1 Detailed Description	93
16.26 Driver Class Reference	93
16.26.1 Detailed Description	94
16.26.2 Member Function Documentation	94
16.26.2.1 Close	94
16.26.2.2 Control	94
16.26.2.3 GetPath	95
16.26.2.4 Open	95
16.26.2.5 Read	95
16.26.2.6 SetName	95
16.26.2.7 Write	96
16.27 DriverList Class Reference	96
16.27.1 Detailed Description	96
16.27.2 Member Function Documentation	97
16.27.2.1 Add	97
16.27.2.2 FindByPath	97
16.27.2.3 Init	97
16.27.2.4 Remove	97
16.28 EventFlag Class Reference	97
16.28.1 Detailed Description	98
16.28.2 Member Function Documentation	98
16.28.2.1 Clear	98
16.28.2.2 ClearTransaction	99
16.28.2.3 GetMask	99
16.28.2.4 ProcessQueue	99
16.28.2.5 Set	99
16.28.2.6 SetTransaction	99
16.28.2.7 TimeoutTransaction	100
16.28.2.8 Wait	100
16.28.2.9 Wait	100
16.28.2.10 WaitTransaction	101
16.29 FixedHeap Class Reference	101
16.29.1 Detailed Description	101
16.29.2 Member Function Documentation	101
16.29.2.1 Alloc	101
16.29.2.2 Create	102
16.29.2.3 Free	102
16.30 Font_t Struct Reference	102
16.30.1 Detailed Description	103
16.31 GamePanelControl Class Reference	103

16.31.1 Detailed Description	103
16.31.2 Member Function Documentation	103
16.31.2.1 Activate	103
16.31.2.2 Draw	104
16.31.2.3 Init	104
16.31.2.4 ProcessEvent	104
16.32GlobalMessagePool Class Reference	104
16.32.1 Detailed Description	105
16.32.2 Member Function Documentation	105
16.32.2.1 Pop	105
16.32.2.2 Push	105
16.33Glyph_t Struct Reference	105
16.33.1 Detailed Description	106
16.34GraphicsDriver Class Reference	106
16.34.1 Detailed Description	107
16.34.2 Member Function Documentation	107
16.34.2.1 Bitmap	107
16.34.2.2 Circle	107
16.34.2.3 DrawPixel	107
16.34.2.4 Ellipse	108
16.34.2.5 Line	108
16.34.2.6 Move	108
16.34.2.7 Point	108
16.34.2.8 ReadPixel	108
16.34.2.9 Rectangle	108
16.34.2.10SetWindow	109
16.34.2.11Stamp	109
16.34.2.12Text	109
16.34.2.13TextFX	109
16.34.2.14TriangleFill	109
16.34.2.15TriangleWire	109
16.35GroupBoxControl Class Reference	110
16.35.1 Detailed Description	110
16.35.2 Member Function Documentation	111
16.35.2.1 Activate	111
16.35.2.2 Draw	111
16.35.2.3 Init	111
16.35.2.4 ProcessEvent	111
16.36GuiControl Class Reference	111
16.36.1 Detailed Description	114

16.36.2 Member Function Documentation	114
16.36.2.1 Activate	114
16.36.2.2 ClearStale	114
16.36.2.3 Draw	114
16.36.2.4 GetControllIndex	115
16.36.2.5 GetControlOffset	115
16.36.2.6 GetHeight	115
16.36.2.7 GetLeft	115
16.36.2.8 GetParentControl	115
16.36.2.9 GetParentWindow	116
16.36.2.10 GetTop	116
16.36.2.11 GetWidth	116
16.36.2.12 GetZOrder	116
16.36.2.13 Init	116
16.36.2.14 IsInFocus	116
16.36.2.15 IsStale	117
16.36.2.16 ProcessEvent	117
16.36.2.17 SetControllIndex	117
16.36.2.18 SetHeight	117
16.36.2.19 SetLeft	117
16.36.2.20 SetParentControl	117
16.36.2.21 SetParentWindow	118
16.36.2.22 SetTop	118
16.36.2.23 SetWidth	118
16.36.2.24 SetZOrder	118
16.36.3 Member Data Documentation	118
16.36.3.1 m_ucControllIndex	118
16.36.3.2 m_ucZOrder	119
16.37 GuiEvent_t Struct Reference	119
16.37.1 Detailed Description	119
16.38 GuiEventSurface Class Reference	119
16.38.1 Detailed Description	120
16.38.2 Member Function Documentation	120
16.38.2.1 AddWindow	120
16.38.2.2 CopyEvent	121
16.38.2.3 Init	121
16.38.2.4 InvalidateRegion	121
16.38.2.5 ProcessEvent	121
16.38.2.6 RemoveWindow	121
16.38.2.7 SendEvent	121

16.39GuiWindow Class Reference	122
16.39.1 Detailed Description	124
16.39.2 Member Function Documentation	124
16.39.2.1 AddControl	124
16.39.2.2 CycleFocus	124
16.39.2.3 GetDriver	124
16.39.2.4 GetHeight	124
16.39.2.5 GetLeft	125
16.39.2.6 GetMaxZOrder	125
16.39.2.7 GetTop	125
16.39.2.8 GetWidth	125
16.39.2.9 Init	125
16.39.2.10InvalidateRegion	125
16.39.2.11IsInFocus	126
16.39.2.12ProcessEvent	127
16.39.2.13Redraw	127
16.39.2.14RemoveControl	127
16.39.2.15SetDriver	127
16.39.2.16SetFocus	127
16.39.2.17SetHeight	128
16.39.2.18SetLeft	128
16.39.2.19SetTop	128
16.39.2.20SetWidth	128
16.39.3 Member Data Documentation	128
16.39.3.1 m_pclDriver	128
16.40HeapConfig Class Reference	129
16.40.1 Detailed Description	129
16.41JoystickEvent_t Struct Reference	129
16.41.1 Detailed Description	130
16.42Kernel Class Reference	130
16.42.1 Detailed Description	131
16.42.2 Member Function Documentation	131
16.42.2.1 Init	131
16.42.2.2 IsPanic	131
16.42.2.3 IsStarted	131
16.42.2.4 Panic	131
16.42.2.5 SetPanic	132
16.42.2.6 Start	132
16.43KernelSWI Class Reference	132
16.43.1 Detailed Description	133

16.43.2 Member Function Documentation	133
16.43.2.1 DI	133
16.43.2.2 RI	133
16.44 KernelTimer Class Reference	133
16.44.1 Detailed Description	134
16.44.2 Member Function Documentation	134
16.44.2.1 GetOvertime	134
16.44.2.2 Read	134
16.44.2.3 RI	134
16.44.2.4 SetExpiry	134
16.44.2.5 SubtractExpiry	135
16.44.2.6 TimeToExpiry	135
16.45 KeyEvent_t Struct Reference	135
16.45.1 Detailed Description	136
16.46 LabelControl Class Reference	136
16.46.1 Detailed Description	137
16.46.2 Member Function Documentation	137
16.46.2.1 Activate	137
16.46.2.2 Draw	137
16.46.2.3 Init	137
16.46.2.4 ProcessEvent	137
16.47 LinkList Class Reference	138
16.47.1 Detailed Description	138
16.47.2 Member Function Documentation	138
16.47.2.1 Add	138
16.47.2.2 GetHead	139
16.47.2.3 GetTail	139
16.47.2.4 Remove	139
16.48 LinkListNode Class Reference	139
16.48.1 Detailed Description	141
16.48.2 Member Function Documentation	141
16.48.2.1 GetNext	141
16.48.2.2 GetPrev	141
16.49 MemUtil Class Reference	141
16.49.1 Detailed Description	142
16.49.2 Member Function Documentation	142
16.49.2.1 Checksum16	142
16.49.2.2 Checksum8	142
16.49.2.3 CompareMemory	143
16.49.2.4 CompareStrings	143

16.49.2.5 CopyMemory	143
16.49.2.6 CopyString	143
16.49.2.7 DecimalToHex	144
16.49.2.8 DecimalToString	144
16.49.2.9 SetMemory	144
16.49.2.10StringLength	144
16.49.2.11StringSearch	144
16.49.2.12Tokenize	145
16.50Message Class Reference	145
16.50.1 Detailed Description	146
16.50.2 Member Function Documentation	146
16.50.2.1 GetCode	146
16.50.2.2 GetData	146
16.50.2.3 SetCode	146
16.50.2.4 SetData	146
16.51MessageQueue Class Reference	147
16.51.1 Detailed Description	147
16.51.2 Member Function Documentation	147
16.51.2.1 GetCount	147
16.51.2.2 Receive	148
16.51.2.3 Receive	148
16.51.2.4 Send	148
16.52MouseEvent_t Struct Reference	148
16.52.1 Detailed Description	149
16.53Mutex Class Reference	149
16.53.1 Detailed Description	150
16.53.2 Member Function Documentation	150
16.53.2.1 Claim	150
16.53.2.2 Claim	150
16.53.2.3 ClaimTransaction	151
16.53.2.4 Release	151
16.53.2.5 ReleaseTransaction	151
16.53.2.6 Timeout	151
16.53.2.7 TimeoutTransaction	151
16.54NLFS Class Reference	152
16.54.1 Detailed Description	154
16.54.2 Member Function Documentation	154
16.54.2.1 Append_Block_To_Node	154
16.54.2.2 Cleanup_Node_Links	154
16.54.2.3 Create_Dir	154

16.54.2.4 Create_File	155
16.54.2.5 Create_File_i	156
16.54.2.6 Delete_File	156
16.54.2.7 Delete_Folder	156
16.54.2.8 File_Names_Match	156
16.54.2.9 Find_File	157
16.54.2.10 Find_Last_Slash	157
16.54.2.11 Find_Parent_Dir	157
16.54.2.12 Format	157
16.54.2.13 GetBlockSize	158
16.54.2.14 GetFirstChild	158
16.54.2.15 GetNextPeer	158
16.54.2.16 GetNumBlocks	158
16.54.2.17 GetNumBlocksFree	159
16.54.2.18 GetNumFiles	159
16.54.2.19 GetNumFilesFree	159
16.54.2.20 GetStat	159
16.54.2.21 Mount	159
16.54.2.22 Pop_Free_Block	160
16.54.2.23 Pop_Free_Node	160
16.54.2.24 Print_Dir_Details	160
16.54.2.25 Print_File_Details	160
16.54.2.26 Print_Free_Details	160
16.54.2.27 Print_Node_Details	160
16.54.2.28 Push_Free_Block	161
16.54.2.29 Push_Free_Node	161
16.54.2.30 Read_Block	161
16.54.2.31 Read_Block_Header	161
16.54.2.32 Read_Node	161
16.54.2.33 RootSync	162
16.54.2.34 Set_Node_Name	162
16.54.2.35 Write_Block	162
16.54.2.36 Write_Block_Header	162
16.54.2.37 Write_Node	162
16.55 NLFS_Block_t Struct Reference	163
16.55.1 Detailed Description	163
16.56 NLFS_File Class Reference	163
16.56.1 Detailed Description	164
16.56.2 Member Function Documentation	164
16.56.2.1 Close	164

16.56.2.2 Open	164
16.56.2.3 Read	165
16.56.2.4 Seek	165
16.56.2.5 Write	165
16.57NLFS_File_Node_t Struct Reference	166
16.57.1 Detailed Description	166
16.58NLFS_File_Stat_t Struct Reference	166
16.58.1 Detailed Description	167
16.59NLFS_Host_t Union Reference	167
16.59.1 Detailed Description	167
16.60NLFS_Node_t Struct Reference	167
16.60.1 Detailed Description	168
16.61NLFS_RAM Class Reference	168
16.61.1 Detailed Description	169
16.61.2 Member Function Documentation	169
16.61.2.1 Read_Block	169
16.61.2.2 Read_Block_Header	169
16.61.2.3 Read_Node	169
16.61.2.4 Write_Block	170
16.61.2.5 Write_Block_Header	170
16.61.2.6 Write_Node	170
16.62NLFS_Root_Node_t Struct Reference	171
16.62.1 Detailed Description	171
16.63NotificationControl Class Reference	171
16.63.1 Detailed Description	172
16.63.2 Member Function Documentation	172
16.63.2.1 Activate	172
16.63.2.2 Draw	172
16.63.2.3 Init	173
16.63.2.4 ProcessEvent	173
16.64Option_t Struct Reference	173
16.64.1 Detailed Description	173
16.65PanelControl Class Reference	174
16.65.1 Detailed Description	175
16.65.2 Member Function Documentation	175
16.65.2.1 Activate	175
16.65.2.2 Draw	175
16.65.2.3 Init	175
16.65.2.4 ProcessEvent	175
16.66Profiler Class Reference	176

16.66.1 Detailed Description	176
16.66.2 Member Function Documentation	176
16.66.2.1 Init	176
16.67 ProfileTimer Class Reference	176
16.67.1 Detailed Description	177
16.67.2 Member Function Documentation	177
16.67.2.1 ComputeCurrentTicks	177
16.67.2.2 GetAverage	178
16.67.2.3 GetCurrent	178
16.67.2.4 Init	178
16.67.2.5 Start	178
16.68 ProgressControl Class Reference	178
16.68.1 Detailed Description	179
16.68.2 Member Function Documentation	179
16.68.2.1 Activate	179
16.68.2.2 Draw	180
16.68.2.3 Init	180
16.68.2.4 ProcessEvent	180
16.69 Quantum Class Reference	180
16.69.1 Detailed Description	181
16.69.2 Member Function Documentation	181
16.69.2.1 AddThread	181
16.69.2.2 RemoveThread	181
16.69.2.3 SetTimer	181
16.69.2.4 UpdateTimer	181
16.70 Scheduler Class Reference	181
16.70.1 Detailed Description	182
16.70.2 Member Function Documentation	182
16.70.2.1 Add	182
16.70.2.2 GetCurrentThread	183
16.70.2.3 GetNextThread	183
16.70.2.4 GetStopList	183
16.70.2.5 GetThreadList	183
16.70.2.6 IsEnabled	183
16.70.2.7 Remove	184
16.70.2.8 Schedule	185
16.70.2.9 SetScheduler	185
16.71 Screen Class Reference	185
16.71.1 Detailed Description	186
16.71.2 Member Function Documentation	186

16.71.2.1 Activate	186
16.71.2.2 Deactivate	186
16.72ScreenList Class Reference	186
16.72.1 Detailed Description	187
16.73ScreenManager Class Reference	187
16.73.1 Detailed Description	187
16.74Semaphore Class Reference	188
16.74.1 Detailed Description	189
16.74.2 Member Function Documentation	189
16.74.2.1 GetCount	189
16.74.2.2 Init	189
16.74.2.3 Pend	189
16.74.2.4 Pend	189
16.74.2.5 PendTransaction	190
16.74.2.6 Post	191
16.74.2.7 PostTransaction	191
16.74.2.8 ProcessQueue	191
16.74.2.9 Timeout	191
16.74.2.10TimeoutTransaction	191
16.75ShellCommand_t Struct Reference	192
16.75.1 Detailed Description	192
16.76ShellSupport Class Reference	192
16.76.1 Detailed Description	193
16.76.2 Member Function Documentation	193
16.76.2.1 CheckForOption	193
16.76.2.2 RunCommand	193
16.76.2.3 TokensToCommandLine	193
16.76.2.4 UnescapeToken	194
16.77SlickButtonControl Class Reference	194
16.77.1 Detailed Description	195
16.77.2 Member Function Documentation	195
16.77.2.1 Activate	195
16.77.2.2 Draw	195
16.77.2.3 Init	195
16.77.2.4 ProcessEvent	195
16.78SlickGroupBoxControl Class Reference	196
16.78.1 Detailed Description	196
16.78.2 Member Function Documentation	197
16.78.2.1 Activate	197
16.78.2.2 Draw	198

16.78.2.3 Init	198
16.78.2.4 ProcessEvent	198
16.79SlickProgressControl Class Reference	198
16.79.1 Detailed Description	199
16.79.2 Member Function Documentation	199
16.79.2.1 Activate	199
16.79.2.2 Draw	199
16.79.2.3 Init	199
16.79.2.4 ProcessEvent	200
16.80Slip Class Reference	200
16.80.1 Detailed Description	201
16.80.2 Member Function Documentation	201
16.80.2.1 DecodeByte	201
16.80.2.2 EncodeByte	201
16.80.2.3 GetDriver	201
16.80.2.4 ReadData	201
16.80.2.5 SetDriver	202
16.80.2.6 WriteData	202
16.80.2.7 WriteVector	202
16.81SlipDataVector Struct Reference	202
16.81.1 Detailed Description	203
16.82SlipMux Class Reference	203
16.82.1 Detailed Description	204
16.82.2 Member Function Documentation	204
16.82.2.1 GetDriver	204
16.82.2.2 GetQueue	204
16.82.2.3 GetSlip	204
16.82.2.4 Init	204
16.82.2.5 InstallHandler	204
16.82.2.6 MessageReceive	205
16.82.2.7 SetQueue	205
16.83SlipTerm Class Reference	205
16.83.1 Detailed Description	206
16.83.2 Member Function Documentation	206
16.83.2.1 Init	206
16.83.2.2 PrintLn	206
16.83.2.3 PrintLn	206
16.83.2.4 SetVerbosity	206
16.83.2.5 StrLen	206
16.83.3 Member Data Documentation	207

16.83.3.1 m_ucVerbosity	207
16.84StubControl Class Reference	207
16.84.1 Detailed Description	207
16.84.2 Member Function Documentation	208
16.84.2.1 Activate	208
16.84.2.2 Draw	208
16.84.2.3 Init	208
16.84.2.4 ProcessEvent	208
16.85SystemHeap Class Reference	208
16.85.1 Detailed Description	209
16.85.2 Member Function Documentation	209
16.85.2.1 Alloc	209
16.85.2.2 Free	209
16.86TextFX_t Struct Reference	210
16.86.1 Detailed Description	210
16.87Thread Class Reference	210
16.87.1 Detailed Description	213
16.87.2 Member Function Documentation	213
16.87.2.1 ContextSwitchSWI	213
16.87.2.2 Exit	213
16.87.2.3 GetCurPriority	213
16.87.2.4 GetCurrent	213
16.87.2.5 GetEventFlagMask	213
16.87.2.6 GetEventFlagMode	214
16.87.2.7 GetID	214
16.87.2.8 GetName	214
16.87.2.9 GetOwner	214
16.87.2.10GetPriority	214
16.87.2.11GetQuantum	214
16.87.2.12GetStackSlack	215
16.87.2.13InheritPriority	215
16.87.2.14Init	215
16.87.2.15SetCurrent	215
16.87.2.16SetEventFlagMask	215
16.87.2.17SetEventFlagMode	216
16.87.2.18SetID	216
16.87.2.19SetName	216
16.87.2.20SetOwner	216
16.87.2.21SetPriority	216
16.87.2.22SetPriorityBase	217

16.87.2.23	SetQuantum	217
16.87.2.24	Sleep	217
16.87.2.25	Stop	217
16.87.2.26	USleep	217
16.87.2.27	Yield	217
16.88	ThreadList Class Reference	218
16.88.1	Detailed Description	218
16.88.2	Member Function Documentation	219
16.88.2.1	Add	219
16.88.2.2	Add	219
16.88.2.3	HighestWaiter	219
16.88.2.4	Remove	219
16.88.2.5	SetFlagPointer	219
16.88.2.6	SetPriority	220
16.89	ThreadPort Class Reference	221
16.89.1	Detailed Description	221
16.89.2	Member Function Documentation	221
16.89.2.1	InitStack	221
16.90	Tile_8x8 Class Reference	222
16.90.1	Detailed Description	222
16.90.2	Member Function Documentation	222
16.90.2.1	LoadTile	222
16.90.2.2	Render	222
16.91	TileDef_t Struct Reference	223
16.91.1	Detailed Description	223
16.92	Timer Class Reference	223
16.92.1	Detailed Description	224
16.92.2	Member Function Documentation	225
16.92.2.1	SetCallback	225
16.92.2.2	SetData	225
16.92.2.3	SetFlags	225
16.92.2.4	SetIntervalMSeconds	225
16.92.2.5	SetIntervalSeconds	225
16.92.2.6	SetIntervalTicks	225
16.92.2.7	SetIntervalUSeconds	226
16.92.2.8	SetOwner	226
16.92.2.9	SetTolerance	226
16.92.2.10	Start	226
16.92.2.11	Start	226
16.92.2.12	Stop	227

16.93TimerEvent_t Struct Reference	227
16.93.1 Detailed Description	227
16.94TimerList Class Reference	227
16.94.1 Detailed Description	228
16.94.2 Member Function Documentation	228
16.94.2.1 Add	228
16.94.2.2 Init	228
16.94.2.3 Process	229
16.94.2.4 Remove	229
16.95TimerScheduler Class Reference	229
16.95.1 Detailed Description	229
16.95.2 Member Function Documentation	230
16.95.2.1 Add	230
16.95.2.2 Init	230
16.95.2.3 Process	230
16.95.2.4 Remove	230
16.96Token_t Struct Reference	230
16.96.1 Detailed Description	231
16.97TouchEvent_t Struct Reference	231
16.97.1 Detailed Description	231
16.98Transaction Class Reference	231
16.98.1 Detailed Description	232
16.98.2 Member Function Documentation	232
16.98.2.1 GetCode	232
16.98.2.2 GetData	232
16.98.2.3 Set	233
16.99TransactionQueue Class Reference	233
16.99.1 Detailed Description	234
16.99.2 Member Function Documentation	234
16.99.2.1 Dequeue	234
16.99.2.2 Enqueue	234
16.99.2.3 Finish	235
16.99.2.4 GlobalQueueInit	235
16.100UnitTest Class Reference	235
16.100.1 Detailed Description	236
16.100.2 Member Function Documentation	237
16.100.2.1 Complete	237
16.100.2.2 GetFailed	237
16.100.2.3 GetName	237
16.100.2.4 GetPassed	237

16.100.2.5	GetResult	237
16.100.2.6	GetTotal	237
16.100.2.7	SetName	238
16.10	WriteBuffer16 Class Reference	239
16.101.1	Detailed Description	239
16.101.2	Member Function Documentation	240
16.101.2.1	SetBuffers	240
16.101.2.2	SetCallback	241
16.101.2.3	WriteData	241
16.101.2.4	WriteVector	241
17	File Documentation	243
17.1	/home/mo/mark3-source/embedded/stage/src/atomic.cpp File Reference	243
17.1.1	Detailed Description	243
17.2	atomic.cpp	243
17.3	/home/mo/mark3-source/embedded/stage/src/atomic.h File Reference	245
17.3.1	Detailed Description	245
17.4	atomic.h	245
17.5	/home/mo/mark3-source/embedded/stage/src/blocking.cpp File Reference	246
17.5.1	Detailed Description	246
17.6	blocking.cpp	246
17.7	/home/mo/mark3-source/embedded/stage/src/blocking.h File Reference	247
17.7.1	Detailed Description	248
17.8	blocking.h	248
17.9	/home/mo/mark3-source/embedded/stage/src/control_button.cpp File Reference	249
17.9.1	Detailed Description	249
17.10	control_button.cpp	249
17.11	/home/mo/mark3-source/embedded/stage/src/control_button.h File Reference	252
17.11.1	Detailed Description	252
17.12	control_button.h	252
17.13	/home/mo/mark3-source/embedded/stage/src/control_checkbox.cpp File Reference	253
17.13.1	Detailed Description	253
17.13.2	Variable Documentation	253
17.13.2.1	aucBox	253
17.13.2.2	aucCheck	254
17.14	control_checkbox.cpp	254
17.15	/home/mo/mark3-source/embedded/stage/src/control_checkbox.h File Reference	256
17.15.1	Detailed Description	257
17.16	control_checkbox.h	257
17.17	/home/mo/mark3-source/embedded/stage/src/control_gamepanel.cpp File Reference	257

17.17.1 Detailed Description	258
17.18control_gamepanel.cpp	258
17.19/home/mo/mark3-source/embedded/stage/src/control_gamepanel.h File Reference	258
17.19.1 Detailed Description	259
17.20control_gamepanel.h	259
17.21/home/mo/mark3-source/embedded/stage/src/control_groupbox.cpp File Reference	259
17.21.1 Detailed Description	260
17.22control_groupbox.cpp	260
17.23/home/mo/mark3-source/embedded/stage/src/control_groupbox.h File Reference	261
17.23.1 Detailed Description	261
17.24control_groupbox.h	261
17.25/home/mo/mark3-source/embedded/stage/src/control_label.h File Reference	262
17.25.1 Detailed Description	262
17.26control_label.h	262
17.27/home/mo/mark3-source/embedded/stage/src/control_notification.cpp File Reference	263
17.27.1 Detailed Description	263
17.28control_notification.cpp	263
17.29/home/mo/mark3-source/embedded/stage/src/control_notification.h File Reference	265
17.29.1 Detailed Description	265
17.30control_notification.h	265
17.31/home/mo/mark3-source/embedded/stage/src/control_panel.cpp File Reference	266
17.31.1 Detailed Description	266
17.32control_panel.cpp	266
17.33/home/mo/mark3-source/embedded/stage/src/control_panel.h File Reference	267
17.33.1 Detailed Description	267
17.34control_panel.h	267
17.35/home/mo/mark3-source/embedded/stage/src/control_progress.cpp File Reference	268
17.35.1 Detailed Description	268
17.36control_progress.cpp	268
17.37/home/mo/mark3-source/embedded/stage/src/control_progress.h File Reference	269
17.37.1 Detailed Description	269
17.38control_progress.h	270
17.39/home/mo/mark3-source/embedded/stage/src/control_slickbutton.h File Reference	270
17.39.1 Detailed Description	271
17.40control_slickbutton.h	271
17.41/home/mo/mark3-source/embedded/stage/src/control_slickprogress.cpp File Reference	271
17.41.1 Detailed Description	272
17.42control_slickprogress.cpp	272
17.43/home/mo/mark3-source/embedded/stage/src/control_slickprogress.h File Reference	273
17.43.1 Detailed Description	273

17.44control_slickprogress.h	273
17.45/home/mo/mark3-source/embedded/stage/src/dcpu.cpp File Reference	274
17.45.1 Detailed Description	274
17.46dcpu.cpp	275
17.47/home/mo/mark3-source/embedded/stage/src/dcpu.h File Reference	285
17.47.1 Detailed Description	287
17.47.2 Macro Definition Documentation	287
17.47.2.1 DCPU_NORMAL_OPCODE_MASK	287
17.47.3 Enumeration Type Documentation	287
17.47.3.1 DCPU_OpBasic	287
17.47.3.2 DCPU_OpExtended	288
17.48dcpu.h	289
17.49/home/mo/mark3-source/embedded/stage/src/debug_tokens.h File Reference	293
17.49.1 Detailed Description	294
17.50debug_tokens.h	295
17.51/home/mo/mark3-source/embedded/stage/src/draw.h File Reference	296
17.51.1 Detailed Description	297
17.52draw.h	297
17.53/home/mo/mark3-source/embedded/stage/src/driver.cpp File Reference	299
17.53.1 Detailed Description	299
17.54driver.cpp	300
17.55/home/mo/mark3-source/embedded/stage/src/driver.h File Reference	301
17.55.1 Detailed Description	301
17.55.2 Intro	301
17.55.3 Driver Design	301
17.55.4 Driver API	302
17.56driver.h	302
17.57/home/mo/mark3-source/embedded/stage/src/eventflag.cpp File Reference	303
17.57.1 Detailed Description	304
17.58eventflag.cpp	304
17.59/home/mo/mark3-source/embedded/stage/src/eventflag.h File Reference	308
17.59.1 Detailed Description	309
17.60eventflag.h	309
17.61/home/mo/mark3-source/embedded/stage/src/fixed_heap.cpp File Reference	310
17.61.1 Detailed Description	310
17.62fixed_heap.cpp	310
17.63/home/mo/mark3-source/embedded/stage/src/fixed_heap.h File Reference	312
17.63.1 Detailed Description	312
17.64fixed_heap.h	312
17.65/home/mo/mark3-source/embedded/stage/src/font.h File Reference	313

17.65.1 Detailed Description	314
17.66font.h	314
17.67/home/mo/mark3-source/embedded/stage/src/graphics.cpp File Reference	314
17.67.1 Detailed Description	315
17.68graphics.cpp	315
17.69/home/mo/mark3-source/embedded/stage/src/graphics.h File Reference	327
17.69.1 Detailed Description	328
17.70graphics.h	328
17.71/home/mo/mark3-source/embedded/stage/src/gui.cpp File Reference	329
17.71.1 Detailed Description	329
17.72gui.cpp	329
17.73/home/mo/mark3-source/embedded/stage/src/gui.h File Reference	337
17.73.1 Detailed Description	339
17.73.2 Enumeration Type Documentation	339
17.73.2.1 GuiEventType_t	339
17.73.2.2 GuiReturn_t	339
17.74gui.h	339
17.75/home/mo/mark3-source/embedded/stage/src/kernel.cpp File Reference	344
17.75.1 Detailed Description	345
17.76kernel.cpp	345
17.77/home/mo/mark3-source/embedded/stage/src/kernel.h File Reference	346
17.77.1 Detailed Description	346
17.78kernel.h	346
17.79/home/mo/mark3-source/embedded/stage/src/kernel_debug.h File Reference	347
17.79.1 Detailed Description	347
17.80kernel_debug.h	347
17.81/home/mo/mark3-source/embedded/stage/src/kernelswi.cpp File Reference	348
17.81.1 Detailed Description	349
17.82kernelswi.cpp	349
17.83/home/mo/mark3-source/embedded/stage/src/kernelswi.h File Reference	350
17.83.1 Detailed Description	350
17.84kernelswi.h	350
17.85/home/mo/mark3-source/embedded/stage/src/kerneltimer.cpp File Reference	351
17.85.1 Detailed Description	351
17.86kerneltimer.cpp	351
17.87/home/mo/mark3-source/embedded/stage/src/kerneltimer.h File Reference	353
17.87.1 Detailed Description	353
17.88kerneltimer.h	354
17.89/home/mo/mark3-source/embedded/stage/src/kerneltypes.h File Reference	354
17.89.1 Detailed Description	355

17.90kerneltypes.h	355
17.91/home/mo/mark3-source/embedded/stage/src/keycodes.h File Reference	356
17.91.1 Detailed Description	356
17.92keycodes.h	356
17.93/home/mo/mark3-source/embedded/stage/src/kprofile.cpp File Reference	358
17.93.1 Detailed Description	358
17.94kprofile.cpp	358
17.95/home/mo/mark3-source/embedded/stage/src/kprofile.h File Reference	359
17.95.1 Detailed Description	360
17.96kprofile.h	360
17.97/home/mo/mark3-source/embedded/stage/src/ksemaphore.cpp File Reference	361
17.97.1 Detailed Description	361
17.98ksemaphore.cpp	361
17.99/home/mo/mark3-source/embedded/stage/src/ksemaphore.h File Reference	365
17.99.1 Detailed Description	365
17.100ksemaphore.h	365
17.101/home/mo/mark3-source/embedded/stage/src/ll.cpp File Reference	366
17.101.1 Detailed Description	366
17.102ll.cpp	366
17.103/home/mo/mark3-source/embedded/stage/src/ll.h File Reference	368
17.103.1 Detailed Description	369
17.104ll.h	369
17.105/home/mo/mark3-source/embedded/stage/src/manual.h File Reference	370
17.105.1 Detailed Description	370
17.106manual.h	371
17.107/home/mo/mark3-source/embedded/stage/src/mark3cfg.h File Reference	371
17.107.1 Detailed Description	372
17.107.2 Macro Definition Documentation	372
17.107.2.1GLOBAL_MESSAGE_POOL_SIZE	372
17.107.2.2KERNEL_TIMERS_TICKLESS	372
17.107.2.3KERNEL_USE_ATOMIC	372
17.107.2.4KERNEL_USE_DYNAMIC_THREADS	373
17.107.2.5KERNEL_USE_EVENTFLAG	373
17.107.2.6KERNEL_USE_MESSAGE	373
17.107.2.7KERNEL_USE_PROFILER	373
17.107.2.8KERNEL_USE_QUANTUM	373
17.107.2.9KERNEL_USE_SEMAPHORE	373
17.107.2.10KERNEL_USE_THREADNAME	373
17.107.2.11KERNEL_USE_TIMERS	374
17.107.2.12SAFE_UNLINK	374

17.107.2.1THREAD_QUANTUM_DEFAULT	374
17.107.2.1TRANSACTION_QUEUE_SIZE	374
17.108mark3cfg.h	374
17.109home/mo/mark3-source/embedded/stage/src/memutil.cpp File Reference	375
17.109.1Detailed Description	375
17.110memutil.cpp	375
17.111home/mo/mark3-source/embedded/stage/src/memutil.h File Reference	381
17.111.1Detailed Description	381
17.112memutil.h	381
17.113home/mo/mark3-source/embedded/stage/src/message.cpp File Reference	382
17.113.1Detailed Description	382
17.114message.cpp	382
17.115home/mo/mark3-source/embedded/stage/src/message.h File Reference	384
17.115.1Detailed Description	385
17.115.2Using Messages, Queues, and the Global Message Pool	385
17.116message.h	385
17.117home/mo/mark3-source/embedded/stage/src/mutex.cpp File Reference	387
17.117.1Detailed Description	387
17.118mutex.cpp	387
17.119home/mo/mark3-source/embedded/stage/src/mutex.h File Reference	391
17.119.1Detailed Description	392
17.119.2initializing	392
17.119.3Resource protection example	392
17.120mutex.h	392
17.121home/mo/mark3-source/embedded/stage/src/nlfs.cpp File Reference	393
17.121.1Detailed Description	393
17.122nlfs.cpp	393
17.123home/mo/mark3-source/embedded/stage/src/nlfs.h File Reference	405
17.123.1Detailed Description	406
17.123.2Enumeration Type Documentation	407
17.123.2.1NLFS_Type_t	407
17.124nlfs.h	407
17.125home/mo/mark3-source/embedded/stage/src/nlfs_config.h File Reference	410
17.125.1Detailed Description	410
17.126nlfs_config.h	410
17.127home/mo/mark3-source/embedded/stage/src/nlfs_file.cpp File Reference	411
17.127.1Detailed Description	411
17.128nlfs_file.cpp	411
17.129home/mo/mark3-source/embedded/stage/src/nlfs_file.h File Reference	414
17.129.1Detailed Description	415

17.129.2 Enumeration Type Documentation	415
17.129.2.1 NLFS_File_Mode	415
17.130 nlfs_file.h	415
17.131 home/mo/mark3-source/embedded/stage/src/nlfs_ram.cpp File Reference	416
17.131.1 Detailed Description	416
17.132 nlfs_ram.cpp	416
17.133 home/mo/mark3-source/embedded/stage/src/nlfs_ram.h File Reference	418
17.133.1 Detailed Description	418
17.134 nlfs_ram.h	418
17.135 home/mo/mark3-source/embedded/stage/src/panic_codes.h File Reference	418
17.135.1 Detailed Description	419
17.136 panic_codes.h	419
17.137 home/mo/mark3-source/embedded/stage/src/profile.cpp File Reference	419
17.137.1 Detailed Description	420
17.138 profile.cpp	420
17.139 home/mo/mark3-source/embedded/stage/src/profile.h File Reference	421
17.139.1 Detailed Description	422
17.140 profile.h	422
17.141 home/mo/mark3-source/embedded/stage/src/quantum.cpp File Reference	423
17.141.1 Detailed Description	423
17.142 quantum.cpp	423
17.143 home/mo/mark3-source/embedded/stage/src/quantum.h File Reference	425
17.143.1 Detailed Description	425
17.144 quantum.h	425
17.145 home/mo/mark3-source/embedded/stage/src/scheduler.cpp File Reference	426
17.145.1 Detailed Description	426
17.146 scheduler.cpp	426
17.147 home/mo/mark3-source/embedded/stage/src/scheduler.h File Reference	427
17.147.1 Detailed Description	428
17.148 scheduler.h	428
17.149 home/mo/mark3-source/embedded/stage/src/screen.cpp File Reference	429
17.149.1 Detailed Description	429
17.150 screen.cpp	429
17.151 home/mo/mark3-source/embedded/stage/src/screen.h File Reference	430
17.151.1 Detailed Description	430
17.152 screen.h	430
17.153 home/mo/mark3-source/embedded/stage/src/shell_support.cpp File Reference	432
17.153.1 Detailed Description	432
17.154 shell_support.cpp	432
17.155 home/mo/mark3-source/embedded/stage/src/shell_support.h File Reference	434

17.155.1Detailed Description	434
17.155.2Typedef Documentation	435
17.155.2.1fp_internal_command	435
17.156hell_support.h	435
17.157home/mo/mark3-source/embedded/stage/src/slip.cpp File Reference	436
17.157.1Detailed Description	436
17.158slip.cpp	436
17.159home/mo/mark3-source/embedded/stage/src/slip.h File Reference	439
17.159.1Detailed Description	440
17.159.2Enumeration Type Documentation	440
17.159.2.1SlipChannel	440
17.160slip.h	440
17.161home/mo/mark3-source/embedded/stage/src/slip_mux.cpp File Reference	441
17.161.1Detailed Description	442
17.161.2Function Documentation	442
17.161.2.1SlipMux_CallBack	442
17.162slip_mux.cpp	442
17.163home/mo/mark3-source/embedded/stage/src/slip_mux.h File Reference	443
17.163.1Detailed Description	444
17.164slip_mux.h	444
17.165home/mo/mark3-source/embedded/stage/src/slipterm.cpp File Reference	444
17.165.1Detailed Description	445
17.166slipterm.cpp	445
17.167home/mo/mark3-source/embedded/stage/src/slipterm.h File Reference	445
17.167.1Detailed Description	446
17.168slipterm.h	446
17.169home/mo/mark3-source/embedded/stage/src/system_heap.cpp File Reference	447
17.169.1Detailed Description	447
17.170system_heap.cpp	447
17.171home/mo/mark3-source/embedded/stage/src/system_heap.h File Reference	449
17.171.1Detailed Description	450
17.171.2Macro Definition Documentation	450
17.171.2.1HEAP_RAW_SIZE	450
17.171.2.2HEAP_RAW_SIZE_1	450
17.172system_heap.h	450
17.173home/mo/mark3-source/embedded/stage/src/system_heap_config.h File Reference	453
17.173.1Detailed Description	453
17.173.2Macro Definition Documentation	454
17.173.2.1HEAP_BLOCK_SIZE_1	454
17.174system_heap_config.h	454

17.175	home/mo/mark3-source/embedded/stage/src/thread.cpp File Reference	454
17.175.1	Detailed Description	455
17.176	hread.cpp	455
17.177	home/mo/mark3-source/embedded/stage/src/thread.h File Reference	459
17.177.1	Detailed Description	460
17.178	hread.h	460
17.179	home/mo/mark3-source/embedded/stage/src/threadlist.cpp File Reference	462
17.179.1	Detailed Description	462
17.180	hreadlist.cpp	462
17.181	home/mo/mark3-source/embedded/stage/src/threadlist.h File Reference	464
17.181.1	Detailed Description	464
17.182	hreadlist.h	464
17.183	home/mo/mark3-source/embedded/stage/src/threadport.cpp File Reference	465
17.183.1	Detailed Description	465
17.184	hreadport.cpp	465
17.185	home/mo/mark3-source/embedded/stage/src/threadport.h File Reference	467
17.185.1	Detailed Description	468
17.185.2	Macro Definition Documentation	468
17.185.2.1	CS_ENTER	468
17.185.2.2	CS_EXIT	468
17.186	hreadport.h	468
17.187	home/mo/mark3-source/embedded/stage/src/timerlist.cpp File Reference	470
17.187.1	Detailed Description	470
17.188	merlist.cpp	470
17.189	home/mo/mark3-source/embedded/stage/src/timerlist.h File Reference	474
17.189.1	Detailed Description	475
17.189.2	Macro Definition Documentation	475
17.189.2.1	TIMERLIST_FLAG_EXPIRED	475
17.190	merlist.h	475
17.191	home/mo/mark3-source/embedded/stage/src/tracebuffer.cpp File Reference	477
17.191.1	Detailed Description	478
17.192	racebuffer.cpp	478
17.193	home/mo/mark3-source/embedded/stage/src/tracebuffer.h File Reference	478
17.193.1	Detailed Description	478
17.194	racebuffer.h	479
17.195	home/mo/mark3-source/embedded/stage/src/transaction.cpp File Reference	479
17.195.1	Detailed Description	479
17.196	ansaction.cpp	479
17.197	home/mo/mark3-source/embedded/stage/src/transaction.h File Reference	480
17.197.1	Detailed Description	481

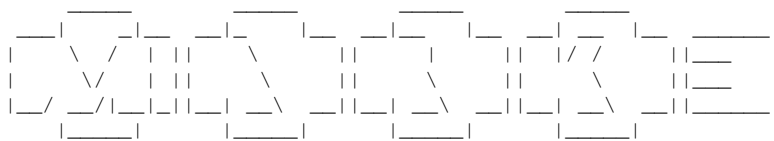
17.198	ransaction.h	481
17.199	home/mo/mark3-source/embedded/stage/src/unit_test.cpp File Reference	482
17.199	Detailed Description	482
17.200	unit_test.cpp	482
17.201	home/mo/mark3-source/embedded/stage/src/unit_test.h File Reference	483
17.201	Detailed Description	483
17.202	unit_test.h	483
17.203	home/mo/mark3-source/embedded/stage/src/writebuf16.cpp File Reference	485
17.203	Detailed Description	485
17.204	writebuf16.cpp	485
17.205	home/mo/mark3-source/embedded/stage/src/writebuf16.h File Reference	487
17.205	Detailed Description	487
17.206	writebuf16.h	487

Index

489

Chapter 1

The Mark3 Realtime Kernel



--[Mark3 Realtime Platform]-----

Copyright (c) 2012-2013 Funkenstein Software Consulting, all rights reserved.
See license.txt for more information

The Mark3 Realtime [Kernel](#) is a completely free, open-source, real-time operating system aimed at bringing multi-tasking to microcontroller systems without MMUs.

It uses modern programming languages and concepts (it's written entirely in C++) to minimize code duplication, and its object-oriented design enhances readability. The API is simple - there are only six functions required to set up the kernel, initialize threads, and start the scheduler.

The source is fully-documented with example code provided to illustrate concepts. The result is a performant RTOS, which is easy to read, easy to understand, and easy to extend to fit your needs.

But Mark3 is bigger than just a real-time kernel, it also contains a number of class-leading features:

- Device driver HAL which provides a meaningful abstraction around device-specific peripherals.
- Capable recursive-make driven build system which can be used to build all libraries, examples, tests, and documentation for any number of targets from the command-line.
- Graphics and UI code designed to simplify the implementation of systems using displays, keypads, joysticks, and touchscreens
- Standards-based custom communications protocol used to simplify the creation of host tools
- A bulletproof, well-documented bootloader for AVR microcontrollers

Chapter 2

Preface

2.1 Who should read this

As the cover clearly states, this is a book about the Mark3 real-time kernel. I assume that if you're reading this book you have an interest in some, if not all, of the following subjects:

- Embedded systems
- Real-time systems
- Operating system kernel design

And if you're interested in those topics, you're likely familiar with C and C++ and the more you know, the easier you'll find this book to read. And if C++ scares you, and you don't like embedded, real-time systems, you're probably looking for another book. If you're unfamiliar with RTOS fundamentals, I highly suggest searching through the vast amount of RTOS-related articles on the internet to familiarize yourself with the concepts.

2.2 Why Mark3?

My first job after graduating from university in 2005 was with a small company that had a very old-school, low-budget philosophy when it came to software development. Every make-or-buy decision ended with "make" when it came to tools. It was the kind of environment where vendors cost us money, but manpower was free. In retrospect, we didn't have a ton of business during the time that I worked there, and that may have had something to do with the fact that we were constantly short on ready cash for things we could code ourselves.

Early on, I asked why we didn't use industry-standard tools - like JTAG debuggers or IDEs. One senior engineer scoffed that debuggers were tools for wimps - and something that a good programmer should be able to do without. After all - we had serial ports, GPIOs, and a bi-color LED on our boards. Since these were built into the hardware, they didn't cost us a thing. We also had a single software "build" server that took 5 minutes to build a 32k binary on its best days, so when we had to debug code, it was a painful process of trial and error, with lots of Youtube between iterations. We complained that tens of thousands of dollars of productivity was being flushed away that could have been solved by implementing a proper build server - and while we eventually got our wish, it took far more time than it should have.

Needless to say, software development was painful at that company. We made life hard on ourselves purely out of pride, and for the right to say that we walked "up-hills both ways through 3 feet of snow, everyday". Our code was tied ever-so-tightly to our hardware platform, and the system code was indistinguishable from the application. While we didn't use an RTOS, we had effectively implemented a 3-priority threading scheme using a carefully designed interrupt nesting scheme with event flags and a while(1) superloop running as a background thread. Nothing was abstracted, and the code was always optimized for the platform, presumably in an effort to save on code size and wasted cycles. I asked why we didn't use an RTOS in any of our systems and received dismissive scoffs - the overhead from thread switching and maintaining multiple threads could not be tolerated in our systems according

to our chief engineers. In retrospect, our ad-hoc system was likely as large as my smallest kernel, and had just as much context switching (although it was hidden by the compiler).

And every time a new iteration of our product was developed, the firmware took far too long to bring up, because the algorithms and data structures had to be re-tooled to work with the peripherals and sensors attached to the new boards. We worked very hard in an attempt to reinvent the wheel, all in the name of producing "efficient" code.

Regardless, I learned a lot about software development.

Most important, I learned that good design is the key to good software; and good design doesn't have to come at a price. In all but the smallest of projects, the well-designed, well-abstracted code is not only more portable, but it's usually smaller, easier to read, and easier to reuse.

Also, since we had all the time in the world to invest in developing our own tools, I gained a lot of experience building them, and making use of good, free PC tools that could be used to develop and debug a large portion of our code. I ended up writing PC-based device and peripheral simulators, state-machine frameworks, and abstractions for our horrible ad-hoc system code. At the end of the day, I had developed enough tools that I could solve a lot of our development problems without having to re-inventing the wheel at each turn. Gaining a background in how these tools worked gave me a better understanding of how to use them - making me more productive at the jobs that I've had since.

I am convinced that designing good software takes honest effort up-front, and that good application code cannot be written unless it is based on a solid framework. Just as the wise man builds his house on rocks, and not on sand, wise developers write applications based on a well-defined platforms. And while you can probably build a house using nothing but a hammer and sheer will, you can certainly build one a lot faster with all the right tools.

This conviction lead me to development my first RTOS kernel in 2009 - FunkOS. It is a small, yet surprisingly full-featured kernel. It has all the basics (semaphores, mutexes, round-robin and preemptive scheduling), and some pretty advanced features as well (device drivers and other middleware). However, it had two major problems - it doesn't scale well, and it doesn't support many devices.

While I had modest success with this kernel (it has been featured on some blogs, and still gets around 125 downloads a month), it was nothing like the success of other RTOS kernels like uC/OS-II and FreeRTOS. To be honest, as a one-man show, I just don't have the resources to support all of the devices, toolchains, and evaluation boards that a real vendor can. I had never expected my kernel to compete with the likes of them, and I don't expect Mark3 to change the embedded landscape either.

My main goal with Mark3 was to solve the technical shortfalls in the FunkOS kernel by applying my experience in kernel development. As a result, Mark3 is better than FunkOS in almost every way; it scales better, has lower interrupt latency, and is generally more thoughtfully designed (all at a small cost to code size).

Another goal I had was to create something easy to understand, that could be documented and serve as a good introduction to RTOS kernel design. The end result of these goals is the kernel as presented in this book - a full source listing of a working OS kernel, with each module completely documented and explained in detail.

Finally, I wanted to prove that a kernel written entirely in C++ could perform just as well as one written in C, without incurring any extra overhead. Comparing the same configuration of Mark2 to Mark3, the code size is remarkably similar, and the execution performance is just as good. Not only that, but there are fewer lines of code. The code is more readable and easier to understand as a result of making use of object-oriented concepts provided by C++. Applications are easier to write because common concepts are encapsulated into objects (Threads, Semaphores, Mutexes, etc.) with their own methods and data, as opposed to APIs which rely on lots of explicit pointer-passing, type casting, and other operations that are typically considered "unsafe" or "advanced topics" in C.

Chapter 3

Can you Afford an RTOS?

Of course, since you're reading the manual for an RTOS that I've been developing for the last few years, you can guess that the conclusion that I draw is a resounding "yes".

If your code is of any sort of non-trivial complexity (say, at least a few-thousand lines), then a more appropriate question would be "can you afford **not** to use an RTOS in your system?".

In short, there are simply too many benefits of an RTOS to ignore.

- Sophisticated synchronization objects
- The ability to efficiently block and wait
- Enhanced responsiveness for high-priority tasks
- Built in timers
- Built in efficient memory management

Sure, these features have a cost in code space and RAM, but from my experience the cost of trying to code around a lack of these features will cost you as much - if not more. The results are often far less maintainable, error prone, and complex. And that simply adds time and cost. Real developers ship, and the RTOS is quickly becoming one of the standard tools that help keep developers shipping.

3.1 Intro

(Note - this article was written for the C-based Mark2 kernel, which is slightly different. While the general principles are the same, the numbers are not an 100% accurate reflection of the current costs of the Mark3 kernel.)

One of the main arguments against using an RTOS in an embedded project is that the overhead incurred is too great to be justified. Concerns over "wasted" RAM caused by using multiple stacks, added CPU utilization, and the "large" code footprint from the kernel cause a large number of developers to shun using a preemptive RTOS, instead favoring a non-preemptive, application-specific solution.

I believe that not only is the impact negligible in most cases, but that the benefits of writing an application with an RTOS can lead to savings around the board (code size, quality, reliability, and development time). While these other benefits provide the most compelling case for using an RTOS, they are far more challenging to demonstrate in a quantitative way, and are clearly documented in numerous industry-based case studies.

While there is some overhead associated with an RTOS, the typical arguments are largely unfounded when an RTOS is correctly implemented in a system. By measuring the true overhead of a preemptive RTOS in a typical application, we will demonstrate that the impact to code space, RAM, and CPU usage is minimal, and indeed acceptable for a wide range of CPU targets.

To illustrate just how little an RTOS impacts the size of an embedded software design we will look at a typical microcontroller project and analyze the various types of overhead associated with using a pre-emptive realtime kernel versus a similar non-preemptive event-based framework.

RTOS overhead can be broken into three distinct areas:

- Code space: The amount of code space eaten up by the kernel (static)
- Memory overhead: The RAM associated with running the kernel and application threads.
- Runtime overhead: The CPU cycles required for the kernel's functionality (primarily scheduling and thread switching)

While there are other notable reasons to include or avoid the use of an RTOS in certain applications (determinism, responsiveness, and interrupt latency among others), these are not considered in this discussion - as they are difficult to consider for the scope of our "canned" application. Application description:

For the purpose of this comparison, we first create an application using the standard preemptive Mark3 kernel with 2 system threads running: A foreground thread and a background thread. This gives three total priority levels in the system - the interrupt level (high), and two application priority threads (medium and low), which is quite a common paradigm for microcontroller firmware designs. The foreground thread processes a variety of time-critical events at a fixed frequency, while the background thread processes lower priority, aperiodic events. When there are no background thread events to process, the processor enters its low-power mode until the next interrupt is acknowledged.

The contents of the threads themselves are unimportant for this comparison, but we can assume they perform a variety of I/O using various user-input devices and a serial graphics display. As a result, a number of Mark3 device drivers are also implemented.

The application is compiled for an ATmega328p processor which contains 32kB of code space in flash, and 2kB of RAM, which is a lower-mid-range microcontroller in Atmel's 8-bit AVR line of microcontrollers. Using the WinAVR GCC compiler with -O2 level optimizations, an executable is produced with the following code/RAM utilization:

31600 Bytes Code Space 2014 Bytes RAM

An alternate version of this project is created using a custom "super-loop" kernel, which uses a single application thread and provides 2 levels of priority (interrupt and application). In this case, the event handler processes the different priority application events to completion from highest to lowest priority.

This approach leaves the application itself largely unchanged. Using the same optimization levels as the preemptive kernel, the code compiles as follows:

29904 Bytes Code Space 1648 Bytes RAM

3.2 Memory overhead:

At first glance, the difference in RAM utilization seems quite a lot higher for the preemptive mode version of the application, but the raw numbers don't tell the whole story.

The first issue is that the cooperative-mode total does not take into account the system stack - whereas these values are included in the totals for RTOS version of the project. As a result, some further analysis is required to determine how the stack sizes truly compare.

In cooperative mode, there is only one thread of execution - so considering that multiple event handlers are executed in turn, the stack requirements for cooperative mode is simply determined by those of the most stack-intensive event handler.

In contrast, the preemptive kernel requires a separate stack for each active thread, and as a result the stack usage of the system is the sum of the stacks for all threads.

Since the application and idle events are the same for both preemptive and cooperative mode, we know that their (independent) stack requirements will be the same in both cases.

For cooperative mode, we see that the idle thread stack utilization is lower than that of the application thread, and so the application thread's determines the stack size requirement. Again, with the preemptive kernel the stack utilization is the sum of the stacks defined for both threads.

As a result, the difference in overhead between the two cases becomes the extra stack required for the idle thread - which in our case is (a somewhat generous) 64 bytes.

The numbers still don't add up completely, but looking into the linker output we see that the rest of the difference comes from the extra data structures used to declare the threads in preemptive mode.

With this taken into account, the true memory cost of a 2-thread system ends up being around 150 bytes of RAM - which is less than 8% of the total memory available on this particular microcontroller. Whether or not this is reasonable certainly depends on the application, but more importantly, it is not so unreasonable as to eliminate an RTOS-based solution from being considered.

3.3 Code Space Overhead:

The difference in code space overhead between the preemptive and cooperative mode solutions is less of an issue. Part of this reason is that both the preemptive and cooperative kernels are relatively small, and even an average target device (like the Atmega328 we've chosen) has plenty of room.

Mark3 can be configured so that only features necessary for the application are included in the RTOS - you only pay for the parts of the system that you use. In this way, we can measure the overhead on a feature-by-feature basis, which is shown below for the kernel as configured for this application:

3466 Bytes

The configuration tested in this comparison uses the thread/port module with timers, drivers, and semaphores, for a total kernel size of ~3.5KB, with the rest of the code space occupied by the application.

The custom cooperative-mode framework has a similar structure which is broken down by module as follows:

1850 Bytes

As can be seen from the compiler's output, the difference in code space between the two versions of the application is about 1.7kB - or about 5% of the available code space on the selected processor. While nearly all of this comes from the added overhead of the kernel, the rest of the difference comes the changes to the application necessary to facilitate the different frameworks.

3.4 Runtime Overhead

On the cooperative kernel, the overhead associated with running the thread is the time it takes the kernel to notice a pending event flag and launch the appropriate event handler, plus the timer interrupt execution time.

Similarly, on the preemptive kernel, the overhead is the time it takes to switch contexts to the application thread, plus the timer interrupt execution time.

The timer interrupt overhead is similar for both cases, so the overhead then becomes the difference between the following:

Preemptive mode:

- Posting the semaphore that wakes the high-priority thread
- Performing a context switch to the high-priority thread

Cooperative mode:

- Setting the high-priority thread's event flag
- Acknowledging the event from the event loop

Using the cycle-accurate AVR simulator, we find the end-to-end event sequence time to be 20.4us for the cooperative mode scheduler and 44.2us for the preemptive, giving a difference of 23.8us.

With a fixed high-priority event frequency of 33Hz, we achieve a runtime overhead of 983.4us per second, or 0.0983% of the total available CPU time. Now, obviously this value would expand at higher event frequencies and/or slower CPU frequencies, but for this typical application we find the difference in runtime overhead to be negligible for a preemptive system. Analysis:

For the selected test application and platform, including a preemptive RTOS is entirely reasonable, as the costs are low relative to a non-preemptive kernel solution. But these costs scale relative to the speed, memory and code space of the target processor. Because of these variables, there is no "magic bullet" environment suitable for every application, but Mark3 attempts to provide a framework suitable for a wide range of targets.

On the one hand, if these tests had been performed on a higher-end microcontroller such as the ATmega1284p (containing 128kB of code space and 16kB of RAM), the overhead would be in the noise. For this type of resource-rich microcontroller, there would be no reason to avoid using the Mark3 preemptive kernel.

Conversely, using a lower-end microcontroller like an ATmega88pa (which has only 8kB of code space and 1kB of RAM), the added overhead would likely be prohibitive for including a preemptive kernel. In this case, the cooperative-mode kernel would be a better choice.

As a rule of thumb, if one budgets 10% of a microcontroller's code space/RAM for a preemptive kernel's overhead, you should only require at minimum a microcontroller with 16k of code space and 2kB of RAM as a base platform for an RTOS. Unless there are serious constraints on the system that require much better latency or responsiveness than can be achieved with RTOS overhead, almost any modern platform is sufficient for hosting a kernel. In the event you find yourself with a microprocessor with external memory, there should be no reason to avoid using an RTOS at all.

Chapter 4

Superloops

4.1 Intro to Superloops

Before we start taking a look at designing a real-time operating system, it's worthwhile taking a look through one of the most-common design patterns that developers use to manage task execution in embedded systems - Superloops.

Systems based on superloops favor the system control logic baked directly into the application code, usually under the guise of simplicity, or memory (code and RAM) efficiency. For simple systems, superloops can definitely get the job done. However, they have some serious limitations, and are not suitable for every kind of project. In a lot of cases you can squeak by using superloops - especially in extremely constrained systems, but in general they are not a solid basis for reusable, portable code.

Nonetheless, a variety of examples are presented here- from the extremely simple, to cooperative and limited-preemptive multitasking systems, all of which are examples are representative of real-world systems that I've either written the firmware for, or have seen in my experience.

4.2 The simplest loop

Let's start with the simplest embedded system design possible - an infinite loop that performs a single task repeatedly:

```
int main()
{
    while(1)
    {
        Do_Something();
    }
}
```

Here, the code inside the loop will run a single function forever and ever. Not much to it, is there? But you might be surprised at just how much embedded system firmware is implemented using essentially the same mechanism - there isn't anything wrong with that, but it's just not that interesting.

While the execution timeline for this program is equally boring, for the sake of completeness it would look like this:

Despite its simplicity we can see the beginnings of some core OS concepts. Here, the `while(1)` statement can be logically seen as the operating system kernel - this one control statement determines what tasks can run in the system, and defines the constraints that could modify their execution. But at the end of the day, that's a big part of what a kernel is - a mechanism that controls the execution of application code.

The second concept here is the task. This is application code provided by the user to perform some useful purpose in a system. In this case `Do_something()` represents that task - it could be monitoring blood pressure, reading a sensor and writing its data to a terminal, or playing an MP3; anything you can think of for an embedded system to do. A simple round-robin multi-tasking system can be built off of this example by simply adding additional tasks in

sequence in the main while-loop. Note that in this example the CPU is always busy running tasks - at no time is the CPU idle, meaning that it is likely burning a lot of power.

While we conceptually have two separate pieces of code involved here (an operating system kernel and a set of running tasks), they are not logically separate. The OS code is indistinguishable from the application. It's like a single-celled organism - everything is crammed together within the walls of an indivisible unit; and specialized to perform its given function relying solely on instinct.

4.3 Interrupt-Driven Super-loop

In the previous example, we had a system without any way to control the execution of the task- it just runs forever. There's no way to control when the task can (or more importantly can't) run, which greatly limits the usefulness of the system. Say you only want your task to run every 100 milliseconds - in the previous code, you have to add a hard-coded delay at the end of your task's execution to ensure your code runs only when it should.

Fortunately, there is a much more elegant way to do this. In this example, we introduce the concept of the synchronization object. A Synchronization object is some data structure which works within the bounds of the operating system to tell tasks when they can run, and in many cases includes special data unique to the synchronization event. There are a whole family of synchronization objects, which we'll get into later. In this example, we make use of the simplest synchronization primitive - the global flag.

With the addition of synchronization brings the addition of event-driven systems. If you're programming a microcontroller system, you generally have scores of peripherals available to you - timers, GPIOs, ADCs, UARTs, ethernet, USB, etc. All of which can be configured to provide a stimulus to your system by means of interrupts. This stimulus gives us the ability not only to program our micros to do_something(), but to do_something() if-and-only-if a corresponding trigger has occurred.

The following concepts are shown in the example below:

```
volatile K_BOOL something_to_do = false;

__interrupt__ My_Interrupt_Source(void)
{
    something_to_do = true;
}

int main()
{
    while(1)
    {
        if( something_to_do )
        {
            Do_something();
            something_to_do = false;
        }
        else
        {
            Idle();
        }
    }
}
```

So there you have it - an event driven system which uses a global variable to synchronize the execution of our task based on the occurrence of an interrupt. It's still just a bare-metal, OS-baked-into-the-application system, but it's introduced a whole bunch of added complexity (and control!) into the system.

The first thing to notice in the source is that the global variable, something_to_do, is used as a synchronization object. When an interrupt occurs from some external event, triggering the My_Interrupt_Source() ISR, program flow in main() is interrupted, the interrupt handler is run, and something_to_do is set to true, letting us know that when we get back to main(), that we should run our Do_something() task.

Another new concept at play here is that of the idle function. In general, when running an event driven system, there are times when the CPU has no application tasks to run. In order to minimize power consumption, CPUs usually contain instructions or registers that can be set up to disable non-essential subsets of the system when there's nothing to do. In general, the sleeping system can be re-activated quickly as a result of an interrupt or other external stimulus, allowing normal processing to resume.

Now, we could just call `Do_something()` from the interrupt itself - but that's generally not a great solution. In general, the more time we spend inside an interrupt, the more time we spend with at least some interrupts disabled. As a result, we end up with interrupt latency. Now, in this system, with only one interrupt source and only one task this might not be a big deal, but say that `Do_something()` takes several seconds to complete, and in that time several other interrupts occur from other sources. While executing in our long-running interrupt, no other interrupts can be processed - in many cases, if two interrupts of the same type occur before the first is processed, one of these interrupt events will be lost. This can be utterly disastrous in a real-time system and should be avoided at all costs. As a result, it's generally preferable to use synchronization objects whenever possible to defer processing outside of the ISR.

Another OS concept that is implicitly introduced in this example is that of task priority. When an interrupt occurs, the normal execution of code in `main()` is preempted: control is swapped over to the ISR (which runs to completion), and then control is given back to `main()` where it left off. The very fact that interrupts take precedence over what's running shows that `main` is conceptually a "low-priority" task, and that all ISRs are "high-priority" tasks. In this example, our "high-priority" task is setting a variable to tell our "low-priority" task that it can do something useful. We will investigate the concept of task priority further in the next example.

Preemption is another key principle in embedded systems. This is the notion that whatever the CPU is doing when an interrupt occurs, it should stop, cache its current state (referred to as its context), and allow the high-priority event to be processed. The context of the previous task is then restored its state before the interrupt, and resumes processing. We'll come back to preemption frequently, since the concept comes up frequently in RTOS-based systems.

4.4 Cooperative multi-tasking

Our next example takes the previous example one step further by introducing cooperative multi-tasking:

```
// Bitfield values used to represent three distinct tasks
#define TASK_1_EVENT (0x01)
#define TASK_2_EVENT (0x02)
#define TASK_3_EVENT (0x04)

volatile K_UCHAR event_flags = 0;

// Interrupt sources used to trigger event execution

__interrupt__ My_Interrupt_1(void)
{
    event_flags |= TASK_1_EVENT;
}

__interrupt__ My_Interrupt_2(void)
{
    event_flags |= TASK_2_EVENT;
}

__interrupt__ My_Interrupt_3(void)
{
    event_flags |= TASK_3_EVENT;
}

// Main tasks
int main(void)
{
    while(1)
    {
        while(event_flags)
        {
            if( event_flags & TASK_1_EVENT)
            {
                Do_Task_1();
                event_flags &= ~TASK_1_EVENT;
            } else if( event_flags & TASK_2_EVENT) {
                Do_Task_2();
                event_flags &= ~TASK_2_EVENT;
            } else if( event_flags & TASK_3_EVENT) {
                Do_Task_3();
                event_flags &= ~TASK_3_EVENT;
            }
        }
        Idle();
    }
}
```

This system is very similar to what we had before - however the differences are worth discussing. First, we have stimulus from multiple interrupt sources: each ISR is responsible for setting a single bit in our global event flag, which is then used to control execution of individual tasks from within main().

Next, we can see that tasks are explicitly given priorities inside the main loop based on the logic of the if/else if structure. As long as there is something set in the event flag, we will always try to execute Task1 first, and only when Task1 isn't set will we attempt to execute Task2, and then Task 3. This added logic provides the notion of priority. However, because each of these tasks exist within the same context (they're just different functions called from our main control loop), we don't have the same notion of preemption that we have when dealing with interrupts.

That means that even through we may be running Task2 and an event flag for Task1 is set by an interrupt, the CPU still has to finish processing Task2 to completion before Task1 can be run. And that's why this kind of scheduling is referred to as cooperative multitasking: we can have as many tasks as we want, but unless they cooperate by means of returning back to main, the system can end up with high-priority tasks getting starved for CPU time by lower-priority, long-running tasks.

This is one of the more popular Os-baked-into-the-application approaches, and is widely used in a variety of real-time embedded systems.

4.5 Hybrid cooperative/preemptive multi-tasking

The final variation on the superloop design utilizes software-triggered interrupts to simulate a hybrid cooperative/preemptive multitasking system. Consider the example code below.

```
// Bitfields used to represent high-priority tasks. Tasks in this group
// can preempt tasks in the group below - but not eachother.
#define HP_TASK_1      (0x01)
#define HP_TASK_2      (0x02)

volatile K_UCHAR hp_tasks = 0;

// Bitfields used to represent low-priority tasks.
#define LP_TASK_1      (0x01)
#define LP_TASK_2      (0x02)

volatile K_UCHAR lp_tasks = 0;

// Interrupt sources, used to trigger both high and low priority tasks.
__interrupt__ System_Interrupt_1(void)
{
    // Set any of the other tasks from here...
    hp_tasks |= HP_TASK_1;
    // Trigger the SWI that calls the High_Priority_Tasks interrupt handler
    SWI();
}

__interrupt__ System_Interrupt_n...(void)
{
    // Set any of the other tasks from here...
}

// Interrupt handler that is used to implement the high-priority event context
__interrupt__ High_Priority_Tasks(void)
{
    // Enabled every interrupt except this one
    Disable_My_Interrupt();
    Enable_Interrupts();
    while( hp_tasks)
    {
        if( hp_tasks & HP_TASK_1)
        {
            HP_Task1();
            hp_tasks &= ~HP_TASK_1;
        }
        else if( hp_tasks & HP_TASK_2)
        {
            HP_Task2();
            hp_tasks &= ~HP_TASK_2;
        }
    }
    Restore_Interrupts();
    Enable_My_Interrupt();
}
```

```
// Main loop, used to implement the low-priority events
int main(void)
{
    // Set the function to run when a SWI is triggered
    Set_SWI(High_Priority_Tasks);

    // Run our super-loop
    while(1)
    {
        while (lp_tasks)
        {
            if (lp_tasks & LP_TASK_1)
            {
                LP_Task1();
                lp_tasks &= ~LP_TASK_1;
            }
            else if (lp_tasks & LP_TASK_2)
            {
                LP_Task2();
                lp_tasks &= ~LP_TASK_2;
            }
        }
        Idle();
    }
}
```

In this example, `High_Priority_Tasks()` can be triggered at any time as a result of a software interrupt (SWI). When a high-priority event is set, the code that sets the event calls the SWI as well, which instantly preempts whatever is happening in main, switching to the high-priority interrupt handler. If the CPU is executing in an interrupt handler already, the current ISR completes, at which point control is given to the high priority interrupt handler.

Once inside the HP ISR, all interrupts (except the software interrupt) are re-enabled, which allows this interrupt to be preempted by other interrupt sources, which is called interrupt nesting. As a result, we end up with two distinct execution contexts (main and `HighPriorityTasks()`), in which all tasks in the high-priority group are guaranteed to preempt main() tasks, and will run to completion before returning control back to tasks in main(). This is a very basic preemptive multitasking scenario, approximating a "real" RTOS system with two threads of different priorities.

4.6 Problems with superloops

As mentioned earlier, a lot of real-world systems are implemented using a superloop design; and while they are simple to understand due to the limited and obvious control logic involved, they are not without their problems.

Hidden Costs

It's difficult to calculate the overhead of the superloop and the code required to implement workarounds for blocking calls, scheduling, and preemption. There's a cost in both the logic used to implement workarounds (usually involving state machines), as well as a cost to maintainability that comes with breaking up into chunks based on execution time instead of logical operations. In moderate firmware systems, this size cost can exceed the overhead of a reasonably well-featured RTOS, and the deficit in maintainability is something that is measurable in terms of lost productivity through debugging and profiling.

Tightly-coupled code

Because the control logic is integrated so closely with the application logic, a lot of care must be taken not to compromise the separation between application and system code. The timing loops, state machines, and architecture-specific control mechanisms used to avoid (or simulate) preemption can all contribute to the problem. As a result, a lot of superloop code ends up being difficult to port without effectively simulating or replicating the underlying system for which the application was written. Abstraction layers can mitigate the risks, but a lot of care should be taken to fully decouple the application code from the system code.

No blocking calls

In a super-loop environment, there's no such thing as a blocking call or blocking objects. Tasks cannot stop mid-execution for event-driven I/O from other contexts - they must always run to completion. If busy-waiting and polling are used as a substitute, it increases latency and wastes cycles. As a result, extra code complexity is often times necessary to work-around this lack of blocking objects, often times through implementing additional state machines. In a large enough system, the added overhead in code size and cycles can add up.

Difficult to guarantee responsiveness

Without multiple levels of priority, it may be difficult to guarantee a certain degree of real-time responsiveness without added profiling and tweaking. The latency of a given task in a priority-based cooperative multitasking system is the length of the longest task. Care must be taken to break tasks up into appropriate sized chunks in order to ensure that higher-priority tasks can run in a timely fashion - a manual process that must be repeated as new tasks are added in the system. Once again, this adds extra complexity that makes code larger, more difficult to understand and maintain due to the artificial subdivision of tasks into time-based components.

Limited preemption capability

As shown in the example code, the way to gain preemption in a superloop is through the use of nested interrupts. While this isn't unwieldy for two levels of priority, adding more levels beyond this becomes complicated. In this case, it becomes necessary to track interrupt nesting manually, and separate sets of tasks that can run within given priority loops - and deadlock becomes more difficult to avoid.

Chapter 5

Mark3 Overview

5.1 Intro

The following section details the overall design of Mark3, the goals I've set out to achieve, the features that I've intended to provide, as well as an introduction to the programming concepts used to make it happen.

5.2 Features

Mark3 is a fully-featured real-time kernel, and is feature-competitive with other open-source and commercial RTOS's in the embedded arena.

The key features of this RTOS are:

- Flexible [Scheduler](#)
 - Unlimited number of threads with 8 priority levels
 - Unlimited threads per priority level
 - Round-robin scheduling for threads at each priority level
 - Time quantum scheduling for each thread in a given priority level
- Configurable stacks for each [Thread](#)
- Resource protection:
 - Integrated mutual-exclusion semaphores ([Mutex](#))
 - Priority-inheritance on [Mutex](#) objects to prevent priority inversion
- Synchronization Objects
 - Binary and counting [Semaphore](#) to coordinate thread execution
 - Event flags with 16-bit bitfields for complex thread synchronization
- Efficient Timers
 - The RTOS is tickless, the OS only wakes up when a timer expires, not at a regular interval
 - One-shot and periodic timers with event callbacks
 - Timers are high-precision and long-counting (about 68000 seconds when used with a 16us resolution timer)
- [Driver](#) API
 - A hardware abstraction layer is provided to simplify driver development
- Robust Interprocess Communications
 - Threadsafe global [Message](#) pool and configurable message queues

5.3 Design Goals

Lightweight

Mark3 can be configured to have an extremely low static memory footprint. Each thread is defined with its own stack, and each thread structure can be configured to take as little as 26 bytes of RAM. The complete Mark3 kernel with all features, setup code, a serial driver, and the Mark3 protocol libraries comes in at under 9K of code space and 1K of RAM on atmel AVR.

Modular

Each system feature can be enabled or disabled by modifying the kernel configuration header file. Include what you want, and ignore the rest to save code space and RAM.

Easily Portable

Mark3 should be portable to a variety of 8, 16 and 32 bit architectures without MMUs. Porting the OS to a new architecture is relatively straightforward, requiring only device-specific implementations for the lowest-level operations such as context switching and timer setup.

Easy To Use

Mark3 is small by design - which gives it the advantage that it's also easy to develop for. This manual, the code itself, and the Doxygen documentation in the code provide ample documentation to get you up to speed quickly. Because you get to see the source, there's nothing left to assumption.

Simple to Understand

Not only is the Mark3 API rigorously documented (hey - that's what this book is for!), but the architecture and naming conventions are intuitive - it's easy to figure out where code lives, and how it works. Individual modules are small due to the "one feature per file" rule used in development. This makes Mark3 an ideal platform for learning about aspects of RTOS design.

Chapter 6

Getting Started

6.1 Kernel Setup

This section details the process of defining threads, initializing the kernel, and adding threads to the scheduler.

If you're at all familiar with real-time operating systems, then these setup and initialization steps should be familiar. I've tried very hard to ensure that as much of the heavy lifting is hidden from the user, so that only the bare minimum of calls are required to get things started.

The examples presented in this chapter are real, working examples taken from the ATmega328p port.

First, you'll need to create the necessary data structures and functions for the threads:

1. Create a [Thread](#) object for all of the "root" or "initial" tasks.
2. Allocate stacks for each of the Threads
3. Define an entry-point function for each [Thread](#)

This is shown in the example code below:

```
//-----  
#include "thread.h"  
#include "kernel.h"  
  
//1) Create a thread object for all of the "root" or "initial" tasks  
static Thread AppThread;  
static Thread IdleThread;  
  
//2) Allocate stacks for each thread  
#define STACK_SIZE_APP      (192)  
#define STACK_SIZE_IDLE     (128)  
  
static K_UCHAR aucAppStack[STACK_SIZE_APP];  
static K_UCHAR aucIdleStack[STACK_SIZE_IDLE];  
  
//3) Define entry point functions for each thread  
void AppThread(void);  
void IdleThread(void);
```

Next, we'll need to add the required kernel initialization code to main. This consists of running the [Kernel's](#) init routine, initializing all of the threads we defined, adding the threads to the scheduler, and finally calling [Kernel::Start\(\)](#), which transfers control of the system to the RTOS.

These steps are illustrated in the following example.

```
int main(void)  
{  
    //1) Initialize the kernel prior to use  
    Kernel::Init();  
  
    //2) Initialize all of the threads we've defined
```

```

AppThread.Init(    aucAppStack,
                  STACK_SIZE_APP,
                  1,
                  (void*)AppEntry,
                  NULL );

IdleThread.Init(  aucIdleStack,
                  STACK_SIZE_IDLE,
                  0,
                  4,
                  (void*)IdleEntry,
                  NULL );

//3) Add the threads to the scheduler
AppThread.Start();
IdleThread.Start();

//4) Give control of the system to the kernel
Kernel::Start();
}

```

Not much to it, is there? There are a few noteworthy points in this code, though.

In order for the kernel to work properly, a system must always contain an idle thread; that is, a thread at priority level 0 that never blocks. This thread is responsible for performing any of the low-level power management on the CPU in order to maximize battery life in an embedded device. The idle thread must also never block, and it must never exit. Either of these operations will cause undefined behavior in the system.

The App thread is at a priority level greater-than 0. This ensures that as long as the App thread has something useful to do, it will be given control of the CPU. In this case, if the app thread blocks, control will be given back to the Idle thread, which will put the CPU into a power-saving mode until an interrupt occurs.

Stack sizes must be large enough to accommodate not only the requirements of the threads, but also the requirements of interrupts - up to the maximum interrupt-nesting level used. Stack overflows are super-easy to run into in an embedded system; if you encounter strange and unexplained behavior in your code, chances are good that one of your threads is blowing its stack.

6.2 Threads

Mark3 Threads act as independent tasks in the system. While they share the same address-space, global data, device-drivers, and system peripherals, each thread has its own set of CPU registers and stack, collectively known as the thread's **context**. The context is what allows the RTOS kernel to rapidly switch between threads at a high rate, giving the illusion that multiple things are happening in a system, when really, only one thread is executing at a time.

6.2.1 Thread Setup

Each instance of the [Thread](#) class represents a thread, its stack, its CPU context, and all of the state and metadata maintained by the kernel. Before a [Thread](#) will be scheduled to run, it must first be initialized with the necessary configuration data.

The Init function gives the user the opportunity to set the stack, stack size, thread priority, entry-point function, entry-function argument, and round-robin time quantum:

[Thread](#) stacks are pointers to blobs of memory (usually K_CHAR arrays) carved out of the system's address space. Each thread must have a stack defined that's large enough to handle not only the requirements of local variables in the thread's code path, but also the maximum depth of the ISR stack.

Priorities should be chosen carefully such that the shortest tasks with the most strict determinism requirements are executed first - and are thus located in the highest priorities. Tasks that take the longest to execute (and require the least degree of responsiveness) must occupy the lower thread priorities. The idle thread must be the only thread occupying the lowest priority level.

The thread quantum only applies when there are multiple threads in the ready queue at the same priority level. This interval is used to kick-off a timer that will cycle execution between the threads in the priority list so that they each get a fair chance to execute.

The entry function is the function that the kernel calls first when the thread instance is first started. Entry functions have at most one argument - a pointer to a data-object specified by the user during initialization.

An example thread initialization is shown below:

```
Thread clMyThread;
K_UCHAR aucStack[192];

void AppEntry(void)
{
    while(1)
    {
        // Do something!
    }
}

...
{
    clMyThread.Init(aucStack,
                    192,
                    1,
                    4,
                    (void*)AppEntry,
                    NULL );
}
```

Once a thread has been initialized, it can be added to the scheduler by calling:

```
clMyThread.Start();
```

The thread will be placed into the [Scheduler's](#) queue at the designated priority, where it will wait its turn for execution.

6.2.2 Entry Functions

Mark3 Threads should not run-to-completion - they should execute as infinite loops that perform a series of tasks, appropriately partitioned to provide the responsiveness characteristics desired in the system.

The most basic [Thread](#) loop is shown below:

```
void Thread( void *param )
{
    while(1)
    {
        // Do Something
    }
}
```

Threads can interact with eachother in the system by means of synchronization objects ([Semaphore](#)), mutual-exclusion objects ([Mutex](#)), Inter-process messaging ([MessageQueue](#)), and timers ([Timer](#)).

Threads can suspend their own execution for a predetermined period of time by using the static [Thread::Sleep\(\)](#) method. Calling this will block the [Thread's](#) executin until the amount of time specified has ellapsed. Upon expiry, the thread will be placed back into the ready queue for its priority level, where it awaits its next turn to run.

6.3 Timers

[Timer](#) objects are used to trigger callback events periodic or on a one-shot (alarm) basis.

While extremely simple to use, they provide one of the most powerful execution contexts in the system. The timer callbacks execute from within the timer callback ISR in an interrupt-enabled context. As such, timer callbacks are considered higher-priority than any thread in the system, but lower priority than other interrupts. Care must be taken to ensure that timer callbacks execute as quickly as possible to minimize the impact of processing on the throughput of tasks in the system. Wherever possible, heavy-lifting should be deferred to the threads by way of semaphores or messages.

Below is an example showing how to start a periodic system timer which will trigger every second:

```

{
    Timer clTimer;
    clTimer.Init();

    clTimer.Start( 1000,
                  1,
                  MyCallback,
                  (void*)&my_data );

    ... // Keep doing work in the thread
}

// Callback function, executed from the timer-expiry context.
void MyCallback( Thread *pclOwner_, void *pvData_ )
{
    LED.Flash(); // Flash an LED.
}

```

6.4 Semaphores

Semaphores are used to synchronized execution of threads based on the availability (and quantity) of application-specific resources in the system. They are extremely useful for solving producer-consumer problems, and are the method-of-choice for creating efficient, low latency systems, where ISRs post semaphores that are handled from within the context of individual threads. (Yes, Semaphores can be posted - but not pended - from the interrupt context).

The following is an example of the producer-consumer usage of a binary semaphore:

```

Semaphore clSemaphore; // Declare a semaphore shared between a producer and a consumer thread.

void Producer()
{
    clSemaphore.Init(0, 1);
    while(1)
    {
        // Do some work, create something to be consumed

        // Post a semaphore, allowing another thread to consume the data
        clSemaphore.Post();
    }
}

void Consumer()
{
    // Assumes semaphore initialized before use...
    While(1)
    {
        // Wait for new data from the producer thread
        clSemaphore.Pend();

        // Consume the data!
    }
}

```

And an example of using semaphores from the ISR context to perform event- driven processing.

```

Semaphore clSemaphore;

__interrupt__ MyISR()
{
    clSemaphore.Post(); // Post the interrupt. Lightweight when uncontested.
}

void MyThread()
{
    clSemaphore.Init(0, 1); // Ensure this is initialized before the MyISR interrupt is enabled.
    while(1)
    {
        // Wait until we get notification from the interrupt
        clSemaphore.Pend();

        // Interrupt has fired, do the necessary work in this thread's context
        HeavyLifting();
    }
}

```

6.5 Mutexes

Mutexes (Mutual exclusion objects) are provided as a means of creating "protected sections" around a particular resource, allowing for access of these objects to be serialized. Only one thread can hold the mutex at a time - other threads have to wait until the region is released by the owner thread before they can take their turn operating on the protected resource. Note that mutexes can only be owned by threads - they are not available to other contexts (i.e. interrupts). Calling the mutex APIs from an interrupt will cause catastrophic system failures.

Note that these objects are also not recursive- that is, the owner thread can not attempt to claim a mutex more than once.

Priority inheritance is provided with these objects as a means to avoid priority inversions. Whenever a thread at a priority than the mutex owner blocks on a mutex, the priority of the current thread is boosted to the highest-priority waiter to ensure that other tasks at intermediate priorities cannot artificially prevent progress from being made.

Mutex objects are very easy to use, as there are only three operations supported: Initialize, Claim and Release. An example is shown below.

```

Mutex clMutex;    // Create a mutex globally.

void Init()
{
    // Initialize the mutex before use.
    clMutex.Init();
}

// Some function called from a thread
void Thread1Function()
{
    clMutex.Claim();

    // Once the mutex is owned, no other thread can
    // enter a block protect by the same mutex

    my_protected_resource.do_something();
    my_protected_resource.do_something_else();

    clMutex.Release();
}

// Some function called from another thread
void Thread2Function()
{
    clMutex.Claim();

    // Once the mutex is owned, no other thread can
    // enter a block protect by the same mutex

    my_protected_resource.do_something();
    my_protected_resource.do_different_things();

    clMutex.Release();
}

```

6.6 Event Flags

Event Flags are another synchronization object, conceptually similar to a semaphore.

Unlike a semaphore, however, the condition on which threads are unblocked is determined by a more complex set of rules. Each Event Flag object contains a 16-bit field, and threads block, waiting for combinations of bits within this field to become set.

A thread can wait on any pattern of bits from this field to be set, and any number of threads can wait on any number of different patterns. Threads can wait on a single bit, multiple bits, or bits from within a subset of bits within the field.

As a result, setting a single value in the flag can result in any number of threads becoming unblocked simultaneously. This mechanism is extremely powerful, allowing for all sorts of complex, yet efficient, thread synchronization schemes that can be created using a single shared object.

Note that Event Flags can be set from interrupts, but you cannot wait on an event flag from within an interrupt.

Examples demonstrating the use of event flags are shown below.

```
// Simple example showing a thread blocking on a multiple bits in the
// fields within an event flag.

EventFlag clEventFlag;

int main()
{
    ...
    clEventFlag.Init(); // Initialize event flag prior to use
    ...
}

void MyInterrupt()
{
    // Some interrupt corresponds to event 0x0020
    clEventFlag.Set(0x0020);
}

void MyThreadFunc()
{
    ...
    while(1)
    {
        ...
        K_USHORT usWakeCondition;

        // Allow this thread to block on multiple flags
        usWakeCondition = clEventFlag.Wait(0x00FF, EVENT_FLAG_ANY);

        // Clear the event condition that caused the thread to wake (in this case,
        // usWakeCondition will equal 0x20 when triggered from the interrupt above)
        clEventFlag.Clear(usWakeCondition);

        // <do something>
    }
}
```

6.7 Messages

Sending messages between threads is the key means of synchronizing access to data, and the primary mechanism to perform asynchronous data processing operations.

Sending a message consists of the following operations:

- Obtain a [Message](#) object from the global message pool
- Set the message data and event fields
- Send the message to the destination message queue

While receiving a message consists of the following steps:

- Wait for a messages in the destination message queue
- Process the message data
- Return the message back to the global message pool

These operations, and the various data objects involved are discussed in more detail in the following section.

6.7.1 Message Objects

[Message](#) objects are used to communicate arbitrary data between threads in a safe and synchronous way.

The message object consists of an event code field and a data field. The event code is used to provide context to the message object, while the data field (essentially a void * data pointer) is used to provide a payload of data corresponding to the particular event.

Access to these fields is marshalled by accessors - the transmitting thread uses the `SetData()` and `SetCode()` methods to seed the data, while the receiving thread uses the `GetData()` and `GetCode()` methods to retrieve it.

By providing the data as a void data pointer instead of a fixed-size message, we achieve an unprecedented measure of simplicity and flexibility. Data can be either statically or dynamically allocated, and sized appropriately for the event without having to format and reformat data by both sending and receiving threads. The choices here are left to the user - and the kernel doesn't get in the way of efficiency.

It is worth noting that you can send messages to message queues from within ISR context. This helps maintain consistency, since the same APIs can be used to provide event-driven programming facilities throughout the whole of the OS.

6.7.2 Global Message Pool

To maintain efficiency in the messaging system (and to prevent over-allocation of data), a global pool of message objects is provided. The size of this message pool is specified in the implementation, and can be adjusted depending on the requirements of the target application as a compile-time option.

Allocating a message from the message pool is as simple as calling the `GlobalMessagePool::Pop()` Method.

Messages are returned back to the `GlobalMessagePool::Push()` method once the message contents are no longer required.

One must be careful to ensure that discarded messages always are returned to the pool, otherwise a resource leak can occur, which may cripple the operating system's ability to pass data between threads.

6.7.3 Message Queues

`Message` objects specify data with context, but do not specify where the messages will be sent. For this purpose we have a `MessageQueue` object. Sending an object to a message queue involves calling the `MessageQueue::Send()` method, passing in a pointer to the `Message` object as an argument.

When a message is sent to the queue, the first thread blocked on the queue (as a result of calling the `MessageQueue::Receive()` method) will wake up, with a pointer to the `Message` object returned.

It's worth noting that multiple threads can block on the same message queue, providing a means for multiple threads to share work in parallel.

6.7.4 Messaging Example

```
// Message queue object shared between threads
MessageQueue cMsgQ;

// Function that initializes the shared message queue
void MsgQInit()
{
    cMsgQ.Init();
}

// Function called by one thread to send message data to
// another
void TxMessage()
{
    // Get a message, initialize its data
    Message *pclMsg = GlobalMessagePool::Pop();

    pclMsg->SetCode(0xAB);
    pclMsg->SetData((void*)some_data);

    // Send the data to the message queue
    cMsgQ.Send(pclMsg);
}

// Function called in the other thread to block until
// a message is received in the message queue.
void RxMessage()
{
    Message *pclMsg;
```

```

// Block until we have a message in the queue
pclMsg = clMsgQ.Receive();

// Do something with the data once the message is received
pclMsg->GetCode();

// Free the message once we're done with it.
GlobalMessagePool::Push(pclMsg);
}

```

6.8 Sleep

There are instances where it may be necessary for a thread to poll a resource, or wait a specific amount of time before proceeding to operate on a peripheral or volatile piece of data.

While the [Timer](#) object is generally a better choice for performing time-sensitive operations (and certainly a better choice for periodic operations), the [Thread::Sleep\(\)](#) method provides a convenient (and efficient) mechanism that allows for a thread to suspend its execution for a specified interval.

Note that when a thread is sleeping it is blocked, during which other threads can operate, or the system can enter its idle state.

```

int GetPeripheralData();
{
    int value;
    // The hardware manual for a peripheral specifies that
    // the "foo()" method will result in data being generated
    // that can be captured using the "bar()" method.
    // However, the value only becomes valid after 10ms

    peripheral.foo();
    Thread::Sleep(10);    // Wait 10ms for data to become valid
    value = peripheral.bar();
    return value;
}

```

6.9 Round-Robin Quantum

Threads at the same thread priority are scheduled using a round-robin scheme. Each thread is given a timeslice (which can be configured) of which it shares time amongst ready threads in the group. Once a thread's timeslice has expired, the next thread in the priority group is chosen to run until its quantum has expired - the cycle continues over and over so long as each thread has work to be done.

By default, the round-robin interval is set at 4ms.

This value can be overridden by calling the thread's [SetQuantum\(\)](#) with a new interval specified in milliseconds.

Chapter 7

Inside The Scheduler

This section details the inner-working of the Mark3 [Scheduler](#) in detail.

7.1 A Bit About Threads

Before we get started talking about the internals of the Mark3 scheduler, it's necessary to go over some background material - starting with: what is a thread, anyway?

Let's look at a very basic CPU without any sort of RTOS, and without interrupts. When the CPU is powered up, the program counter is loaded with some default location, at which point the processor core will start executing instructions sequentially - running forever and ever according to whatever has been loaded into program memory. This single instance of a simple program sequence is the only thing that runs on the processor, and the execution of the program can be predicted entirely by looking at the CPU's current register state, its program, and any affected system memory (the CPU's "context").

It's simple enough, and that's exactly the definition we have for a thread in an RTOS.

Each thread contains an instance of a CPU's register context, its own stack, and any other bookkeeping information necessary to define the minimum unique execution state of a system at runtime. It is the job of a RTOS to multiplex the execution of multiple threads on a single physical CPU, thereby creating the illusion that many programs are being executed simultaneously. In reality there can only ever be one thread truly executing at any given moment on a CPU core, so it's up to the scheduler to set and enforce rules about what thread gets to run when, for how long, and under what conditions. As mentioned earlier, any system without an RTOS executes as a single thread, so at least two threads are required for an RTOS to serve any useful purpose.

Note that all of this information is common to pretty well every RTOS in existence - the implementation details, including the scheduler rules, are all part of what differentiates one RTOS from another.

7.2 Thread States and Thread Lists

Since only one thread can run on a CPU at a time, the scheduler relies on thread information to make its decisions. Mark3's scheduler relies on a variety of such information, including: The thread's current priority Round-Robin execution quanta Whether or not the thread is blocked on a synchronization object, such as a mutex or semaphore Whether or not the thread is currently suspended The scheduler further uses this information to logically place each thread into 1 of 4 possible states: Ready - The thread is currently running Running - The thread is able to run Blocked - The thread cannot run until a system condition is met Stopped - The thread cannot run because its execution has been suspended In order to determine a thread's state, threads are placed in "buckets" corresponding to these states. Ready and running threads exist in the scheduler's buckets, blocked threads exist in a buckets belonging to the object they're blocked on, and stopped threads exist in a bucket of all stopped threads.

In reality, the various buckets are just doubly-linked lists of [Thread](#) objects

- implemented in something called the [ThreadList](#) class. To facilitate this, the [Thread](#) class directly inherits

from the [LinkListNode](#) class, which contains the node pointers required to implement a doubly-linked list. As a result, Threads may be effortlessly moved from one state to another using efficient linked-list operations built into the [ThreadList](#) class.

7.3 About Blocking and Unblocking

While many developers new to the concept of an RTOS assume that all threads in a system are entirely separate from each other, the reality is that practical systems typically involve multiple threads working together, or at the very least sharing resources. In order to synchronize the execution of threads for that purpose, a number of synchronization primitives (blocking objects) are implemented to create specific sets of conditions under which threads can continue execution. The concept of "blocking" a thread until a specific condition is met is fundamental to understanding RTOS applications design, as well as any highly-multithreaded applications.

Blocking objects and primitives provided by Mark3 include:

- Semaphores (binary and counting)
- Mutexes
- Event Flags
- [Thread](#) Sleep
- [Message](#) Queues

Each of these objects inherit from the [BlockingObject](#) class, which itself contains a [ThreadList](#) object. This class contains methods to `Block()` a thread (remove it from the [Scheduler](#)'s "Ready" or "Running" ThreadLists), as well as `UnBlock()` a thread (move a thread back to the "Ready" lists). This object handles transitioning threads from list-to-list (and state-to-state), as well as taking care of any other [Scheduler](#) bookkeeping required in the process. While each of the Blocking types implement a different condition, they are effectively variations on the same theme. Many simple Blocking objects are also used to build complex blocking objects - for instance, the [Thread](#) Sleep mechanism is essentially a binary semaphore and a timer object, while a message queue is a linked-list of message objects combined with a semaphore.

7.4 The Scheduling Algorithm

At this point we've covered the following concepts:

- Threads
- [Thread](#) States and [Thread](#) Lists
- Blocking and Un-Blocking Threads

Thankfully, this is all the background required to understand how the Mark3 [Scheduler](#) works. In technical terms, Mark3 implements "strict priority scheduling, with round-robin scheduling among threads in each priority group". In plain English, this boils down to a scheduler which follows a few simple rules:

- Find the highest-priority "Ready" list that has at least one Threads.
- If the first thread in that bucket is not the current thread, select it to run next
- Otherwise, rotate the linked list, and choose the next thread in the list to run

Since context switching is one of the most common and frequent operation performed by an RTOS, this needs to be as fast and deterministic as possible. While the logic is simple, a lot of care must be put into optimizing the scheduler to achieve those goals. In the section below we discuss the optimization approaches taken in Mark3.

There are a number of ways to find the highest-priority thread. The naive approach would be to simply iterate through the scheduler's array of ThreadLists from highest to lowest, stopping when the first non-empty list is found, such as in the following block of code:

```
for (prio = num_prio - 1; prio >= 0; prio--)
{
    if (thread_list[prio].get_head() != NULL)
    {
        break;
    }
}
```

While that would certainly work and be sufficient for a variety of systems, it's a non-deterministic approach (complexity $O(n)$) whose cost varies substantially based on how many priorities have to be evaluated. It's simple to read and understand, but it's non-optimal.

Fortunately, a functionally-equivalent and more deterministic approach can be implemented with a few tricks.

In addition to maintaining an array of ThreadLists, Mark3 also maintains a bitmap (one bit per priority level) that indicates which thread lists have ready threads. This bitmap is maintained automatically by the [ThreadList](#) class, and is updated every time a thread is moved to/from the [Scheduler's](#) ready lists.

By inspecting this bitmap using a technique to count the leading zero bits in the bitmap, we determine which threadlist to choose in fixed time.

Now, to implement the leading-zeros check, this can once again be performed iteratively using bitshifts and compares (which isn't any more efficient than the raw list traversal), but it can also be evaluated using either a lookup table, or via a special CPU instruction to count the leading zeros in a value. In Mark3, we opt for the lookup-table approach since we have a limited number of priorities and not all supported CPU architectures support a count leading zero instruction. To achieve a balance between performance and memory use, we use a 4-bit lookup table (costing 16 bytes) to perform the lookup.

(As a sidenote - this is actually a very common approach in OS schedulers. It's actually part of the reason why modern ARM cores implement a dedicated count-leading-zeros [CLZ] instruction!)

With a 4-bit lookup table and an 8-bit priority-level bitmap, the priority check algorithm looks something like this:

```
// Check the highest 4 priority levels, represented in the
// upper 4 bits in the bitmap
priority = priority_lookup_table[(priority_bitmap >> 4)];

// priority is non-zero if we found something there
if( priority )
{
    // Add 4 because we were looking at the higher levels
    priority += 4;
}
else
{
    // Nothing in the upper 4, look at the lowest 4 priority levels
    // represented by the lowest 4 bits in the bitmap
    priority = priority_lookup_table[priority_bitmap & 0x0F];
}
```

Deconstructing this algorithm, you can see that the priority lookup will have an $O(1)$ complexity - and is extremely low-cost. This operation is thus fully deterministic and time bound - no matter how many threads are scheduled, the operation will always be time-bound to the most expensive of these two code paths. Even with only 8 priority levels, this is still much faster than iteratively checking the thread lists manually, compared with the previous example implementation.

Once the priority level has been found, selecting the next thread to run is trivial, consisting of something like this:

```
next_thread = thread_list[prio].get_head();
```

In the case of the `get_head()` calls, this evaluates to an inline-load of the "head" pointer in the particular thread list. One important thing to take away from this analysis is that the scheduler is only responsible for selecting the next-to-run thread. In fact, these two operations are totally decoupled - no context switching is performed by the scheduler, and the scheduler isn't called from the context switch. The scheduler simply produces new "next thread" values that are consumed from within the context switch code.

7.5 Considerations For Round-Robin Scheduling

One thing that isn't considered directly from the scheduler algorithm is the problem of dealing with multiple threads within a single priority group; all of the algorithms that have been explored above simply look at the firstThread in each group.

Mark3 addresses this issue indirectly, using a software timer to manage round-robin scheduling, as follows. In some instances where the scheduler is run by the kernel directly (typically as a result of calling `Thread::Yield()`), the kernel will perform an additional check after running the `Scheduler` to determine whether or there are multiple ready Threads in the priority of the next ready thread. If there are multiple threads within that priority, the kernel adds a one-shot software timer which is programmed to expire at the next `Thread`'s configured quantum. When this timer expires, the timer's callback function executes to perform two simple operations: "Pivot" the current `Thread`'s priority list.

Set a flag telling the kernel to trigger a Yield after exiting the main `TimerScheduler` processing loop. Pivoting the thread list basically moves the head of a circular-linked-list to its next value, which in our case ensures that a new thread will be chosen the next time the scheduler is run (the scheduler only looks at the head node of the priority lists). And by calling Yield, the system forces the scheduler to run, a new round-robin software timer to be installed (if necessary), and triggers a context switch SWI to load the newly-chosen thread. Note that if the thread attached to the round-robin timer is pre-empted, the kernel will take steps to abort and invalidate that round-robin software timer, installing a new one tied to the next thread to run if necessary. Because the round-robin software timer is dynamically installed when there are multiple ready threads at the highest ready priority level, there is no CPU overhead with this feature unless that condition is met. The cost of round-robin scheduling is also fixed - no matter how many threads there are, and the cost is identical to any other one-shot software timer in the system.

7.6 Context Switching

There's really not much to say about the actual context switch operation at a high level. Context switches are triggered whenever it has been determined that a new thread needs to be swapped into the CPU core when the scheduler is run. Mark3 implements also context switches as a call to a software interrupt - on AVR platforms, we typically use INT0 or INT2 for this (although any pin-change GPIO interrupt can be used), and on ARM we achieve this by triggering a PendSV exception.

However, regardless of the architecture, the context-switch ISR will perform the following three operations:

- Save the current `Thread`'s context to the current `Thread` stack
- Make the "next to run" thread the "currently running" thread
- Restore the context of the next `Thread` from the `Thread` stack

The code to implement the context switch is entirely architecture-specific, so it won't be discussed in detail here. It's almost always gory inline-assembly which is used to load and store various CPU registers, and is highly-optimized for speed. I will dive into how this important bit of code works (on ARM Cortex-M0+) in a separate whitepaper.

7.7 Putting It All Together

In short, we can say that the Mark3 scheduler works as follows:

- The scheduler is run whenever a `Thread::Yield()` is called by a user, as part of blocking calls, or whenever a new thread is started
- The Mark3 scheduler is deterministic, selecting the next thread to run in fixed-time
- The scheduler only chooses the next thread to run, the context switch SWI consumes that information to get that thread running

- Where there are multiple ready threads in the highest populated priority level, a software timer is used to manage round-robin scheduling

While we've covered a lot of ground in this chapter, there's not a whole lot of code involved. However, the code that performs these operations is quite nuanced and subtle. If you're interested in seeing how this all works in practice, I suggest reading through the Mark3 source code (which is heavily annotated), and stepping through the code with a simulator/emulator.

Chapter 8

Porting Mark3 - An Example Using ARM Cortex-M0

This document serves as both a real-world example of how Mark3 can be ported to new architectures, and as a practical reference for using the RTOS support functionality baked in modern ARM Cortex-M series microcontrollers.

Most of this documentation here is taken directly from the source code found in the `/kernel/cpu/cm0/` ports directory, with additional annotations to explain the port in more detail. Note that a familiarity with Cortex-M series parts will go a long way to understanding the subject matter presented, especially a basic understanding of the ARM CPU registers, exception models, and OS support features (PendSV, SysTick and SVC).

Porting Mark3 to a new architecture consists of a few basic pieces; for developers familiar with the target architecture and the porting process, it's not a tremendously onerous endeavour to get Mark3 up-and-running somewhere new. For starters, all non-portable components are completely isolated in the source-tree under `/embedded/kernel/<CPU>/<VARIANT>/<TOOLCHAIN>/`, where `<CPU>` is the architecture, `<VARIANT>` is the vendor/part, and `<TOOLCHAIN>` is the compiler tool suite used to build the code.

From within the specific port folder, a developer needs only implement a few classes and headers that define the port-specific behavior of Mark3:

- [KernelSWI](#) ([kernelswi.cpp](#)/kernelswi.h) - Provides a maskable software-triggered interrupt used to perform context switching.
- [KernelTimer](#) ([kerneltimer.cpp](#)/kerneltimer.h) - Provides either a fixed-frequency or programmable-interval timer, which triggers an interrupt on expiry. This is used for implementing round-robin scheduling, thread-sleeps, and generic software timers.
- [Profiler](#) ([kprofile.cpp](#)/kprofile.h) - Contains code for runtime code-profiling. This is optional and may be stubbed out if left unimplemented (we won't cover profiling timers here).
- [ThreadPort](#) ([threadport.cpp](#)/threadport.h) - The meat-and-potatoes of the port code lives here. This class contains architecture/part-specific code used to initialize threads, implement critical-sections, perform context-switching, and start the kernel. Most of the time spent in this article focuses on the code found here.

Summarizing the above, these modules provide the following list of functionality:

- [Thread](#) stack initialization
- [Kernel](#) startup and first thread entry
- Context switch and SWI
- [Kernel](#) timers
- Critical Sections

The implementation of each of these pieces will be analyzed in detail in the sections that follow.

8.1 Thread Stack Initialization

Before a thread can be used, its stack must first be initialized to its default state. This default state ensures that when the thread is scheduled for the first time and its context restored, that it will cause the CPU to jump to the user's specified entry-point function.

All of the platform independent thread setup is handled by the generic kernel code. However, since every CPU architecture has its own register set, and stacks different information as part of an interrupt/exception, we have to implement this thread setup code for each platform we want the kernel to support (Combination of Architecture + Variant + Toolchain).

In the ARM Cortex-M0 architecture, the stack frame consists of the following information:

a) Exception Stack Frame

Contains the 8 registers which the ARM Cortex-M0 CPU automatically pushes to the stack when entering an exception. The following registers are included (in stack'd order):

```
[ XPSR ] <-- Highest address in context
[ PC   ]
[ LR   ]
[ R12  ]
[ R3   ]
[ R2   ]
[ R1   ]
[ R0   ]
```

XPSR – This is the CPU's status register. We need to set this to 0x01000000 (the "T" bit), which indicates that the CPU is executing in "thumb" mode. Note that ARMv6m and ARMv7m processors only run thumb2 instructions, so an exception is liable to occur if this bit ever gets cleared.

PC – Program Counter. This should be set with the initial entry point (function pointer) for that the user wishes to start executing with this thread.

LR - The base link register. Normally, this register contains the return address of the calling function, which is where the CPU jumps when a function returns. However, our threads generally don't return (and if they do, they're placed into the stop state). As a result we can leave this as 0.

The other registers in the stack frame are generic working registers, and have no special meaning, with the exception that R0 will hold the user's argument value passed into the entrypoint.

b) Complimentary CPU Register Context

```
[ R11   ]
...
[ R4    ] <-- Lowest address in context
```

These are the other general-purpose CPU registers that need to be backed up/restored on a context switch, but aren't stacked by default on a Cortex-M0 exception. If there were any additional hardware registers to back up, then we'd also have to include them in this part of the context as well.

As a result, these registers all need to be manually pushed to the stack on stack creation, and will need to be explicitly pushed and pop as part of a normal context switch.

With this default exception state in mind, the following code is used to initialize a thread's stack for a Cortex-M0.

```
void ThreadPort::InitStack(Thread *pclThread_)
{
    K_ULONG *pulStack;
    K_ULONG *pulTemp;
    K_ULONG ulAddr;
    K_USHORT i;

    // Get the entrypoint for the thread
    ulAddr = (K_ULONG)(pclThread_>m_pfEntryPoint);

    // Get the top-of-stack pointer for the thread
    pulStack = (K_ULONG*)pclThread_>m_pwStackTop;

    // Initialize the stack to all FF's to aid in stack depth checking
    pulTemp = (K_ULONG*)pclThread_>m_pwStack;
```



```

for (i = 0; i < pclThread->m_usStackSize / sizeof(K_ULONG); i++)
{
    pulTemp[i] = 0xFFFFFFFF;
}

PUSH_TO_STACK(pulStack, 0);          // Apply one word of padding

/-- Simulated Exception Stack Frame --
PUSH_TO_STACK(pulStack, 0x01000000); // XSPR - set "T" bit for thumb-mode
PUSH_TO_STACK(pulStack, ulAddr);     // PC
PUSH_TO_STACK(pulStack, 0);          // LR
PUSH_TO_STACK(pulStack, 0x12);
PUSH_TO_STACK(pulStack, 0x3);
PUSH_TO_STACK(pulStack, 0x2);
PUSH_TO_STACK(pulStack, 0x1);
PUSH_TO_STACK(pulStack, (K_ULONG)pclThread->m_pvArg); // R0 = argument

/-- Simulated Manually-Stacked Registers --
PUSH_TO_STACK(pulStack, 0x11);
PUSH_TO_STACK(pulStack, 0x10);
PUSH_TO_STACK(pulStack, 0x09);
PUSH_TO_STACK(pulStack, 0x08);
PUSH_TO_STACK(pulStack, 0x07);
PUSH_TO_STACK(pulStack, 0x06);
PUSH_TO_STACK(pulStack, 0x05);
PUSH_TO_STACK(pulStack, 0x04);
pulStack++;

pclThread->m_pwStackTop = pulStack;
}

```

8.2 Kernel Startup

The same general process applies to starting the kernel on an ARM Cortex-M0 as on other platforms. Here, we initialize and start the platform specific timer and software-interrupt modules, find the first thread to run, and then jump to that first thread.

Now, to perform that last step, we have two options:

1) Simulate a return from an exception manually to start the first thread, or.. 2) Use a software interrupt to trigger the first "Context Restore/Return from Interrupt"

For 1), we basically have to restore the whole stack manually, not relying on the CPU to do any of this for us. That's certainly doable, but not all Cortex parts support this (other members of the family support privileged modes, etc.). That, and the code required to do this is generally more complex due to all of the exception-state simulation. So, we will opt for the second option instead.

To implement a software to start our first thread, we will use the SVC instruction to generate an exception. From that exception, we can then restore the context from our first thread, set the CPU up to use the right "process" stack, and return-from-exception back to our first thread. We'll explore the code for that later.

But, before we can call the SVC exception, we're going to do a couple of things.

First, we're going to reset the default MSP stack pointer to its original top-of-stack value. The rationale here is that we no longer care about the data on the MSP stack, since calling the SVC instruction triggers a chain of events from which we never return. The MSP is also used by all exception-handling, so regaining a few words of stack here can be useful. We'll also enable all maskable exceptions at this point, since this code results in the kernel being started with the CPU executing the RTOS threads, at which point a user would expect interrupts to be enabled.

Note, the default stack pointer location is stored at address 0x00000000 on all ARM Cortex M0 parts. That explains the code below.

```

void ThreadPort_StartFirstThread( void )
{
    asm(
        " ldr r1, [r0] \n" // Reset the MSP to the default base address
        " msr msp, r1 \n"
        " cpsie i \n"      // Enable interrupts
        " svc 0 \n"        // Jump to SVC Call
    );
}

```

8.3 First Thread Entry

This handler has the job of taking the first thread object's stack, and restoring the default state data in a way that ensures that the thread starts executing when returning from the call.

We also keep in mind that there's an 8-byte offset from the beginning of the thread object to the location of the thread stack pointer. This offset is a result of the thread object inheriting from the linked-list node class, which has 8-bytes of data. This is stored first in the object, before the first element of the class, which is the "stack top" pointer.

The following assembly code shows how the SVC call is implemented in Mark3 for the purpose of starting the first thread.

```
get_thread_stack:
    ; Get the stack pointer for the current thread
    ldr r0, g_pstCurrent
    ldr r1, [r0]
    add r1, #8
    ldr r2, [r1]          ; r2 contains the current stack-top

load_manually_placed_context_r11_r8:
    ; Handle the bottom 32-bytes of the stack frame
    ; Start with r11-r8, because only r0-r7 can be used
    ; with ldmia on CM0.
    add r2, #16
    ldmia r2!, {r4-r7}
    mov r11, r7
    mov r10, r6
    mov r9, r5
    mov r8, r4

set_psp:
    ; Since r2 is coincidentally back to where the stack pointer should be,
    ; Set the program stack pointer such that returning from the exception handler
    msp psp, r2

load_manually_placed_context_r7_r4:
    ; Get back to the bottom of the manually stacked registers and pop.
    sub r2, #32
    ldmia r2!, {r4-r7} ; Register r4-r11 are restored.

set_thread_and_privilege_modes:
    ; Also modify the control register to force use of thread mode as well
    ; For CM3 forward-compatibility, also set user mode.
    mrs r0, control
    mov r1, #0x03
    orr r0, r1
    control, r0

set_lr:
    ; Set up the link register such that on return, the code operates in thread mode using the PSP
    ; To do this, we or 0x0D to the value stored in the lr by the exception hardware EXC_RETURN.
    ; Alternately, we could just force lr to be 0xFFFFFFF0 (we know that's what we want from the hardware,
    ; anyway)
    mov r0, #0x0D
    mov r1, lr
    orr r0, r1

exit_exception:
    ; Return from the exception handler. The CPU will automatically unstack R0-R3, R12, PC, LR, and xPSR
    ; for us. If all goes well, our thread will start execution at the entrypoint, with the us-specified
    ; argument.
    bx r0
```

8.4 Context Switching

On ARM Cortex parts, there's dedicated hardware that's used primarily to support RTOS (or RTOS-like) functionality. This functionality includes the SysTick timer, and the PendSV Exception. SysTick is used for a tick-based kernel timer, while the PendSV exception is used for performing context switches. In reality, it's a "special SVC" call that's designed to be lower-overhead, in that it isn't mux'd with a bunch of other system or application functionality.

So how do we go about actually implementing a context switch here? There are a lot of different parts involved, but it essentially comes down to 3 steps:

- 1) Saving the context. [Thread](#)'s top-of-stack value is stored, all registers are stacked. We're good to go!
- 2) Swap threads. We swap the [Scheduler](#)'s "next" thread with the "current" thread.

3) Restore Context. This is more or less identical to what we did when restoring the first context. Some operations may be optimized for data already stored in registers.

The code used to implement these steps on Cortex-M0 is presented below:

```
void PendSV_Handler(void)
{
    ASM(
        // Thread_SaveContext()
        " ldr r1, CURR_ \n"
        " ldr r1, [r1] \n "
        " mov r3, r1 \n "
        " add r3, #8 \n "

        // Grab the psp and adjust it by 32 based on the extra registers we're going
        // to be manually stacking.
        " mrs r2, psp \n "
        " sub r2, #32 \n "

        // While we're here, store the new top-of-stack value
        " str r2, [r3] \n "

        // And, while r2 is at the bottom of the stack frame, stack r7-r4
        " stmia r2!, {r4-r7} \n "

        // Stack r11-r8
        " mov r7, r11 \n "
        " mov r6, r10 \n "
        " mov r5, r9 \n "
        " mov r4, r8 \n "
        " stmia r2!, {r4-r7} \n "

        // Equivalent of Thread_Swap() - performs g_pstCurrent = g_pstNext
        " ldr r1, CURR_ \n"
        " ldr r0, NEXT_ \n"
        " ldr r0, [r0] \n"
        " str r0, [r1] \n"

        // Thread_RestoreContext()
        // Get the pointer to the next thread's stack
        " add r0, #8 \n "
        " ldr r2, [r0] \n "

        // Stack pointer is in r2, start loading registers from the "manually-stacked" set
        // Start with r11-r8, since these can't be accessed directly.
        " add r2, #16 \n "
        " ldmia r2!, {r4-r7} \n "
        " mov r11, r7 \n "
        " mov r10, r6 \n "
        " mov r9, r5 \n "
        " mov r8, r4 \n "

        // After subbing R2 #16 manually, and #16 through ldmia, our PSP is where it
        // needs to be when we return from the exception handler
        " msr psp, r2 \n "

        // Pop manually-stacked R4-R7
        " sub r2, #32 \n "
        " ldmia r2!, {r4-r7} \n "

        // lr contains the proper EXC_RETURN value
        // we're done with the exception, so return back to the newly-chosen thread
        " bx lr \n "
        " nop \n "

        // Must be 4-byte aligned. Also - GNU assembler, I hate you for making me resort to this.
        " NEXT_: .word g_pstNext \n"
        " CURR_: .word g_pstCurrent \n"
    );
}
```

8.5 Kernel Timers

ARM Cortex-M series microcontrollers each contain a SysTick timer, which was designed to facilitate a fixed-interval RTOS timer-tick. This timer is a precise 24-bit down-count timer, run at the main CPU clock frequency, that can be programmed to trigger an exception when the timer expires. The handler for this exception can thus be used to drive software timers throughout the system on a fixed interval.

Unfortunately, this hardware is extremely simple, and does not offer the flexibility of other timer hardware commonly

implemented by MCU vendors - specifically a suitable timer prescaler that can be used to generate efficient, long-counting intervals. As a result, while the "generic" port of Mark3 for Cortex-M0 leverages the common SysTick timer interface, it only supports the tick-based version of the kernel's timer (note that specific Cortex-M0 ports such as the Atmel SAMD20 do have tickless timers).

Setting up a tick-based [KernelTimer](#) class to use the SysTick timer is, however, extremely easy, as is illustrated below:

```
void KernelTimer::Start(void)
{
    SysTick_Config(SYSTEM_FREQ / 1000); // 1KHz fixed clock...
    NVIC_EnableIRQ(SysTick_IRQn);
}
```

In this instance, the call to `SysTick_Config()` generates a 1kHz system-tick signal, and the `NVIC_EnableIRQ()` call ensures that a SysTick exception is generated for each tick. All other functions in the Cortex version of the [KernelTimer](#) class are essentially stubbed out (see the source for more details).

Note that the functions used in this call are part of the ARM Cortex Microcontroller Software Interface Standard (CMSIS), and are supplied by all parts vendors selling Cortex hardware. This greatly simplifies the design of our port-code, since we can be reasonably assured that these APIs will work the same on all devices.

The handler code called when a SysTick exception occurs is basically the same as on other platforms (such as AVR), except that we explicitly clear the "exception pending" bit before returning. This is implemented in the following code:

```
void SysTick_Handler(void)
{
    #if KERNEL_USE_TIMERS
        TimerScheduler::Process();
    #endif
    #if KERNEL_USE_QUANTUM
        Quantum::UpdateTimer();
    #endif

    // Clear the systick interrupt pending bit.
    SCB->ICSR |= SCB_ICSR_PENDSTCLR_Msk;
}
```

8.6 Critical Sections

A "critical section" is a block of code whose execution cannot be interrupted by means of context switches or an interrupt. In a traditional single-core operating system, it is typically implemented as a block of code where the interrupts are disabled - this is also the approach taken by Mark3. Given that every CPU has its own means of disabling/enabling interrupts, the implementation of the critical section APIs is also non-portable.

In the Cortex-M0 port, we implement the two critical section APIs ([CS_ENTER\(\)](#) and [CS_EXIT\(\)](#)) as function-like macros containing inline assembly. All uses of these calls are called in pairs within a function and must take place at the same level-of-scope. Also, as nesting may occur (critical section within a critical section), this must be taken into account in the code.

In general, [CS_ENTER\(\)](#) performs the following tasks:

- Cache the current interrupt-enabled state within a local variable in the thread's state
- Disable interrupts

Conversely, [CS_EXIT\(\)](#) performs the following tasks:

- Read the original interrupt-enabled state from the cached value
- Restore interrupts to the original value

On Cortex-M series microcontrollers, the PRIMASK special register contains a single status bit which can be used to enable/disable all maskable interrupts at once. This register can be read directly to examine or modify its state. For

convenience, ARMv6m provides two instructions to enable/disable interrupts - cpsid (disable interrupts) and cpsie (enable interrupts). Mark3 implements these steps according to the following code:

```
//-----
#define CS_ENTER() \
{ \
    K_ULONG __ulRegState; \
    asm ( \
        " mrs r0, PRIMASK \n" \
        " mov %[STATUS], r0 \n" \
        " cpsid i \n " \
        : [STATUS] "=r" (__ulRegState) \
        ); \
}

//-----
#define CS_EXIT() \
    asm ( \
        " mov r0, %[STATUS] \n" \
        " msr primask, r0 \n" \
        : \
        : [STATUS] "r" (__ulRegState) \
        ); \
}
```

8.7 Conclusion

In this chapter we have investigated how the main non-portable areas of the Mark3 RTOS are implemented on a Cortex-M0 microcontroller. Mark3 leverages all of the hardware blocks designed to enable RTOS functionality on ARM Cortex-M series microcontrollers: the SVC call provides the mechanism by which we start the kernel, the PendSV exception provides the necessary software interrupt, and the SysTick timer provides an RTOS tick. As a result, Mark3 is a perfect fit for these devices - and as a result of this approach, the same RTOS port code should work with little to no modification on all ARM Cortex-M parts.

We have discussed what functionality in the RTOS is not portable, and what interfaces must be implemented in order to complete a fully-functional port. The five specific areas which are non-portable (stack initialization, kernel startup/entry, kernel timers, context switching, and critical sections) have been discussed in detail, with the platform-specific source provided as a practical reference to ARM-specific OS features, as well as Mark3's porting infrastructure. From this example (and the accompanying source), it should be possible for an experienced developers to create a port Mark3 to other microcontroller targets.

Chapter 9

Build System

Mark3 is distributed with a recursive makefile build system, allowing the entire source tree to be built into a series of libraries with simple make commands.

The way the scripts work, every directory with a valid makefile is scanned, as well as all of its subdirectories. The build then generates binary components for all of the components it finds -libraries and executables. All libraries that are generated can then be imported into an application using the linker without having to copy-and-paste files on a module-by-module basis. Applications built during this process can then be loaded onto a device directly, without requiring a GUI-based IDE. As a result, Mark3 integrates well with 3rd party tools for continuous-integration and automated testing.

This modular framework allows for large volumes of libraries and binaries to be built at once - the default build script leverages this to build all of the examples and unit tests at once, linking against the pre-built kernel, services, and drivers. Whatever can be built as a library is built as a library, promoting reuse throughout the platform, and enabling Mark3 to be used as a platform, with an ecosystem of libraries, services, drivers and applications.

9.1 Source Layout

One key aspect of Mark3 is that system features are organized into their own separate modules. These modules are further grouped together into folders based on the type of features represented:

Root	Base folder, contains recursive makefiles for build system
bootloader	Mark3 Bootloader code for AVR
build	Makefile support for various platforms
doc	Documentation (including this)
drivers	Device driver code
example	Example applications
kernel	Basic Mark3 Components (the focus of this manual)
cpu	CPU-specific porting code
services	Utility code and services, extended system features
stage	Staging directory, where the build system places artifacts
tests	Unit tests, written as C/C++ applications

9.2 Building the kernel

The base.mak file determines how the kernel, drivers, and libraries are built, for what targets, and with what options. Most of these options can be copied directly from the options found in your IDE managed projects. Below is an overview of the main variables used to configure the build.

STAGE	- Location in the filesystem where the build output is stored
ROOT_DIR	- The location of the root source tree
ARCH	- The CPU architecture to build against
VARIANT	- The variant of the above CPU to target
TOOLCHAIN	- Which toolchain to build with (dependent on ARCH and VARIANT)

Build.mak contains the logic which is used to perform the recursive make in all directories. Unless you really know what you're doing, it's best to leave this as-is.

You must make sure that all required paths are set in your system environment variables so that they are accessible through from the command-line.

Once configured, you can build the source tree using the various make targets:

- make headers
 - copy all headers in each module's /public subdirectory to the location specified by STAGE environment variable's ./inc subdirectory.
- make library
 - regenerate all objects copy marked as libraries (i.e. the kernel + drivers). Resulting binaries are copied into STAGE's ./lib subdirectory.
- make binary
 - build all executable projects in the root directory structure. In the default distribution, this includes the basic set of demos.

To add new components to the recursive build system, simply add your code into a new folder beneath the root install location.

Source files, the module makefile and private header files go directly in the new folder, while public headers are placed in a ./public subdirectory. Create a ./obj directory to hold the output from the builds.

The contents of the module makefile looks something like this:

```
# Include common prelude make file
include $(ROOT_DIR)base.mak

# If we're building a library, set IS_LIB and LIBNAME
# If we're building an app, set IS_APP and APPNAME
IS_LIB=1
LIBNAME=mylib

#this is the list of the source modules required to build the kernel
CPP_SOURCE = mylib.cpp \
             someotherfile.cpp

# Similarly, C-language source would be under the C_SOURCE variable.

# Include the rest of the script that is actually used for building the
# outputs
include $(ROOT_DIR)build.mak
```

Once you've placed your code files in the right place, and configured the makefile appropriately, a fresh call to make headers, make library, then make binary will guarantee that your code is built.

Now, you can still copy-and-paste the required kernel, port, and drivers, directly into your application avoiding the whole process of using make from the command line. To do this, run "make source" from the root directory in svn, and copy the contents of /stage/src into your project. This should contain the source to the kernel, all drivers, and all services that are in the tree - along with the necessary header files.

9.3 Building on Windows

Building Mark3 on Windows is the same as on Linux, but there are a few prerequisites that need to be taken into consideration before the build scripts and makefiles will work as expected.

Step 1 - Install Latest Atmel Studio IDE

Atmel Studio contains the AVR8 GCC toolchain, which contains the necessary compilers, assemblers, and platform support required to turn the source modules into libraries and executables.

To get Atmel Studio, go to the Atmel website (<http://www.atmel.com>) and register to download the latest version. This is a free download (and rather large). The included IDE (if you choose to use it) is very slick, as it's based on Visual Studio, and contains a wonderful cycle-accurate simulator for AVR devices. In fact, the simulator is so good that most of the kernel and its drivers were developed using this tool.

Once you have downloaded and installed Atmel Studio, you will need to add the location of the AVR toolchain to the PATH environment variable.

To do this, go to Control Panel -> System and Security -> System -> Advanced System Settings, and edit the PATH variable. Append the location of the toolchain bin folder to the end of the variable.

On Windows 7 x64, it should look something like this:

C: Files (x86) Toolchain GCC\Native\3.4.2.1002-gnu-toolchain

Step 2 - Install MinGW and MinSys

MinGW (and MinSys in particular) provide a unix-like environment that runs under windows. Some of the utilities provided include a version of the bash shell, and GNU standard make - both which are required by the Mark3 recursive build system.

The MinGW installer can be downloaded from its project page on SourceForge. When installing, be sure to select the "MinSys" component.

Once installed, add the MinSys binary path to the PATH environment variable, in a similar fashion as with Atmel Studio in Step 1.

Step 3 - Setup Include Paths in Platform Makefile

The AVR header file path must be added to the "platform.mak" makefile for each AVR Target you are attempting to build for. These files can be located under /embedded/build/avr/atmegaXXX/. The path to the includes directory should be added to the end of the CFLAGS and CPPFLAGS variables, as shown in the following:

```
TEST_INC="/c/Program Files (x86)/Atmel/Atmel Toolchain/AVR8
GCC/Native/3.4.2.1002/avr8-gnu-toolchain/include"
CFLAGS += -I$(TEST_INC)
CPPFLAGS += -I$(TEST_INC)
```

Step 4 - Build Mark3 using Bash

Launch a terminal to your Mark3 base directory, and cd into the "embedded" folder. You should now be able to build Mark3 by running "bash ./build.sh" from the command-line.

Alternately, you can run bash itself, building Mark3 by running ./build.sh or the various make targets using the same syntax as documented previously.

Note - building on Windows is *slow*. This has a lot to do with how "make" performs under windows. There are faster substitutes for make (such as cs-make) that are exponentially quicker, and approach the performance of make on Linux. Other mechanisms, such as running make with multiple concurrent jobs (i.e. "make -j4") also helps significantly, especially on systems with multicore CPUs.

9.4 Exporting the Source

In addition to providing a full recursive-make based build system, the kernel source for a given target can be exported directly to a .zip file for convenience. Run export.sh for any supported target (the full list of targets is listed below) to create a .zip archive of the kernel source and port code. If doxygen and pdflatex are available from your OS, documentation will also be generated (HTML and PDF) on the fly and included in the archive.

9.4.1 Supported targets

Currently, Mark3 supports the following AVR parts:

- atmega328p

- arduino
- atmega644
- atmega1284p
- atxmega256a3 (*experimental)

The following Cortex M0 parts are supported as well:

- Atmel samd20
- ST Micro stm32f0

Chapter 10

License

10.1 License

Copyright (c) 2013, Funkenstein Software Consulting All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Funkenstein Software Consulting, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL FUNKENSTEIN SOFTWARE (MARK SLEVINSKY) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 11

Profiling Results

The following profiling results were obtained using an ATmega328p @ 16MHz.

The test cases are designed to make use of the kernel profiler, which accurately measures the performance of the fundamental system APIs, in order to provide information for user comparison, as well as to ensure that regressions are not being introduced into the system.

11.1 Date Performed

Sat Jun 1 10:43:06 EDT 2013

11.2 Compiler Information

The kernel and test code used in these results were built using the following compiler: `./profile.sh: 55: ./profile.sh: /home/moslevin/atmel/bin/avr-gcc: not found`

11.3 Profiling Results

- Semaphore Initialization: 7 cycles (averaged over 83 iterations)
- Semaphore Post (uncontested): 180 cycles (averaged over 83 iterations)
- Semaphore Pend (uncontested): 67 cycles (averaged over 83 iterations)
- Semaphore Flyback Time (Contested Pend): 1553 cycles (averaged over 83 iterations)
- Mutex Init: 0 cycles (averaged over 83 iterations)
- Mutex Claim: 143 cycles (averaged over 83 iterations)
- Mutex Release: 49 cycles (averaged over 83 iterations)
- Thread Initialize: 7800 cycles (averaged over 83 iterations)
- Thread Start: 803 cycles (averaged over 83 iterations)
- Context Switch: 198 cycles (averaged over 83 iterations)
- Thread Schedule: 47 cycles (averaged over 83 iterations)

Chapter 12

Code Size Profiling

The following report details the size of each module compiled into the kernel.

The size of each component is dependent on the flags specified in [mark3cfg.h](#) at compile time. Note that these sizes represent the maximum size of each module before dead code elimination and any additional link-time optimization, and represent the maximum possible size that any module can take.

The results below are for profiling on Atmel AVR atmega328p-based targets using gcc. Results are not necessarily indicative of relative or absolute performance on other platforms or toolchains.

12.1 Information

Subversion Repository Information:

- Repository Root: <https://svn.code.sf.net/p/mark3/source>
- Revision: 154
- URL: <https://svn.code.sf.net/p/mark3/source/trunk/embedded>

Date Profiled: Sat Jan 25 15:29:37 EST 2014

12.2 Compiler Version

avr-gcc (GCC) 4.7.2 Copyright (C) 2012 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

12.3 Profiling Results

Mark3 Module Size Report:

- [Atomic](#) Operations..... : 478 Bytes
- Synchronization Objects - Base Class..... : 270 Bytes
- Device [Driver](#) Framework (including /dev/null)... : 236 Bytes
- Synchronization Object - Event Flag..... : 1160 Bytes
- Fundamental [Kernel](#) Linked-List Classes..... : 610 Bytes

- Message-based IPC..... : 500 Bytes
- [Mutex](#) (Synchronization Object)..... : 1004 Bytes
- Performance-profiling timers..... : 556 Bytes
- Round-Robin Scheduling Support..... : 299 Bytes
- [Thread](#) Scheduling..... : 499 Bytes
- [Semaphore](#) (Synchronization Object)..... : 868 Bytes
- [Thread](#) Implementation..... : 1441 Bytes
- Fundamental [Kernel](#) Thread-list Data Structures.. : 212 Bytes
- Mark3 [Kernel](#) Base Class..... : 80 Bytes
- Software [Timer](#) Implementation..... : 1035 Bytes
- [Kernel Transaction](#) Queues..... : 308 Bytes
- Runtime [Kernel](#) Trace Implementation..... : 0 Bytes
- Circular Logging Buffer Base Class..... : 524 Bytes
- Atmel AVR - Basic Threading Support..... : 528 Bytes
- Atmel AVR - [Kernel](#) Interrupt Implemenation..... : 56 Bytes
- Atmel AVR - [Kernel Timer](#) Implementation..... : 338 Bytes
- Atmel AVR - Profiling [Timer](#) Implementation..... : 256 Bytes

Mark3 [Kernel](#) Size Summary:

- [Kernel](#) : 3420 Bytes
- Synchronization Objects : 3532 Bytes
- Port : 1178 Bytes
- Features : 2650 Bytes
- Total Size : 10780 Bytes

Chapter 13

Hierarchical Index

13.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Atomic	63
BitStreamer	65
BlockHeap	66
BlockingObject	68
EventFlag	97
Mutex	149
Semaphore	188
CommandLine_t	75
DCPU	76
DCPU_Registers	80
DrawBitmap_t	86
DrawCircle_t	87
DrawEllipse_t	88
DrawLine_t	88
DrawMove_t	89
DrawPoint_t	89
DrawPoly_t	90
DrawRectangle_t	90
DrawStamp_t	91
DrawText_t	92
DrawVector_t	92
DrawWindow_t	93
DriverList	96
FixedHeap	101
Font_t	102
GlobalMessagePool	104
Glyph_t	105
GuiEvent_t	119
GuiEventSurface	119
HeapConfig	129
JoystickEvent_t	129
Kernel	130
KernelSWI	132
KernelTimer	133
KeyEvent_t	135
LinkList	138
CircularLinkList	74

ThreadList	218
DoubleLinkedList	85
TimerList	227
TransactionQueue	233
LinkedListNode	139
DCPUPlugin	81
Driver	93
DevNull	83
GraphicsDriver	106
GuiControl	111
ButtonControl	70
CheckBoxControl	72
GamePanelControl	103
GroupBoxControl	110
LabelControl	136
NotificationControl	171
PanelControl	174
ProgressControl	178
SlickButtonControl	194
SlickGroupBoxControl	196
SlickProgressControl	198
StubControl	207
GuiWindow	122
Message	145
Screen	185
Thread	210
Timer	223
Transaction	231
MemUtil	141
MessageQueue	147
MouseEvent_t	148
NLFS	152
NLFS_RAM	168
NLFS_Block_t	163
NLFS_File	163
NLFS_File_Node_t	166
NLFS_File_Stat_t	166
NLFS_Host_t	167
NLFS_Node_t	167
NLFS_Root_Node_t	171
Option_t	173
Profiler	176
ProfileTimer	176
Quantum	180
Scheduler	181
ScreenList	186
ScreenManager	187
ShellCommand_t	192
ShellSupport	192
Slip	200
SlipDataVector	202
SlipMux	203
SlipTerm	205
SystemHeap	208
TextFX_t	210
ThreadPort	221
Tile_8x8	222

TileDef_t	223
TimerEvent_t	227
TimerScheduler	229
Token_t	230
TouchEvent_t	231
UnitTest	235
WriteBuffer16	239

Chapter 14

Class Index

14.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Atomic	
The Atomic class	63
BitStreamer	65
BlockHeap	
Single-block-size heap	66
BlockingObject	
Class implementing thread-blocking primitives	68
ButtonControl	70
CheckBoxControl	72
CircularLinkedList	
Circular-linked-list data type, inherited from the base LinkedList type	74
CommandLine_t	
Structure containing multiple representations for command-line data	75
DCPU	
DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH	76
DCPU_Registers	
Structure defining the DCPU hardware registers	80
DCPUPlugin	
Class used to provide the hardware device abstraction between the DCPU -16 emulator/VM and the host system	81
DevNull	
This class implements the "default" driver (/dev/null)	83
DoubleLinkedList	
Doubly-linked-list data type, inherited from the base LinkedList type	85
DrawBitmap_t	
Defines a bitmap	86
DrawCircle_t	
Defines a circle	87
DrawEllipse_t	
Defines a ellipse	88
DrawLine_t	
Defines a simple line	88
DrawMove_t	
Simple 2D copy/paste	89
DrawPoint_t	
Defines a pixel	89

DrawPoly_t	Defines the structure of an arbitrary polygon	90
DrawRectangle_t	Defines a rectangle	90
DrawStamp_t	Defines a 1-bit 2D bitmap of arbitrary resolution	91
DrawText_t	Defines a bitmap-rendered string	92
DrawVector_t	Specifies a single 2D point	92
DrawWindow_t	Defines the active window - establishes boundaries for drawing on the current display	93
Driver	Base device-driver class used in hardware abstraction	93
DriverList	List of Driver objects used to keep track of all device drivers in the system	96
EventFlag	Blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system	97
FixedHeap	Fixed-size-block heap allocator with multiple block sizes	101
Font_t	102
GamePanelControl	103
GlobalMessagePool	Implements a list of message objects shared between all threads	104
Glyph_t	105
GraphicsDriver	Defines the base graphics driver class, which is inherited by all other graphics drivers	106
GroupBoxControl	110
GuiControl	GUI Control Base Class	111
GuiEvent_t	Composite UI event structure	119
GuiEventSurface	GUI Event Surface Object	119
GuiWindow	Basic Window Class	122
HeapConfig	Heap configuration object	129
JoystickEvent_t	Joystick UI event structure	129
Kernel	Class that encapsulates all of the kernel startup functions	130
KernelSWI	Class providing the software-interrupt required for context-switching in the kernel	132
KernelTimer	Hardware timer interface, used by all scheduling/timer subsystems	133
KeyEvent_t	Keyboard UI event structure definition	135
LabelControl	136
LinkedList	Abstract-data-type from which all other linked-lists are derived	138
LinkedListNode	Basic linked-list node data structure	139
MemUtil	String and Memory manipulation class	141
Message	Class to provide message-based IPC services in the kernel	145

MessageQueue	
List of messages, used as the channel for sending and receiving messages between threads	147
MouseEvent_t	
Mouse UI event structure	148
Mutex	
Mutual-exclusion locks, based on BlockingObject	149
NLFS	
Nice Little File System class	152
NLFS_Block_t	
Block data structure	163
NLFS_File	
The NLFS_File class	163
NLFS_File_Node_t	
Data structure for the "file" FS-node type	166
NLFS_File_Stat_t	
Structure used to report the status of a given file	166
NLFS_Host_t	
Union used for managing host-specific pointers/data-types	167
NLFS_Node_t	
Filesystem node data structure	167
NLFS_RAM	
The NLFS_RAM class	168
NLFS_Root_Node_t	
Data structure for the Root-configuration FS-node type	171
NotificationControl	171
Option_t	
Structure used to represent a command-line option with its arguments	173
PanelControl	174
Profiler	
System profiling timer interface	176
ProfileTimer	
Profiling timer	176
ProgressControl	178
Quantum	
Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling	180
Scheduler	
Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping	181
Screen	185
ScreenList	186
ScreenManager	187
Semaphore	
Counting semaphore, based on BlockingObject base class	188
ShellCommand_t	
Data structure defining a lookup table correlating a command name to its handler function	192
ShellSupport	
Features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell	192
SlickButtonControl	194
SlickGroupBoxControl	196
SlickProgressControl	198
Slip	
Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP)	200
SlipDataVector	
Data structure used for vector-based SLIP data transmission	202
SlipMux	
Static-class which implements a multiplexed stream of SLIP data over a single interface	203

SlipTerm	Class implementing a simple debug terminal interface	205
StubControl	Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented	207
SystemHeap	Implements a heap which is accessible from all components in the system	208
TextFX_t	210
Thread	Object providing fundamental multitasking support in the kernel	210
ThreadList	This class is used for building thread-management facilities, such as schedulers, and blocking objects	218
ThreadPort	Class defining the architecture specific functions required by the kernel	221
Tile_8x8	222
TileDef_t	TileDef_t Structure defining parameters for a color-indexed tile	223
Timer	Timer - an event-driven execution context based on a specified time interval	223
TimerEvent_t	Timer UI event structure	227
TimerList	TimerList class - a doubly-linked-list of timer objects	227
TimerScheduler	"Static" Class used to interface a global TimerList with the rest of the kernel	229
Token_t	Token descriptor struct format	230
TouchEvent_t	Touch UI event structure	231
Transaction	The Transaction class	231
TransactionQueue	The TransactionQueue class	233
UnitTest	Class used to implement a simple unit-testing framework	235
WriteBuffer16	This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc . . .	239

Chapter 15

File Index

15.1 File List

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

/home/mo/mark3-source/embedded/stage/src/ atomic.cpp	
Basic Atomic Operations	243
/home/mo/mark3-source/embedded/stage/src/ atomic.h	
Basic Atomic Operations	245
/home/mo/mark3-source/embedded/stage/src/ bitstream.cpp	??
/home/mo/mark3-source/embedded/stage/src/ bitstream.h	??
/home/mo/mark3-source/embedded/stage/src/ blocking.cpp	
Implementation of base class for blocking objects	246
/home/mo/mark3-source/embedded/stage/src/ blocking.h	
Blocking object base class declarations	248
/home/mo/mark3-source/embedded/stage/src/ colordepth.h	??
/home/mo/mark3-source/embedded/stage/src/ colorspace.h	??
/home/mo/mark3-source/embedded/stage/src/ control_button.cpp	
GUI Button Control Implementation	249
/home/mo/mark3-source/embedded/stage/src/ control_button.h	
GUI Button Control	252
/home/mo/mark3-source/embedded/stage/src/ control_checkbox.cpp	
Checkbox Control	254
/home/mo/mark3-source/embedded/stage/src/ control_checkbox.h	
Checkbox Control	257
/home/mo/mark3-source/embedded/stage/src/ control_gamepanel.cpp	
GUI Panel Control Implementation with joystick control and tick-based state machine updates	258
/home/mo/mark3-source/embedded/stage/src/ control_gamepanel.h	
GUI Game Panel Control	259
/home/mo/mark3-source/embedded/stage/src/ control_groupbox.cpp	
GUI GroupBox Control Implementation	260
/home/mo/mark3-source/embedded/stage/src/ control_groupbox.h	
GUI Group Box Control	261
/home/mo/mark3-source/embedded/stage/src/ control_label.cpp	??
/home/mo/mark3-source/embedded/stage/src/ control_label.h	
GUI Label Control	262
/home/mo/mark3-source/embedded/stage/src/ control_notification.cpp	
Notification pop-up control	263
/home/mo/mark3-source/embedded/stage/src/ control_notification.h	
Notification pop-up control	265
/home/mo/mark3-source/embedded/stage/src/ control_panel.cpp	
GUI Panel Control Implementation	266

/home/mo/mark3-source/embedded/stage/src/control_panel.h	
GUI Panel Control	267
/home/mo/mark3-source/embedded/stage/src/control_progress.cpp	
GUI Progress Bar Control	268
/home/mo/mark3-source/embedded/stage/src/control_progress.h	
GUI Progress Bar Control	270
/home/mo/mark3-source/embedded/stage/src/control_slickbutton.cpp	??
/home/mo/mark3-source/embedded/stage/src/control_slickbutton.h	
GUI Button Control, with a flare	271
/home/mo/mark3-source/embedded/stage/src/control_slickgroupbox.cpp	??
/home/mo/mark3-source/embedded/stage/src/control_slickgroupbox.h	??
/home/mo/mark3-source/embedded/stage/src/control_slickprogress.cpp	
GUI Progress Bar Control, with flare	272
/home/mo/mark3-source/embedded/stage/src/control_slickprogress.h	
GUI Progress Bar Control, with flare	273
/home/mo/mark3-source/embedded/stage/src/dcpu.cpp	
Portable DCPU-16 CPU emulator	275
/home/mo/mark3-source/embedded/stage/src/dcpu.h	
DCPU-16 emulator	289
/home/mo/mark3-source/embedded/stage/src/debug_tokens.h	
Hex codes/translation tables used for efficient string tokenization	295
/home/mo/mark3-source/embedded/stage/src/draw.h	
Raster graphics APIs Description: Implements basic drawing functionality	297
/home/mo/mark3-source/embedded/stage/src/driver.cpp	
Device driver/hardware abstraction layer	300
/home/mo/mark3-source/embedded/stage/src/driver.h	
Driver abstraction framework	302
/home/mo/mark3-source/embedded/stage/src/eventflag.cpp	
Event Flag Blocking Object/IPC-Object implementation	304
/home/mo/mark3-source/embedded/stage/src/eventflag.h	
Event Flag Blocking Object/IPC-Object definition	309
/home/mo/mark3-source/embedded/stage/src/fixed_heap.cpp	
Fixed-block-size memory management	310
/home/mo/mark3-source/embedded/stage/src/fixed_heap.h	
Fixed-block-size heaps	312
/home/mo/mark3-source/embedded/stage/src/font.h	
Font structure definitions	314
/home/mo/mark3-source/embedded/stage/src/fontport.h	??
/home/mo/mark3-source/embedded/stage/src/graphics.cpp	
Generic graphics driver implementation	315
/home/mo/mark3-source/embedded/stage/src/graphics.h	
Graphics driver class declaration	328
/home/mo/mark3-source/embedded/stage/src/gui.cpp	
Graphical User Interface classes and data structure definitions	329
/home/mo/mark3-source/embedded/stage/src/gui.h	
Graphical User Interface classes and data structure declarations	339
/home/mo/mark3-source/embedded/stage/src/kernel.cpp	
Kernel initialization and startup code	345
/home/mo/mark3-source/embedded/stage/src/kernel.h	
Kernel initialization and startup class	346
/home/mo/mark3-source/embedded/stage/src/kernel_debug.h	
Macros and functions used for assertions, kernel traces, etc	347
/home/mo/mark3-source/embedded/stage/src/kernelswi.cpp	
Kernel Software interrupt implementation for ATMega328p	349
/home/mo/mark3-source/embedded/stage/src/kernelswi.h	
Kernel Software interrupt declarations	350
/home/mo/mark3-source/embedded/stage/src/kerneltimer.cpp	
Kernel Timer Implementation for ATMega328p	351

/home/mo/mark3-source/embedded/stage/src/kerneltimer.h	
Kernel Timer Class declaration	354
/home/mo/mark3-source/embedded/stage/src/kerneltypes.h	
Basic data type primitives used throughout the OS	355
/home/mo/mark3-source/embedded/stage/src/keycodes.h	
Standard ASCII keyboard codes	356
/home/mo/mark3-source/embedded/stage/src/kprofile.cpp	
ATMega328p Profiling timer implementation	358
/home/mo/mark3-source/embedded/stage/src/kprofile.h	
Profiling timer hardware interface	360
/home/mo/mark3-source/embedded/stage/src/ksemaphore.cpp	
Semaphore Blocking-Object Implementation	361
/home/mo/mark3-source/embedded/stage/src/ksemaphore.h	
Semaphore Blocking Object class declarations	365
/home/mo/mark3-source/embedded/stage/src/ll.cpp	
Core Linked-List implementation, from which all kernel objects are derived	366
/home/mo/mark3-source/embedded/stage/src/ll.h	
Core linked-list declarations, used by all kernel list types	369
/home/mo/mark3-source/embedded/stage/src/manual.h	
Ascii-format documentation, used by doxygen to create various printable and viewable forms	371
/home/mo/mark3-source/embedded/stage/src/mark3cfg.h	
Mark3 Kernel Configuration	374
/home/mo/mark3-source/embedded/stage/src/memutil.cpp	
Implementation of memory, string, and conversion routines	375
/home/mo/mark3-source/embedded/stage/src/memutil.h	
Utility class containing memory, string, and conversion routines	381
/home/mo/mark3-source/embedded/stage/src/message.cpp	
Inter-thread communications via message passing	382
/home/mo/mark3-source/embedded/stage/src/message.h	
Inter-thread communication via message-passing	385
/home/mo/mark3-source/embedded/stage/src/mutex.cpp	
Mutual-exclusion object	387
/home/mo/mark3-source/embedded/stage/src/mutex.h	
Mutual exclusion class declaration	392
/home/mo/mark3-source/embedded/stage/src/nlfs.cpp	
Nice Little Filesystem (NLFS) implementation for Mark3	393
/home/mo/mark3-source/embedded/stage/src/nlfs.h	
Nice Little Filesystem (NLFS) - a simple, embeddable filesystem	407
/home/mo/mark3-source/embedded/stage/src/nlfs_config.h	
NLFS configuration parameters	410
/home/mo/mark3-source/embedded/stage/src/nlfs_file.cpp	
Nice Little Filesystem - File Access Class	411
/home/mo/mark3-source/embedded/stage/src/nlfs_file.h	
NLFS file access class	415
/home/mo/mark3-source/embedded/stage/src/nlfs_ram.cpp	
RAM-based Nice Little Filesystem (NLFS) driver	416
/home/mo/mark3-source/embedded/stage/src/nlfs_ram.h	
RAM-based Nice Little Filesystem (NLFS) driver	418
/home/mo/mark3-source/embedded/stage/src/panic_codes.h	
Define and enumerate the possible causes of a kernel panic	419
/home/mo/mark3-source/embedded/stage/src/profile.cpp	
Code profiling utilities	420
/home/mo/mark3-source/embedded/stage/src/profile.h	
High-precision profiling timers	422
/home/mo/mark3-source/embedded/stage/src/profiling_results.h	??
/home/mo/mark3-source/embedded/stage/src/quantum.cpp	
Thread Quantum Implementation for Round-Robin Scheduling	423

/home/mo/mark3-source/embedded/stage/src/quantum.h	
Thread Quantum declarations for Round-Robin Scheduling	425
/home/mo/mark3-source/embedded/stage/src/scheduler.cpp	
Strict-Priority + Round-Robin thread scheduler implementation	426
/home/mo/mark3-source/embedded/stage/src/scheduler.h	
Thread scheduler function declarations	428
/home/mo/mark3-source/embedded/stage/src/screen.cpp	
Higher level window management framework	429
/home/mo/mark3-source/embedded/stage/src/screen.h	
Higher level window management framework	430
/home/mo/mark3-source/embedded/stage/src/shell_support.cpp	
Support functions & data structures useful in implementing a shell	432
/home/mo/mark3-source/embedded/stage/src/shell_support.h	
Support functions & data structures useful in implementing a shell	435
/home/mo/mark3-source/embedded/stage/src/size_report.h	??
/home/mo/mark3-source/embedded/stage/src/slip.cpp	
Serial Line IP framing code	436
/home/mo/mark3-source/embedded/stage/src/slip.h	
Serial Line IP framing code	440
/home/mo/mark3-source/embedded/stage/src/slip_mux.cpp	
FunkenSlip Channel Multiplexer	442
/home/mo/mark3-source/embedded/stage/src/slip_mux.h	
FunkenSlip Channel Multiplexer	444
/home/mo/mark3-source/embedded/stage/src/slipterm.cpp	
Serial debug interface using SLIP protocol, and FunkenSlip multiplexing	445
/home/mo/mark3-source/embedded/stage/src/slipterm.h	
Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing	446
/home/mo/mark3-source/embedded/stage/src/system_heap.cpp	
Global system-heap implementation	447
/home/mo/mark3-source/embedded/stage/src/system_heap.h	
Global system-heap implmentation	450
/home/mo/mark3-source/embedded/stage/src/system_heap_config.h	
System heap configuration - defines the block sizes and counts used to fulfill system/service allocations	454
/home/mo/mark3-source/embedded/stage/src/thread.cpp	
Platform-Independent thread class Definition	455
/home/mo/mark3-source/embedded/stage/src/thread.h	
Platform independent thread class declarations	460
/home/mo/mark3-source/embedded/stage/src/threadlist.cpp	
Thread linked-list definitions	462
/home/mo/mark3-source/embedded/stage/src/threadlist.h	
Thread linked-list declarations	464
/home/mo/mark3-source/embedded/stage/src/threadport.cpp	
ATMega328p Multithreading	465
/home/mo/mark3-source/embedded/stage/src/threadport.h	
ATMega328p Multithreading support	468
/home/mo/mark3-source/embedded/stage/src/tiles.cpp	??
/home/mo/mark3-source/embedded/stage/src/tiles.h	??
/home/mo/mark3-source/embedded/stage/src/timerlist.cpp	
Timer data structure + scheduler implementations	470
/home/mo/mark3-source/embedded/stage/src/timerlist.h	
Timer list and timer-scheduling declarations	475
/home/mo/mark3-source/embedded/stage/src/tracebuffer.cpp	
Kernel trace buffer class definition	478
/home/mo/mark3-source/embedded/stage/src/tracebuffer.h	
Kernel trace buffer class declaration	479
/home/mo/mark3-source/embedded/stage/src/transaction.cpp	
Transaction Queue Implementation	479

/home/mo/mark3-source/embedded/stage/src/ transaction.h	
Transaction Queue Implementation	481
/home/mo/mark3-source/embedded/stage/src/ unit_test.cpp	
Unit test class definition	482
/home/mo/mark3-source/embedded/stage/src/ unit_test.h	
Unit test class declarations	483
/home/mo/mark3-source/embedded/stage/src/ writebuf16.cpp	
16 bit circular buffer implementation with callbacks	485
/home/mo/mark3-source/embedded/stage/src/ writebuf16.h	
Thread-safe circular buffer implementation with 16-bit elements	487

Chapter 16

Class Documentation

16.1 Atomic Class Reference

The [Atomic](#) class.

```
#include <atomic.h>
```

Static Public Member Functions

- static K_UCHAR [Set](#) (K_UCHAR *pucSource_, K_UCHAR ucVal_)
Set Set a variable to a given value in an uninterruptable operation.
- static K_USHORT **Set** (K_USHORT *pusSource_, K_USHORT usVal_)
- static K_ULONG **Set** (K_ULONG *pulSource_, K_ULONG ulVal_)
- static K_UCHAR [Add](#) (K_UCHAR *pucSource_, K_UCHAR ucVal_)
Add Add a value to a variable in an uninterruptable operation.
- static K_USHORT **Add** (K_USHORT *pusSource_, K_USHORT usVal_)
- static K_ULONG **Add** (K_ULONG *pulSource_, K_ULONG ulVal_)
- static K_UCHAR [Sub](#) (K_UCHAR *pucSource_, K_UCHAR ucVal_)
Sub Subtract a value from a variable in an uninterruptable operation.
- static K_USHORT **Sub** (K_USHORT *pusSource_, K_USHORT usVal_)
- static K_ULONG **Sub** (K_ULONG *pulSource_, K_ULONG ulVal_)
- static K_BOOL [TestAndSet](#) (K_BOOL *pbLock)
TestAndSet Test to see if a variable is set, and set it if is not already set.

16.1.1 Detailed Description

The [Atomic](#) class.

This utility class provides primitives for atomic operations - that is, operations that are guaranteed to execute uninterrupted. Basic atomic primitives provided here include Set/Add/Delete for 8, 16, and 32-bit integer types, as well as an atomic test-and-set.

Definition at line 39 of file [atomic.h](#).

16.1.2 Member Function Documentation

16.1.2.1 K_UCHAR Atomic::Add (K_UCHAR * *pucSource_*, K_UCHAR *ucVal_*) [static]

Add Add a value to a variable in an uninterruptable operation.

Parameters

<i>pucSource_</i>	Pointer to a variable
<i>ucVal_</i>	Value to add to the variable

Returns

Previously-held value in *pucSource_*

Definition at line 60 of file [atomic.cpp](#).

16.1.2.2 `K_UCHAR Atomic::Set (K_UCHAR * pucSource_, K_UCHAR ucVal_) [static]`

Set Set a variable to a given value in an uninterruptable operation.

Parameters

<i>pucSource_</i>	Pointer to a variable to set the value of
<i>ucVal_</i>	New value to set in the variable

Returns

Previously-set value

Definition at line 29 of file [atomic.cpp](#).

16.1.2.3 `K_UCHAR Atomic::Sub (K_UCHAR * pucSource_, K_UCHAR ucVal_) [static]`

Sub Subtract a value from a variable in an uninterruptable operation.

Parameters

<i>pucSource_</i>	Pointer to a variable
<i>ucVal_</i>	Value to subtract from the variable

Returns

Previously-held value in *pucSource_*

Definition at line 93 of file [atomic.cpp](#).

16.1.2.4 `K_BOOL Atomic::TestAndSet (K_BOOL * pbLock) [static]`

TestAndSet Test to see if a variable is set, and set it if is not already set.

This is an uninterruptable operation.

If the value is false, set the variable to true, and return the previously-held value.

If the value is already true, return true.

Parameters

<i>pbLock</i>	Pointer to a value to test against. This will always be set to "true" at the end of a call to TestAndSet.
---------------	---

Returns

true - Lock value was "true" on entry, false - Lock was set

Definition at line 126 of file [atomic.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/atomic.h](#)
- [/home/mo/mark3-source/embedded/stage/src/atomic.cpp](#)

16.2 BitStreamer Class Reference

Public Member Functions

- void [Init](#) (K_UCHAR *pucData_, K_USHORT usSize_)
Init.
- void [AdvanceByte](#) (void)
AdvanceByte.
- K_UCHAR [ReadBits](#) (K_UCHAR ucNumBits_)
ReadBits.

Private Attributes

- K_UCHAR * [m_pucData](#)
Pointer to the data being streamed.
- K_UCHAR [m_ucBitIndex](#)
Current "bit" index in the current byte.
- K_USHORT [m_usByteIndex](#)
Current "byte" index in the stream.
- K_USHORT [m_usSize](#)
Length of data (in bytes)

16.2.1 Detailed Description

Definition at line 21 of file [bitstream.h](#).

16.2.2 Member Function Documentation

16.2.2.1 void BitStreamer::AdvanceByte (void)

AdvanceByte.

Advance byte index to the next full byte if the current bit index is non-zero. If the current bit index is zero, no action is taken. This is used to byte-align 2-dimensional data, such as images.

Definition at line 28 of file [bitstream.cpp](#).

16.2.2.2 void BitStreamer::Init (K_UCHAR * pucData_, K_USHORT usSize_)

Init.

Initialize the [BitStreamer](#) object prior to use

Parameters

<i>pucData_</i>	Pointer to raw data to be streamed
<i>usSize_</i>	Size of pucData_ in bytes

Definition at line 19 of file [bitstream.cpp](#).

16.2.2.3 K_UCHAR BitStreamer::ReadBits (K_UCHAR ucNumBits_)

ReadBits.

Read the next "n" bits from the stream, returning the result into an 8-bit unsigned integer.

Parameters

<i>ucNumBits_</i>	Number of bits to read (less than 8)
-------------------	--------------------------------------

Returns

Bits read as an 8-bit unsigned integer.

Definition at line 38 of file [bitstream.cpp](#).

The documentation for this class was generated from the following files:

- /home/mo/mark3-source/embedded/stage/src/bitstream.h
- /home/mo/mark3-source/embedded/stage/src/bitstream.cpp

16.3 BlockHeap Class Reference

Single-block-size heap.

```
#include <fixed_heap.h>
```

Public Member Functions

- void * [Create](#) (void *pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_)
Create a single list heap in the blob of memory provided, with the selected heap size, and the selected number of blocks.
- void * [Alloc](#) ()
Allocate a block of memory from this heap.
- void [Free](#) (void *pvData_)
Free a previously allocated block of memory.
- K_BOOL [IsFree](#) ()
Returns the state of a heap - whether or not it has free elements.

Protected Attributes

- K_USHORT [m_usBlocksFree](#)
Number of blocks free in the heap.

Private Attributes

- [DoubleLinkedList m_clList](#)
Linked list used to manage the blocks.

16.3.1 Detailed Description

Single-block-size heap.

Definition at line 29 of file [fixed_heap.h](#).

16.3.2 Member Function Documentation

16.3.2.1 void * BlockHeap::Alloc ()

Allocate a block of memory from this heap.

Returns

pointer to a block of memory, or 0 on failure

Definition at line 83 of file [fixed_heap.cpp](#).

16.3.2.2 void * BlockHeap::Create (void * pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_)

Create a single list heap in the blob of memory provided, with the selected heap size, and the selected number of blocks.

Will create as many blocks as will fit in the usSize_ parameter

Parameters

<i>pvHeap_</i>	Pointer to the heap data to initialize
<i>usSize_</i>	Size of the heap range in bytes
<i>usBlockSize_</i>	Size of each heap block in bytes

Returns

Pointer to the next heap element to initialize

Definition at line 48 of file [fixed_heap.cpp](#).

16.3.2.3 void BlockHeap::Free (void * pvData_)

Free a previously allocated block of memory.

Parameters

<i>pvData_</i>	Pointer to a block of data previously allocated off the heap.
----------------	---

Definition at line 102 of file [fixed_heap.cpp](#).

16.3.2.4 K_BOOL BlockHeap::IsFree () [inline]

Returns the state of a heap - whether or not it has free elements.

Returns

true if the heap is not full, false if the heap is full

Definition at line 74 of file [fixed_heap.h](#).

The documentation for this class was generated from the following files:

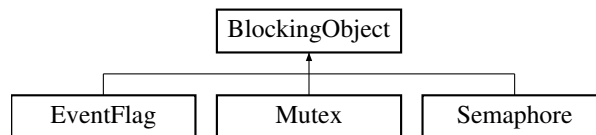
- [/home/mo/mark3-source/embedded/stage/src/fixed_heap.h](#)
- [/home/mo/mark3-source/embedded/stage/src/fixed_heap.cpp](#)

16.4 BlockingObject Class Reference

Class implementing thread-blocking primitives.

```
#include <blocking.h>
```

Inheritance diagram for BlockingObject:



Protected Member Functions

- void **Block** (**Thread** *pciThread_)
- void **UnBlock** (**Thread** *pciThread_)
- K_UCHAR **UnLock** ()
Unlock.
- K_BOOL **LockAndQueue** (K_USHORT usCode_, void *pvData_, K_BOOL *pbSchedState_)
LockAndQueue.

Protected Attributes

- **ThreadList** m_clBlockList
ThreadList which is used to hold the list of threads blocked on a given object.
- **TransactionQueue** m_clKTQ
Kernel Transaction Queue used to serialize acceses to this blocking object.
- K_UCHAR m_ucLocks
The current count of locks held by this blocking object.

16.4.1 Detailed Description

Class implementing thread-blocking primitives.

Used for implementing things like semaphores, mutexes, message queues, or anything else that could cause a thread to suspend execution on some external stimulus.

Definition at line 67 of file [blocking.h](#).

16.4.2 Member Function Documentation

16.4.2.1 void BlockingObject::Block (Thread * pciThread_) [protected]

Parameters

<i>pciThread_</i>	Pointer to the thread object that will be blocked.
-------------------	--

Blocks a thread on this object. This is the fundamental operation performed by any sort of blocking operation in the operating system. All semaphores/mutexes/sleeping/messaging/etc ends up going through the blocking code at some point as part of the code that manages a transition from an "active" or "waiting" thread to a "blocked" thread.

The steps involved in blocking a thread (which are performed in the function itself) are as follows;

1) Remove the specified thread from the current owner's list (which is likely one of the scheduler's thread lists) 2) Add the thread to this object's thread list 3) Setting the thread's "current thread-list" point to reference this object's threadlist.

Definition at line 36 of file [blocking.cpp](#).

16.4.2.2 `K_BOOL BlockingObject::LockAndQueue (K_USHORT usCode_, void * pvData_, K_BOOL * pbSchedState_)`
[protected]

LockAndQueue.

Lock the object and enqueue data on its transaction queue. If the object is already locked, enqueue the data and return back. Otherwise, disable the scheduler and return its state in addition to enqueueing the given transaction

Parameters

<i>usCode_</i>	Transaction code value
<i>pvData_</i>	Abstract transaction data pointer
<i>pbSchedState_</i>	Pointer to a flag used to store the scheduler's original state.

Returns

true - Object was previously locked, false - object was not previously locked.

Definition at line 87 of file [blocking.cpp](#).

16.4.2.3 `void BlockingObject::UnBlock (Thread * pclThread_)` [protected]

Parameters

<i>pclThread_</i>	Pointer to the thread to unblock.
-------------------	-----------------------------------

Unblock a thread that is already blocked on this object, returning it to the "ready" state by performing the following steps:

1) Removing the thread from this object's threadlist 2) Restoring the thread to its "original" owner's list

Definition at line 54 of file [blocking.cpp](#).

16.4.2.4 `K_UCHAR BlockingObject::UnLock ()` [protected]

Unlock.

See Also

Lock

Returns

Count of pending locks held on this blocking object

This function will atomically-decrement the internal lock count held on the object, returning the new lock count value.

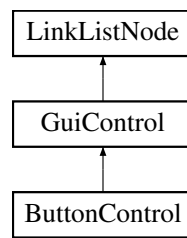
Definition at line 73 of file [blocking.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/blocking.h](#)
- [/home/mo/mark3-source/embedded/stage/src/blocking.cpp](#)

16.5 ButtonControl Class Reference

Inheritance diagram for ButtonControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetBGColor](#) (COLOR eColor_)
- void [SetLineColor](#) (COLOR eColor_)
- void [SetFillColor](#) (COLOR eColor_)
- void [SetTextColor](#) (COLOR eColor_)
- void [SetActiveColor](#) (COLOR eColor_)
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *szCaption_)
- void [SetCallback](#) (ButtonCallback pfCallback_, void *pvData_)

Private Attributes

- const K_CHAR * [m_szCaption](#)
- [Font_t](#) * [m_pstFont](#)
- COLOR [m_uBGColor](#)
- COLOR [m_uActiveColor](#)
- COLOR [m_uLineColor](#)
- COLOR [m_uFillColor](#)
- COLOR [m_uTextColor](#)
- bool [m_bState](#)
- void * [m_pvCallbackData](#)
- ButtonCallback [m_pfCallback](#)

Additional Inherited Members

16.5.1 Detailed Description

Definition at line 32 of file [control_button.h](#).

16.5.2 Member Function Documentation

16.5.2.1 void ButtonControl::Activate (bool *bActivate_*) [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 215 of file [control_button.cpp](#).

16.5.2.2 void ButtonControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 39 of file [control_button.cpp](#).

16.5.2.3 void ButtonControl::Init () [virtual]

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 25 of file [control_button.cpp](#).

16.5.2.4 GuiReturn_t ButtonControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

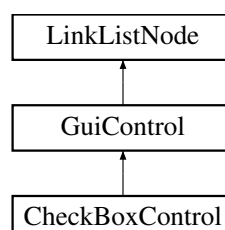
Definition at line 117 of file [control_button.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_button.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_button.cpp](#)

16.6 CheckBoxControl Class Reference

Inheritance diagram for CheckBoxControl:



Public Member Functions

- virtual void **Init** ()
Initialiize the control - must be called before use.
- virtual void **Draw** ()
Redraw the control "cleanly".
- virtual **GuiReturn_t** **ProcessEvent** (**GuiEvent_t** *pstEvent_)
Process an event sent to the control.
- virtual void **Activate** (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void **SetFont** (**Font_t** *pstFont_)
- void **SetCaption** (const char *szCaption_)
- void **SetCheck** (bool bChecked_)
- void **SetFontColor** (COLOR uFontColor_)
- void **SetBoxColor** (COLOR uBoxColor_)
- void **SetBackColor** (COLOR uBackColor_)
- bool **IsChecked** (void)

Private Attributes

- const char * **m_szCaption**
- COLOR **m_uBackColor**
- COLOR **m_uBoxColor**
- COLOR **m_uFontColor**
- **Font_t** * **m_pstFont**
- bool **m_bChecked**

Additional Inherited Members

16.6.1 Detailed Description

Definition at line 29 of file [control_checkbox.h](#).

16.6.2 Member Function Documentation

16.6.2.1 virtual void **CheckBoxControl::Activate** (bool *bActivate_*) [inline],[virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 35 of file [control_checkbox.h](#).

16.6.2.2 void **CheckBoxControl::Draw** () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 59 of file [control_checkbox.cpp](#).

16.6.2.3 void CheckBoxControl::Init () [virtual]

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 53 of file [control_checkbox.cpp](#).

16.6.2.4 GuiReturn_t CheckBoxControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 130 of file [control_checkbox.cpp](#).

The documentation for this class was generated from the following files:

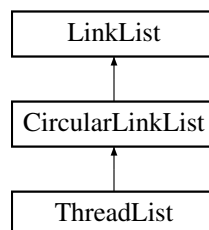
- [/home/mo/mark3-source/embedded/stage/src/control_checkbox.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_checkbox.cpp](#)

16.7 CircularLinkedList Class Reference

Circular-linked-list data type, inherited from the base [LinkList](#) type.

```
#include <ll.h>
```

Inheritance diagram for CircularLinkedList:



Public Member Functions

- virtual void [Add](#) ([LinkListNode](#) *node_)
Add the linked list node to this linked list.
- virtual void [Remove](#) ([LinkListNode](#) *node_)
Add the linked list node to this linked list.
- void [PivotForward](#) ()
Pivot the head of the circularly linked list forward (Head = Head->next, Tail = Tail->next)
- void [PivotBackward](#) ()
Pivot the head of the circularly linked list backward (Head = Head->prev, Tail = Tail->prev)

Additional Inherited Members

16.7.1 Detailed Description

Circular-linked-list data type, inherited from the base [LinkedList](#) type.

Definition at line 197 of file [ll.h](#).

16.7.2 Member Function Documentation

16.7.2.1 void CircularLinkedList::Add ([LinkedListNode](#) * node_) [virtual]

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to add
--------------	----------------------------

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

Definition at line 102 of file [ll.cpp](#).

16.7.2.2 void CircularLinkedList::Remove ([LinkedListNode](#) * node_) [virtual]

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to remove
--------------	-------------------------------

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

Definition at line 127 of file [ll.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/ll.h](#)
- [/home/mo/mark3-source/embedded/stage/src/ll.cpp](#)

16.8 CommandLine_t Struct Reference

Structure containing multiple representations for command-line data.

```
#include <shell_support.h>
```

Public Attributes

- [Token_t](#) * [pastTokenList](#)
Pointer to the list of tokens in the commandline.
- K_UCHAR [ucTokenCount](#)
Count of tokens in the token list.
- [Token_t](#) * [pstCommand](#)
Pointer to the token corresponding to the shell command.
- [Option_t](#) [astOptions](#) [12]
Option strucure array built from the token list.

- K_UCHAR [ucNumOptions](#)
Number of options parsed from the token list.

16.8.1 Detailed Description

Structure containing multiple representations for command-line data.

Definition at line 93 of file [shell_support.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/shell_support.h](#)

16.9 DCPU Class Reference

DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

```
#include <dcpu.h>
```

Public Member Functions

- void [Init](#) (K_USHORT *pusRAM_, K_USHORT usRAMSize_, const K_USHORT *pusROM_, K_USHORT usROMSize_)
Initialize the CPU emulator, specifying which driver supplies the memory read interface.
- void [RunOpcode](#) ()
Execute the next opcode at the VM's current PC.
- [DCPU_Registers](#) * [GetRegisters](#) ()
Return a pointer to the VM's register structure.
- void [SendInterrupt](#) (K_USHORT usMessage_)
Send an interrupt to the CPU with a given message.
- void [AddPlugin](#) ([DCPUPlugin](#) *pclPlugin_)
Add a plugin to the CPU.

Private Member Functions

- void **SET** ()
- void **ADD** ()
- void **SUB** ()
- void **MUL** ()
- void **MLI** ()
- void **DIV** ()
- void **DVI** ()
- void **MOD** ()
- void **MDI** ()
- void **AND** ()
- void **BOR** ()
- void **XOR** ()
- void **SHR** ()
- void **ASR** ()
- void **SHL** ()
- bool **IFB** ()
- bool **IFC** ()
- bool **IFE** ()

- bool **IFN** ()
- bool **IFG** ()
- bool **IFA** ()
- bool **IFL** ()
- bool **IFU** ()
- void **ADX** ()
- void **SBX** ()
- void **STI** ()
- void **STD** ()
- void **JSR** ()
- void **INT** ()
- void **IAG** ()
- void **IAS** ()
- void **RFI** ()
- void **IAQ** ()
- void **HWN** ()
- void **HWQ** ()
- void **HWI** ()
- K_UCHAR **GetOperand** (K_UCHAR ucOpType_, K_USHORT **pusResult_)
- void **ProcessInterruptQueue** ()

Process the next interrupt in the Queue.

Private Attributes

- **DCPU_Registers m_stRegisters**
CPU Register file.
- K_USHORT * **a**
Temporary "a" operand pointer.
- K_USHORT * **b**
Temporary "b" operand pointer.
- K_USHORT **m_usTempA**
Local-storage for staging literal "a" values.
- K_USHORT * **m_pusRAM**
Pointer to the RAM buffer.
- K_USHORT **m_usRAMSize**
Size to the RAM (including stack)
- K_USHORT * **m_pusROM**
Pointer to the CPU ROM storage.
- K_USHORT **m_usROMSize**
Size of the ROM.
- K_ULONG **m_ulCycleCount**
Current cycle count.
- K_BOOL **m_bInterruptQueueing**
CPU flag indicating whether or not interrupts are queued.
- K_UCHAR **m_ucQueueLevel**
Current interrupt Queue level.
- K_USHORT **m_ausInterruptQueue** [8]
Interrupt queue.
- **DoubleLinkedList m_clPluginList**
Linked-list of plug-ins.

16.9.1 Detailed Description

DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Definition at line 359 of file [dcpu.h](#).

16.9.2 Member Function Documentation

16.9.2.1 void DCPU::AddPlugin (DCPUPlugin * *pclPlugin_*)

Add a plugin to the CPU.

Parameters

<i>pclPlugin_</i>	Pointer to the plugin object to add
-------------------	-------------------------------------

Definition at line 948 of file [dcpu.cpp](#).

16.9.2.2 K_UCHAR DCPU::GetOperand (K_UCHAR *ucOpType_*, K_USHORT ** *pusResult_*) [private]

Parameters

<i>ucOpType_</i>	The operand type, as specified in DCPU_Argument
<i>pusResult_</i>	Pointer to the pointer that corresponds to the argument's location in memory.

Definition at line 722 of file [dcpu.cpp](#).

16.9.2.3 DCPU_Registers * DCPU::GetRegisters () [inline]

Return a pointer to the VM's register structure.

Returns

Pointer to the VM's register structure

Definition at line 391 of file [dcpu.h](#).

16.9.2.4 void DCPU::HWN () [private]

Returns the number of connected hardware devices to "a"

Definition at line 642 of file [dcpu.cpp](#).

16.9.2.5 void DCPU::IAQ () [private]

Add an interrupt to the interrupt queue if non-zero, if a = 0 then interrupts will be triggered as normal

Interrupts queued

Interrupts triggered

Definition at line 624 of file [dcpu.cpp](#).

16.9.2.6 void DCPU::Init (K_USHORT * *pusRAM_*, K_USHORT *usRAMSize_*, const K_USHORT * *pusROM_*, K_USHORT *usROMSize_*)

Initialize the CPU emulator, specifying which driver supplies the memory read interface.

This allows us to abstract RAM/FLASH/EEPROM or other memory. The VM must be initialized before any other method in the class is run.

Parameters

<i>pusRAM_</i>	Pointer to the CPU's RAM buffer
<i>usRAMSize_</i>	Size of the RAM Buffer in words
<i>pusROM_</i>	Pointer to the CPU's ROM buffer
<i>usROMSize_</i>	Size of the ROM buffer in words

Definition at line 697 of file [dcpu.cpp](#).

16.9.2.7 void DCPU::RFI() [private]

Disables interrupt queueing, pop A from the stack, then pops PC from the stack. By disabling interrupt Queueing, we're essentially re-enabling interrupts.

Definition at line 609 of file [dcpu.cpp](#).

16.9.2.8 void DCPU::SendInterrupt (K_USHORT usMessage_)

Send an interrupt to the CPU with a given message.

Parameters

<i>usMessage_</i>	Message to send along with the interrupt
-------------------	--

Definition at line 922 of file [dcpu.cpp](#).

16.9.3 Member Data Documentation**16.9.3.1 DoubleLinkedList DCPU::m_clPluginList [private]**

Linked-list of plug-ins.

Definition at line 489 of file [dcpu.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/dcpu.h](#)
- [/home/mo/mark3-source/embedded/stage/src/dcpu.cpp](#)

16.10 DCPU_Registers Struct Reference

Structure defining the [DCPU](#) hardware registers.

```
#include <dcpu.h>
```

Public Attributes

- union {
 struct {
 K_USHORT A
 K_USHORT B
 K_USHORT C
 K_USHORT X
 K_USHORT Y
 K_USHORT Z
 K_USHORT I
 K_USHORT J
 }


```

    K_USHORT PC
    K_USHORT SP
    K_USHORT EX
    K_USHORT IA
}
K_USHORT ausRegisters [12]
};

```

16.10.1 Detailed Description

Structure defining the [DCPU](#) hardware registers.

Definition at line 72 of file [dcpu.h](#).

The documentation for this struct was generated from the following file:

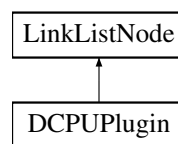
- [/home/mo/mark3-source/embedded/stage/src/dcpu.h](#)

16.11 DCPUPugin Class Reference

Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system.

```
#include <dcpu.h>
```

Inheritance diagram for DCPUPugin:



Public Member Functions

- void [Init](#) (K_USHORT usDeviceNumber_, K_ULONG ulHWID_, K_ULONG ulVID_, K_USHORT usVersion_, [DCPU_Callback](#) pfCallback_)
Initialize the [DCPU](#) plugin extension.
- void [Enumerate](#) ([DCPU_Registers](#) *pstRegisters_)
Perform hardware enumeration to the target VM specified by the register set.
- void [Interrupt](#) ([DCPU](#) *pdcPU_)
Execute the hardware callback.
- K_USHORT [GetDeviceNumber](#) ()
Return the device number associated with this plugin.

Private Attributes

- K_USHORT [m_usDeviceNumber](#)
Location of the device on the "bus".
- K_ULONG [m_ulHWID](#)
Hardware ID.
- K_ULONG [m_ulVID](#)
Vendor ID.

- K_USHORT [m_usVersion](#)
Hardware Version.
- DCPU_Callback [m_pfCallback](#)
HWI Callback.

Friends

- class **DCPUPluginList**

Additional Inherited Members

16.11.1 Detailed Description

Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system.
Definition at line [267](#) of file [dcpu.h](#).

16.11.2 Member Function Documentation

16.11.2.1 void DCPUPlugin::Enumerate (DCPU_Registers * *pstRegisters_*) [\[inline\]](#)

Perform hardware enumeration to the target VM specified by the register set.

Parameters

<i>pstRegisters_</i>	Pointer to the VM's CPU registers, which are filled with enumeration data. See the DCPU 1.7 spec for details.
----------------------	---

Definition at line [311](#) of file [dcpu.h](#).

16.11.2.2 K_USHORT DCPUPlugin::GetDeviceNumber () [\[inline\]](#)

Return the device number associated with this plugin.

Returns

Device number associated with this plugin

Definition at line [339](#) of file [dcpu.h](#).

16.11.2.3 void DCPUPlugin::Init (K_USHORT *usDeviceNumber_*, K_ULONG *ulHWID_*, K_ULONG *ulVID_*, K_USHORT *usVersion_*, DCPU_Callback *pfCallback_*) [\[inline\]](#)

Initialize the [DCPU](#) plugin extension.

Plug

Parameters

<i>usDevice-Number_</i>	Unique plugin device enumeration associated with this plugin
-------------------------	--

<i>ulHWID_</i>	Unique hardware type identifier
<i>ulVID_</i>	Hardware Vendor ID
<i>usVersion_</i>	Version identifier for this hardware piece
<i>pfCallback_</i>	Callback function invoked from the VM when a HWI instruction is called on this device. This is essentially the interrupt handler.

Definition at line 288 of file [dcpu.h](#).

16.11.2.4 void DCPUPugin::Interrupt (DCPU * *pcCPU_*) [inline]

Execute the hardware callback.

Parameters

<i>pcCPU_</i>	Pointer to the VM triggering the interrupt
---------------	--

Definition at line 327 of file [dcpu.h](#).

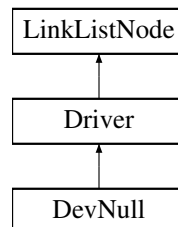
The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/dcpu.h](#)

16.12 DevNull Class Reference

This class implements the "default" driver (/dev/null)

Inheritance diagram for DevNull:



Public Member Functions

- virtual void [Init](#) ()
Initialize a driver, must be called prior to use.
- virtual K_UCHAR [Open](#) ()
Open a device driver prior to use.
- virtual K_UCHAR [Close](#) ()
Close a previously-opened device driver.
- virtual K_USHORT [Read](#) (K_USHORT usBytes_, K_UCHAR *pucData_)
Read a specified number of bytes from the device into a specific buffer.
- virtual K_USHORT [Write](#) (K_USHORT usBytes_, K_UCHAR *pucData_)
Write a payload of data of a given length to the device.
- virtual K_USHORT [Control](#) (K_USHORT usEvent_, void *pvDataIn_, K_USHORT usSizeIn_, void *pvDataOut_, K_USHORT usSizeOut_)
This is the main entry-point for device-specific io and control operations.

Additional Inherited Members

16.12.1 Detailed Description

This class implements the "default" driver (/dev/null)

Definition at line 40 of file [driver.cpp](#).

16.12.2 Member Function Documentation

16.12.2.1 virtual K_UCHAR DevNull::Close () [inline],[virtual]

Close a previously-opened device driver.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements [Driver](#).

Definition at line 45 of file [driver.cpp](#).

16.12.2.2 virtual K_USHORT DevNull::Control (K_USHORT *usEvent_*, void * *pvDataIn_*, K_USHORT *usSizeIn_*, void * *pvDataOut_*, K_USHORT *usSizeOut_*) [inline],[virtual]

This is the main entry-point for device-specific io and control operations.

This is used for implementing all "side-channel" communications with a device, and any device-specific IO operations that do not conform to the typical POSIX read/write paradigm. Use of this function is analagous to the non-POSIX (yet still common) devctl() or ioctl().

Parameters

<i>usEvent_</i>	Code defining the io event (driver-specific)
<i>pvDataIn_</i>	Pointer to the input data
<i>usSizeIn_</i>	Size of the input data (in bytes)
<i>pvDataOut_</i>	Pointer to the output data
<i>usSizeOut_</i>	Size of the output data (in bytes)

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements [Driver](#).

Definition at line 53 of file [driver.cpp](#).

16.12.2.3 virtual K_UCHAR DevNull::Open () [inline],[virtual]

Open a device driver prior to use.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implements [Driver](#).

Definition at line 44 of file [driver.cpp](#).

16.12.2.4 `virtual K_USHORT DevNull::Read (K_USHORT usBytes_, K_UCHAR * pucData_) [inline], [virtual]`

Read a specified number of bytes from the device into a specific buffer.

Depending on the driver-specific implementation, this may be a number less than the requested number of bytes read, indicating that there was less input than desired, or that as a result of buffering, the data may not be available.

Parameters

<i>usBytes_</i>	Number of bytes to read (<= size of the buffer)
<i>pucData_</i>	Pointer to a data buffer receiving the read data

Returns

Number of bytes actually read

Implements [Driver](#).

Definition at line 47 of file [driver.cpp](#).

16.12.2.5 `virtual K_USHORT DevNull::Write (K_USHORT usBytes_, K_UCHAR * pucData_) [inline], [virtual]`

Write a payload of data of a given length to the device.

Depending on the implementation of the driver, the amount of data written to the device may be less than the requested number of bytes. A result less than the requested size may indicate that the device buffer is full, indicating that the user must retry the write at a later point with the remaining data.

Parameters

<i>usBytes_</i>	Number of bytes to write (<= size of the buffer)
<i>pucData_</i>	Pointer to a data buffer containing the data to write

Returns

Number of bytes actually written

Implements [Driver](#).

Definition at line 50 of file [driver.cpp](#).

The documentation for this class was generated from the following file:

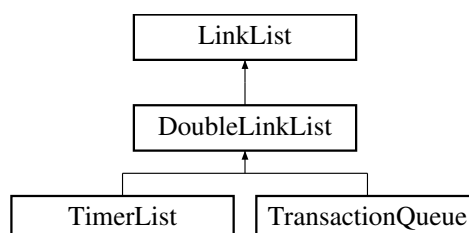
- [/home/mo/mark3-source/embedded/stage/src/driver.cpp](#)

16.13 DoubleLinkedList Class Reference

Doubly-linked-list data type, inherited from the base [LinkedList](#) type.

```
#include <ll.h>
```

Inheritance diagram for DoubleLinkedList:



Public Member Functions

- [DoubleLinkedList](#) ()
Default constructor - initializes the head/tail nodes to NULL.
- virtual void [Add](#) ([LinkedListNode](#) *node_)
Add the linked list node to this linked list.
- virtual void [Remove](#) ([LinkedListNode](#) *node_)
Add the linked list node to this linked list.

Additional Inherited Members

16.13.1 Detailed Description

Doubly-linked-list data type, inherited from the base [LinkedList](#) type.

Definition at line 166 of file [ll.h](#).

16.13.2 Member Function Documentation

16.13.2.1 void [DoubleLinkedList::Add](#) ([LinkedListNode](#) * node_) [virtual]

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to add
--------------	----------------------------

Implements [LinkedList](#).

Definition at line 41 of file [ll.cpp](#).

16.13.2.2 void [DoubleLinkedList::Remove](#) ([LinkedListNode](#) * node_) [virtual]

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to remove
--------------	-------------------------------

Implements [LinkedList](#).

Definition at line 65 of file [ll.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/ll.h](#)
- [/home/mo/mark3-source/embedded/stage/src/ll.cpp](#)

16.14 DrawBitmap_t Struct Reference

Defines a bitmap.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)

- *Leftmost pixel.*
K_USHORT [usY](#)
- *Uppermost pixel.*
K_USHORT [usWidth](#)
- *Width of the bitmap in pixels.*
K_USHORT [usHeight](#)
- *Height of the bitmap in pixels.*
K_UCHAR [ucBPP](#)
- *Bits-per-pixel.*
K_UCHAR * [pucData](#)
- *Pixel data pointer.*

16.14.1 Detailed Description

Defines a bitmap.

Definition at line 117 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.15 DrawCircle_t Struct Reference

Defines a circle.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)
Center X pixel.
- K_USHORT [usY](#)
Center Y pixel.
- K_USHORT [usRadius](#)
Radius in pixels.
- COLOR [uLineColor](#)
Color of the circle perimeter.
- K_BOOL [bFill](#)
Whether or not to fill the interior of the circle.
- COLOR [uFillColor](#)
Fill color for the circle.

16.15.1 Detailed Description

Defines a circle.

Definition at line 92 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.16 DrawEllipse_t Struct Reference

Defines a ellipse.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)
Center X pixel.
- K_USHORT [usY](#)
Center Y pixel.
- K_USHORT [usHeight](#)
Height of the ellipse.
- K_USHORT [usWidth](#)
Width of the ellipse.
- COLOR [uColor](#)
Color of the ellipse perimeter.

16.16.1 Detailed Description

Defines a ellipse.

Definition at line [105](#) of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.17 DrawLine_t Struct Reference

Defines a simple line.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX1](#)
Starting X coordinate.
- K_USHORT [usX2](#)
Ending X coordinate.
- K_USHORT [usY1](#)
Starting Y Coordinate.
- K_USHORT [usY2](#)
Ending Y coordinate.
- COLOR [uColor](#)
Color of the pixel.

16.17.1 Detailed Description

Defines a simple line.

Definition at line 66 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.18 DrawMove_t Struct Reference

Simple 2D copy/paste.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usSrcX](#)
Source X pixel (leftmost)
- K_USHORT [usSrcY](#)
Source Y pixel (topmost)
- K_USHORT [usDstX](#)
Destination X pixel (leftmost)
- K_USHORT [usDstY](#)
Destination Y pixel (topmost)
- K_USHORT [usCopyHeight](#)
Number of rows to copy.
- K_USHORT [usCopyWidth](#)
Number of columns to copy.

16.18.1 Detailed Description

Simple 2D copy/paste.

Moves a bitmap specified by the given source coordinates on-surface to the destination coordinates.

Definition at line 188 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.19 DrawPoint_t Struct Reference

Defines a pixel.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)
X coordinate of the pixel.
- K_USHORT [usY](#)

- *Y coordinate of the pixel.*
- COLOR [uColor](#)
Color of the pixel.

16.19.1 Detailed Description

Defines a pixel.

Definition at line 55 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.20 DrawPoly_t Struct Reference

Defines the structure of an arbitrary polygon.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usNumPoints](#)
Number of points in the polygon.
- COLOR [uColor](#)
Color to use for lines/fill.
- K_BOOL [bFill](#)
Display as wireframe or filled.
- [DrawVector_t](#) * [pstVector](#)
Vector points making the polygon.

16.20.1 Detailed Description

Defines the structure of an arbitrary polygon.

Can be used to specify the

Definition at line 215 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.21 DrawRectangle_t Struct Reference

Defines a rectangle.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usLeft](#)
Leftmost pixel of the rectangle.

- K_USHORT [usTop](#)
Topmost pixel of the rectangle.
- K_USHORT [usRight](#)
Rightmost pixel of the rectangle.
- K_USHORT [usBottom](#)
Bottom pixel of the rectangle.
- COLOR [uLineColor](#)
Color of the line.
- K_BOOL [bFill](#)
Whether or not to floodfill the interior.
- COLOR [uFillColor](#)
Color of the interior of the rectangle.

16.21.1 Detailed Description

Defines a rectangle.

Definition at line 78 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.22 DrawStamp_t Struct Reference

Defines a 1-bit 2D bitmap of arbitrary resolution.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)
Leftmost pixel.
- K_USHORT [usY](#)
Uppermost pixel.
- K_USHORT [usWidth](#)
Width of the stamp.
- K_USHORT [usHeight](#)
Height of the stamp.
- COLOR [uColor](#)
Color of the stamp.
- K_UCHAR * [pucData](#)
Pointer to the stamp data.

16.22.1 Detailed Description

Defines a 1-bit 2D bitmap of arbitrary resolution.

Definition at line 130 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.23 DrawText_t Struct Reference

Defines a bitmap-rendered string.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usLeft](#)
Leftmost pixel of the text.
- K_USHORT [usTop](#)
Uppermost pixel of the text.
- COLOR [uColor](#)
Color of the text.
- [Font_t](#) * [pstFont](#)
Pointer to the font used to render the text.
- const K_CHAR * [pcString](#)
ASCII String to render.

16.23.1 Detailed Description

Defines a bitmap-rendered string.

Definition at line 144 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.24 DrawVector_t Struct Reference

Specifies a single 2D point.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usX](#)
- K_USHORT [usY](#)

16.24.1 Detailed Description

Specifies a single 2D point.

When used in arrays, this provides a way to draw vector paths, which form the basis of the polygon data structures.

Definition at line 204 of file [draw.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.25 DrawWindow_t Struct Reference

Defines the active window - establishes boundaries for drawing on the current display.

```
#include <draw.h>
```

Public Attributes

- K_USHORT [usLeft](#)
Left boundary.
- K_USHORT [usRight](#)
Right boundary.
- K_USHORT [usTop](#)
Upper boundary.
- K_USHORT [usBottom](#)
Bottom boundary.

16.25.1 Detailed Description

Defines the active window - establishes boundaries for drawing on the current display.

Only pixels drawn inside the surface boundaries are rendered to the output

Definition at line 175 of file [draw.h](#).

The documentation for this struct was generated from the following file:

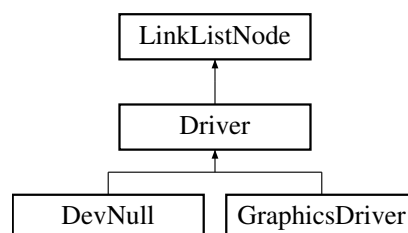
- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.26 Driver Class Reference

Base device-driver class used in hardware abstraction.

```
#include <driver.h>
```

Inheritance diagram for Driver:



Public Member Functions

- virtual void [Init](#) ()=0
Initialize a driver, must be called prior to use.
- virtual K_UCHAR [Open](#) ()=0
Open a device driver prior to use.
- virtual K_UCHAR [Close](#) ()=0
Close a previously-opened device driver.

- virtual K_USHORT [Read](#) (K_USHORT usBytes_, K_UCHAR *pucData_)=0
Read a specified number of bytes from the device into a specific buffer.
- virtual K_USHORT [Write](#) (K_USHORT usBytes_, K_UCHAR *pucData_)=0
Write a payload of data of a given length to the device.
- virtual K_USHORT [Control](#) (K_USHORT usEvent_, void *pvDataIn_, K_USHORT usSizeIn_, void *pvDataOut_, K_USHORT usSizeOut_)=0
This is the main entry-point for device-specific io and control operations.
- void [SetName](#) (const K_CHAR *pcName_)
Set the path for the driver.
- const K_CHAR * [GetPath](#) ()
Returns a string containing the device path.

Private Attributes

- const K_CHAR * [m_pcPath](#)
string pointer that holds the driver path (name)

Additional Inherited Members

16.26.1 Detailed Description

Base device-driver class used in hardware abstraction.

All other device drivers inherit from this class

Definition at line 121 of file [driver.h](#).

16.26.2 Member Function Documentation

16.26.2.1 K_UCHAR Driver::Close () [pure virtual]

Close a previously-opened device driver.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in [DevNull](#).

16.26.2.2 K_USHORT Driver::Control (K_USHORT usEvent_, void * pvDataIn_, K_USHORT usSizeIn_, void * pvDataOut_, K_USHORT usSizeOut_) [pure virtual]

This is the main entry-point for device-specific io and control operations.

This is used for implementing all "side-channel" communications with a device, and any device-specific IO operations that do not conform to the typical POSIX read/write paradigm. Use of this function is analagous to the non-POSIX (yet still common) `devctl()` or `ioctl()`.

Parameters

<i>usEvent_</i>	Code defining the io event (driver-specific)
-----------------	--

<i>pvDataIn_</i>	Pointer to the input data
<i>usSizeIn_</i>	Size of the input data (in bytes)
<i>pvDataOut_</i>	Pointer to the output data
<i>usSizeOut_</i>	Size of the output data (in bytes)

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in [DevNull](#).

16.26.2.3 `const K_CHAR * Driver::GetPath () [inline]`

Returns a string containing the device path.

Returns

pcName_ Return the string constant representing the device path

Definition at line 231 of file [driver.h](#).

16.26.2.4 `K_UCHAR Driver::Open () [pure virtual]`

Open a device driver prior to use.

Returns

Driver-specific return code, 0 = OK, non-0 = error

Implemented in [DevNull](#).

16.26.2.5 `K_USHORT Driver::Read (K_USHORT usBytes_, K_UCHAR * pucData_) [pure virtual]`

Read a specified number of bytes from the device into a specific buffer.

Depending on the driver-specific implementation, this may be a number less than the requested number of bytes read, indicating that there there was less input than desired, or that as a result of buffering, the data may not be available.

Parameters

<i>usBytes_</i>	Number of bytes to read (<= size of the buffer)
<i>pucData_</i>	Pointer to a data buffer receiving the read data

Returns

Number of bytes actually read

Implemented in [DevNull](#).

16.26.2.6 `void Driver::SetName (const K_CHAR * pcName_) [inline]`

Set the path for the driver.

Name must be set prior to access (since driver access is name-based).

Parameters

<i>pcName_</i>	String constant containing the device path
----------------	--

Definition at line 222 of file [driver.h](#).

16.26.2.7 K_USHORT Driver::Write (K_USHORT *usBytes_*, K_UCHAR * *pucData_*) [pure virtual]

Write a payload of data of a given length to the device.

Depending on the implementation of the driver, the amount of data written to the device may be less than the requested number of bytes. A result less than the requested size may indicate that the device buffer is full, indicating that the user must retry the write at a later point with the remaining data.

Parameters

<i>usBytes_</i>	Number of bytes to write (<= size of the buffer)
<i>pucData_</i>	Pointer to a data buffer containing the data to write

Returns

Number of bytes actually written

Implemented in [DevNull](#).

The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/driver.h](#)

16.27 DriverList Class Reference

List of [Driver](#) objects used to keep track of all device drivers in the system.

```
#include <driver.h>
```

Static Public Member Functions

- static void [Init](#) ()
Initialize the list of drivers.
- static void [Add](#) ([Driver](#) **pclDriver_*)
Add a [Driver](#) object to the managed global driver-list.
- static void [Remove](#) ([Driver](#) **pclDriver_*)
Remove a driver from the global driver list.
- static [Driver](#) * [FindByPath](#) (const K_CHAR **m_pcPath*)
Look-up a driver in the global driver-list based on its path.

Static Private Attributes

- static [DoubleLinkedList](#) [m_clDriverList](#)
LinkedList object used to implementing the driver object management.

16.27.1 Detailed Description

List of [Driver](#) objects used to keep track of all device drivers in the system.

By default, the list contains a single entity, "/dev/null".

Definition at line 244 of file [driver.h](#).

16.27.2 Member Function Documentation

16.27.2.1 DriverList::Add (Driver * *pclDriver_*) [inline],[static]

Add a [Driver](#) object to the managed global driver-list.

Parameters

<i>pclDriver_</i>	pointer to the driver object to add to the global driver list.
-------------------	--

Definition at line 264 of file [driver.h](#).

16.27.2.2 Driver * DriverList::FindByPath (const K_CHAR * *m_pcPath*) [static]

Look-up a driver in the global driver-list based on its path.

In the event that the driver is not found in the list, a pointer to the default "/dev/null" object is returned. In this way, unimplemented drivers are automatically stubbed out.

Definition at line 94 of file [driver.cpp](#).

16.27.2.3 void DriverList::Init () [static]

Initialize the list of drivers.

Must be called prior to using the device driver library.

Definition at line 85 of file [driver.cpp](#).

16.27.2.4 void DriverList::Remove (Driver * *pclDriver_*) [inline],[static]

Remove a driver from the global driver list.

Parameters

<i>pclDriver_</i>	Pointer to the driver object to remove from the global table
-------------------	--

Definition at line 274 of file [driver.h](#).

The documentation for this class was generated from the following files:

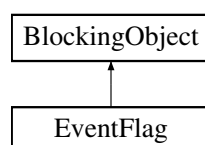
- [/home/mo/mark3-source/embedded/stage/src/driver.h](#)
- [/home/mo/mark3-source/embedded/stage/src/driver.cpp](#)

16.28 EventFlag Class Reference

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

```
#include <eventflag.h>
```

Inheritance diagram for EventFlag:



Public Member Functions

- void [Init](#) ()
Init Initializes the [EventFlag](#) object prior to use.
- K_USHORT [Wait](#) (K_USHORT usMask_, EventFlagOperation_t eMode_)
Wait - Block a thread on the specific flags in this event flag group.
- K_USHORT [Wait](#) (K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_)
Wait - Block a thread on the specific flags in this event flag group.
- void [Timeout](#) ([Thread](#) *pclOwner_)
- void [Set](#) (K_USHORT usMask_)
Set - Set additional flags in this object (logical OR).
- void [Clear](#) (K_USHORT usMask_)
ClearFlags - Clear a specific set of flags within this object, specific by bitmask.
- K_USHORT [GetMask](#) ()
GetMask Returns the state of the 16-bit bitmask within this object.

Private Member Functions

- K_BOOL [ProcessQueue](#) ()
ProcessQueue.
- void [WaitTransaction](#) ([Transaction](#) *pclTRX_, K_BOOL *pbReschedule_)
WaitTransaction.
- void [SetTransaction](#) ([Transaction](#) *pclTRX_, K_BOOL *pbReschedule_)
SetTransaction.
- void [ClearTransaction](#) ([Transaction](#) *pclTRX_, K_BOOL *pbReschedule_)
ClearTransaction.
- void [TimeoutTransaction](#) ([Transaction](#) *pclTRX_, K_BOOL *pbReschedule_)
TimeoutTransaction.

Private Attributes

- K_USHORT [m_usSetMask](#)
Currently set bits in the event mask.

Additional Inherited Members

16.28.1 Detailed Description

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

Each [EventFlag](#) object contains a 16-bit bitmask, which is used to trigger events on associated threads. Threads wishing to block, waiting for a specific event to occur can wait on any pattern within this 16-bit bitmask to be set. Here, we provide the ability for a thread to block, waiting for ANY bits in a specified mask to be set, or for ALL bits within a specific mask to be set. Depending on how the object is configured, the bits that triggered the wakeup can be automatically cleared once a match has occurred.

Definition at line 47 of file [eventflag.h](#).

16.28.2 Member Function Documentation

16.28.2.1 void EventFlag::Clear (K_USHORT usMask_)

ClearFlags - Clear a specific set of flags within this object, specific by bitmask.

Parameters

<i>usMask_</i>	- Bitmask of flags to clear
----------------	-----------------------------

Definition at line 368 of file [eventflag.cpp](#).

16.28.2.2 void EventFlag::ClearTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

ClearTransaction.

Clear event flags synchronously, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 326 of file [eventflag.cpp](#).

16.28.2.3 K_USHORT EventFlag::GetMask ()

GetMask Returns the state of the 16-bit bitmask within this object.

Returns

The state of the 16-bit bitmask

Definition at line 386 of file [eventflag.cpp](#).

16.28.2.4 K_BOOL EventFlag::ProcessQueue () [private]

ProcessQueue.

Process the kernel transaction queue associated with this event-flag object. This executes all queued accesses in first-in first-out order, ensuring that state is preserved and results are deterministic. When this function returns, the event flag object is no longer blocked.

Returns

true - the sheduler must be re-run when enabled. false - the scheduler does not need to be re-run.

Definition at line 124 of file [eventflag.cpp](#).

16.28.2.5 void EventFlag::Set (K_USHORT *usMask_*)

Set - Set additional flags in this object (logical OR).

This API can potentially result in threads blocked on [Wait\(\)](#) to be unblocked.

Parameters

<i>usMask_</i>	- Bitmask of flags to set.
----------------	----------------------------

Definition at line 350 of file [eventflag.cpp](#).

16.28.2.6 void EventFlag::SetTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

SetTransaction.

Set an event-flag mask in a synchronous operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 219 of file [eventflag.cpp](#).

16.28.2.7 void EventFlag::TimeoutTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

TimeoutTransaction.

Perform an event flag "timeout" operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 333 of file [eventflag.cpp](#).

16.28.2.8 K_USHORT EventFlag::Wait (K_USHORT *usMask_*, EventFlagOperation_t *eMode_*)

Wait - Block a thread on the specific flags in this event flag group.

Parameters

<i>usMask_</i>	- 16-bit bitmask to block on
<i>eMode_</i>	- EVENT_FLAG_ANY: Thread will block on any of the bits in the mask <ul style="list-style-type: none"> • EVENT_FLAG_ALL: Thread will block on all of the bits in the mask

Returns

Bitmask condition that caused the thread to unblock, or 0 on error or timeout

Definition at line 73 of file [eventflag.cpp](#).

16.28.2.9 K_USHORT EventFlag::Wait (K_USHORT *usMask_*, EventFlagOperation_t *eMode_*, K_ULONG *ulTimeMS_*)

Wait - Block a thread on the specific flags in this event flag group.

Parameters

<i>usMask_</i>	- 16-bit bitmask to block on
<i>eMode_</i>	- EVENT_FLAG_ANY: Thread will block on any of the bits in the mask <ul style="list-style-type: none"> • EVENT_FLAG_ALL: Thread will block on all of the bits in the mask
<i>ulTimeMS_</i>	- Time to block (in ms)

Returns

Bitmask condition that caused the thread to unblock, or 0 on error or timeout

! If the Yield operation causes a new thread to be chosen, there will ! Be a context switch at the above SetScheduler() call. The original calling ! thread will not return back until a matching SetFlags call is made.

Definition at line 77 of file [eventflag.cpp](#).

16.28.2.10 void EventFlag::WaitTransaction (Transaction * *pcITRX_*, K_BOOL * *pbReschedule_*) [private]

WaitTransaction.

Perform a synchronous even-flag blocking operation, as specified from an object on the transaction queue.

Parameters

<i>pcITRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 160 of file [eventflag.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/eventflag.h](#)
- [/home/mo/mark3-source/embedded/stage/src/eventflag.cpp](#)

16.29 FixedHeap Class Reference

Fixed-size-block heap allocator with multiple block sizes.

```
#include <fixed_heap.h>
```

Public Member Functions

- void [Create](#) (void *pvHeap_, [HeapConfig](#) *pcITHeapConfig_)
Creates a heap in a provided blob of memory with lists of fixed-size blocks configured based on the associated configuration data.
- void * [Alloc](#) (K_USHORT usSize_)
Allocate a blob of memory from the heap.

Static Public Member Functions

- static void [Free](#) (void *pvNode_)
Free a previously-allocated block of memory to the heap it was originally allocated from.

Private Attributes

- [HeapConfig](#) * [m_pacITHeaps](#)
Pointer to the configuration data used by the heap.

16.29.1 Detailed Description

Fixed-size-block heap allocator with multiple block sizes.

Definition at line 104 of file [fixed_heap.h](#).

16.29.2 Member Function Documentation

16.29.2.1 void * FixedHeap::Alloc (K_USHORT usSize_)

Allocate a blob of memory from the heap.

If no appropriately-sized data block is available, will return NULL. Note, this API is thread- safe, and interrupt safe.

Parameters

<i>usSize_</i>	Size (in bytes) to allocate from the heap
----------------	---

Returns

Pointer to a block of data allocated, or 0 on error.

Definition at line 130 of file [fixed_heap.cpp](#).

16.29.2.2 void FixedHeap::Create (void * *pvHeap_*, HeapConfig * *pclHeapConfig_*)

Creates a heap in a provided blob of memory with lists of fixed-size blocks configured based on the associated configuration data.

A heap must be created before it can be allocated/freed.

Parameters

<i>pvHeap_</i>	Pointer to the data blob that will contain the heap
<i>pclHeapConfig_</i>	Pointer to the array of config objects that define how the heap is laid out in memory, and how many blocks of what size are included. The objects in the array must be initialized, starting from smallest block-size to largest, with the final entry in the table have a 0-block size, indicating end-of-configuration.

Definition at line 113 of file [fixed_heap.cpp](#).

16.29.2.3 void FixedHeap::Free (void * *pvNode_*) [static]

Free a previously-allocated block of memory to the heap it was originally allocated from.

This must point to the block of memory at its originally-returned pointer, and not an address within an allocated blob (as supported by some allocators).

Parameters

<i>pvNode_</i>	Pointer to the previously-allocated block of memory
----------------	---

Definition at line 160 of file [fixed_heap.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/fixed_heap.h](#)
- [/home/mo/mark3-source/embedded/stage/src/fixed_heap.cpp](#)

16.30 Font_t Struct Reference

Public Attributes

- K_UCHAR **ucSize**
- K_UCHAR **ucFlags**
- K_UCHAR **ucStartChar**
- K_UCHAR **ucMaxChar**
- const K_CHAR * **szName**
- const FONT_STORAGE_TYPE * **pucFontData**

16.30.1 Detailed Description

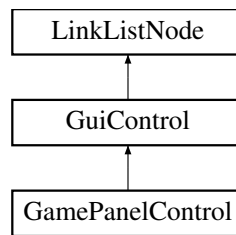
Definition at line 43 of file [font.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/font.h](#)

16.31 GamePanelControl Class Reference

Inheritance diagram for GamePanelControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.

Private Attributes

- [JoystickEvent_t m_stLastJoy](#)
- [JoystickEvent_t m_stCurrentJoy](#)

Additional Inherited Members

16.31.1 Detailed Description

Definition at line 32 of file [control_gamepanel.h](#).

16.31.2 Member Function Documentation

16.31.2.1 virtual void [GamePanelControl::Activate](#) (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<code>bActivate_</code>	- true to activate, false to deactivate
-------------------------	---

Implements [GuiControl](#).

Definition at line 38 of file [control_gamepanel.h](#).

16.31.2.2 void GamePanelControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 27 of file [control_gamepanel.cpp](#).

16.31.2.3 virtual void GamePanelControl::Init () [inline],[virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 35 of file [control_gamepanel.h](#).

16.31.2.4 GuiReturn_t GamePanelControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<code>pstEvent_</code>	Pointer to a struct containing the event data
------------------------	---

Implements [GuiControl](#).

Definition at line 33 of file [control_gamepanel.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_gamepanel.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_gamepanel.cpp](#)

16.32 GlobalMessagePool Class Reference

Implements a list of message objects shared between all threads.

```
#include <message.h>
```

Static Public Member Functions

- static void [Init](#) ()
Initialize the message queue prior to use.
- static void [Push](#) ([Message](#) *pclMessage_)
Return a previously-claimed message object back to the global queue.
- static [Message](#) * [Pop](#) ()
Pop a message from the global queue, returning it to the user to be populated before sending by a transmitter.

Static Private Attributes

- static [Message m_aclMessagePool](#) [GLOBAL_MESSAGE_POOL_SIZE]
Array of message objects that make up the message pool.
- static [DoubleLinkedList m_clList](#)
Linked list used to manage the [Message](#) objects.

16.32.1 Detailed Description

Implements a list of message objects shared between all threads.

Definition at line 157 of file [message.h](#).

16.32.2 Member Function Documentation

16.32.2.1 [Message * GlobalMessagePool::Pop \(\)](#) [static]

Pop a message from the global queue, returning it to the user to be populated before sending by a transmitter.

Returns

Pointer to a [Message](#) object

Definition at line 69 of file [message.cpp](#).

16.32.2.2 [void GlobalMessagePool::Push \(Message * pclMessage_ \)](#) [static]

Return a previously-claimed message object back to the global queue.

Used once the message has been processed by a receiver.

Parameters

pclMessage_	Pointer to the Message object to return back to the global queue
-----------------------------	--

Definition at line 57 of file [message.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/message.h](#)
- [/home/mo/mark3-source/embedded/stage/src/message.cpp](#)

16.33 Glyph_t Struct Reference

Public Attributes

- K_UCHAR [ucWidth](#)
Width of this font glyph in pixels.
- K_UCHAR [ucHeight](#)
Height of this font glyph in pixels.
- K_UCHAR [ucVOffset](#)
Vertical offset of this glyph.
- K_UCHAR [aucData](#) [1]
Glyph data array.

16.33.1 Detailed Description

Definition at line 26 of file [font.h](#).

The documentation for this struct was generated from the following file:

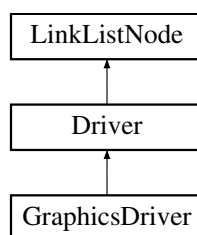
- [/home/mo/mark3-source/embedded/stage/src/font.h](#)

16.34 GraphicsDriver Class Reference

Defines the base graphics driver class, which is inherited by all other graphics drivers.

```
#include <graphics.h>
```

Inheritance diagram for GraphicsDriver:



Public Member Functions

- virtual void [DrawPixel](#) ([DrawPoint_t](#) *pstPoint_)
Draw a single pixel to the display.
- virtual void [ReadPixel](#) ([DrawPoint_t](#) *pstPoint_)
Read a single pixel from the display.
- virtual void [ClearScreen](#) ()
Clear the screen (initializes to all black pixels)
- virtual void [Point](#) ([DrawPoint_t](#) *pstPoint_)
Draw a pixel to the display.
- virtual void [Line](#) ([DrawLine_t](#) *pstLine_)
Draw a line to the display using Bresenham's line drawing algorithm.
- virtual void [Rectangle](#) ([DrawRectangle_t](#) *pstRectangle_)
Draws a rectangle on the display.
- virtual void [Circle](#) ([DrawCircle_t](#) *pstCircle_)
Draw a circle to the display.
- virtual void [Ellipse](#) ([DrawEllipse_t](#) *pstEllipse_)
Draw an ellipse to the display.
- virtual void [Bitmap](#) ([DrawBitmap_t](#) *pstBitmap_)
Draw an RGB image on the display.
- virtual void [Stamp](#) ([DrawStamp_t](#) *pstStamp_)
Draws a stamp (a 1-bit bitmap) on the display.
- virtual void [Move](#) ([DrawMove_t](#) *pstMove_)
Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.
- virtual void [TriangleWire](#) ([DrawPoly_t](#) *pstPoly_)
Draw a wireframe triangle to the display.
- virtual void [TriangleFill](#) ([DrawPoly_t](#) *pstPoly_)
Draw a filled triangle to the display.

- virtual void **Polygon** ([DrawPoly_t](#) *pstPoly_)
- virtual void **Text** ([DrawText_t](#) *pstText_)
 - Draw a string of text to the display using a bitmap font.*
- void **TextFX** ([DrawText_t](#) *pstText_, [TextFX_t](#) *pstFX_)
- virtual K_USHORT **TextWidth** ([DrawText_t](#) *pstText_)
- void **SetWindow** ([DrawWindow_t](#) *pstWindow_)
 - Set the drawable window of the screen.*
- void **ClearWindow** ()
 - Clear the window - resetting the boundaries to the entire drawable area of the screen.*

Protected Attributes

- K_USHORT **m_usResX**
- K_USHORT **m_usResY**
- K_USHORT **m_usLeft**
- K_USHORT **m_usTop**
- K_USHORT **m_usRight**
- K_USHORT **m_usBottom**
- K_UCHAR **m_ucBPP**

Additional Inherited Members

16.34.1 Detailed Description

Defines the base graphics driver class, which is inherited by all other graphics drivers.

Per-pixel rendering functions for all raster operations is provided by default. These can be overridden with more efficient hardware-supported operations where available.

Definition at line 32 of file [graphics.h](#).

16.34.2 Member Function Documentation

16.34.2.1 void GraphicsDriver::Bitmap ([DrawBitmap_t](#) * *pstBitmap_*) [virtual]

Draw an RGB image on the display.

Parameters

<i>pstBitmap_</i>	- pointer to the bitmap object to display
-------------------	---

Definition at line 302 of file [graphics.cpp](#).

16.34.2.2 void GraphicsDriver::Circle ([DrawCircle_t](#) * *pstCircle_*) [virtual]

Draw a circle to the display.

Parameters

<i>pstCircle_</i>	- pointer to the circle to draw
-------------------	---------------------------------

Definition at line 178 of file [graphics.cpp](#).

16.34.2.3 void GraphicsDriver::DrawPixel ([DrawPoint_t](#) * *pstPoint_*) [inline], [virtual]

Draw a single pixel to the display.

Parameters

<i>pstPoint_</i>	Structure containing the pixel data (color/location) to be written.
------------------	---

Definition at line 49 of file [graphics.h](#).

16.34.2.4 void GraphicsDriver::Ellipse (DrawEllipse_t * *pstEllipse_*) [virtual]

Draw an ellipse to the display.

Parameters

<i>pstEllipse_</i>	- pointer to the ellipse to draw on the display
--------------------	---

Definition at line 250 of file [graphics.cpp](#).

16.34.2.5 void GraphicsDriver::Line (DrawLine_t * *pstLine_*) [virtual]

Draw a line to the display using Bresenham's line drawing algorithm.

Parameters

<i>pstLine_</i>	- pointer to the line structure
-----------------	---------------------------------

Definition at line 50 of file [graphics.cpp](#).

16.34.2.6 void GraphicsDriver::Move (DrawMove_t * *pstMove_*) [virtual]

Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.

Parameters

<i>pstMove_</i>	- object describing the graphics movement operation (framebuffer operations only).
-----------------	--

Definition at line 440 of file [graphics.cpp](#).

16.34.2.7 void GraphicsDriver::Point (DrawPoint_t * *pstPoint_*) [virtual]

Draw a pixel to the display.

Parameters

<i>pstPoint_</i>	- pointer to the struct containing the pixel to draw
------------------	--

Definition at line 44 of file [graphics.cpp](#).

16.34.2.8 void GraphicsDriver::ReadPixel (DrawPoint_t * *pstPoint_*) [inline],[virtual]

Read a single pixel from the display.

Parameters

<i>pstPoint_</i>	Structure containing the pixel location of the pixel to be read. The color value will contain the value from the display when read.
------------------	---

Definition at line 58 of file [graphics.h](#).

16.34.2.9 void GraphicsDriver::Rectangle (DrawRectangle_t * *pstRectangle_*) [virtual]

Draws a rectangle on the display.

Parameters

<i>pstRectangle_</i>	- pointer to the rectangle struct
----------------------	-----------------------------------

Definition at line 133 of file [graphics.cpp](#).

16.34.2.10 void GraphicsDriver::SetWindow (DrawWindow_t * *pstWindow_*)

Set the drawable window of the screen.

Parameters

<i>pstWindow_</i>	- pointer to the window struct defining the drawable area
-------------------	---

Definition at line 1050 of file [graphics.cpp](#).

16.34.2.11 void GraphicsDriver::Stamp (DrawStamp_t * *pstStamp_*) [virtual]

Draws a stamp (a 1-bit bitmap) on the display.

Parameters

<i>pstStamp_</i>	- pointer to the stamp object to draw
------------------	---------------------------------------

Definition at line 401 of file [graphics.cpp](#).

16.34.2.12 void GraphicsDriver::Text (DrawText_t * *pstText_*) [virtual]

Draw a string of text to the display using a bitmap font.

Parameters

<i>pstText_</i>	- pointer to the text object to render
-----------------	--

Definition at line 501 of file [graphics.cpp](#).

16.34.2.13 GraphicsDriver::TextFX (DrawText_t * *pstText_*, TextFX_t * *pstFX_*)

Render a string of text to the display with effects.

Parameters

<i>pstText_</i>	- pointer to the text object to render
<i>pstFX_</i>	- struct defining special text formatting to apply

ToDo - Add rotation

Definition at line 589 of file [graphics.cpp](#).

16.34.2.14 void GraphicsDriver::TriangleFill (DrawPoly_t * *pstPoly_*) [virtual]

Draw a filled triangle to the display.

Parameters

<i>pstPoly_</i>	Pointer to the polygon to draw.
-----------------	---------------------------------

Definition at line 823 of file [graphics.cpp](#).

16.34.2.15 void GraphicsDriver::TriangleWire (DrawPoly_t * *pstPoly_*) [virtual]

Draw a wireframe triangle to the display.

Parameters

<i>pstPoly_</i>	Pointer to the polygon to draw.
-----------------	---------------------------------

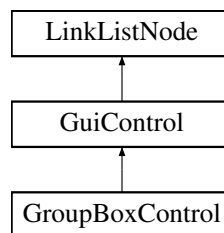
Definition at line 798 of file [graphics.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/graphics.h](#)
- [/home/mo/mark3-source/embedded/stage/src/graphics.cpp](#)

16.35 GroupBoxControl Class Reference

Inheritance diagram for GroupBoxControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetPanelColor](#) (COLOR eColor_)
- void [SetLineColor](#) (COLOR eColor_)
- void [SetFontColor](#) (COLOR eColor_)
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *pcCaption_)

Private Attributes

- COLOR [m_uPanelColor](#)
- COLOR [m_uLineColor](#)
- COLOR [m_uFontColor](#)
- [Font_t](#) * [m_pstFont](#)
- const K_CHAR * [m_pcCaption](#)

Additional Inherited Members

16.35.1 Detailed Description

Definition at line 29 of file [control_groupbox.h](#).

16.35.2 Member Function Documentation

16.35.2.1 `virtual void GroupBoxControl::Activate (bool bActivate_) [inline], [virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 38 of file [control_groupbox.h](#).

16.35.2.2 `void GroupBoxControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 30 of file [control_groupbox.cpp](#).

16.35.2.3 `virtual void GroupBoxControl::Init () [inline], [virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 32 of file [control_groupbox.h](#).

16.35.2.4 `virtual GuiReturn_t GroupBoxControl::ProcessEvent (GuiEvent_t * pstEvent_) [inline], [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 37 of file [control_groupbox.h](#).

The documentation for this class was generated from the following files:

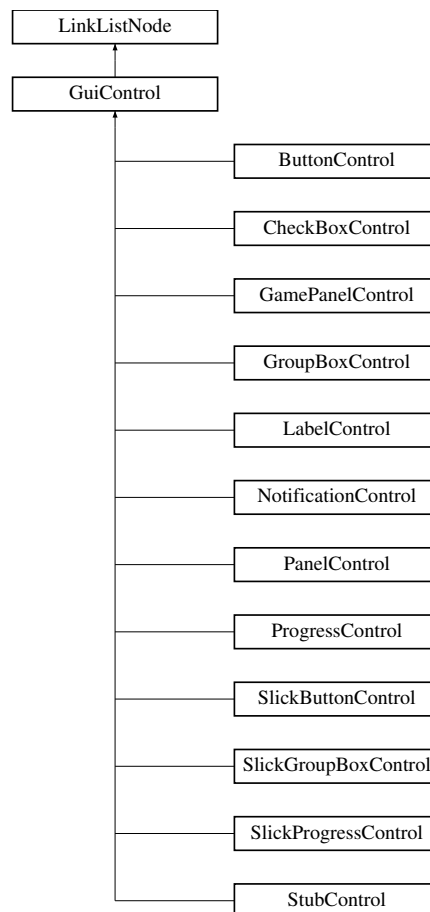
- [/home/mo/mark3-source/embedded/stage/src/control_groupbox.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_groupbox.cpp](#)

16.36 GuiControl Class Reference

GUI Control Base Class.

```
#include <gui.h>
```

Inheritance diagram for GuiControl:



Public Member Functions

- virtual void **Init** ()=0
Initialize the control - must be called before use.
- virtual void **Draw** ()=0
Redraw the control "cleanly".
- virtual **GuiReturn_t** **ProcessEvent** (**GuiEvent_t** *pstEvent_)=0
Process an event sent to the control.
- void **SetTop** (**K_USHORT** usTop_)
Set the location of the topmost pixel of the control.
- void **SetLeft** (**K_USHORT** usLeft_)
Set the location of the leftmost pixel of the control.
- void **SetHeight** (**K_USHORT** usHeight_)
Set the height of the control (in pixels)
- void **SetWidth** (**K_USHORT** usWidth_)
Set the width of the control (in pixels)
- void **SetZOrder** (**K_UCHAR** ucZ_)
Set the Z-order (depth) of the control.
- void **SetControlIndex** (**K_UCHAR** ucIdx_)
Set the index of the control, used for cycling through focus (ala tab order in VB).
- **K_USHORT** **GetTop** ()
Return the topmost pixel of the control.
- **K_USHORT** **GetLeft** ()
Return the leftmost pixel of the control.

- K_USHORT [GetHeight](#) ()
Get the height of the control in pixels.
- K_USHORT [GetWidth](#) ()
Get the width of the control in pixels.
- K_UCHAR [GetZOrder](#) ()
Return the Z-order of the control.
- K_UCHAR [GetControllIndex](#) ()
Return the Control Index of the control.
- K_BOOL [IsStale](#) ()
Return whether or not the control needs to be redrawn or not.
- void [GetControlOffset](#) (K_USHORT *pusX_, K_USHORT *pusY_)
Return the absolute offset of the control within an event surface.
- K_BOOL [IsInFocus](#) ()
Return whether or not the current control has the focus in the window.
- virtual void [Activate](#) (bool bActivate_)=0
Activate or deactivate the current control - used when switching from one active control to another.

Protected Member Functions

- void [SetParentControl](#) (GuiControl *pclParent_)
Set the parent control of this control.
- void [SetParentWindow](#) (GuiWindow *pclWindow_)
Set the parent window of this control.
- GuiControl * [GetParentControl](#) ()
Return the pointer to the control's currently-assigned parent control.
- GuiWindow * [GetParentWindow](#) ()
Get the parent window of this control.
- void [ClearStale](#) ()
Clear the stale flag for this control.
- void [SetStale](#) ()
Signal that the object needs to be redrawn.
- void [SetAcceptFocus](#) (bool bFocus_)
Tell the control whether or not to accept focus.
- bool [AcceptsFocus](#) ()
Returns whether or not this control accepts focus.

Private Attributes

- K_BOOL [m_bStale](#)
true if the control is stale and needs to be redrawn, false otherwise
- K_BOOL [m_bAcceptsFocus](#)
Whether or not the control accepts focus or not.
- K_UCHAR [m_ucZOrder](#)
The Z-Order (depth) of the control.
- K_UCHAR [m_ucControllIndex](#)
Index of the control in the window.
- K_USHORT [m_usTop](#)
Topmost location of the control on the window.
- K_USHORT [m_usLeft](#)
Leftmost location of the control on the window.

- `K_USHORT m_usWidth`
Width of the control in pixels.
- `K_USHORT m_usHeight`
Height of the control in pixels.
- `GuiControl * m_pclParentControl`
Pointer to the parent control.
- `GuiWindow * m_pclParentWindow`
Pointer to the parent window associated with this control.

Friends

- class **GuiWindow**
- class **GuiEventSurface**

Additional Inherited Members

16.36.1 Detailed Description

GUI Control Base Class.

This class is the common ancestor to all GUI control elements. It defines a base set of properties common to all controls, as well as methods for initialization, event handling, and redrawing. Controls are directly related to Windows, which are used to manage and organize controls.

Definition at line 538 of file [gui.h](#).

16.36.2 Member Function Documentation

16.36.2.1 `void GuiControl::Activate (bool bActivate_) [pure virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<code>bActivate_</code>	- true to activate, false to deactivate
-------------------------	---

Implemented in [StubControl](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

16.36.2.2 `void GuiControl::ClearStale () [inline],[protected]`

Clear the stale flag for this control.

Should only be done after a redraw has been completed

Definition at line 741 of file [gui.h](#).

16.36.2.3 `void GuiControl::Draw () [pure virtual]`

Redraw the control "cleanly".

Subclass specific.

Implemented in [StubControl](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

16.36.2.4 `K_UCHAR GuiControl::GetControlIndex () [inline]`

Return the Control Index of the control.

Returns

The control index of the control

Definition at line 648 of file [gui.h](#).

16.36.2.5 `void GuiControl::GetControlOffset (K_USHORT * pusX_, K_USHORT * pusY_)`

Return the absolute offset of the control within an event surface.

This function will traverse through all of the object's parents, and their parents, until the root control and root window are identified. The absolute pixel locations of the Topmost (Y) and Leftmost (X) pixels are populated in the

Parameters

<i>pusX_</i>	Pointer to the K_USHORT containing the leftmost pixel
<i>pusY_</i>	Pointer to the K_USHORT containing the topmost pixel

Definition at line 669 of file [gui.cpp](#).

16.36.2.6 `K_USHORT GuiControl::GetHeight () [inline]`

Get the height of the control in pixels.

Returns

Height of the control in pixels

Definition at line 627 of file [gui.h](#).

16.36.2.7 `K_USHORT GuiControl::GetLeft () [inline]`

Return the leftmost pixel of the control.

Returns

Leftmost pixel of the control

Definition at line 620 of file [gui.h](#).

16.36.2.8 `GuiControl * GuiControl::GetParentControl () [inline], [protected]`

Return the pointer to the control's currently-assigned parent control.

Returns

Pointer to the Control's currently assigned parent control.

Definition at line 725 of file [gui.h](#).

16.36.2.9 GuiWindow * GuiControl::GetParentWindow () [inline], [protected]

Get the parent window of this control.

Returns

Pointer to the control's window

Definition at line 733 of file [gui.h](#).

16.36.2.10 K_USHORT GuiControl::GetTop () [inline]

Return the topmost pixel of the control.

Returns

Topmost pixel of the control

Definition at line 613 of file [gui.h](#).

16.36.2.11 K_USHORT GuiControl::GetWidth () [inline]

Get the width of the control in pixels.

Returns

Width of the control in pixels

Definition at line 634 of file [gui.h](#).

16.36.2.12 K_UCHAR GuiControl::GetZOrder () [inline]

Return the Z-order of the control.

Returns

Z-order of the control

Definition at line 641 of file [gui.h](#).

16.36.2.13 void GuiControl::Init () [pure virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implemented in [StubControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [LabelControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), [GroupBoxControl](#), [NotificationControl](#), and [SlickGroupBoxControl](#).

16.36.2.14 K_BOOL GuiControl::IsInFocus () [inline]

Return whether or not the current control has the focus in the window.

Returns

true if this control is in focus, false otherwise

Definition at line 677 of file [gui.h](#).

16.36.2.15 `K_BOOL GuiControl::IsStale () [inline]`

Return whether or not the control needs to be redrawn or not.

Returns

true - control needs redrawing, false - control is intact.

Definition at line 655 of file [gui.h](#).

16.36.2.16 `GuiReturn_t GuiControl::ProcessEvent (GuiEvent_t * pstEvent_) [pure virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implemented in [StubControl](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

16.36.2.17 `void GuiControl::SetControllIndex (K_UCHAR ucIdx_) [inline]`

Set the index of the control, used for cycling through focus (ala tab order in VB).

Parameters

<i>ucIdx_</i>	Focus index of the control
---------------	----------------------------

Definition at line 606 of file [gui.h](#).

16.36.2.18 `void GuiControl::SetHeight (K_USHORT usHeight_) [inline]`

Set the height of the control (in pixels)

Parameters

<i>usHeight_</i>	Height of the control in pixels
------------------	---------------------------------

Definition at line 584 of file [gui.h](#).

16.36.2.19 `void GuiControl::SetLeft (K_USHORT usLeft_) [inline]`

Set the location of the leftmost pixel of the control.

Parameters

<i>usLeft_</i>	Leftmost pixel of the control
----------------	-------------------------------

Definition at line 577 of file [gui.h](#).

16.36.2.20 `void GuiControl::SetParentControl (GuiControl * pclParent_) [inline], [protected]`

Set the parent control of this control.

When a control has its parent set, it is considered "nested" within that control. Moving the control will thus result in all of its child controls to become invalidated, thus requiring redraws. The control's object offsets (Top, Bottom, Height, and Width) also become relative to the origin of the parent control.

Parameters

<i>pciParent_</i>	Pointer to the control's parent control
-------------------	---

Definition at line 706 of file [gui.h](#).

16.36.2.21 void GuiControl::SetParentWindow (GuiWindow * *pciWindow_*) [inline], [protected]

Set the parent window of this control.

All controls within the same window are all associated together, and share events targetted towards a specific window. Event tabbing, focus, and Z-ordering is also shared between controls within a window.

Parameters

<i>pciWindow_</i>	Pointer to the control's parent window.
-------------------	---

Definition at line 717 of file [gui.h](#).

16.36.2.22 void GuiControl::SetTop (K_USHORT *usTop_*) [inline]

Set the location of the topmost pixel of the control.

Parameters

<i>usTop_</i>	Topmost pixel of the control
---------------	------------------------------

Definition at line 570 of file [gui.h](#).

16.36.2.23 void GuiControl::SetWidth (K_USHORT *usWidth_*) [inline]

Set the width of the control (in pixels)

Parameters

<i>usWidth_</i>	Width of the control in pixels
-----------------	--------------------------------

Definition at line 591 of file [gui.h](#).

16.36.2.24 void GuiControl::SetZOrder (K_UCHAR *ucZ_*) [inline]

Set the Z-order (depth) of the control.

Parameters

<i>ucZ_</i>	Z order of the control
-------------	------------------------

Definition at line 598 of file [gui.h](#).

16.36.3 Member Data Documentation

16.36.3.1 K_UCHAR GuiControl::m_ucControlIndex [private]

Index of the control in the window.

This is used for setting focus when transitioning from control to control on a window

Definition at line 770 of file [gui.h](#).

16.36.3.2 K_UCHAR GuiControl::m_ucZOrder [private]

The Z-Order (depth) of the control.

Only the highest order controls are visible at any given location

Definition at line 766 of file [gui.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)
- [/home/mo/mark3-source/embedded/stage/src/gui.cpp](#)

16.37 GuiEvent_t Struct Reference

Composite UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_UCHAR [ucEventType](#)
GuiEventType_t event type.
- K_UCHAR [ucTargetID](#)
Control index that this event is targeted towards.
- union {
 [KeyEvent_t stKey](#)
 Keyboard data.
 [MouseEvent_t stMouse](#)
 Mouse data.
 [TouchEvent_t stTouch](#)
 Touchscreen data.
 [JoystickEvent_t stJoystick](#)
 Joystick data.
 [TimerEvent_t stTimer](#)
 Timer data.
};

16.37.1 Detailed Description

Composite UI event structure.

Depending on the event type, can contain either a keyboard, mouse, touch, joystick, timer event, etc.

Definition at line 187 of file [gui.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.38 GuiEventSurface Class Reference

GUI Event Surface Object.

```
#include <gui.h>
```

Public Member Functions

- void [Init](#) ()
Initialize an event surface before use.
- void [AddWindow](#) ([GuiWindow](#) *pclWindow_)
Add a window to the event surface.
- void [RemoveWindow](#) ([GuiWindow](#) *pclWindow_)
Remove a window from the event surface.
- K_BOOL [SendEvent](#) ([GuiEvent_t](#) *pstEvent_)
Send an event to this window surface.
- K_BOOL [ProcessEvent](#) ()
Process an event in the event queue.
- K_UCHAR [GetEventCount](#) ()
Get the count of pending events in the event surface's queue.
- [GuiWindow](#) * [FindWindowByName](#) (const K_CHAR *szName_)
Return a pointer to a window by name, or NULL on failure.
- void [InvalidateRegion](#) (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)
Invalidate a region of the window, specified by the bounding box.

Private Member Functions

- void [CopyEvent](#) ([GuiEvent_t](#) *pstDst_, [GuiEvent_t](#) *pstSrc_)
Copy the contents of one message structure to another.

Private Attributes

- [DoubleLinkedList](#) [m_clWindowList](#)
List of windows managed on this event surface.
- [MessageQueue](#) [m_clMessageQueue](#)
Message queue used to manage window events.

16.38.1 Detailed Description

GUI Event Surface Object.

An event surface is the lowest-level UI object. It maintains a list of windows which are associated with it, and manages the transmission and routing of events to each window, and their appropriate controls

All windows located on the event surface are assumed to share a common display, and coordinate frame. In this way, multiple GUIs can be implemented in the system, each tied to separate physical or virtual displays.

Definition at line 452 of file [gui.h](#).

16.38.2 Member Function Documentation

16.38.2.1 void [GuiEventSurface::AddWindow](#) ([GuiWindow](#) * [pclWindow_](#))

Add a window to the event surface.

Parameters

<i>pciWindow_</i>	Pointer to the window object to add to the surface
-------------------	--

Definition at line 525 of file [gui.cpp](#).

16.38.2.2 void GuiEventSurface::CopyEvent (GuiEvent_t * *pstDst_*, GuiEvent_t * *pstSrc_*) [private]

Copy the contents of one message structure to another.

Parameters

<i>pstDst_</i>	Destination event pointer
<i>pstSrc_</i>	Source event pointer

Definition at line 645 of file [gui.cpp](#).

16.38.2.3 void GuiEventSurface::Init () [inline]

Initialize an event surface before use.

Must be called prior to any other object methods.

Definition at line 459 of file [gui.h](#).

16.38.2.4 void GuiEventSurface::InvalidateRegion (K_USHORT *usLeft_*, K_USHORT *usTop_*, K_USHORT *usWidth_*, K_USHORT *usHeight_*)

Invalidate a region of the window, specified by the bounding box.

The coordinates specified in the parameters (top and left) refer to absolute display coordinates, and are not relative to coordinates within a window.

Definition at line 658 of file [gui.cpp](#).

16.38.2.5 K_BOOL GuiEventSurface::ProcessEvent ()

Process an event in the event queue.

If no events are pending, the call will block until an event is available.

Definition at line 577 of file [gui.cpp](#).

16.38.2.6 void GuiEventSurface::RemoveWindow (GuiWindow * *pciWindow_*)

Remove a window from the event surface.

Parameters

<i>pciWindow_</i>	Pointer to the window object to remove from the surface
-------------------	---

Definition at line 533 of file [gui.cpp](#).

16.38.2.7 K_BOOL GuiEventSurface::SendEvent (GuiEvent_t * *pstEvent_*)

Send an event to this window surface.

The event will be forwarded to all windows managed by this service.

Parameters

<code>pstEvent_</code>	Pointer to an event to send
------------------------	-----------------------------

Returns

true on success, false on failure

Definition at line 541 of file [gui.cpp](#).

The documentation for this class was generated from the following files:

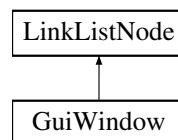
- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)
- [/home/mo/mark3-source/embedded/stage/src/gui.cpp](#)

16.39 GuiWindow Class Reference

Basic Window Class.

```
#include <gui.h>
```

Inheritance diagram for GuiWindow:



Public Member Functions

- void [Init](#) ()
Initialize the GUI Window object prior to use.
- void [SetDriver](#) ([GraphicsDriver](#) *pclDriver_)
Set the graphics driver to use for rendering controls on the window.
- [GraphicsDriver](#) * [GetDriver](#) ()
Set the graphics driver to use for rendering controls on the window.
- void [AddControl](#) ([GuiControl](#) *pclControl_, [GuiControl](#) *pclParent_)
Assign a GUI Control to this window object.
- void [RemoveControl](#) ([GuiControl](#) *pclControl_)
Removes a previously-added control from the Window.
- K_UCHAR [GetMaxZOrder](#) ()
Returns the highest Z-Order of all controls attached to this window.
- void [Redraw](#) (K_BOOL bRedrawAll_)
Redraw objects in the window.
- void [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to this window.
- void [SetFocus](#) ([GuiControl](#) *pclControl_)
Set the control used to accept "focus" events.
- K_BOOL [IsInFocus](#) ([GuiControl](#) *pclControl_)
Return whether or not the selected control is in focus or not.
- void [SetTop](#) (K_USHORT usTop_)
Set the location of the topmost pixel of the window.

- void [SetLeft](#) (K_USHORT usLeft_)
Set the location of the leftmost pixel of the window.
- void [SetHeight](#) (K_USHORT usHeight_)
Set the height of the window (in pixels)
- void [SetWidth](#) (K_USHORT usWidth_)
Set the width of the window (in pixels)
- K_USHORT [GetTop](#) ()
Return the topmost pixel of the window.
- K_USHORT [GetLeft](#) ()
Return the leftmost pixel of the window.
- K_USHORT [GetHeight](#) ()
Get the height of the window in pixels.
- K_USHORT [GetWidth](#) ()
Get the width of the window in pixels.
- K_UCHAR [GetZOrder](#) ()
Get the Z-order of the window on the event surface.
- void [SetZOrder](#) (K_UCHAR ucZ_)
Set the Z-order of the window on the event surface.
- void [CycleFocus](#) (bool bForward_)
Cycle the focus to the next active control in the window.
- void [SetName](#) (const K_CHAR *szName_)
Set the name for this window.
- const K_CHAR * [GetName](#) ()
Return the name of this window.
- void [InvalidateRegion](#) (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)
Invalidate a region of the window, specified by the bounding box.

Private Attributes

- K_USHORT [m_usTop](#)
Topmost pixel of the window on the event surface.
- K_USHORT [m_usLeft](#)
Leftmost pixel of the window on the event surface.
- K_USHORT [m_usHeight](#)
Height of the window in pixels.
- K_USHORT [m_usWidth](#)
Width of the window in pixels.
- K_UCHAR [m_ucZ](#)
Z-order of the window on the event surface.
- const K_CHAR * [m_szName](#)
Name applied to this window.
- [DoubleLinkedList](#) [m_clControlList](#)
List of controls managed by this window.
- [GuiControl](#) * [m_pclInFocus](#)
Pointer to the control in event focus.
- K_UCHAR [m_ucControlCount](#)
Number of controls in this window.
- [GraphicsDriver](#) * [m_pclDriver](#)
Graphics driver for this window.

Additional Inherited Members

16.39.1 Detailed Description

Basic Window Class.

A Window is loosely defined as a container of controls, all sharing a coordinate reference coordinate frame. Events are managed on a per-window basis, and each window is isolated from eachother.

Definition at line 223 of file [gui.h](#).

16.39.2 Member Function Documentation

16.39.2.1 GuiWindow::AddControl (*GuiControl* * *pclControl_*, *GuiControl* * *pclParent_*)

Assign a GUI Control to this window object.

Adding an object to a window ensures that the object will be drawn on the specific window surface, and ensures that events directed to this window will be forwarded to the controls appropriately.

Parameters

<i>pclControl_</i>	Pointer to the control object to add
<i>pclParent_</i>	Pointer to the control's "parent" object (or NULL)

Definition at line 27 of file [gui.cpp](#).

16.39.2.2 void GuiWindow::CycleFocus (*bool bForward_*)

Cycle the focus to the next active control in the window.

Parameters

<i>bForward_</i>	- Cycle to the next control when true, previous control when false
------------------	--

Definition at line 395 of file [gui.cpp](#).

16.39.2.3 *GraphicsDriver* * GuiWindow::GetDriver () [inline]

Set the graphics driver to use for rendering controls on the window.

Returns

Pointer to the Window's graphics driver

Definition at line 252 of file [gui.h](#).

16.39.2.4 *K_USHORT* GuiWindow::GetHeight () [inline]

Get the height of the window in pixels.

Returns

Height of the window in pixels

Definition at line 379 of file [gui.h](#).

16.39.2.5 `K_USHORT GuiWindow::GetLeft () [inline]`

Return the leftmost pixel of the window.

Returns

Leftmost pixel of the window

Definition at line 372 of file [gui.h](#).

16.39.2.6 `K_UCHAR GuiWindow::GetMaxZOrder ()`

Returns the highest Z-Order of all controls attached to this window.

Returns

The highest Z-Order used by controls in this window

Definition at line 61 of file [gui.cpp](#).

16.39.2.7 `K_USHORT GuiWindow::GetTop () [inline]`

Return the topmost pixel of the window.

Returns

Topmost pixel of the window

Definition at line 365 of file [gui.h](#).

16.39.2.8 `K_USHORT GuiWindow::GetWidth () [inline]`

Get the width of the window in pixels.

Returns

Width of the window in pixels

Definition at line 386 of file [gui.h](#).

16.39.2.9 `void GuiWindow::Init () [inline]`

Initialize the GUI Window object prior to use.

Must be called before calling other methods on this object

Definition at line 231 of file [gui.h](#).

16.39.2.10 `void GuiWindow::InvalidateRegion (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)`

Invalidate a region of the window, specified by the bounding box.

The coordinates specified in the parameters (top and left) refer to absolute display coordinates, and are not relative to coordinates within a window.

Definition at line 127 of file [gui.cpp](#).

16.39.2.11 K_BOOL GuiWindow::IsInFocus (**GuiControl** * *pcControl_*) [inline]

Return whether or not the selected control is in focus or not.

Parameters

<i>pclControl_</i>	Pointer to the control object to evaluate
--------------------	---

Returns

true - the selected control is the active control on the window false - otherwise

Definition at line 323 of file [gui.h](#).

16.39.2.12 void GuiWindow::ProcessEvent (GuiEvent_t * pstEvent_)

Process an event sent to this window.

This method handles all of the plumbing required to target the event towards specific controls, or all controls in the window depending on the event payload.

Definition at line 245 of file [gui.cpp](#).

16.39.2.13 void GuiWindow::Redraw (K_BOOL bRedrawAll_)

Redraw objects in the window.

Typically, only the affected controls will need to be redrawn, but in some cases (such as window initialization), the entire window will need to be redrawn cleanly. This behavior is defined by the value of the bRedrawAll_ parameter.

Definition at line 85 of file [gui.cpp](#).

16.39.2.14 GuiWindow::RemoveControl (GuiControl * pclControl_)

Removes a previously-added control from the Window.

Parameters

<i>pclControl_</i>	Pointer to the control object to remove
--------------------	---

Definition at line 40 of file [gui.cpp](#).

16.39.2.15 void GuiWindow::SetDriver (GraphicsDriver * pclDriver_) [inline]

Set the graphics driver to use for rendering controls on the window.

Parameters

<i>pclDriver_</i>	Pointer to the graphics driver
-------------------	--------------------------------

Definition at line 244 of file [gui.h](#).

16.39.2.16 void GuiWindow::SetFocus (GuiControl * pclControl_)

Set the control used to accept "focus" events.

Such events include keyboard events.

Parameters

<i>pciControl_</i>	Pointer to the control object to set focus on.
--------------------	--

Definition at line 387 of file [gui.cpp](#).

16.39.2.17 void GuiWindow::SetHeight (K_USHORT *usHeight_*) [inline]

Set the height of the window (in pixels)

Parameters

<i>usHeight_</i>	Height of the window in pixels
------------------	--------------------------------

Definition at line 351 of file [gui.h](#).

16.39.2.18 void GuiWindow::SetLeft (K_USHORT *usLeft_*) [inline]

Set the location of the leftmost pixel of the window.

Parameters

<i>usLeft_</i>	Leftmost pixel of the window
----------------	------------------------------

Definition at line 344 of file [gui.h](#).

16.39.2.19 void GuiWindow::SetTop (K_USHORT *usTop_*) [inline]

Set the location of the topmost pixel of the window.

Parameters

<i>usTop_</i>	Topmost pixel of the window
---------------	-----------------------------

Definition at line 337 of file [gui.h](#).

16.39.2.20 void GuiWindow::SetWidth (K_USHORT *usWidth_*) [inline]

Set the width of the window (in pixels)

Parameters

<i>usWidth_</i>	Width of the window in pixels
-----------------	-------------------------------

Definition at line 358 of file [gui.h](#).

16.39.3 Member Data Documentation

16.39.3.1 GraphicsDriver* GuiWindow::m_pciDriver [private]

Graphics driver for this window.

Definition at line 436 of file [gui.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)
- [/home/mo/mark3-source/embedded/stage/src/gui.cpp](#)

16.40 HeapConfig Class Reference

Heap configuration object.

```
#include <fixed_heap.h>
```

Public Attributes

- K_USHORT [m_usBlockSize](#)
Block size in bytes.
- K_USHORT [m_usBlockCount](#)
Number of blocks to create @ this size.

Protected Attributes

- [BlockHeap m_clHeap](#)
[BlockHeap](#) object used by the allocator.

Friends

- class **FixedHeap**

16.40.1 Detailed Description

Heap configuration object.

Definition at line 90 of file [fixed_heap.h](#).

The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/fixed_heap.h](#)

16.41 JoystickEvent_t Struct Reference

Joystick UI event structure.

```
#include <gui.h>
```

Public Attributes

- union {
 K_USHORT [usRawData](#)
 Raw joystick data.
 struct {
 unsigned int [bUp](#):1
 D-pad UP state.
 unsigned int [bDown](#):1
 D-pad DOWN state.
 unsigned int [bLeft](#):1
 D-pad LEFT state.
 unsigned int [bRight](#):1
 D-pad RIGHT state.
 unsigned int [bButton1](#):1

```

        Joystick Button1 state.
    unsigned int bButton2:1
        Joystick Button2 state.
    unsigned int bButton3:1
        Joystick Button3 state.
    unsigned int bButton4:1
        Joystick Button4 state.
    unsigned int bButton5:1
        Joystick Button5 state.
    unsigned int bButton6:1
        Joystick Button6 state.
    unsigned int bButton7:1
        Joystick Button7 state.
    unsigned int bButton8:1
        Joystick Button8 state.
    unsigned int bButton9:1
        Joystick Button9 state.
    unsigned int bButton10:1
        Joystick Button10 state.
    unsigned int bSelect:1
        Start button state.
    unsigned int bStart:1
        Select button state.
}
};

```

16.41.1 Detailed Description

Joystick UI event structure.

Definition at line 144 of file [gui.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.42 Kernel Class Reference

Class that encapsulates all of the kernel startup functions.

```
#include <kernel.h>
```

Static Public Member Functions

- static void [Init](#) (void)
Kernel Initialization Function, call before any other OS function.
- static void [Start](#) (void)
Start the kernel; function never returns.
- static bool [IsStarted](#) ()
IsStarted.
- static void [SetPanic](#) (panic_func_t pfPanic_)
SetPanic Set a function to be called when a kernel panic occurs, giving the user to determine the behavior when a catastrophic failure is observed.
- static bool [IsPanic](#) ()

IsPanic Returns whether or not the kernel is in a panic state.

- static void [Panic](#) (K_USHORT usCause_)

Panic Cause the kernel to enter its panic state.

Static Private Attributes

- static bool [m_bIsStarted](#)
true if kernel is running, false otherwise
- static bool [m_bIsPanic](#)
true if kernel is in panic state, false otherwise
- static panic_func_t [m_pfPanic](#)
user-set panic function

16.42.1 Detailed Description

Class that encapsulates all of the kernel startup functions.

Definition at line 42 of file [kernel.h](#).

16.42.2 Member Function Documentation

16.42.2.1 Kernel::Init (void) [static]

[Kernel](#) Initialization Function, call before any other OS function.

Initializes all global resources used by the operating system. This must be called before any other kernel function is invoked.

Definition at line 48 of file [kernel.cpp](#).

16.42.2.2 static bool Kernel::IsPanic () [inline],[static]

[IsPanic](#) Returns whether or not the kernel is in a panic state.

Returns

Whether or not the kernel is in a panic state

Definition at line 89 of file [kernel.h](#).

16.42.2.3 static bool Kernel::IsStarted () [inline],[static]

[IsStarted](#).

Returns

Whether or not the kernel has started - true = running, false = not started

Definition at line 74 of file [kernel.h](#).

16.42.2.4 void Kernel::Panic (K_USHORT usCause_) [static]

[Panic](#) Cause the kernel to enter its panic state.

Parameters

<i>usCause_</i>	Reason for the kernel panic
-----------------	-----------------------------

Definition at line 88 of file [kernel.cpp](#).

16.42.2.5 `static void Kernel::SetPanic (panic_func_t pfPanic_) [inline],[static]`

SetPanic Set a function to be called when a kernel panic occurs, giving the user to determine the behavior when a catastrophic failure is observed.

Parameters

<i>pfPanic_</i>	Panic function pointer
-----------------	------------------------

Definition at line 83 of file [kernel.h](#).

16.42.2.6 `Kernel::Start (void) [static]`

Start the kernel; function never returns.

Start the operating system kernel - the current execution context is cancelled, all kernel services are started, and the processor resumes execution at the entrypoint for the highest-priority thread.

You must have at least one thread added to the kernel before calling this function, otherwise the behavior is undefined.

Definition at line 78 of file [kernel.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/kernel.h](#)
- [/home/mo/mark3-source/embedded/stage/src/kernel.cpp](#)

16.43 KernelSWI Class Reference

Class providing the software-interrupt required for context-switching in the kernel.

```
#include <kernelswi.h>
```

Static Public Member Functions

- static void [Config](#) (void)
Configure the software interrupt - must be called before any other software interrupt functions are called.
- static void [Start](#) (void)
Enable ("Start") the software interrupt functionality.
- static void [Stop](#) (void)
Disable the software interrupt functionality.
- static void [Clear](#) (void)
Clear the software interrupt.
- static void [Trigger](#) (void)
Call the software interrupt.
- static K_UCHAR [DI](#) ()
Disable the SWI flag itself.
- static void [RI](#) (K_UCHAR bEnable_)
Restore the state of the SWI to the value specified.

16.43.1 Detailed Description

Class providing the software-interrupt required for context-switching in the kernel.

Definition at line 32 of file [kernelswi.h](#).

16.43.2 Member Function Documentation

16.43.2.1 K_UCHAR KernelSWI::DI () [static]

Disable the SWI flag itself.

Returns

previous status of the SWI, prior to the DI call

Definition at line 50 of file [kernelswi.cpp](#).

16.43.2.2 void KernelSWI::RI (K_UCHAR bEnable_) [static]

Restore the state of the SWI to the value specified.

Parameters

<i>bEnable_</i>	true - enable the SWI, false - disable SWI
-----------------	--

Definition at line 58 of file [kernelswi.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/kernelswi.h](#)
- [/home/mo/mark3-source/embedded/stage/src/kernelswi.cpp](#)

16.44 KernelTimer Class Reference

Hardware timer interface, used by all scheduling/timer subsystems.

```
#include <kerneltimer.h>
```

Static Public Member Functions

- static void [Config](#) (void)
Initializes the kernel timer before use.
- static void [Start](#) (void)
Starts the kernel time (must be configured first)
- static void [Stop](#) (void)
Shut down the kernel timer, used when no timers are scheduled.
- static K_UCHAR [DI](#) (void)
Disable the kernel timer's expiry interrupt.
- static void [RI](#) (K_UCHAR bEnable_)
Restore the state of the kernel timer's expiry interrupt.
- static void [EI](#) (void)
Enable the kernel timer's expiry interrupt.
- static K_ULONG [SubtractExpiry](#) (K_ULONG ulInterval_)
Subtract the specified number of ticks from the timer's expiry count register.

- static K_ULONG [TimeToExpiry](#) (void)
Returns the number of ticks remaining before the next timer expiry.
- static K_ULONG [SetExpiry](#) (K_ULONG ulInterval_)
Resets the kernel timer's expiry interval to the specified value.
- static K_ULONG [GetOvertime](#) (void)
Return the number of ticks that have elapsed since the last expiry.
- static void [ClearExpiry](#) (void)
Clear the hardware timer expiry register.

Static Private Member Functions

- static K_USHORT [Read](#) (void)
Safely read the current value in the timer register.

16.44.1 Detailed Description

Hardware timer interface, used by all scheduling/timer subsystems.

Definition at line 33 of file [kerneltimer.h](#).

16.44.2 Member Function Documentation

16.44.2.1 K_ULONG KernelTimer::GetOvertime (void) [static]

Return the number of ticks that have elapsed since the last expiry.

Returns

Number of ticks that have elapsed after last timer expiration

Definition at line 115 of file [kerneltimer.cpp](#).

16.44.2.2 K_USHORT KernelTimer::Read (void) [static],[private]

Safely read the current value in the timer register.

Returns

Value held in the timer register

Definition at line 66 of file [kerneltimer.cpp](#).

16.44.2.3 void KernelTimer::RI (K_UCHAR bEnable_) [static]

Retstore the state of the kernel timer's expiry interrupt.

Parameters

<i>bEnable_</i>	1 enable, 0 disable
-----------------	---------------------

Definition at line 168 of file [kerneltimer.cpp](#).

16.44.2.4 K_ULONG KernelTimer::SetExpiry (K_ULONG ulInterval_) [static]

Resets the kernel timer's expiry interval to the specified value.

Parameters

<i>ulInterval_</i>	Desired interval in ticks to set the timer for
--------------------	--

Returns

Actual number of ticks set (may be less than desired)

Definition at line 121 of file [kerneltimer.cpp](#).

16.44.2.5 K_ULONG KernelTimer::SubtractExpiry (K_ULONG *ulInterval_*) [static]

Subtract the specified number of ticks from the timer's expiry count register.

Returns the new expiry value stored in the register.

Parameters

<i>ulInterval_</i>	Time (in HW-specific) ticks to subtract
--------------------	---

Returns

Value in ticks stored in the timer's expiry register

Definition at line 84 of file [kerneltimer.cpp](#).

16.44.2.6 K_ULONG KernelTimer::TimeToExpiry (void) [static]

Returns the number of ticks remaining before the next timer expiry.

Returns

Time before next expiry in platform-specific ticks

Definition at line 95 of file [kerneltimer.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/kerneltimer.h](#)
- [/home/mo/mark3-source/embedded/stage/src/kerneltimer.cpp](#)

16.45 KeyEvent_t Struct Reference

Keyboard UI event structure definition.

```
#include <gui.h>
```

Public Attributes

- K_UCHAR [ucKeyCode](#)
8-bit value representing a keyboard scan code
- union {
 K_UCHAR [ucFlags](#)
 Flags indicating modifiers to the event.
 struct {
 unsigned int [bKeyState](#):1

```

        Key is being pressed or released.
        unsigned int bShiftState:1
            Whether or not shift is pressed.
        unsigned int bCtrlState:1
            Whether or not CTRL is pressed.
        unsigned int bAltState:1
            Whether or not ALT it pressed.
        unsigned int bWinState:1
            Whether or not the Window/Clover key is pressed.
        unsigned int bFnState:1
            Whether or not a special function key is pressed.
    }
};

```

16.45.1 Detailed Description

Keyboard UI event structure definition.

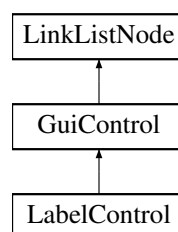
Definition at line 80 of file [gui.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.46 LabelControl Class Reference

Inheritance diagram for LabelControl:



Public Member Functions

- virtual void [Init](#) ()

Initialize the control - must be called before use.
- virtual void [Draw](#) ()

Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
- virtual void [Activate](#) (bool bActivate_)
- void [SetBackColor](#) (COLOR eColor_)
- void [SetFontColor](#) (COLOR eColor_)
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *pcData_)

Private Attributes

- [Font_t](#) * **m_pstFont**
- const K_CHAR * **m_pcCaption**
- COLOR **m_uBackColor**
- COLOR **m_uFontColor**

Additional Inherited Members

16.46.1 Detailed Description

Definition at line 30 of file [control_label.h](#).

16.46.2 Member Function Documentation

16.46.2.1 virtual void LabelControl::Activate (bool *bActivate_*) [inline],[virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 40 of file [control_label.h](#).

16.46.2.2 void LabelControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 26 of file [control_label.cpp](#).

16.46.2.3 virtual void LabelControl::Init () [inline],[virtual]

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 33 of file [control_label.h](#).

16.46.2.4 virtual GuiReturn_t LabelControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [inline],[virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 39 of file [control_label.h](#).

The documentation for this class was generated from the following files:

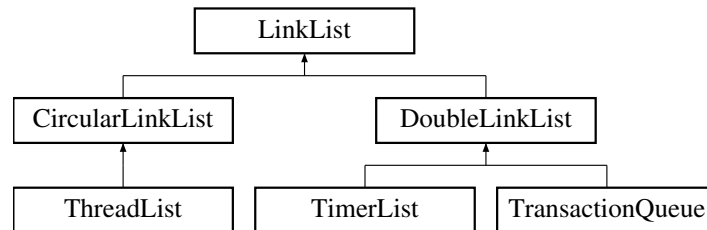
- [/home/mo/mark3-source/embedded/stage/src/control_label.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_label.cpp](#)

16.47 LinkList Class Reference

Abstract-data-type from which all other linked-lists are derived.

```
#include <ll.h>
```

Inheritance diagram for LinkList:



Public Member Functions

- void [Init](#) ()
Clear the linked list.
- virtual void [Add](#) ([LinkListNode](#) *node_)=0
Add the linked list node to this linked list.
- virtual void [Remove](#) ([LinkListNode](#) *node_)=0
Add the linked list node to this linked list.
- [LinkListNode](#) * [GetHead](#) ()
Get the head node in the linked list.
- [LinkListNode](#) * [GetTail](#) ()
Get the tail node of the linked list.

Protected Attributes

- [LinkListNode](#) * [m_pstHead](#)
Pointer to the head node in the list.
- [LinkListNode](#) * [m_pstTail](#)
Pointer to the tail node in the list.

16.47.1 Detailed Description

Abstract-data-type from which all other linked-lists are derived.

Definition at line [113](#) of file [ll.h](#).

16.47.2 Member Function Documentation

16.47.2.1 void [LinkList::Add](#) ([LinkListNode](#) * node_) [pure virtual]

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to add
--------------	----------------------------

Implemented in [CircularLinkedList](#), [DoubleLinkedList](#), and [ThreadList](#).

16.47.2.2 `LinkedListNode * LinkedList::GetHead () [inline]`

Get the head node in the linked list.

Returns

Pointer to the head node in the list

Definition at line [150](#) of file [ll.h](#).

16.47.2.3 `LinkedListNode * LinkedList::GetTail () [inline]`

Get the tail node of the linked list.

Returns

Pointer to the tail node in the list

Definition at line [159](#) of file [ll.h](#).

16.47.2.4 `void LinkedList::Remove (LinkedListNode * node_) [pure virtual]`

Add the linked list node to this linked list.

Parameters

<i>node_</i>	Pointer to the node to remove
--------------	-------------------------------

Implemented in [CircularLinkedList](#), [DoubleLinkedList](#), and [ThreadList](#).

The documentation for this class was generated from the following file:

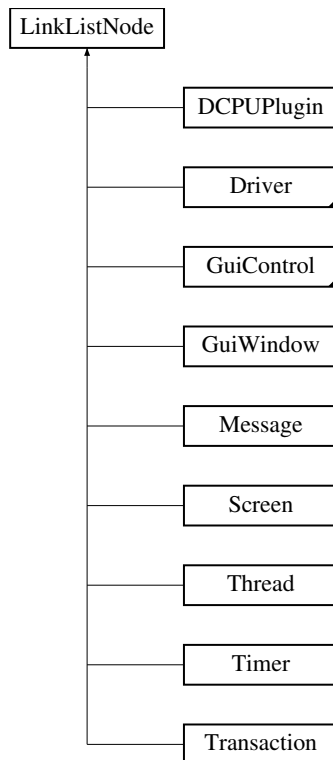
- [/home/mo/mark3-source/embedded/stage/src/ll.h](#)

16.48 LinkedListNode Class Reference

Basic linked-list node data structure.

```
#include <ll.h>
```

Inheritance diagram for LinkedListNode:



Public Member Functions

- [LinkListNode * GetNext](#) (void)
Returns a pointer to the next node in the list.
- [LinkListNode * GetPrev](#) (void)
Returns a pointer to the previous node in the list.

Protected Member Functions

- void [ClearNode](#) ()
Initialize the linked list node, clearing its next and previous node.

Protected Attributes

- [LinkListNode * next](#)
Pointer to the next node in the list.
- [LinkListNode * prev](#)
Pointer to the previous node in the list.

Friends

- class **LinkList**
- class **DoubleLinkList**
- class **CircularLinkList**

16.48.1 Detailed Description

Basic linked-list node data structure.

This data is managed by the linked-list class types, and can be used transparently between them.

Definition at line 69 of file [ll.h](#).

16.48.2 Member Function Documentation

16.48.2.1 `LinkedListNode * LinkedListNode::GetNext (void) [inline]`

Returns a pointer to the next node in the list.

Returns

a pointer to the next node in the list.

Definition at line 93 of file [ll.h](#).

16.48.2.2 `LinkedListNode * LinkedListNode::GetPrev (void) [inline]`

Returns a pointer to the previous node in the list.

Returns

a pointer to the previous node in the list.

Definition at line 102 of file [ll.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/ll.h](#)
- [/home/mo/mark3-source/embedded/stage/src/ll.cpp](#)

16.49 MemUtil Class Reference

String and Memory manipulation class.

```
#include <memutil.h>
```

Static Public Member Functions

- static void [DecimalToHex](#) (K_UCHAR ucData_, char *szText_)
Convert an 8-bit unsigned binary value as a hexadecimal string.
- static void **DecimalToHex** (K_USHORT usData_, char *szText_)
- static void **DecimalToHex** (K_ULONG ulData_, char *szText_)
- static void [DecimalToString](#) (K_UCHAR ucData_, char *szText_)
Convert an 8-bit unsigned binary value as a decimal string.
- static void **DecimalToString** (K_USHORT usData_, char *szText_)
- static void **DecimalToString** (K_ULONG ulData_, char *szText_)
- static K_UCHAR [Checksum8](#) (const void *pvSrc_, K_USHORT usLen_)
Compute the 8-bit additive checksum of a memory buffer.
- static K_USHORT [Checksum16](#) (const void *pvSrc_, K_USHORT usLen_)
Compute the 16-bit additive checksum of a memory buffer.

- static K_USHORT [StringLength](#) (const char *szStr_)
Compute the length of a string in bytes.
- static bool [CompareStrings](#) (const char *szStr1_, const char *szStr2_)
Compare the contents of two zero-terminated string buffers to eachother.
- static void [CopyMemory](#) (void *pvDst_, const void *pvSrc_, K_USHORT usLen_)
Copy one buffer in memory into another.
- static void [CopyString](#) (char *szDst_, const char *szSrc_)
Copy a string from one buffer into another.
- static K_SHORT [StringSearch](#) (const char *szBuffer_, const char *szPattern_)
Search for the presence of one string as a substring within another.
- static bool [CompareMemory](#) (const void *pvMem1_, const void *pvMem2_, K_USHORT usLen_)
Compare the contents of two memory buffers to eachother.
- static void [SetMemory](#) (void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_)
Initialize a buffer of memory to a specified 8-bit pattern.
- static K_UCHAR [Tokenize](#) (const char *szBuffer_, [Token_t](#) *pastTokens_, K_UCHAR ucMaxTokens_)
Tokenize Function to tokenize a string based on a space delimiter.

16.49.1 Detailed Description

String and Memory manipulation class.

Utility method class implementing common memory and string manipulation functions, without relying on an external standard library implementation which might not be available on some toolchains, may be closed source, or may not be thread-safe.

Definition at line 47 of file [memutil.h](#).

16.49.2 Member Function Documentation

16.49.2.1 static K_USHORT MemUtil::Checksum16 (const void * pvSrc_, K_USHORT usLen_) [static]

Compute the 16-bit addative checksum of a memory buffer.

Parameters

<i>pvSrc_</i>	Memory buffer to compute a 16-bit checksum of.
<i>usLen_</i>	Length of the buffer in bytes.

Returns

16-bit checksum of the memory block.

Definition at line 215 of file [memutil.cpp](#).

16.49.2.2 static K_USHORT MemUtil::Checksum8 (const void * pvSrc_, K_USHORT usLen_) [static]

Compute the 8-bit addative checksum of a memory buffer.

Parameters

<i>pvSrc_</i>	Memory buffer to compute a 8-bit checksum of.
---------------	---

<i>usLen_</i>	Length of the buffer in bytes.
---------------	--------------------------------

Returns

8-bit checksum of the memory block.

Definition at line 199 of file [memutil.cpp](#).

16.49.2.3 static bool MemUtil::CompareMemory (const void * *pvMem1_*, const void * *pvMem2_*, K_USHORT *usLen_*)
[static]

Compare the contents of two memory buffers to eachother.

Parameters

<i>pvMem1_</i>	First buffer to compare
<i>pvMem2_</i>	Second buffer to compare
<i>usLen_</i>	Length of buffer (in bytes) to compare

Returns

true if the buffers match, false if they do not.

Definition at line 342 of file [memutil.cpp](#).

16.49.2.4 static bool MemUtil::CompareStrings (const char * *szStr1_*, const char * *szStr2_*) [static]

Compare the contents of two zero-terminated string buffers to eachother.

Parameters

<i>szStr1_</i>	First string to compare
<i>szStr2_</i>	Second string to compare

Returns

true if strings match, false otherwise.

Definition at line 247 of file [memutil.cpp](#).

16.49.2.5 static void MemUtil::CopyMemory (void * *pvDst_*, const void * *pvSrc_*, K_USHORT *usLen_*) [static]

Copy one buffer in memory into another.

Parameters

<i>pvDst_</i>	Pointer to the destination buffer
<i>pvSrc_</i>	Pointer to the source buffer
<i>usLen_</i>	Number of bytes to copy from source to destination

Definition at line 273 of file [memutil.cpp](#).

16.49.2.6 static void MemUtil::CopyString (char * *szDst_*, const char * *szSrc_*) [static]

Copy a string from one buffer into another.

Parameters

<i>szDst_</i>	Pointer to the buffer to copy into
<i>szSrc_</i>	Pointer to the buffer to copy data from

Definition at line 290 of file [memutil.cpp](#).

16.49.2.7 static void MemUtil::DecimalToHex (K_UCHAR *ucData_*, char * *szText_*) [static]

Convert an 8-bit unsigned binary value as a hexadecimal string.

Parameters

<i>ucData_</i>	Value to convert into a string
<i>szText_</i>	Destination string buffer (3 bytes minimum)

Definition at line 28 of file [memutil.cpp](#).

16.49.2.8 static void MemUtil::DecimalToString (K_UCHAR *ucData_*, char * *szText_*) [static]

Convert an 8-bit unsigned binary value as a decimal string.

Parameters

<i>ucData_</i>	Value to convert into a string
<i>szText_</i>	Destination string buffer (4 bytes minimum)

Definition at line 122 of file [memutil.cpp](#).

16.49.2.9 static void MemUtil::SetMemory (void * *pvDst_*, K_UCHAR *ucVal_*, K_USHORT *usLen_*) [static]

Initialize a buffer of memory to a specified 8-bit pattern.

Parameters

<i>pvDst_</i>	Destination buffer to set
<i>ucVal_</i>	8-bit pattern to initialize each byte of destination with
<i>usLen_</i>	Length of the buffer (in bytes) to initialize

Definition at line 363 of file [memutil.cpp](#).

16.49.2.10 static K_USHORT MemUtil::StringLength (const char * *szStr_*) [static]

Compute the length of a string in bytes.

Parameters

<i>szStr_</i>	Pointer to the zero-terminated string to calculate the length of
---------------	--

Returns

length of the string (in bytes), not including the 0-terminator.

Definition at line 232 of file [memutil.cpp](#).

16.49.2.11 static K_SHORT MemUtil::StringSearch (const char * *szBuffer_*, const char * *szPattern_*) [static]

Search for the presence of one string as a substring within another.

Parameters

<i>szBuffer_</i>	Buffer to search for pattern within
<i>szPattern_</i>	Pattern to search for in the buffer

Returns

Index of the first instance of the pattern in the buffer, or -1 on no match.

Definition at line 307 of file [memutil.cpp](#).

16.49.2.12 `K_UCHAR MemUtil::Tokenize (const char * szBuffer_, Token_t * pastTokens_, K_UCHAR ucMaxTokens_)`
`[static]`

Tokenize Function to tokenize a string based on a space delimiter.

This is a non-destructive function, which populates a `Token_t` descriptor array.

Parameters

<i>szBuffer_</i>	String to tokenize
<i>pastTokens_</i>	Pointer to the array of token descriptors
<i>ucMaxTokens_</i>	Maximum number of tokens to parse (i.e. size of <i>pastTokens_</i>)

Returns

Count of tokens parsed

Definition at line 376 of file [memutil.cpp](#).

The documentation for this class was generated from the following files:

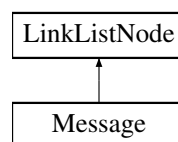
- [/home/mo/mark3-source/embedded/stage/src/memutil.h](#)
- [/home/mo/mark3-source/embedded/stage/src/memutil.cpp](#)

16.50 Message Class Reference

Class to provide message-based IPC services in the kernel.

```
#include <message.h>
```

Inheritance diagram for Message:



Public Member Functions

- void [Init](#) ()
Initialize the data and code in the message.
- void [SetData](#) (void *pvData_)
Set the data pointer for the message before transmission.
- void * [GetData](#) ()

- *Get the data pointer stored in the message upon receipt.*
- void [SetCode](#) (K_USHORT usCode_)
- *Set the code in the message before transmission.*
- K_USHORT [GetCode](#) ()
- *Return the code set in the message upon receipt.*

Private Attributes

- void * [m_pvData](#)
- *Pointer to the message data.*
- K_USHORT [m_usCode](#)
- *Message code, providing context for the message.*

Additional Inherited Members

16.50.1 Detailed Description

Class to provide message-based IPC services in the kernel.

Definition at line 99 of file [message.h](#).

16.50.2 Member Function Documentation

16.50.2.1 K_USHORT Message::GetCode () [inline]

Return the code set in the message upon receipt.

Returns

User code set in the object

Definition at line 143 of file [message.h](#).

16.50.2.2 void * Message::GetData () [inline]

Get the data pointer stored in the message upon receipt.

Returns

Pointer to the data set in the message object

Definition at line 125 of file [message.h](#).

16.50.2.3 Message::SetCode (K_USHORT usCode_) [inline]

Set the code in the message before transmission.

Parameters

<i>usCode_</i>	Data code to set in the object
----------------	--------------------------------

Definition at line 134 of file [message.h](#).

16.50.2.4 void Message::SetData (void * pvData_) [inline]

Set the data pointer for the message before transmission.

Parameters

<code>pvData_</code>	Pointer to the data object to send in the message
----------------------	---

Definition at line 116 of file [message.h](#).

The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/message.h](#)

16.51 MessageQueue Class Reference

List of messages, used as the channel for sending and receiving messages between threads.

```
#include <message.h>
```

Public Member Functions

- `void Init ()`
Initialize the message queue prior to use.
- `Message * Receive ()`
Receive a message from the message queue.
- `Message * Receive (K_ULONG ulTimeWaitMS_)`
Receive a message from the message queue.
- `void Send (Message *pclSrc_)`
Send a message object into this message queue.
- `K_USHORT GetCount ()`
Return the number of messages pending in the "receive" queue.

Private Attributes

- `Semaphore m_clSemaphore`
Counting semaphore used to manage thread blocking.
- `DoubleLinkedList m_clLinkList`
List object used to store messages.

16.51.1 Detailed Description

List of messages, used as the channel for sending and receiving messages between threads.

Definition at line 201 of file [message.h](#).

16.51.2 Member Function Documentation

16.51.2.1 K_USHORT MessageQueue::GetCount ()

Return the number of messages pending in the "receive" queue.

Returns

Count of pending messages in the queue.

Definition at line 149 of file [message.cpp](#).

16.51.2.2 Message * MessageQueue::Receive ()

Receive a message from the message queue.

If the message queue is empty, the thread will block until a message is available.

Returns

Pointer to a message object at the head of the queue

Definition at line 91 of file [message.cpp](#).

16.51.2.3 Message * MessageQueue::Receive (K_ULONG ulWaitTimeMS_)

Receive a message from the message queue.

If the message queue is empty, the thread will block until a message is available for the duration specified. If no message arrives within that duration, the call will return with NULL.

Parameters

<i>ulWaitTimeMS_</i>	The amount of time in ms to wait for a message before timing out and unblocking the waiting thread.
----------------------	---

Returns

Pointer to a message object at the head of the queue or NULL on timeout.

Definition at line 111 of file [message.cpp](#).

16.51.2.4 void MessageQueue::Send (Message * pclSrc_)

Send a message object into this message queue.

Will un-block the first waiting thread blocked on this queue if that occurs.

Parameters

<i>pclSrc_</i>	Pointer to the message object to add to the queue
----------------	---

Definition at line 133 of file [message.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/message.h](#)
- [/home/mo/mark3-source/embedded/stage/src/message.cpp](#)

16.52 MouseEvent_t Struct Reference

Mouse UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_USHORT [usX](#)
absolute X location of the mouse (pixel)
- K_USHORT [usY](#)

```

        absolute Y location of the mouse (pixel)
    • union {
        K_UCHAR ucFlags
        modifier flags for the event
    struct {
        unsigned int bLeftState:1
        State of the left mouse button.
        unsigned int bRightState:1
        State of the right mouse button.
        unsigned int bMiddleState:1
        State of the middle mouse button.
        unsigned int bScrollUp:1
        State of the scroll wheel (UP)
        unsigned int bScrollDown:1
        State of the scroll wheel (DOWN)
    }
};

```

16.52.1 Detailed Description

Mouse UI event structure.

Definition at line 102 of file [gui.h](#).

The documentation for this struct was generated from the following file:

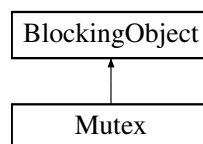
- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.53 Mutex Class Reference

Mutual-exclusion locks, based on [BlockingObject](#).

```
#include <mutex.h>
```

Inheritance diagram for Mutex:



Public Member Functions

- void [Init](#) ()

Initialize a mutex object for use - must call this function before using the object.
- void [Claim](#) ()

Claim the mutex.
- bool [Claim](#) (K_ULONG ulWaitTimeMS_)
- void [Timeout](#) (Thread *pclOwner_)

Wake a thread blocked on the mutex.
- void [Release](#) ()

Release the mutex.

Private Member Functions

- `K_UCHAR WakeNext ()`
Wake the next thread waiting on the [Mutex](#).
- `K_BOOL ProcessQueue ()`
- `void ClaimTransaction (Transaction *pclTRX_, K_BOOL *pbReschedule_)`
ClaimTransaction.
- `void ReleaseTransaction (Transaction *pclTRX_, K_BOOL *pbReschedule_)`
ReleaseTransaction.
- `void TimeoutTransaction (Transaction *pclTRX_, K_BOOL *pbReschedule_)`
TimeoutTransaction.

Private Attributes

- `K_UCHAR m_ucRecurse`
The recursive lock-count when a mutex is claimed multiple times by the same owner.
- `K_UCHAR m_bReady`
State of the mutex - true = ready, false = claimed.
- `K_UCHAR m_ucMaxPri`
Maximum priority of thread in queue, used for priority inheritance.
- `Thread * m_pclOwner`
Pointer to the thread that owns the mutex (when claimed)

Additional Inherited Members

16.53.1 Detailed Description

Mutual-exclusion locks, based on [BlockingObject](#).

Definition at line 69 of file [mutex.h](#).

16.53.2 Member Function Documentation

16.53.2.1 void Mutex::Claim ()

Claim the mutex.

When the mutex is claimed, no other thread can claim a region protected by the object.

Definition at line 282 of file [mutex.cpp](#).

16.53.2.2 bool Mutex::Claim (K_ULONG ulWaitTimeMS_)

Parameters

<code>ulWaitTimeMS_</code>	
----------------------------	--

Returns

true - mutex was claimed within the time period specified
false - mutex operation timed-out before the claim operation.

Definition at line 286 of file [mutex.cpp](#).

16.53.2.3 void Mutex::ClaimTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

ClaimTransaction.

Perform a mutex Claim (Lock) operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 115 of file [mutex.cpp](#).

16.53.2.4 void Mutex::Release ()

Release the mutex.

When the mutex is released, another object can enter the mutex-protected region.

Definition at line 325 of file [mutex.cpp](#).

16.53.2.5 void Mutex::ReleaseTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

ReleaseTransaction.

Perform a [Mutex](#) Release/Unlock operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 185 of file [mutex.cpp](#).

16.53.2.6 void Mutex::Timeout (Thread * *pciOwner_*)

Wake a thread blocked on the mutex.

This is an internal function used for implementing timed mutexes relying on timer callbacks. Since these do not have access to the private data of the mutex and its base classes, we have to wrap this as a public method - do not use this for any other purposes.

Parameters

<i>pciOwner_</i>	Thread to unblock from this object.
------------------	---

Definition at line 55 of file [mutex.cpp](#).

16.53.2.7 void Mutex::TimeoutTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

TimeoutTransaction.

Perform a [Mutex](#) "timeout" operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 233 of file [mutex.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/mutex.h](#)

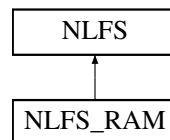
- [/home/mo/mark3-source/embedded/stage/src/mutex.cpp](#)

16.54 NLFS Class Reference

Nice Little File System class.

```
#include <nlfs.h>
```

Inheritance diagram for NLFS:



Public Member Functions

- void [Format](#) ([NLFS_Host_t](#) *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_, K_USHORT usDataBlockSize_)
Format/Create a new filesystem with the configuration specified in the parameters.
- void [Mount](#) ([NLFS_Host_t](#) *puHost_)
Re-mount a previously-created filesystem using this FS object.
- K_USHORT [Create_File](#) (const K_CHAR *szPath_)
Create_File creates a new file object at the specified path.
- K_USHORT [Create_Dir](#) (const K_CHAR *szPath_)
Create_Dir creates a new directory at the specified path.
- K_USHORT [Delete_File](#) (const K_CHAR *szPath_)
Delete_File Removes a file from disk.
- K_USHORT [Delete_Folder](#) (const K_CHAR *szPath_)
Delete_Folder Remove a folder from disk.
- void [Cleanup_Node_Links](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Cleanup_Node_Links Remove the links between the given node and its parent/peer nodes.
- K_USHORT [Find_Parent_Dir](#) (const K_CHAR *szPath_)
Find_Parent_Dir returns the directory under which the specified file object lives.
- K_USHORT [Find_File](#) (const K_CHAR *szPath_)
Find_File returns the file node ID of the object at a given path.
- void [Print](#) (void)
Print displays a summary of files in the filesystem.
- K_ULONG [GetBlockSize](#) (void)
GetBlockSize retrieves the data block size for the filesystem.
- K_ULONG [GetNumBlocks](#) (void)
GetNumBlocks retrieves the number of data blocks in the filesystem.
- K_ULONG [GetNumBlocksFree](#) (void)
GetNumBlocksFree retrieves the number of free data blocks in the filesystem.
- K_ULONG [GetNumFiles](#) (void)
GetNumFiles retrieves the maximum number of files in the filesystem.
- K_USHORT [GetNumFilesFree](#) (void)
GetNumFilesFree retrieves the number of free blocks in the filesystem.
- K_USHORT [GetFirstChild](#) (K_USHORT usNode_)
GetFirstChild Return the first child node for a node representing a directory.

- K_USHORT [GetNextPeer](#) (K_USHORT usNode_)
GetNextPeer Return the Node ID of a File/Directory's next peer.
- K_BOOL [GetStat](#) (K_USHORT usNode_, [NLFS_File_Stat_t](#) *pstStat_)
GetStat Get the status of a file on-disk.

Protected Member Functions

- K_CHAR [Find_Last_Slash](#) (const K_CHAR *szPath_)
Find_Last_Slash Finds the location of the last '/' character in a path.
- K_BOOL [File_Names_Match](#) (const K_CHAR *szPath_, [NLFS_Node_t](#) *pstNode_)
File_Names_Match Determines if a given path matches the name in a file node.
- virtual void [Read_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)=0
Read_Node is an implementation-specific method used to read a file node from physical storage into a local data struture.
- virtual void [Write_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)=0
Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.
- virtual void [Read_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstBlock_)=0
Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.
- virtual void [Write_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstFileBlock_)=0
Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.
- virtual void [Read_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)=0
Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.
- virtual void [Write_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)=0
Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.
- void [RootSync](#) ()
RootSync Synchronize the filesystem config in the object back to the underlying storage mechanism.
- void [Repair](#) ()
Repair Checks a filesystem for inconsistencies and makes repairs in order to avoid losing storage blocks.
- void [Print_Free_Details](#) (K_USHORT usNode_)
Print_Free_Details Print details about a free node.
- void [Print_File_Details](#) (K_USHORT usNode_)
Print_File_Details displays information about a given file node.
- void [Print_Dir_Details](#) (K_USHORT usNode_)
Print_Dir_Details displays information about a given directory node.
- void [Print_Node_Details](#) (K_USHORT usNode_)
Print_Node_Details prints details about a node, the details differ based on whether it's a file/directory/root node.
- void [Push_Free_Node](#) (K_USHORT usNode_)
Push_Free_Node returns a file node back to the free node list.
- K_USHORT [Pop_Free_Node](#) (void)
Pop_Free_Node returns the first free file node in the free list.
- void [Push_Free_Block](#) (K_ULONG ulBlock_)
Push_Free_Block returns a file block back to the head of the free block list.
- K_ULONG [Pop_Free_Block](#) (void)
Pop_Free_Block pops a file data block from the head of the free list.
- K_ULONG [Append_Block_To_Node](#) ([NLFS_Node_t](#) *pstFile_)
Append_Block_To_Node adds a file data block to the end of a file.
- K_USHORT [Create_File_i](#) (const K_CHAR *szPath_, [NLFS_Type_t](#) eType_)
Create_File_i is the private method used to create a file or directory.
- void [Set_Node_Name](#) ([NLFS_Node_t](#) *pstFileNode_, const K_CHAR *szPath_)
Set_Node_Name sets the name of a file or directory node.

Protected Attributes

- [NLFS_Host_t * m_puHost](#)
Local, cached copy of host FS pointer.
- [NLFS_Root_Node_t m_stLocalRoot](#)
Local, cached copy of root.

Friends

- class **NLFS_File**

16.54.1 Detailed Description

Nice Little File System class.

Definition at line 280 of file [nlfs.h](#).

16.54.2 Member Function Documentation

16.54.2.1 K_ULONG NLFS::Append_Block_To_Node (NLFS_Node_t * *pstFile_*) [protected]

Append_Block_To_Node adds a file data block to the end of a file.

Parameters

in	<i>pstFile_</i>	- Pointer to the file node to add a block to
----	-----------------	--

Returns

Data block ID of the allocated block, or INVALID_BLOCK on failure.

Definition at line 245 of file [nlfs.cpp](#).

16.54.2.2 void NLFS::Cleanup_Node_Links (K_USHORT *usNode_*, NLFS_Node_t * *pstNode_*)

Cleanup_Node_Links Remove the links between the given node and its parent/peer nodes.

Parameters

<i>usNode_</i>	Index of the node
<i>pstNode_</i>	Pointer to a local copy of the node data

Definition at line 598 of file [nlfs.cpp](#).

16.54.2.3 K_USHORT NLFS::Create_Dir (const K_CHAR * *szPath_*)

Create_Dir creates a new directory at the specified path.

Parameters

in	<i>szPath_</i>	- Path to the directory to create
----	----------------	-----------------------------------

Returns

ID of the created dir, or INVALID_NODE if the path cannot be resolved, or the file already exists.

Definition at line 586 of file [nlfs.cpp](#).

16.54.2.4 K_USHORT NLFS::Create_File (const K_CHAR * *szPath_*)

Create_File creates a new file object at the specified path.

Parameters

<i>in</i>	<i>szPath_</i>	- Path to the file to create
-----------	----------------	------------------------------

Returns

ID of the created file, or INVALID_NODE if the path cannot be resolved, or the file already exists.

Definition at line 573 of file [nlfs.cpp](#).

16.54.2.5 K_USHORT NLFS::Create_File_i (const K_CHAR * *szPath_*, NLFS_Type_t *eType_*) [protected]

Create_File_i is the private method used to create a file or directory.

Parameters

<i>in</i>	<i>szPath_</i>	- Path of the file or directory to create
<i>in</i>	<i>eType_</i>	- Type of file to create

Returns

File node ID of the newly created file, or INVALID_NODE on failure.

! ToDo - set real user/group IDs

Definition at line 490 of file [nlfs.cpp](#).

16.54.2.6 K_USHORT NLFS::Delete_File (const K_CHAR * *szPath_*)

Delete_File Removes a file from disk.

Parameters

<i>szPath_</i>	Path of the file to remove
----------------	----------------------------

Returns

Index of the node deleted or INVALID_NODE on error

Definition at line 705 of file [nlfs.cpp](#).

16.54.2.7 K_USHORT NLFS::Delete_Folder (const K_CHAR * *szPath_*)

Delete_Folder Remove a folder from disk.

Parameters

<i>szPath_</i>	Path of the folder to remove
----------------	------------------------------

Returns

Index of the node deleted or INVALID_NODE on error

Definition at line 662 of file [nlfs.cpp](#).

16.54.2.8 K_BOOL NLFS::File_Names_Match (const K_CHAR * *szPath_*, NLFS_Node_t * *pstNode_*) [protected]

File_Names_Match Determines if a given path matches the name in a file node.

Parameters

in	<i>szPath_</i>	- file path to search for
in	<i>pstNode_</i>	- pointer to a fs node

Returns

true if the filename in the path matches the filename in the node.

Definition at line 42 of file [nlfs.cpp](#).

16.54.2.9 K_USHORT NLFS::Find_File (const K_CHAR * *szPath_*)

Find_File returns the file node ID of the object at a given path.

Parameters

in	<i>szPath_</i>	- Path of the file to search for
----	----------------	----------------------------------

Returns

file node ID, or INVALID_NODE if the path is invalid.

Definition at line 405 of file [nlfs.cpp](#).

16.54.2.10 K_CHAR NLFS::Find_Last_Slash (const K_CHAR * *szPath_*) [protected]

Find_Last_Slash Finds the location of the last '/' character in a path.

Parameters

in	<i>szPath_</i>	- String representing a '/' delimited path.
----	----------------	---

Returns

the byte offset of the last slash char in the path.

Definition at line 26 of file [nlfs.cpp](#).

16.54.2.11 K_USHORT NLFS::Find_Parent_Dir (const K_CHAR * *szPath_*)

Find_Parent_Dir returns the directory under which the specified file object lives.

Parameters

in	<i>szPath_</i>	- Path of the file to find parent directory node for
----	----------------	--

Returns

directory node ID, or INVALID_NODE if the path is invalid.

Definition at line 289 of file [nlfs.cpp](#).

16.54.2.12 void NLFS::Format (NLFS_Host_t * *puHost_*, K_ULONG *ulTotalSize_*, K_USHORT *usNumFiles_*, K_USHORT *usDataBlockSize_*)

Format/Create a new filesystem with the configuration specified in the parameters.

Parameters

in	<i>puHost_</i>	- Pointer to the FS storage object, interpreted by the physical medium driver.
in	<i>ulTotalSize_</i>	- Total size of the object to format (in bytes)
in	<i>usNumFiles_</i>	- Number of file nodes to create in the FS. This parameter determines the maximum number of files and directories that can exist simultaneously in the filesystem. All filesystem storage not allocated towards file nodes is automatically used as data-blocks.
	<i>usDataBlock-Size_</i>	- Size of each data block (in bytes). Setting a lower block size is a good way to avoid wasting space in small-files due to over-allocation of storage (size on-disk vs. actual file size). However, each block requires a metadata object, which can also add to overhead. Also, file read/write speed can vary significantly based on the block size - in many scenarios, larger blocks can lead to higher throughput.

Definition at line 756 of file [nlfs.cpp](#).

16.54.2.13 `K_ULONG NLFS::GetBlockSize (void) [inline]`

GetBlockSize retrieves the data block size for the filesystem.

Returns

The size of a data block in the filesystem, as configured at format.

Definition at line 382 of file [nlfs.h](#).

16.54.2.14 `K_USHORT NLFS::GetFirstChild (K_USHORT usNode_)`

GetFirstChild Return the first child node for a node representing a directory.

Parameters

<i>usNode_</i>	Index of a directory node
----------------	---------------------------

Returns

Node ID of the first child node or INVALID_NODE on failure

Definition at line 890 of file [nlfs.cpp](#).

16.54.2.15 `K_USHORT NLFS::GetNextPeer (K_USHORT usNode_)`

GetNextPeer Return the Node ID of a File/Directory's next peer.

Parameters

<i>usNode_</i>	Node index of the current object
----------------	----------------------------------

Returns

Node ID of the next peer object

Definition at line 908 of file [nlfs.cpp](#).

16.54.2.16 `K_ULONG NLFS::GetNumBlocks (void) [inline]`

GetNumBlocks retrieves the number of data blocks in the filesystem.

Returns

The total number of blocks in the filesystem

Definition at line 388 of file [nlfs.h](#).

16.54.2.17 `K_ULONG NLFS::GetNumBlocksFree (void) [inline]`

GetNumBlocksFree retrieves the number of free data blocks in the filesystem.

Returns

The number of available blocks in the filesystem

Definition at line 395 of file [nlfs.h](#).

16.54.2.18 `K_ULONG NLFS::GetNumFiles (void) [inline]`

GetNumFiles retrieves the maximum number of files in the filesystem.

Returns

The maximum number of files that can be allocated in the system

Definition at line 401 of file [nlfs.h](#).

16.54.2.19 `K_USHORT NLFS::GetNumFilesFree (void) [inline]`

GetNumFilesFree retrieves the number of free blocks in the filesystem.

Returns

The number of free file nodes in the filesystem

Definition at line 407 of file [nlfs.h](#).

16.54.2.20 `K_BOOL NLFS::GetStat (K_USHORT usNode_, NLFS_File_Stat_t * pstStat_)`

GetStat Get the status of a file on-disk.

Parameters

<i>usNode_</i>	Node representing the file
<i>pstStat_</i>	Pointer to the object containing the status

Returns

true on success, false on failure

Definition at line 920 of file [nlfs.cpp](#).

16.54.2.21 `void NLFS::Mount (NLFS_Host_t * puHost_)`

Re-mount a previously-created filesystem using this FS object.

Parameters

<i>in</i>	<i>puHost_</i>	- Pointer to the filesystem object
-----------	----------------	------------------------------------

! Must set the host pointer first.

Definition at line 859 of file [nlfs.cpp](#).

16.54.2.22 `K_ULONG NLFS::Pop_Free_Block (void) [protected]`

Pop_Free_Block pops a file data block from the head of the free list.

Returns

the block index of the file node popped from the head of the free block list

Definition at line 192 of file [nlfs.cpp](#).

16.54.2.23 `K_USHORT NLFS::Pop_Free_Node (void) [protected]`

Pop_Free_Node returns the first free file node in the free list.

Returns

the index of the file node popped off the free list

Definition at line 145 of file [nlfs.cpp](#).

16.54.2.24 `void NLFS::Print_Dir_Details (K_USHORT usNode_) [protected]`

Print_Dir_Details displays information about a given directory node.

Parameters

<i>in</i>	<i>usNode_</i>	- directory index to display details for
-----------	----------------	--

Definition at line 90 of file [nlfs.cpp](#).

16.54.2.25 `void NLFS::Print_File_Details (K_USHORT usNode_) [protected]`

Print_File_Details displays information about a given file node.

Parameters

<i>in</i>	<i>usNode_</i>	- file index to display details for
-----------	----------------	-------------------------------------

Definition at line 68 of file [nlfs.cpp](#).

16.54.2.26 `void NLFS::Print_Free_Details (K_USHORT usNode_) [protected]`

Print_Free_Details Print details about a free node.

Parameters

<i>usNode_</i>	Node to print details for
----------------	---------------------------

Definition at line 106 of file [nlfs.cpp](#).

16.54.2.27 `void NLFS::Print_Node_Details (K_USHORT usNode_) [protected]`

Print_Node_Details prints details about a node, the details differ based on whether it's a file/directory/root node.

Parameters

in	<i>usNode_</i>	- node to show details for
----	----------------	----------------------------

Definition at line 115 of file [nlfs.cpp](#).

16.54.2.28 void NLFS::Push_Free_Block (K_ULONG *ulBlock_*) [protected]

Push_Free_Block returns a file block back to the head of the free block list.

Parameters

in	<i>ulBlock_</i>	- index of the data block to free
----	-----------------	-----------------------------------

Definition at line 224 of file [nlfs.cpp](#).

16.54.2.29 void NLFS::Push_Free_Node (K_USHORT *usNode_*) [protected]

Push_Free_Node returns a file node back to the free node list.

Parameters

in	<i>usNode_</i>	- index of the file node to push back to the free list.
----	----------------	---

Definition at line 172 of file [nlfs.cpp](#).

16.54.2.30 virtual void NLFS::Read_Block (K_ULONG *ulBlock_*, K_ULONG *ulOffset_*, void * *pvData_*, K_ULONG *ulLen_*) [protected], [pure virtual]

Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
out	<i>pvData_</i>	- output buffer to read into
in	<i>ulLen_</i>	- length of data to read (in bytes)

Implemented in [NLFS_RAM](#).

16.54.2.31 virtual void NLFS::Read_Block_Header (K_ULONG *ulBlock_*, NLFS_Block_t * *pstBlock_*) [protected], [pure virtual]

Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
out	<i>pstBlock_</i>	- block header structure to read into

Implemented in [NLFS_RAM](#).

16.54.2.32 virtual void NLFS::Read_Node (K_USHORT *usNode_*, NLFS_Node_t * *pstNode_*) [protected], [pure virtual]

Read_Node is an implementation-specific method used to read a file node from physical storage into a local data structure.

Parameters

in	<i>usNode_</i>	- File node index
out	<i>pstNode_</i>	- Pointer to the file node object to read into

Implemented in [NLFS_RAM](#).

16.54.2.33 void NLFS::RootSync () [protected]

RootSync Synchronize the filesystem config in the object back to the underlying storage mechanism.

This needs to be called to ensure that underlying storage is kept consistent when creating or deleting files.

Definition at line 879 of file [nlfs.cpp](#).

16.54.2.34 void NLFS::Set_Node_Name (NLFS_Node_t * *pstFileNode_*, const K_CHAR * *szPath_*) [protected]

Set_Node_Name sets the name of a file or directory node.

Parameters

in	<i>pstFileNode_</i>	- Pointer to a file node structure to name
in	<i>szPath_</i>	- Name for the file

Definition at line 458 of file [nlfs.cpp](#).

16.54.2.35 virtual void NLFS::Write_Block (K_ULONG *ulBlock_*, K_ULONG *ulOffset_*, void * *pvData_*, K_ULONG *ulLen_*)
[protected],[pure virtual]

Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
in	<i>pvData_</i>	- data buffer to write to disk
in	<i>ulLen_</i>	- length of data to write (in bytes)

Implemented in [NLFS_RAM](#).

16.54.2.36 virtual void NLFS::Write_Block_Header (K_ULONG *ulBlock_*, NLFS_Block_t * *pstFileBlock_*)
[protected],[pure virtual]

Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
in	<i>pstFileBlock_</i>	- pointer to the local data structure to write from

Implemented in [NLFS_RAM](#).

16.54.2.37 virtual void NLFS::Write_Node (K_USHORT *usNode_*, NLFS_Node_t * *pstNode_*) [protected],[pure virtual]

Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.

Parameters

in	<i>usNode_</i>	- File node index
in	<i>pstNode_</i>	- Pointer to the file node object to write from

Implemented in [NLFS_RAM](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)
- [/home/mo/mark3-source/embedded/stage/src/nlfs.cpp](#)

16.55 NLFS_Block_t Struct Reference

Block data structure.

```
#include <nlfs.h>
```

Public Attributes

- K_ULONG [uNextBlock](#)
Index of the next block.
- union {
 K_UCHAR [ucFlags](#)
 Block Flags.
 struct {
 unsigned int [uAllocated](#)
 1 if allocated
 unsigned int [uCheckBit](#)
 Used for continuity checks.
 }
};

16.55.1 Detailed Description

Block data structure.

Contains the block index of the next data block (either in the file, or in the free-data pool), as well as any special flags.

Definition at line [232](#) of file [nlfs.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.56 NLFS_File Class Reference

The [NLFS_File](#) class.

```
#include <nlfs_file.h>
```

Public Member Functions

- int [Open](#) (NLFS *pclFS_, const K_CHAR *szPath_, NLFS_File_Mode_t eMode_)

- Open Opens a file from a given filesystem.*
- int [Read](#) (void *pvBuf_, K_ULONG ulLen_)
Read Read bytes from a file into a specified data buffer.
- int [Write](#) (void *pvBuf_, K_ULONG ulLen_)
Write Write a specified blob of data to the file.
- int [Seek](#) (K_ULONG ulOffset_)
Seek Seek to the specified byte offset within the file.
- int [Close](#) (void)
Close Is used to close an open file buffer.

Private Attributes

- [NLFS * m_pclFileSystem](#)
Pointer to the host filesystem.
- K_ULONG [m_ulOffset](#)
Current byte offset within the file.
- K_ULONG [m_ulCurrentBlock](#)
Index of the current filesystem block.
- K_USHORT [m_usFile](#)
File index of the current file.
- [NLFS_File_Mode_t m_ucFlags](#)
File mode flags.
- [NLFS_Node_t m_stNode](#)
Local copy of the file node.

16.56.1 Detailed Description

The [NLFS_File](#) class.

This class contains an implementation of file-level access built on-top of the [NLFS](#) filesystem architecture. An instance of this class represents an active/open file from inside the NLFSfilesystem.

Definition at line 45 of file [nlfs_file.h](#).

16.56.2 Member Function Documentation

16.56.2.1 int NLFS_File::Close (void)

Close Is used to close an open file buffer.

Returns

0 on success, -1 on failure.

Definition at line 272 of file [nlfs_file.cpp](#).

16.56.2.2 int NLFS_File::Open (NLFS * pclFS_, const K_CHAR * szPath_, NLFS_File_Mode_t eMode_)

Open Opens a file from a given filesystem.

Parameters

<i>pcIFS_</i>	- Pointer to the NLFS filesystem containing the file
<i>szPath_</i>	- Path to the file within the NLFS filesystem
<i>eMode_</i>	- File open mode

Returns

0 on success, -1 on failure

Definition at line 26 of file [nlfs_file.cpp](#).

16.56.2.3 int NLFS_File::Read (void * *pvBuf_*, K_ULONG *ulLen_*)

Read Read bytes from a file into a specified data buffer.

Parameters

in	<i>ulLen_</i>	- Length (in bytes) of data to read
out	<i>pvBuf_</i>	- Pointer to the buffer to read into

Returns

Number of bytes read from the file

Definition at line 151 of file [nlfs_file.cpp](#).

16.56.2.4 int NLFS_File::Seek (K_ULONG *ulOffset_*)

Seek Seek to the specified byte offset within the file.

Parameters

in	<i>ulOffset_</i>	Offset in bytes from the beginning of the file
----	------------------	--

Returns

0 on success, -1 on failure

Definition at line 112 of file [nlfs_file.cpp](#).

16.56.2.5 int NLFS_File::Write (void * *pvBuf_*, K_ULONG *ulLen_*)

Write Write a specified blob of data to the file.

Parameters

in	<i>ulLen_</i>	- Length (in bytes) of the source buffer
in	<i>pvBuf_</i>	- Pointer to the data buffer containing the data to be written

Returns

Number of bytes written to the file

Definition at line 217 of file [nlfs_file.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/nlfs_file.h](#)
- [/home/mo/mark3-source/embedded/stage/src/nlfs_file.cpp](#)

16.57 NLFS_File_Node_t Struct Reference

Data structure for the "file" FS-node type.

```
#include <nlfs.h>
```

Public Attributes

- K_CHAR [acFileName](#) [16]
Arbitrary, 16-char filename.
- K_USHORT [usNextPeer](#)
Index of the next peer file node.
- K_USHORT [usPrevPeer](#)
Index of the previous peer node.
- K_UCHAR [ucGroup](#)
Group ID of the owner.
- K_UCHAR [ucUser](#)
User ID of the owner.
- K_USHORT [usPerms](#)
File permissions (POSIX-style)
- K_USHORT [usParent](#)
Index of the parent file node.
- K_USHORT [usChild](#)
Index of the first child node.
- K_ULONG [ulAllocSize](#)
Size of the file (allocated)
- K_ULONG [ulFileSize](#)
Size of the file (in-bytes)
- K_ULONG [ulFirstBlock](#)
Index of the first file block.
- K_ULONG [ulLastBlock](#)
Index of the last file block.

16.57.1 Detailed Description

Data structure for the "file" FS-node type.

Note that this is the same as for a directory node (although fewer fields are used for that case, as documented).

Definition at line 168 of file [nlfs.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.58 NLFS_File_Stat_t Struct Reference

Structure used to report the status of a given file.

```
#include <nlfs.h>
```

Public Attributes

- K_ULONG [ulAllocSize](#)
Size of the file including partial blocks.
- K_ULONG [ulFileSize](#)
Actual size of the file.
- K_USHORT [usPerms](#)
Permissions attached to the file.
- K_UCHAR [ucUser](#)
User associated with this file.
- K_UCHAR [ucGroup](#)
Group associated with this file.
- K_CHAR [acFileName](#) [16]
Copy of the file name.

16.58.1 Detailed Description

Structure used to report the status of a given file.

Definition at line 266 of file [nlfs.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.59 NLFS_Host_t Union Reference

Union used for managing host-specific pointers/data-types.

```
#include <nlfs.h>
```

Public Attributes

- void * **pvData**
- uint32_t **u32Data**
- uint64_t **u64Data**
- K_ADDR **kaData**

16.59.1 Detailed Description

Union used for managing host-specific pointers/data-types.

This is all pretty abstract, as the data represented here is only accessed by the underlying physical media drive.

Definition at line 253 of file [nlfs.h](#).

The documentation for this union was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.60 NLFS_Node_t Struct Reference

Filesystem node data structure.

```
#include <nlfs.h>
```

Public Attributes

- [NLFS_Type_t eBlockType](#)
Block type ID.
- union {
 [NLFS_Root_Node_t stRootNode](#)
 Root Filesystem Node.
 [NLFS_File_Node_t stFileNode](#)
 File/Directory Node.
};

16.60.1 Detailed Description

Filesystem node data structure.

Contains the block type, as well as the union between the various FS-node data structures. This is also the same data format as how data is stored "on-disk"

Definition at line 215 of file [nlfs.h](#).

The documentation for this struct was generated from the following file:

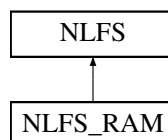
- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.61 NLFS_RAM Class Reference

The [NLFS_RAM](#) class.

```
#include <nlfs_ram.h>
```

Inheritance diagram for NLFS_RAM:



Private Member Functions

- virtual void [Read_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Read_Node is an implementation-specific method used to read a file node from physical storage into a local data struture.
- virtual void [Write_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.
- virtual void [Read_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstBlock_)
Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.
- virtual void [Write_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstFileBlock_)
Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.
- virtual void [Read_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)
Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.

- void [Write_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)

Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Additional Inherited Members

16.61.1 Detailed Description

The [NLFS_RAM](#) class.

This class implements an [NLFS](#) filesystem in a RAM buffer. In this case, the host pointer passed into the "format" call is a pointer to the locally- allocated buffer in which the filesystem lives.

Definition at line 31 of file [nlfs_ram.h](#).

16.61.2 Member Function Documentation

- 16.61.2.1 void [NLFS_RAM::Read_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void * pvData_, K_ULONG ulLen_)
[private], [virtual]

[Read_Block](#) is an implementation-specific method used to read raw file data from physical storage into a local buffer.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
out	<i>pvData_</i>	- output buffer to read into
in	<i>ulLen_</i>	- length of data to read (in bytes)

Implements [NLFS](#).

Definition at line 63 of file [nlfs_ram.cpp](#).

- 16.61.2.2 void [NLFS_RAM::Read_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) * pstBlock_) [private],
[virtual]

[Read_Block_Header](#) is an implementation-specific method used to read a file block header from physical storage into a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
out	<i>pstBlock_</i>	- block header structure to read into

Implements [NLFS](#).

Definition at line 43 of file [nlfs_ram.cpp](#).

- 16.61.2.3 void [NLFS_RAM::Read_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) * pstNode_) [private],
[virtual]

[Read_Node](#) is an implementation-specific method used to read a file node from physical storage into a local data structure.

Parameters

in	<i>usNode_</i>	- File node index
out	<i>pstNode_</i>	- Pointer to the file node object to read into

Implements [NLFS](#).

Definition at line 25 of file [nlfs_ram.cpp](#).

16.61.2.4 `void NLFS_RAM::Write_Block (K_ULONG ulBlock_, K_ULONG ulOffset_, void * pvData_, K_ULONG ulLen_)`
`[private], [virtual]`

Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
in	<i>pvData_</i>	- data buffer to write to disk
in	<i>ulLen_</i>	- length of data to write (in bytes)

Implements [NLFS](#).

Definition at line 73 of file [nlfs_ram.cpp](#).

16.61.2.5 `void NLFS_RAM::Write_Block_Header (K_ULONG ulBlock_, NLFS_Block_t * pstFileBlock_)` `[private], [virtual]`

Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
in	<i>pstFileBlock_</i>	- pointer to the local data structure to write from

Implements [NLFS](#).

Definition at line 53 of file [nlfs_ram.cpp](#).

16.61.2.6 `void NLFS_RAM::Write_Node (K_USHORT usNode_, NLFS_Node_t * pstNode_)` `[private], [virtual]`

Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.

Parameters

in	<i>usNode_</i>	- File node index
in	<i>pstNode_</i>	- Pointer to the file node object to write from

Implements [NLFS](#).

Definition at line 34 of file [nlfs_ram.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/nlfs_ram.h](#)
- [/home/mo/mark3-source/embedded/stage/src/nlfs_ram.cpp](#)

16.62 NLFS_Root_Node_t Struct Reference

Data structure for the Root-configuration FS-node type.

```
#include <nlfs.h>
```

Public Attributes

- K_USHORT [usNumFiles](#)
Number of file nodes in the FS.
- K_USHORT [usNumFilesFree](#)
Number of free file nodes.
- K_USHORT [usNextFreeNode](#)
Index of the next free file.
- K_ULONG [ulNumBlocks](#)
Number of blocks in the FS.
- K_ULONG [ulNumBlocksFree](#)
Number of free blocks.
- K_ULONG [ulNextFreeBlock](#)
Index of the next free block.
- K_ULONG [ulBlockSize](#)
Size of each block on disk.
- K_ULONG [ulBlockOffset](#)
Byte-offset to the first block struct.
- K_ULONG [ulDataOffset](#)
Byte-offset to the first data block.

16.62.1 Detailed Description

Data structure for the Root-configuration FS-node type.

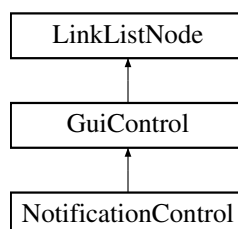
Definition at line 194 of file [nlfs.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/nlfs.h](#)

16.63 NotificationControl Class Reference

Inheritance diagram for NotificationControl:



Public Member Functions

- virtual void **Init** ()
Initialiize the control - must be called before use.
- virtual void **Draw** ()
Redraw the control "cleanly".
- virtual **GuiReturn_t ProcessEvent** (**GuiEvent_t** *pstEvent_)
Process an event sent to the control.
- virtual void **Activate** (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void **SetFont** (**Font_t** *pstFont_)
- void **SetCaption** (const K_CHAR *szCaption_)
- void **Trigger** (K_USHORT usTimeout_)

Private Attributes

- const K_CHAR * **m_szCaption**
- **Font_t** * **m_pstFont**
- K_USHORT **m_usTimeout**
- bool **m_bTrigger**
- bool **m_bVisible**

Additional Inherited Members

16.63.1 Detailed Description

Definition at line 29 of file [control_notification.h](#).

16.63.2 Member Function Documentation

16.63.2.1 virtual void NotificationControl::Activate (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 43 of file [control_notification.h](#).

16.63.2.2 void NotificationControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 26 of file [control_notification.cpp](#).

16.63.2.3 `virtual void NotificationControl::Init() [inline],[virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 32 of file [control_notification.h](#).

16.63.2.4 `GuiReturn_t NotificationControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<code>pstEvent_</code>	Pointer to a struct containing the event data
------------------------	---

Implements [GuiControl](#).

Definition at line 92 of file [control_notification.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_notification.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_notification.cpp](#)

16.64 Option_t Struct Reference

Structure used to represent a command-line option with its arguments.

```
#include <shell_support.h>
```

Public Attributes

- [Token_t](#) * [pstStart](#)
Pointer to the beginning of a token array contain the option and its arguments.
- [K_UCHAR](#) [ucCount](#)
Number of tokens in the token array.

16.64.1 Detailed Description

Structure used to represent a command-line option with its arguments.

An option is defined as any token beginning with a "-" value. The tokens arguments are subsequent tokens that do not begin with "-".

Where no "-" values are specified, each token becomes its own option.

i.e. given the following command-line

```
mycmd -opt1 a b c -opt2 d e f -opt 3
```

The possible [Option_t](#) structures would be:

```
pstStart => Array containing tokens for -opt1, a, b, c
ucCount  => 4 (4 tokens, including the option token, "-opt1")

pstStart => Array containing tokens for -opt2, d, e, f
```

```
ucCount  => 4 (4 tokens, including the option token, "-opt2")

pstStart => Array containing tokens for -opt, 3
ucCount  => 2 (2 tokens, including the option token, "-opt3")
```

in the case of:

```
mycmd a b c
```

Possible token values would be:

```
pstStart => Array containing tokens for a
ucCount  => 1

pstStart => Array containing tokens for b
ucCount  => 1

pstStart => Array containing tokens for c
ucCount  => 1
```

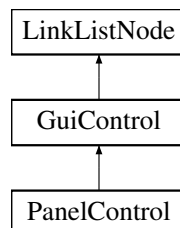
Definition at line 83 of file [shell_support.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/shell_support.h](#)

16.65 PanelControl Class Reference

Inheritance diagram for PanelControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetColor](#) ([COLOR](#) eColor_)

Private Attributes

- [COLOR](#) [m_uColor](#)

Additional Inherited Members

16.65.1 Detailed Description

Definition at line 33 of file [control_panel.h](#).

16.65.2 Member Function Documentation

16.65.2.1 `virtual void PanelControl::Activate (bool bActivate_) [inline], [virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 39 of file [control_panel.h](#).

16.65.2.2 `void PanelControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 26 of file [control_panel.cpp](#).

16.65.2.3 `virtual void PanelControl::Init () [inline], [virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 36 of file [control_panel.h](#).

16.65.2.4 `virtual GuiReturn_t PanelControl::ProcessEvent (GuiEvent_t * pstEvent_) [inline], [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 38 of file [control_panel.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_panel.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_panel.cpp](#)

16.66 Profiler Class Reference

System profiling timer interface.

```
#include <kprofile.h>
```

Static Public Member Functions

- static void [Init](#) ()
Initialize the global system profiler.
- static void [Start](#) ()
Start the global profiling timer service.
- static void [Stop](#) ()
Stop the global profiling timer service.
- static K_USHORT [Read](#) ()
Read the current tick count in the timer.
- static void [Process](#) ()
Process the profiling counters from ISR.
- static K_ULONG [GetEpoch](#) ()
Return the current timer epoch.

Static Private Attributes

- static K_ULONG **m_ulEpoch**

16.66.1 Detailed Description

System profiling timer interface.

Definition at line 37 of file [kprofile.h](#).

16.66.2 Member Function Documentation

16.66.2.1 void Profiler::Init (void) [static]

Initialize the global system profiler.

Must be called prior to use.

Definition at line 32 of file [kprofile.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/kprofile.h](#)
- [/home/mo/mark3-source/embedded/stage/src/kprofile.cpp](#)

16.67 ProfileTimer Class Reference

Profiling timer.

```
#include <profile.h>
```


Public Member Functions

- void [Init](#) ()
Initialize the profiling timer prior to use.
- void [Start](#) ()
Start a profiling session, if the timer is not already active.
- void [Stop](#) ()
Stop the current profiling session, adding to the cumulative time for this timer, and the total iteration count.
- K_ULONG [GetAverage](#) ()
Get the average time associated with this operation.
- K_ULONG [GetCurrent](#) ()
Return the current tick count held by the profiler.

Private Member Functions

- K_ULONG [ComputeCurrentTicks](#) (K_USHORT usCount_, K_ULONG ulEpoch_)
Figure out how many ticks have elapsed in this iteration.

Private Attributes

- K_ULONG [m_ulCumulative](#)
Cumulative tick-count for this timer.
- K_ULONG [m_ulCurrentIteration](#)
Tick-count for the current iteration.
- K_USHORT [m_usInitial](#)
Initial count.
- K_ULONG [m_ulInitialEpoch](#)
Initial Epoch.
- K_USHORT [m_usIterations](#)
Number of iterations executed for this profiling timer.
- K_UCHAR [m_bActive](#)
Whether or not the timer is active or stopped.

16.67.1 Detailed Description

Profiling timer.

This class is used to perform high-performance profiling of code to see how K_LONG certain operations take. Useful in instrumenting the performance of key algorithms and time-critical operations to ensure real-time behavior.

Definition at line 69 of file [profile.h](#).

16.67.2 Member Function Documentation

16.67.2.1 K_ULONG ProfileTimer::ComputeCurrentTicks (K_USHORT usCount_, K_ULONG ulEpoch_) [private]

Figure out how many ticks have elapsed in this iteration.

Parameters

<i>usCount_</i>	Current timer count
<i>ulEpoch_</i>	Current timer epoch

Returns

Current tick count

Definition at line 106 of file [profile.cpp](#).

16.67.2.2 K_ULONG ProfileTimer::GetAverage ()

Get the average time associated with this operation.

Returns

Average tick count normalized over all iterations

Definition at line 79 of file [profile.cpp](#).

16.67.2.3 K_ULONG ProfileTimer::GetCurrent ()

Return the current tick count held by the profiler.

Valid for both active and stopped timers.

Returns

The currently held tick count.

Definition at line 89 of file [profile.cpp](#).

16.67.2.4 void ProfileTimer::Init (void)

Initialize the profiling timer prior to use.

Can also be used to reset a timer that's been used previously.

Definition at line 37 of file [profile.cpp](#).

16.67.2.5 void ProfileTimer::Start (void)

Start a profiling session, if the timer is not already active.

Has no effect if the timer is already active.

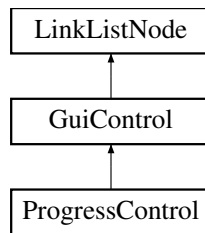
Definition at line 46 of file [profile.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/profile.h](#)
- [/home/mo/mark3-source/embedded/stage/src/profile.cpp](#)

16.68 ProgressControl Class Reference

Inheritance diagram for ProgressControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetBackColor](#) (COLOR eColor_)
- void [SetProgressColor](#) (COLOR eColor_)
- void [SetBorderColor](#) (COLOR eColor_)
- void [SetProgress](#) (K_UCHAR ucProgress_)

Private Attributes

- COLOR [m_uBackColor](#)
- COLOR [m_uProgressColor](#)
- COLOR [m_uBorderColor](#)
- K_UCHAR [m_ucProgress](#)

Additional Inherited Members

16.68.1 Detailed Description

Definition at line 30 of file [control_progress.h](#).

16.68.2 Member Function Documentation

16.68.2.1 virtual void [ProgressControl::Activate](#) (bool *bActivate_*) [inline], [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 36 of file [control_progress.h](#).

16.68.2.2 void ProgressControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 36 of file [control_progress.cpp](#).

16.68.2.3 void ProgressControl::Init () [virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 27 of file [control_progress.cpp](#).

16.68.2.4 GuiReturn_t ProgressControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 102 of file [control_progress.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_progress.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_progress.cpp](#)

16.69 Quantum Class Reference

Static-class used to implement [Thread](#) quantum functionality, which is a key part of round-robin scheduling.

```
#include <quantum.h>
```

Static Public Member Functions

- static void [UpdateTimer](#) ()
This function is called to update the thread quantum timer whenever something in the scheduler has changed.
- static void [AddThread](#) ([Thread](#) *pclThread_)
Add the thread to the quantum timer.
- static void [RemoveThread](#) ()
Remove the thread from the quantum timer.

Static Private Member Functions

- static void [SetTimer](#) ([Thread](#) *pclThread_)
Set up the quantum timer in the timer scheduler.

Static Private Attributes

- static [Timer](#) **m_clQuantumTimer**
- static K_UCHAR **m_bActive**

16.69.1 Detailed Description

Static-class used to implement [Thread](#) quantum functionality, which is a key part of round-robin scheduling.

Definition at line 39 of file [quantum.h](#).

16.69.2 Member Function Documentation

16.69.2.1 void Quantum::AddThread (Thread * *pclThread_*) [static]

Add the thread to the quantum timer.

Only one thread can own the quantum, since only one thread can be running on a core at a time.

Definition at line 70 of file [quantum.cpp](#).

16.69.2.2 void Quantum::RemoveThread (void) [static]

Remove the thread from the quantum timer.

This will cancel the timer.

Definition at line 87 of file [quantum.cpp](#).

16.69.2.3 void Quantum::SetTimer (Thread * *pclThread_*) [static],[private]

Set up the quantum timer in the timer scheduler.

This creates a one-shot timer, which calls a static callback in [quantum.cpp](#) that on expiry will pivot the head of the threadlist for the thread's priority. This is the mechanism that provides round-robin scheduling in the system.

Parameters

<i>pclThread_</i>	Pointer to the thread to set the Quantum timer on
-------------------	---

Definition at line 60 of file [quantum.cpp](#).

16.69.2.4 void Quantum::UpdateTimer (void) [static]

This function is called to update the thread quantum timer whenever something in the scheduler has changed.

This can result in the timer being re-loaded or started. The timer is never stopped, but it may be ignored on expiry.

Definition at line 100 of file [quantum.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/quantum.h](#)
- [/home/mo/mark3-source/embedded/stage/src/quantum.cpp](#)

16.70 Scheduler Class Reference

Priority-based round-robin [Thread](#) scheduling, using ThreadLists for housekeeping.

```
#include <scheduler.h>
```

Static Public Member Functions

- static void [Init](#) ()
Intialize the scheduler, must be called before use.
- static void [Schedule](#) ()
Run the scheduler, determines the next thread to run based on the current state of the threads.
- static void [Add](#) ([Thread](#) *[pclThread_](#))
Add a thread to the scheduler at its current priority level.
- static void [Remove](#) ([Thread](#) *[pclThread_](#))
Remove a thread from the scheduler at its current priority level.
- static K_BOOL [SetScheduler](#) (K_BOOL [bEnable_](#))
Set the active state of the scheduler.
- static [Thread](#) * [GetCurrentThread](#) ()
Return the pointer to the currently-running thread.
- static [Thread](#) * [GetNextThread](#) ()
Return the pointer to the thread that should run next, according to the last run of the scheduler.
- static [ThreadList](#) * [GetThreadList](#) (K_UCHAR [ucPriority_](#))
Return the pointer to the active list of threads that are at the given priority level in the scheduler.
- static [ThreadList](#) * [GetStopList](#) ()
Return the pointer to the list of threads that are in the scheduler's stopped state.
- static K_UCHAR [IsEnabled](#) ()
Return the current state of the scheduler - whether or not scheudling is enabled or disabled.
- static void [QueueScheduler](#) ()

Static Private Attributes

- static K_BOOL [m_bEnabled](#)
[Scheduler](#)'s state - enabled or disabled.
- static K_BOOL [m_bQueuedSchedule](#)
Variable representing whether or not there's a queued scheduler operation.
- static [ThreadList](#) [m_clStopList](#)
[ThreadList](#) for all stopped threads.
- static [ThreadList](#) [m_aclPriorities](#) [NUM_PRIORITIES]
[ThreadLists](#) for all threads at all priorities.
- static K_UCHAR [m_ucPriFlag](#)
Bitmap flag for each.

16.70.1 Detailed Description

Priority-based round-robin [Thread](#) scheduling, using [ThreadLists](#) for housekeeping.

Definition at line 62 of file [scheduler.h](#).

16.70.2 Member Function Documentation

16.70.2.1 void Scheduler::Add ([Thread](#) * [pclThread_](#)) [static]

Add a thread to the scheduler at its current priority level.

Parameters

<i>pcThread_</i>	Pointer to the thread to add to the scheduler
------------------	---

Definition at line 81 of file [scheduler.cpp](#).

16.70.2.2 static Thread* Scheduler::GetCurrentThread () [inline],[static]

Return the pointer to the currently-running thread.

Returns

Pointer to the currently-running thread

Definition at line 119 of file [scheduler.h](#).

16.70.2.3 static Thread* Scheduler::GetNextThread () [inline],[static]

Return the pointer to the thread that should run next, according to the last run of the scheduler.

Returns

Pointer to the next-running thread

Definition at line 127 of file [scheduler.h](#).

16.70.2.4 static ThreadList* Scheduler::GetStopList () [inline],[static]

Return the pointer to the list of threads that are in the scheduler's stopped state.

Returns

Pointer to the [ThreadList](#) containing the stopped threads

Definition at line 145 of file [scheduler.h](#).

16.70.2.5 static ThreadList* Scheduler::GetThreadList (K_UCHAR ucPriority_) [inline],[static]

Return the pointer to the active list of threads that are at the given priority level in the scheduler.

Parameters

<i>ucPriority_</i>	Priority level of
--------------------	-------------------

Returns

Pointer to the [ThreadList](#) for the given priority level

Definition at line 137 of file [scheduler.h](#).

16.70.2.6 K_UCHAR Scheduler::IsEnabled () [inline],[static]

Return the current state of the scheduler - whether or not scheduling is enabled or disabled.

Returns

true - scheduler enabled, false - disabled

Definition at line 155 of file [scheduler.h](#).

16.70.2.7 `void Scheduler::Remove (Thread * pcThread_) [static]`

Remove a thread from the scheduler at its current priority level.

Parameters

<code>pcThread_</code>	Pointer to the thread to be removed from the scheduler
------------------------	--

Definition at line 88 of file [scheduler.cpp](#).

16.70.2.8 Scheduler::Schedule () [static]

Run the scheduler, determines the next thread to run based on the current state of the threads.

Note that the next-thread chosen from this function is only valid while in a critical section.

Definition at line 64 of file [scheduler.cpp](#).

16.70.2.9 void Scheduler::SetScheduler (K_BOOL bEnable_) [static]

Set the active state of the scheduler.

When the scheduler is disabled, the *next thread* is never set; the currently running thread will run forever until the scheduler is enabled again. Care must be taken to ensure that we don't end up trying to block while the scheduler is disabled, otherwise the system ends up in an unusable state.

Parameters

<code>bEnable_</code>	true to enable, false to disable the scheduler
-----------------------	--

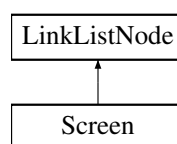
Definition at line 95 of file [scheduler.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/scheduler.h](#)
- [/home/mo/mark3-source/embedded/stage/src/scheduler.cpp](#)

16.71 Screen Class Reference

Inheritance diagram for Screen:



Public Member Functions

- void [Activate](#) ()
This is called when a new screen needs to be created.
- void [Deactivate](#) ()
This is called when a screen is torn-down.
- void [SetWindowAffinity](#) (const K_CHAR *szWindowName_)
Indicate by name which window this screen is to be bound.
- void [SetName](#) (const K_CHAR *szName_)
Set the name of the current screen.
- const K_CHAR * [GetName](#) ()
Return the name of the current screen.

Protected Member Functions

- void [SetManager](#) ([ScreenManager](#) *pclScreenManager_)
Function called by the [ScreenManager](#) to set the screen affinity.

Protected Attributes

- const K_CHAR * **m_szName**
- [ScreenManager](#) * **m_pclScreenManager**
- [GuiWindow](#) * **m_pclWindow**

Private Member Functions

- virtual void **Create** ()=0
- virtual void **Destroy** ()=0

Friends

- class **ScreenManager**

16.71.1 Detailed Description

Definition at line 31 of file [screen.h](#).

16.71.2 Member Function Documentation

16.71.2.1 void Screen::Activate () [inline]

This is called when a new screen needs to be created.

This calls the underlying virtual "create" method, which performs all control object initialization and allocation. Calling a redraw(true) on the bound window will result in the new window being rendered to display.

Definition at line 40 of file [screen.h](#).

16.71.2.2 void Screen::Deactivate () [inline]

This is called when a screen is torn-down.

Essentially removes the controls from the named window and deallocates any memory used to build up the screen.

Definition at line 47 of file [screen.h](#).

The documentation for this class was generated from the following files:

- /home/mo/mark3-source/embedded/stage/src/[screen.h](#)
- /home/mo/mark3-source/embedded/stage/src/[screen.cpp](#)

16.72 ScreenList Class Reference

Public Member Functions

- void [Add](#) ([Screen](#) *pclScreen_)

- *Add a screen to the screen list.*
- void [Remove](#) ([Screen](#) *pclScreen_)
- *Remove a screen from the screen list.*
- [Screen](#) * [GetHead](#) ()
- *Get the beginning of the screen list.*

Private Attributes

- [DoubleLinkedList](#) [m_clList](#)
- *Double link-list used to manage screen objects.*

16.72.1 Detailed Description

Definition at line 84 of file [screen.h](#).

The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/screen.h](#)

16.73 ScreenManager Class Reference

Public Member Functions

- void [AddScreen](#) ([Screen](#) *pclScreen_)
- *Add a new screen to the screen manager.*
- void [RemoveScreen](#) ([Screen](#) *pclScreen_)
- *Remove an existing screen from the screen manager.*
- void [SetEventSurface](#) ([GuiEventSurface](#) *pclSurface_)
- *Set the event surface on which this screen manager's screens will be displayed.*
- [GuiWindow](#) * [FindWindowByName](#) (const K_CHAR *m_szName_)
- *Return a pointer to a window by name.*
- [Screen](#) * [FindScreenByName](#) (const K_CHAR *m_szName_)
- *Return a pointer to a screen by name.*

Private Attributes

- [ScreenList](#) [m_clScreenList](#)
- *[Screen](#) list object used to manage individual screens.*
- [GuiEventSurface](#) * [m_pclSurface](#)
- *Pointer to the GUI Event Surface on which the screens are displayed.*

16.73.1 Detailed Description

Definition at line 109 of file [screen.h](#).

The documentation for this class was generated from the following files:

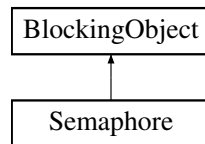
- [/home/mo/mark3-source/embedded/stage/src/screen.h](#)
- [/home/mo/mark3-source/embedded/stage/src/screen.cpp](#)

16.74 Semaphore Class Reference

Counting semaphore, based on [BlockingObject](#) base class.

```
#include <ksemaphore.h>
```

Inheritance diagram for Semaphore:



Public Member Functions

- void [Init](#) (K_USHORT usInitVal_, K_USHORT usMaxVal_)
Initialize a semaphore before use.
- void [Post](#) ()
Increment the semaphore count.
- void [Pend](#) ()
Decrement the semaphore count.
- K_USHORT [GetCount](#) ()
Return the current semaphore counter.
- bool [Pend](#) (K_ULONG ulWaitTimeMS_)
Decrement the semaphore count.
- void [Timeout](#) (Thread *pclChosenOne_)
Wake a thread blocked on the semaphore.

Private Member Functions

- K_UCHAR [WakeNext](#) ()
Wake the next thread waiting on the semaphore.
- K_BOOL [ProcessQueue](#) ()
ProcessQueue.
- void [PostTransaction](#) (Transaction *pclTRX_, K_BOOL *pbReschedule_)
PostTransaction.
- void [PendTransaction](#) (Transaction *pclTRX_, K_BOOL *pbReschedule_)
PendTransaction.
- void [TimeoutTransaction](#) (Transaction *pclTRX_, K_BOOL *pbReschedule_)
TimeoutTransaction.

Private Attributes

- K_USHORT [m_usValue](#)
Current value in the semaphore.
- K_USHORT [m_usMaxValue](#)
Maximum value that the semaphore can hold.

Additional Inherited Members

16.74.1 Detailed Description

Counting semaphore, based on [BlockingObject](#) base class.

Definition at line 39 of file [ksemaphore.h](#).

16.74.2 Member Function Documentation

16.74.2.1 K_USHORT Semaphore::GetCount ()

Return the current semaphore counter.

This can be used by a thread to bypass blocking on a semaphore - allowing it to do other things until a non-zero count is returned, instead of blocking until the semaphore is posted.

Returns

The current semaphore counter value.

Definition at line 283 of file [ksemaphore.cpp](#).

16.74.2.2 void Semaphore::Init (K_USHORT *usInitVal_*, K_USHORT *usMaxVal_*)

Initialize a semaphore before use.

Must be called before post/pend operations.

Parameters

<i>usInitVal_</i>	Initial value held by the semaphore
<i>usMaxVal_</i>	Maximum value for the semaphore

Definition at line 194 of file [ksemaphore.cpp](#).

16.74.2.3 void Semaphore::Pend ()

Decrement the semaphore count.

If the count is zero, the thread will block until the semaphore is pended.

Definition at line 232 of file [ksemaphore.cpp](#).

16.74.2.4 bool Semaphore::Pend (K_ULONG *ulWaitTimeMS_*)

Decrement the semaphore count.

If the count is zero, the thread will block until the semaphore is pended. If the specified interval expires before the thread is unblocked, then the status is returned back to the user.

Returns

true - semaphore was acquired before the timeout false - timeout occurred before the semaphore was claimed.

Definition at line 237 of file [ksemaphore.cpp](#).

16.74.2.5 void Semaphore::PendTransaction (Transaction * *pciTRX_*, K_BOOL * *pbReschedule_*) [private]

PendTransaction.

Perform a semaphore "pend" operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 130 of file [ksemaphore.cpp](#).

16.74.2.6 void Semaphore::Post ()

Increment the semaphore count.

Returns

true if the semaphore was posted, false if the count is already maxed out.

Definition at line 206 of file [ksemaphore.cpp](#).

16.74.2.7 void Semaphore::PostTransaction (Transaction * pciTRX_, K_BOOL * pbReschedule_) [private]

PostTransaction.

Perform a semaphore "post" operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 109 of file [ksemaphore.cpp](#).

16.74.2.8 K_BOOL Semaphore::ProcessQueue () [private]

ProcessQueue.

Process all pending actions on the semaphore's transaction queue. This should only be called from within a context where the blocking object's Lock() value has already been called. When ProcessQueue returns, the Lock() value will be reset to 0 - as all pending transactions have been processed.

Returns

true - A thread scheduling operation must be performed. false - No rescheduling is required.

Definition at line 78 of file [ksemaphore.cpp](#).

16.74.2.9 void Semaphore::Timeout (Thread * pciChosenOne_)

Wake a thread blocked on the semaphore.

This is an internal function used for implementing timed semaphores relying on timer callbacks. Since these do not have access to the private data of the semaphore and its base classes, we have to wrap this as a public method - do not use this for any other purposes.

Definition at line 60 of file [ksemaphore.cpp](#).

16.74.2.10 void Semaphore::TimeoutTransaction (Transaction * pciTRX_, K_BOOL * pbReschedule_) [private]

TimeoutTransaction.

Perform a semaphore "timeout" operation, as specified from an object on the transaction queue.

Parameters

<i>pciTRX_</i>	- Pointer to the transaction object
<i>pbReschedule_</i>	- Pointer to boolean to be set true if rescheduling is required.

Definition at line 161 of file [ksemaphore.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/ksemaphore.h](#)
- [/home/mo/mark3-source/embedded/stage/src/ksemaphore.cpp](#)

16.75 ShellCommand_t Struct Reference

Data structure defining a lookup table correlating a command name to its handler function.

```
#include <shell_support.h>
```

Public Attributes

- const K_CHAR * [szCommand](#)
Command name.
- [fp_internal_command](#) pfHandler
Command handler function.

16.75.1 Detailed Description

Data structure defining a lookup table correlating a command name to its handler function.

Definition at line 117 of file [shell_support.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/shell_support.h](#)

16.76 ShellSupport Class Reference

The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

```
#include <shell_support.h>
```

Static Public Member Functions

- static K_CHAR [RunCommand](#) ([CommandLine_t](#) *pstCommand_, const [ShellCommand_t](#) *pastShellCommands_)
RunCommand Given a command-line, attempts to run the corresponding shell command based where it exists within the supplied [ShellCommand_t](#) array.
- static void [UnescapeToken](#) ([Token_t](#) *pstToken_, K_CHAR *szDest_)
UnescapeToken Convert a token which has special parsing characters in it to a "flattened" string, where all unescaped double quotes and escaped tab, newline, space, etc.
- static [Option_t](#) * [CheckForOption](#) ([CommandLine_t](#) *pstCommand_, const K_CHAR *szOption_)
CheckForOption Check to see whether or not a specific option has been set within the commandline arguments.

- static K_CHAR [TokensToCommandLine](#) (Token_t *pastTokens_, K_UCHAR ucTokens_, CommandLine_t *pstCommand_)

TokensToCommandLine Convert an array of tokens to a commandline object.

16.76.1 Detailed Description

The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

Definition at line 129 of file [shell_support.h](#).

16.76.2 Member Function Documentation

16.76.2.1 Option_t * ShellSupport::CheckForOption (CommandLine_t * pstCommand_, const K_CHAR * szOption_)
[static]

CheckForOption Check to see whether or not a specific option has been set within the commandline arguments.

Parameters

<i>pstCommand_</i>	Pointer to the commandline object containing the options
<i>szOption_</i>	0-terminated string corresponding to the command-line option.

Returns

Pointer to the command line option on match, or 0 on failure.

Definition at line 104 of file [shell_support.cpp](#).

16.76.2.2 K_CHAR ShellSupport::RunCommand (CommandLine_t * pstCommand_, const ShellCommand_t * pastShellCommands_) [static]

RunCommand Given a command-line, attempts to run the corresponding shell command based where it exists within the supplied [ShellCommand_t](#) array.

Parameters

<i>pstCommand_</i>	Pointer to the command-line to execute
<i>pstCommands_</i>	Pointer to an array of shell commands to execute against

Returns

1 on success, 0 on error (command not found)

Definition at line 28 of file [shell_support.cpp](#).

16.76.2.3 K_CHAR ShellSupport::TokensToCommandLine (Token_t * pastTokens_, K_UCHAR ucTokens_, CommandLine_t * pstCommand_) [static]

TokensToCommandLine Convert an array of tokens to a commandline object.

This operation is non-destructive to the source token array.

Parameters

<i>pstTokens_</i>	Pointer to the token array to process
<i>ucTokens_</i>	Number of tokens in the token array
<i>pstCommand_</i>	Pointer to the CommandLine_t object which will represent the shell command and its arguments.

Returns

Number of options processed

Definition at line 123 of file [shell_support.cpp](#).

16.76.2.4 void ShellSupport::UnescapeToken (Token_t * *pstToken_*, K_CHAR * *szDest_*) [static]

UnescapeToken Convert a token which has special parsing characters in it to a "flattened" string, where all un-escaped double quotes and escaped tab, newline, space, etc.

characters are converted into their ascii-code equivalents.

Parameters

<i>pstToken_</i>	Pointer to the source token to convert
<i>szDest_</i>	Pointer to a destination string which will contain the parsed result string

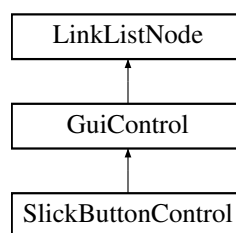
Definition at line 49 of file [shell_support.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/shell_support.h](#)
- [/home/mo/mark3-source/embedded/stage/src/shell_support.cpp](#)

16.77 SlickButtonControl Class Reference

Inheritance diagram for SlickButtonControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) **pstEvent_*)
Process an event sent to the control.
- virtual void [Activate](#) (bool *bActivate_*)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetFont](#) ([Font_t](#) **pstFont_*)
- void [SetCaption](#) (const K_CHAR **szCaption_*)
- void [SetCallback](#) (ButtonCallback *pfCallback_*, void **pvData_*)

Private Attributes

- const K_CHAR * **m_szCaption**
- [Font_t](#) * **m_pstFont**
- bool **m_bState**
- K_UCHAR **m_ucTimeout**
- void * **m_pvCallbackData**
- ButtonCallback **m_pfCallback**

Additional Inherited Members

16.77.1 Detailed Description

Definition at line 32 of file [control_slickbutton.h](#).

16.77.2 Member Function Documentation

16.77.2.1 void SlickButtonControl::Activate (bool *bActivate_*) [virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 286 of file [control_slickbutton.cpp](#).

16.77.2.2 void SlickButtonControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 51 of file [control_slickbutton.cpp](#).

16.77.2.3 void SlickButtonControl::Init () [virtual]

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 40 of file [control_slickbutton.cpp](#).

16.77.2.4 GuiReturn_t SlickButtonControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<code>pstEvent_</code>	Pointer to a struct containing the event data
------------------------	---

Implements [GuiControl](#).

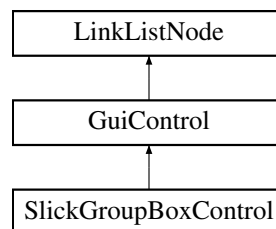
Definition at line 164 of file [control_slickbutton.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_slickbutton.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_slickbutton.cpp](#)

16.78 SlickGroupBoxControl Class Reference

Inheritance diagram for SlickGroupBoxControl:



Public Member Functions

- virtual void [Init](#) ()
Initialiize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *pcCaption_)
- void [SetBGColor](#) (COLOR uColor_)

Private Attributes

- [Font_t](#) * [m_pstFont](#)
- const K_CHAR * [m_pcCaption](#)
- COLOR [m_uBGColor](#)

Additional Inherited Members

16.78.1 Detailed Description

Definition at line 29 of file [control_slickgroupbox.h](#).

16.78.2 Member Function Documentation

16.78.2.1 `virtual void SlickGroupBoxControl::Activate (bool bActivate_) [inline], [virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 35 of file [control_slickgroupbox.h](#).

16.78.2.2 void SlickGroupBoxControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 31 of file [control_slickgroupbox.cpp](#).

16.78.2.3 virtual void SlickGroupBoxControl::Init () [inline],[virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 32 of file [control_slickgroupbox.h](#).

16.78.2.4 virtual GuiReturn_t SlickGroupBoxControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [inline],[virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

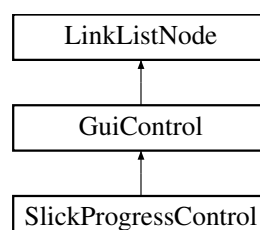
Definition at line 34 of file [control_slickgroupbox.h](#).

The documentation for this class was generated from the following files:

- /home/mo/mark3-source/embedded/stage/src/control_slickgroupbox.h
- /home/mo/mark3-source/embedded/stage/src/control_slickgroupbox.cpp

16.79 SlickProgressControl Class Reference

Inheritance diagram for SlickProgressControl:



Public Member Functions

- virtual void [Init](#) ()
Initialiize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetProgress](#) (K_UCHAR ucProgress_)

Private Attributes

- K_UCHAR [m_ucProgress](#)

Additional Inherited Members

16.79.1 Detailed Description

Definition at line 30 of file [control_slickprogress.h](#).

16.79.2 Member Function Documentation

16.79.2.1 virtual void [SlickProgressControl::Activate](#) (bool *bActivate_*) [inline],[virtual]

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 36 of file [control_slickprogress.h](#).

16.79.2.2 void [SlickProgressControl::Draw](#) () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 33 of file [control_slickprogress.cpp](#).

16.79.2.3 void [SlickProgressControl::Init](#) () [virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 27 of file [control_slickprogress.cpp](#).

16.79.2.4 **GuiReturn_t** SlickProgressControl::ProcessEvent (**GuiEvent_t** * *pstEvent_*) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 107 of file [control_slickprogress.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/control_slickprogress.h](#)
- [/home/mo/mark3-source/embedded/stage/src/control_slickprogress.cpp](#)

16.80 Slip Class Reference

Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

```
#include <slip.h>
```

Public Member Functions

- void [SetDriver](#) ([Driver](#) *pclDriver_)
Set the driver to attach to this object.
- [Driver](#) * [GetDriver](#) ()
Return the pointer to the driver attached to this object.
- void [WriteData](#) (K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_)
Write a packet of data in the FunkenSlip format.
- K_USHORT [ReadData](#) (K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_)
Read a packet from a specified device, parse, and copy to a specified output buffer.
- void [WriteVector](#) (K_UCHAR ucChannel_, [SlipDataVector](#) *astData_, K_USHORT usLen_)
Write a single message composed of multiple data-vector fragments.
- void [SendAck](#) ()
Send an acknowledgement character to the host.
- void [SendNack](#) ()
Send a negative-acknowledgement character to the host.

Static Public Member Functions

- static K_USHORT [EncodeByte](#) (K_UCHAR ucChar_, K_UCHAR *aucBuf_)
Encode a single byte into a stream, returning the size of the encoded value (either 1 or 2 bytes).
- static K_USHORT [DecodeByte](#) (K_UCHAR *ucChar_, const K_UCHAR *aucBuf_)
Decode a byte from a stream into a specified value.

Private Member Functions

- void [WriteByte](#) (K_UCHAR ucData_)

Private Attributes

- [Driver](#) * `m_pclDriver`

16.80.1 Detailed Description

Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

Definition at line 70 of file [slip.h](#).

16.80.2 Member Function Documentation

16.80.2.1 K_USHORT Slip::DecodeByte (K_UCHAR * *ucChar_*, const K_UCHAR * *aucBuf_*) [static]

Decode a byte from a stream into a specified value.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

<i>ucChar_</i>	Destination K_CHAR
<i>aucBuf_</i>	Source buffer

Returns

bytes read, or 0 on terminating character (192)

Definition at line 56 of file [slip.cpp](#).

16.80.2.2 K_USHORT Slip::EncodeByte (K_UCHAR *ucChar_*, K_UCHAR * *aucBuf_*) [static]

Encode a single byte into a stream, returning the size of the encoded value (either 1 or 2 bytes).

Parameters

<i>ucChar_</i>	Character to encode
<i>aucBuf_</i>	Buffer to encode into

Returns

bytes read

Definition at line 34 of file [slip.cpp](#).

16.80.2.3 Driver* Slip::GetDriver () [inline]

Return the pointer to the driver attached to this object.

Returns

Pointer to the driver attached

Definition at line 85 of file [slip.h](#).

16.80.2.4 K_USHORT Slip::ReadData (K_UCHAR * *pucChannel_*, K_CHAR * *aucBuf_*, K_USHORT *usLen_*)

Read a packet from a specified device, parse, and copy to a specified output buffer.

Parameters

<i>pucChannel_</i>	Pointer to a uchar that stores the message channel
<i>aucBuf_</i>	Buffer where the message will be decoded
<i>usLen_</i>	Length of the buffer to decode

Returns

data bytes read, 0 on failure.

Definition at line 104 of file [slip.cpp](#).

16.80.2.5 void Slip::SetDriver (Driver * *pclDriver_*) [inline]

Set the driver to attach to this object.

Parameters

<i>pclDriver_</i>	Pointer to the driver to attach
-------------------	---------------------------------

Definition at line 78 of file [slip.h](#).

16.80.2.6 void Slip::WriteData (K_UCHAR *ucChannel_*, const K_CHAR * *aucBuf_*, K_USHORT *usLen_*)

Write a packet of data in the FunkenSlip format.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

<i>ucChannel_</i>	Channel to encode the packet to
<i>aucBuf_</i>	Payload to encode
<i>usLen_</i>	Length of payload data

Definition at line 164 of file [slip.cpp](#).

16.80.2.7 void Slip::WriteVector (K_UCHAR *ucChannel_*, SlipDataVector * *astData_*, K_USHORT *usLen_*)

Write a single message composed of multiple data-vector fragments.

Allows for transmitting complex data structures without requiring buffering. This operation is zero-copy.

Parameters

<i>ucChannel_</i>	Message channel
<i>astData_</i>	Pointer to the data vector
<i>usLen_</i>	Number of elements in the data vector

Definition at line 223 of file [slip.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/slip.h](#)
- [/home/mo/mark3-source/embedded/stage/src/slip.cpp](#)

16.81 SlipDataVector Struct Reference

Data structure used for vector-based SLIP data transmission.

```
#include <slip.h>
```

Public Attributes

- K_UCHAR [ucSize](#)
Size of the data buffer.
- K_UCHAR * [pucData](#)
Pointer to the data buffer.

16.81.1 Detailed Description

Data structure used for vector-based SLIP data transmission.

Allows for building and transmitting complex data structures without having to copy data into intermediate buffers.

Definition at line 59 of file [slip.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/slip.h](#)

16.82 SlipMux Class Reference

Static-class which implements a multiplexed stream of SLIP data over a single interface.

```
#include <slip_mux.h>
```

Static Public Member Functions

- static void [Init](#) (const K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_)
Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.
- static void [InstallHandler](#) (K_UCHAR ucChannel_, Slip_Channel pfHandler_)
Install a slip handler function for the given communication channel.
- static void [MessageReceive](#) ()
Wait for a valid packet to arrive, and call the appropriate handler function for the channel the message was attached to.
- static Driver * [GetDriver](#) ()
Return the pointer of the current driver used by the [SlipMux](#) module.
- static MessageQueue * [GetQueue](#) ()
Return the pointer to the message queue attached to the slip mux channel.
- static void [SetQueue](#) (MessageQueue *pclMessageQueue_)
Set the message queue that will receive the notification when the slip mux channel has received data.
- static Slip * [GetSlip](#) ()
Return the pointer to the [SlipMux](#)' Slip object.

Static Private Attributes

- static MessageQueue * [m_pclMessageQueue](#)
- static Driver * [m_pclDriver](#)
- static Slip_Channel [m_apfChannelHandlers](#) [SLIP_CHANNEL_COUNT] = {0}
- static K_UCHAR [m_aucData](#) [SLIP_BUFFER_SIZE]
- static Semaphore [m_clSlipSem](#)
- static Slip [m_clSlip](#)

16.82.1 Detailed Description

Static-class which implements a multiplexed stream of SLIP data over a single interface.

Definition at line 43 of file [slip_mux.h](#).

16.82.2 Member Function Documentation

16.82.2.1 static Driver* SlipMux::GetDriver () [inline],[static]

Return the pointer of the current driver used by the [SlipMux](#) module.

Returns

Pointer to the current handle owned by [SlipMux](#)

Definition at line 91 of file [slip_mux.h](#).

16.82.2.2 static MessageQueue* SlipMux::GetQueue () [inline],[static]

Return the pointer to the message queue attached to the slip mux channel.

Returns

Pointer to the message Queue

Definition at line 99 of file [slip_mux.h](#).

16.82.2.3 static Slip* SlipMux::GetSlip () [inline],[static]

Return the pointer to the [SlipMux](#)' [Slip](#) object.

Returns

Pointer to the [Slip](#) object

Definition at line 117 of file [slip_mux.h](#).

16.82.2.4 void SlipMux::Init (const K_CHAR * pcDriverPath_, K_USHORT usRxSize_, K_UCHAR * aucRx_, K_USHORT usTxSize_, K_UCHAR * aucTx_) [static]

Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.

Must be called before any of the other functions in this module are called.

Parameters

<i>pcDriverPath_</i>	Filesystem path to the driver to attach to
<i>usRxSize_</i>	Size of the RX Buffer to attach to the driver
<i>aucRx_</i>	Pointer to the RX Buffer to attach to the driver
<i>usTxSize_</i>	Size of the TX Buffer to attach to the driver
<i>aucTx_</i>	Pointer to the TX Buffer to attach to the driver

Definition at line 59 of file [slip_mux.cpp](#).

16.82.2.5 void SlipMux::InstallHandler (K_UCHAR ucChannel_, Slip_Channel pfHandler_) [static]

Install a slip handler function for the given communication channel.

Parameters

<i>ucChannel_</i>	Channel to attach the handler to
<i>pfHandler_</i>	Pointer to the handler function to attach

Definition at line 76 of file [slip_mux.cpp](#).

16.82.2.6 void SlipMux::MessageReceive (void) [static]

Wait for a valid packet to arrive, and call the appropriate handler function for the channel the message was attached to.

This is essentially the entry point for a thread whose purpose is to service slip Rx data.

Definition at line 85 of file [slip_mux.cpp](#).

16.82.2.7 static void SlipMux::SetQueue (MessageQueue * *pcMessageQueue_*) [inline],[static]

Set the message queue that will receive the notification when the slip mux channel has received data.

Parameters

<i>pcMessage-Queue_</i>	Pointer to the message queue to use for notification.
-------------------------	---

Definition at line 108 of file [slip_mux.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/slip_mux.h](#)
- [/home/mo/mark3-source/embedded/stage/src/slip_mux.cpp](#)

16.83 SlipTerm Class Reference

Class implementing a simple debug terminal interface.

```
#include <slipterm.h>
```

Public Member Functions

- void [Init](#) ()
Initialize the terminal by opening a handle to the serial interface attached at /dev/tty.
- void [PrintLn](#) (const char *szLine_)
Print a string of text to the SLIP interface, multiplexed using the FunkenSlip terminal channel.
- void [PrintLn](#) (K_UCHAR ucSeverity_, const char *szLine_)
Print a string of text to the SLIP interface, but only if the current logging verbosity level is greater than or equal to the specified message severity.
- void [SetVerbosity](#) (K_UCHAR ucLevel_)
Set the logging verbosity level - the minimum severity level that will be printed to the terminal.

Private Member Functions

- K_USHORT [StrLen](#) (const char *szString_)
Quick 'n' dirty StrLen functionality used for printing the string.

Private Attributes

- K_UCHAR [m_ucVerbosity](#)
level greater than this Are not displayed.
- Slip [m_clSlip](#)
Slip object that this module interfaces with.

16.83.1 Detailed Description

Class implementing a simple debug terminal interface.

This is useful for printf style debugging.

Definition at line 40 of file [slipterm.h](#).

16.83.2 Member Function Documentation

16.83.2.1 void SlipTerm::Init (void)

Initialize the terminal by opening a handle to the serial interface attached at /dev/tty.

Must be called prior to using the print functionality.

Definition at line 26 of file [slipterm.cpp](#).

16.83.2.2 void SlipTerm::PrintLn (const char * szLine_)

Print a string of text to the SLIP interface, multiplexed using the FunkenSlip terminal channel.

Parameters

szLine_	String to print
-------------------------	-----------------

Definition at line 44 of file [slipterm.cpp](#).

16.83.2.3 void SlipTerm::PrintLn (K_UCHAR ucSeverity_, const char * szLine_)

Print a string of text to the SLIP interface, but only if the current logging verbosity level is greater than or equal to the specified message severity.

Parameters

ucSeverity_	Message severity level, 0 = highest severity
szLine_	String to print

Definition at line 56 of file [slipterm.cpp](#).

16.83.2.4 void SlipTerm::SetVerbosity (K_UCHAR ucLevel_) [inline]

Set the logging verbosity level - the minimum severity level that will be printed to the terminal.

The higher the number, the more chatty the output.

Definition at line 81 of file [slipterm.h](#).

16.83.2.5 K_USHORT SlipTerm::StrLen (const char * szString_) [private]

Quick 'n' dirty StrLen functionality used for printing the string.

Returns

Length of the string (in bytes)

Definition at line 33 of file [slipterm.cpp](#).

16.83.3 Member Data Documentation**16.83.3.1 K_UCHAR SlipTerm::m_ucVerbosity [private]**

level greater than this Are not displayed.

Verbosity level. Messages with a severity

Definition at line 92 of file [slipterm.h](#).

The documentation for this class was generated from the following files:

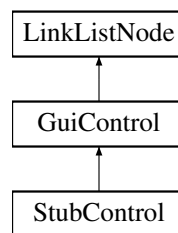
- [/home/mo/mark3-source/embedded/stage/src/slipterm.h](#)
- [/home/mo/mark3-source/embedded/stage/src/slipterm.cpp](#)

16.84 StubControl Class Reference

Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

```
#include <gui.h>
```

Inheritance diagram for StubControl:

**Public Member Functions**

- virtual void [Init](#) ()
Initailize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.

Additional Inherited Members**16.84.1 Detailed Description**

Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Definition at line 796 of file [gui.h](#).

16.84.2 Member Function Documentation

16.84.2.1 `virtual void StubControl::Activate (bool bActivate_) [inline],[virtual]`

Activate or deactivate the current control - used when switching from one active control to another.

Parameters

<i>bActivate_</i>	- true to activate, false to deactivate
-------------------	---

Implements [GuiControl](#).

Definition at line 802 of file [gui.h](#).

16.84.2.2 `virtual void StubControl::Draw () [inline],[virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 800 of file [gui.h](#).

16.84.2.3 `virtual void StubControl::Init () [inline],[virtual]`

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 799 of file [gui.h](#).

16.84.2.4 `virtual GuiReturn_t StubControl::ProcessEvent (GuiEvent_t * pstEvent_) [inline],[virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<i>pstEvent_</i>	Pointer to a struct containing the event data
------------------	---

Implements [GuiControl](#).

Definition at line 801 of file [gui.h](#).

The documentation for this class was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.85 SystemHeap Class Reference

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

```
#include <system_heap.h>
```

Static Public Member Functions

- static void [Init](#) (void)

Init Initialize the system heap prior to usage.

- static void * [Alloc](#) (K_USHORT usSize_)

Alloc allocate a block of data from the heap.

- static void [Free](#) (void *pvData_)

Free free a block of data previously allocated from the heap.

Static Private Attributes

- static K_UCHAR [m_pucRawHeap](#) [HEAP_RAW_SIZE]

Raw heap buffer.

- static [HeapConfig](#) [m_pclSystemHeapConfig](#) [HEAP_NUM_SIZES+1]

Heap configuration metadata.

- static [FixedHeap](#) [m_clSystemHeap](#)

Heap management object.

- static bool [m_bInit](#)

True if initialized, false if uninitialized.

16.85.1 Detailed Description

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

Definition at line 189 of file [system_heap.h](#).

16.85.2 Member Function Documentation

16.85.2.1 void * SystemHeap::Alloc (K_USHORT usSize_) [static]

[Alloc](#) allocate a block of data from the heap.

Parameters

usSize_	size of the block (in bytes) to allocate
-------------------------	--

Returns

pointer to a block of data allocated from the heap, or NULL on failure.

Definition at line 130 of file [system_heap.cpp](#).

16.85.2.2 void SystemHeap::Free (void * pvData_) [static]

[Free](#) free a block of data previously allocated from the heap.

Parameters

pvData_	Pointer to a block of data allocated from the system heap
-------------------------	---

Definition at line 140 of file [system_heap.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/system_heap.h](#)
- [/home/mo/mark3-source/embedded/stage/src/system_heap.cpp](#)

16.86 TextFX_t Struct Reference

Public Attributes

- K_UCHAR [ucFlags](#)
Text effects applied.
- COLOR [uBGColor](#)
Background color for opaque backgrounds.
- K_USHORT [usRotateDeg](#)
Rotation in degrees.
- K_USHORT [usScaleX100](#)
Scaling factor, fixed point modulo 100.
- K_USHORT [usScaleY100](#)
Scaling factor, fixed point modulo 100.

16.86.1 Detailed Description

Definition at line 160 of file [draw.h](#).

The documentation for this struct was generated from the following file:

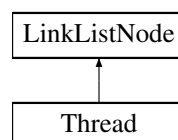
- [/home/mo/mark3-source/embedded/stage/src/draw.h](#)

16.87 Thread Class Reference

Object providing fundamental multitasking support in the kernel.

```
#include <thread.h>
```

Inheritance diagram for Thread:



Public Member Functions

- void [Init](#) (K_WORD *paucStack_, K_USHORT usStackSize_, K_UCHAR ucPriority_, [ThreadEntry_t](#) pfEntryPoint_, void *pvArg_)
Initialize a thread prior to its use.
- void [Start](#) ()
Start the thread - remove it from the stopped list, add it to the scheduler's list of threads (at the thread's set priority), and continue along.
- void [Stop](#) ()
Stop a thread that's actively scheduled without destroying its stacks.
- void [SetName](#) (const K_CHAR *szName_)
Set the name of the thread - this is purely optional, but can be useful when identifying issues that come along when multiple threads are at play in a system.
- const K_CHAR * [GetName](#) ()
- [ThreadList](#) * [GetOwner](#) (void)

- Return the [ThreadList](#) where the thread belongs when it's in the active/ready state in the scheduler.

 - [ThreadList](#) * [GetCurrent](#) (void)

Return the [ThreadList](#) where the thread is currently located.
- K_UCHAR [GetPriority](#) (void)

Return the priority of the current thread.
- K_UCHAR [GetCurPriority](#) (void)

Return the priority of the current thread.
- void [SetQuantum](#) (K_USHORT usQuantum_)

Set the thread's round-robin execution quantum.
- K_USHORT [GetQuantum](#) (void)

Get the thread's round-robin execution quantum.
- void [SetCurrent](#) ([ThreadList](#) *pclNewList_)

Set the thread's current to the specified thread list.
- void [SetOwner](#) ([ThreadList](#) *pclNewList_)

Set the thread's owner to the specified thread list.
- void [SetPriority](#) (K_UCHAR ucPriority_)

Set the priority of the [Thread](#) (running or otherwise) to a different level.
- void [InheritPriority](#) (K_UCHAR ucPriority_)

Allow the thread to run at a different priority level (temporarily) for the purpose of avoiding priority inversions.
- void [Exit](#) ()

Remove the thread from being scheduled again.
- void [SetID](#) (K_UCHAR ucID_)

Set an 8-bit ID to uniquely identify this thread.
- K_UCHAR [GetID](#) ()

Return the 8-bit ID corresponding to this thread.
- K_USHORT [GetStackSlack](#) ()

Performs a (somewhat lengthy) check on the thread stack to check the amount of stack margin (or "slack") remaining on the stack.
- K_USHORT [GetEventFlagMask](#) ()

[GetEventFlagMask](#) returns the thread's current event-flag mask, which is used in conjunction with the [EventFlag](#) blocking object type.
- void [SetEventFlagMask](#) (K_USHORT usMask_)

[SetEventFlagMask](#) Sets the active event flag bitfield mask.
- void [SetEventFlagMode](#) (EventFlagOperation_t eMode_)

[SetEventFlagMode](#) Sets the active event flag operation mode.
- EventFlagOperation_t [GetEventFlagMode](#) ()

[GetEventFlagMode](#) Returns the thread's event flag's operating mode.
- [Timer](#) * [GetTimer](#) ()

Return a pointer to the thread's timer object.
- void [SetExpired](#) (K_BOOL bExpired_)
- K_BOOL [GetExpired](#) ()

Static Public Member Functions

- static void [Sleep](#) (K_ULONG ulTimeMs_)

Put the thread to sleep for the specified time (in milliseconds).
- static void [USleep](#) (K_ULONG ulTimeUs_)

Put the thread to sleep for the specified time (in microseconds).
- static void [Yield](#) (void)

Yield the thread - this forces the system to call the scheduler and determine what thread should run next.

Private Member Functions

- void [SetPriorityBase](#) (K_UCHAR ucPriority_)

Static Private Member Functions

- static void [ContextSwitchSWI](#) (void)
This code is used to trigger the context switch interrupt.

Private Attributes

- K_WORD * [m_pwStackTop](#)
Pointer to the top of the thread's stack.
- K_WORD * [m_pwStack](#)
Pointer to the thread's stack.
- K_USHORT [m_usStackSize](#)
Size of the stack (in bytes)
- K_USHORT [m_usQuantum](#)
Thread quantum (in milliseconds)
- K_UCHAR [m_ucThreadID](#)
Thread ID.
- K_UCHAR [m_ucPriority](#)
Default priority of the thread.
- K_UCHAR [m_ucCurPriority](#)
Current priority of the thread (priority inheritance)
- [ThreadEntry_t](#) [m_pfEntryPoint](#)
The entry-point function called when the thread starts.
- void * [m_pvArg](#)
Pointer to the argument passed into the thread's entrypoint.
- const K_CHAR * [m_szName](#)
Thread name.
- K_USHORT [m_usFlagMask](#)
Event-flag mask.
- [EventFlagOperation_t](#) [m_eFlagMode](#)
Event-flag mode.
- [Timer](#) [m_clTimer](#)
Timer used for blocking-object timeouts.
- K_BOOL [m_bExpired](#)
- [ThreadList](#) * [m_pclCurrent](#)
Pointer to the thread-list where the thread currently resides.
- [ThreadList](#) * [m_pclOwner](#)
Pointer to the thread-list where the thread resides when active.

Friends

- class [ThreadPort](#)

Additional Inherited Members

16.87.1 Detailed Description

Object providing fundamental multitasking support in the kernel.

Definition at line 57 of file [thread.h](#).

16.87.2 Member Function Documentation

16.87.2.1 void Thread::ContextSwitchSWI(void) [static],[private]

This code is used to trigger the context switch interrupt.

Called whenever the kernel decides that it is necessary to swap out the current thread for the "next" thread.

Definition at line 351 of file [thread.cpp](#).

16.87.2.2 void Thread::Exit()

Remove the thread from being scheduled again.

The thread is effectively destroyed when this occurs. This is extremely useful for cases where a thread encounters an unrecoverable error and needs to be restarted, or in the context of systems where threads need to be created and destroyed dynamically.

This must not be called on the idle thread.

Definition at line 149 of file [thread.cpp](#).

16.87.2.3 K_UCHAR Thread::GetCurPriority(void) [inline]

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 160 of file [thread.h](#).

16.87.2.4 ThreadList * Thread::GetCurrent(void) [inline]

Return the [ThreadList](#) where the thread is currently located.

Returns

Pointer to the thread's current list

Definition at line 141 of file [thread.h](#).

16.87.2.5 K_USHORT Thread::GetEventFlagMask() [inline]

GetEventFlagMask returns the thread's current event-flag mask, which is used in conjunction with the [EventFlag](#) blocking object type.

Returns

A copy of the thread's event flag mask

Definition at line 313 of file [thread.h](#).

16.87.2.6 EventFlagOperation_t Thread::GetEventFlagMode () [inline]

GetEventFlagMode Returns the thread's event flag's operating mode.

Returns

The thread's event flag mode.

Definition at line 332 of file [thread.h](#).

16.87.2.7 K_UCHAR Thread::GetID () [inline]

Return the 8-bit ID corresponding to this thread.

Returns

[Thread](#)'s 8-bit ID, set by the user

Definition at line 288 of file [thread.h](#).

16.87.2.8 const K_CHAR * Thread::GetName () [inline]

Returns

Pointer to the name of the thread. If this is not set, will be NULL.

Definition at line 121 of file [thread.h](#).

16.87.2.9 ThreadList * Thread::GetOwner (void) [inline]

Return the [ThreadList](#) where the thread belongs when it's in the active/ready state in the scheduler.

Returns

Pointer to the [Thread](#)'s owner list

Definition at line 132 of file [thread.h](#).

16.87.2.10 K_UCHAR Thread::GetPriority (void) [inline]

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 151 of file [thread.h](#).

16.87.2.11 K_USHORT Thread::GetQuantum (void) [inline]

Get the thread's round-robin execution quantum.

Returns

The thread's quantum

Definition at line 179 of file [thread.h](#).

16.87.2.12 K_USHORT Thread::GetStackSlack ()

Performs a (somewhat lengthy) check on the thread stack to check the amount of stack margin (or "slack") remaining on the stack.

If you're having problems with blowing your stack, you can run this function at points in your code during development to see what operations cause problems. Also useful during development as a tool to optimally size thread stacks.

Returns

The amount of slack (unused bytes) on the stack

! ToDo: Take into account stacks that grow up

Definition at line 240 of file [thread.cpp](#).

16.87.2.13 void Thread::InheritPriority (K_UCHAR ucPriority_)

Allow the thread to run at a different priority level (temporarily) for the purpose of avoiding priority inversions.

This should only be called from within the implementation of blocking-objects.

Parameters

<i>ucPriority_</i>	New Priority to boost to.
--------------------	---------------------------

Definition at line 344 of file [thread.cpp](#).

16.87.2.14 void Thread::Init (K_WORD * paucStack_, K_USHORT usStackSize_, K_UCHAR ucPriority_, ThreadEntry_t pfEntryPoint_, void * pvArg_)

Initialize a thread prior to its use.

Initialized threads are placed in the stopped state, and are not scheduled until the thread's start method has been invoked first.

Parameters

<i>paucStack_</i>	Pointer to the stack to use for the thread
<i>usStackSize_</i>	Size of the stack (in bytes)
<i>ucPriority_</i>	Priority of the thread (0 = idle, 7 = max)
<i>pfEntryPoint_</i>	This is the function that gets called when the thread is started
<i>pvArg_</i>	Pointer to the argument passed into the thread's entrypoint function.

< Default round-robin thread quantum of 4ms

Definition at line 41 of file [thread.cpp](#).

16.87.2.15 void Thread::SetCurrent (ThreadList * pclNewList_) [inline]

Set the thread's current to the specified thread list.

Parameters

<i>pclNewList_</i>	Pointer to the threadlist to apply thread ownership
--------------------	---

Definition at line 189 of file [thread.h](#).

16.87.2.16 void Thread::SetEventFlagMask (K_USHORT usMask_) [inline]

SetEventFlagMask Sets the active event flag bitfield mask.

Parameters

<i>usMask_</i>	
----------------	--

Definition at line 319 of file [thread.h](#).

16.87.2.17 void Thread::SetEventFlagMode (EventFlagOperation_t *eMode_*) [inline]

SetEventFlagMode Sets the active event flag operation mode.

Parameters

<i>eMode_</i>	Event flag operation mode, defines the logical operator to apply to the event flag.
---------------	---

Definition at line 326 of file [thread.h](#).

16.87.2.18 void Thread::SetID (K_UCHAR *ucID_*) [inline]

Set an 8-bit ID to uniquely identify this thread.

Parameters

<i>ucID_</i>	8-bit Thread ID, set by the user
--------------	--

Definition at line 279 of file [thread.h](#).

16.87.2.19 void Thread::SetName (const K_CHAR * *szName_*) [inline]

Set the name of the thread - this is purely optional, but can be useful when identifying issues that come along when multiple threads are at play in a system.

Parameters

<i>szName_</i>	Char string containing the thread name
----------------	--

Definition at line 113 of file [thread.h](#).

16.87.2.20 void Thread::SetOwner (ThreadList * *pclNewList_*) [inline]

Set the thread's owner to the specified thread list.

Parameters

<i>pclNewList_</i>	Pointer to the threadlist to apply thread ownership
--------------------	---

Definition at line 198 of file [thread.h](#).

16.87.2.21 void Thread::SetPriority (K_UCHAR *ucPriority_*)

Set the priority of the [Thread](#) (running or otherwise) to a different level.

This activity involves re-scheduling, and must be done so with due caution, as it may effect the determinism of the system.

This should *always* be called from within a critical section to prevent system issues.

Parameters

<i>ucPriority_</i>	New priority of the thread
--------------------	----------------------------

Definition at line 301 of file [thread.cpp](#).

16.87.2.22 void Thread::SetPriorityBase (K_UCHAR *ucPriority_*) [private]

Parameters

<i>ucPriority_</i>	
--------------------	--

Definition at line 291 of file [thread.cpp](#).

16.87.2.23 void Thread::SetQuantum (K_USHORT *usQuantum_*) [inline]

Set the thread's round-robin execution quantum.

Parameters

<i>usQuantum_</i>	Thread's execution quantum (in milliseconds)
-------------------	--

Definition at line 170 of file [thread.h](#).

16.87.2.24 void Thread::Sleep (K_ULONG *ulTimeMs_*) [static]

Put the thread to sleep for the specified time (in milliseconds).

Actual time slept may be longer (but not less than) the interval specified.

Parameters

<i>ulTimeMs_</i>	Time to sleep (in ms)
------------------	-----------------------

Definition at line 195 of file [thread.cpp](#).

16.87.2.25 void Thread::Stop (void)

Stop a thread that's actively scheduled without destroying its stacks.

Stopped threads can be restarted using the [Start\(\)](#) API.

Definition at line 121 of file [thread.cpp](#).

16.87.2.26 void Thread::USleep (K_ULONG *ulTimeUs_*) [static]

Put the thread to sleep for the specified time (in microseconds).

Actual time slept may be longer (but not less than) the interval specified.

Parameters

<i>ulTimeUs_</i>	Time to sleep (in microseconds)
------------------	---------------------------------

Definition at line 217 of file [thread.cpp](#).

16.87.2.27 void Thread::Yield (void) [static]

Yield the thread - this forces the system to call the scheduler and determine what thread should run next.

This is typically used when threads are moved in and out of the scheduler.

Definition at line 261 of file [thread.cpp](#).

The documentation for this class was generated from the following files:

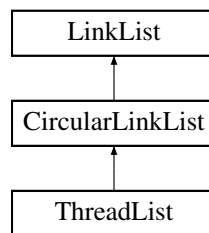
- [/home/mo/mark3-source/embedded/stage/src/thread.h](#)
- [/home/mo/mark3-source/embedded/stage/src/thread.cpp](#)

16.88 ThreadList Class Reference

This class is used for building thread-management facilities, such as schedulers, and blocking objects.

```
#include <threadlist.h>
```

Inheritance diagram for ThreadList:



Public Member Functions

- [ThreadList](#) ()
Default constructor - zero-initializes the data.
- void [SetPriority](#) (K_UCHAR ucPriority_)
Set the priority of this threadlist (if used for a scheduler).
- void [SetFlagPointer](#) (K_UCHAR *pucFlag_)
Set the pointer to a bitmap to use for this threadlist.
- void [Add](#) (LinkListNode *node_)
Add a thread to the threadlist.
- void [Add](#) (LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_)
Add a thread to the threadlist, specifying the flag and priority at the same time.
- void [Remove](#) (LinkListNode *node_)
Remove the specified thread from the threadlist.
- [Thread *](#) [HighestWaiter](#) ()
Return a pointer to the highest-priority thread in the thread-list.

Private Attributes

- K_UCHAR [m_ucPriority](#)
Priority of the threadlist.
- K_UCHAR * [m_pucFlag](#)
Pointer to the bitmap/flag to set when used for scheduling.

Additional Inherited Members

16.88.1 Detailed Description

This class is used for building thread-management facilities, such as schedulers, and blocking objects.

Definition at line 34 of file [threadlist.h](#).

16.88.2 Member Function Documentation

16.88.2.1 void ThreadList::Add (LinkListNode * *node_*) [virtual]

Add a thread to the threadlist.

Parameters

<i>node_</i>	Pointer to the thread (link list node) to add to the list
--------------	---

Reimplemented from [CircularLinkList](#).

Definition at line 46 of file [threadlist.cpp](#).

16.88.2.2 void ThreadList::Add (LinkListNode * *node_*, K_UCHAR * *pucFlag_*, K_UCHAR *ucPriority_*)

Add a thread to the threadlist, specifying the flag and priority at the same time.

Parameters

<i>node_</i>	Pointer to the thread to add (link list node)
<i>pucFlag_</i>	Pointer to the bitmap flag to set (if used in a scheduler context), or NULL for non-scheduler.
<i>ucPriority_</i>	Priority of the threadlist

Definition at line 62 of file [threadlist.cpp](#).

16.88.2.3 Thread * ThreadList::HighestWaiter ()

Return a pointer to the highest-priority thread in the thread-list.

Returns

Pointer to the highest-priority thread

Definition at line 87 of file [threadlist.cpp](#).

16.88.2.4 void ThreadList::Remove (LinkListNode * *node_*) [virtual]

Remove the specified thread from the threadlist.

Parameters

<i>node_</i>	Pointer to the thread to remove
--------------	---------------------------------

Reimplemented from [CircularLinkList](#).

Definition at line 71 of file [threadlist.cpp](#).

16.88.2.5 void ThreadList::SetFlagPointer (K_UCHAR * *pucFlag_*)

Set the pointer to a bitmap to use for this threadlist.

Once again, only needed when the threadlist is being used for scheduling purposes.

Parameters

<i>pucFlag_</i>	Pointer to the bitmap flag
-----------------	----------------------------

Definition at line 40 of file [threadlist.cpp](#).

16.88.2.6 void ThreadList::SetPriority (K_UCHAR *ucPriority_*)

Set the priority of this threadlist (if used for a scheduler).

Parameters

<i>ucPriority_</i>	Priority level of the thread list
--------------------	-----------------------------------

Definition at line 34 of file [threadlist.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/threadlist.h](#)
- [/home/mo/mark3-source/embedded/stage/src/threadlist.cpp](#)

16.89 ThreadPort Class Reference

Class defining the architecture specific functions required by the kernel.

```
#include <threadport.h>
```

Static Public Member Functions

- static void [StartThreads](#) ()
Function to start the scheduler, initial threads, etc.

Static Private Member Functions

- static void [InitStack](#) ([Thread](#) *pstThread_)
Initialize the thread's stack.

Friends

- class [Thread](#)

16.89.1 Detailed Description

Class defining the architecture specific functions required by the kernel.

This is limited (at this point) to a function to start the scheduler, and a function to initialize the default stack-frame for a thread.

Definition at line 167 of file [threadport.h](#).

16.89.2 Member Function Documentation

16.89.2.1 void [ThreadPort::InitStack](#) ([Thread](#) * *pstThread_*) [static], [private]

Initialize the thread's stack.

Parameters

<i>pstThread_</i>	Pointer to the thread to initialize
-------------------	-------------------------------------

Definition at line 37 of file [threadport.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/threadport.h](#)
- [/home/mo/mark3-source/embedded/stage/src/threadport.cpp](#)

16.90 Tile_8x8 Class Reference

Public Member Functions

- void [LoadTile](#) ([TileDef_t](#) *pstTileDef_)
LoadTile.
- void [Render](#) ([GraphicsDriver](#) *pclDriver_, K_USHORT usX_, K_USHORT usY_)
Render.

Private Attributes

- COLOR [m_auTileBuffer](#) [TILE_8x8_BUFFER_SIZE]
m_auTileBuffer Object's local storage for tile data
- K_UCHAR [m_ucWidth](#)
m_ucWidth Width of the tile (may be smaller than width of buffer)
- K_UCHAR [m_ucHeight](#)
m_ucHeight Height of the tile (may be smaler than the height of buffer)

16.90.1 Detailed Description

Definition at line 63 of file [tiles.h](#).

16.90.2 Member Function Documentation

16.90.2.1 void Tile_8x8::LoadTile (TileDef_t * pstTileDef_)

LoadTile.

Load the tile specified by pstTileDef_ into memory. This takes some time as it parses the indexed colors, does a lookup, and then writes to the local tile buffer. Once a tile has been loaded, it can be rendered any number of times.

Parameters

<i>pstTileDef_</i>	Pointer to a struct containing configuration data for the tile to be loaded.
--------------------	--

Definition at line 24 of file [tiles.cpp](#).

16.90.2.2 void Tile_8x8::Render (GraphicsDriver * pclDriver_, K_USHORT usX_, K_USHORT usY_)

Render.

Render loaded tile data to a specific location on a specified display.

Parameters

<i>pclDriver_</i>	Pointer to the graphics driver to render with
<i>usX_</i>	Leftmost pixel index
<i>usY_</i>	Topmost pixel index

Definition at line 51 of file [tiles.cpp](#).

The documentation for this class was generated from the following files:

- /home/mo/mark3-source/embedded/stage/src/tiles.h
- /home/mo/mark3-source/embedded/stage/src/tiles.cpp

16.91 TileDef_t Struct Reference

[TileDef_t](#) Structure defining parameters for a color-indexed tile.

```
#include <tiles.h>
```

Public Attributes

- [TileFormat_t m_eFormat](#)
Color-indexing of the tile (bits-per-pixel)
- [K_UCHAR * m_pucData](#)
Pointer to color-indexed tile data.
- [COLOR * m_puPalette](#)
Pointer to a palette assigned to this tile.
- [K_UCHAR m_ucHeight](#)
Height of the tile (in pixels)
- [K_UCHAR m_ucWidth](#)
Width of the tile (in pixels)

16.91.1 Detailed Description

[TileDef_t](#) Structure defining parameters for a color-indexed tile.

Definition at line 48 of file [tiles.h](#).

The documentation for this struct was generated from the following file:

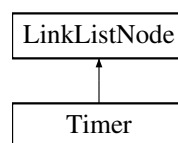
- /home/mo/mark3-source/embedded/stage/src/tiles.h

16.92 Timer Class Reference

[Timer](#) - an event-driven execution context based on a specified time interval.

```
#include <timerlist.h>
```

Inheritance diagram for [Timer](#):



Public Member Functions

- [Timer](#) ()
Default Constructor - zero-initializes all internal data.
- void [Init](#) ()
Re-initialize the [Timer](#) to default values.
- void [Start](#) ([K_UCHAR](#) bRepeat_, [K_ULONG](#) ulIntervalMs_, [TimerCallback_t](#) pfCallback_, void *pvData_)
Start a timer using default ownership, using repeats as an option, and millisecond resolution.
- void [Start](#) ([K_UCHAR](#) bRepeat_, [K_ULONG](#) ulIntervalMs_, [K_ULONG](#) ulToleranceMs_, [TimerCallback_t](#) pfCallback_, void *pvData_)

- Start a timer using default ownership, using repeats as an option, and millisecond resolution.*

 - void [Stop](#) ()

Stop a timer already in progress.
 - void [SetFlags](#) (K_UCHAR ucFlags_)

Set the timer's flags based on the bits in the ucFlags_ argument.
 - void [SetCallback](#) (TimerCallback_t pfCallback_)

Define the callback function to be executed on expiry of the timer.
 - void [SetData](#) (void *pvData_)

Define a pointer to be sent to the timer callbacak on timer expiry.
 - void [SetOwner](#) (Thread *pclOwner_)

Set the owner-thread of this timer object (all timers must be owned by a thread).
 - void [SetIntervalTicks](#) (K_ULONG ulTicks_)

Set the timer expiry in system-ticks (platform specific!)
 - void [SetIntervalSeconds](#) (K_ULONG ulSeconds_)

! The next three cost us 330 bytes of flash on AVR...
 - K_ULONG [GetInterval](#) ()
 - void [SetIntervalMSeconds](#) (K_ULONG ulMSeconds_)

Set the timer expiry interval in milliseconds (platform agnostic)
 - void [SetIntervalUSeconds](#) (K_ULONG ulUSeconds_)

Set the timer expiry interval in microseconds (platform agnostic)
 - void [SetTolerance](#) (K_ULONG ulTicks_)

Set the timer's maximum tolerance in order to synchronize timer processing with other timers in the system.

Private Attributes

- K_UCHAR [m_ucFlags](#)

Flags for the timer, defining if the timer is one-shot or repeated.
- TimerCallback_t [m_pfCallback](#)

Pointer to the callback function.
- K_ULONG [m_ulInterval](#)

Interval of the timer in timer ticks.
- K_ULONG [m_ulTimeLeft](#)

Time remaining on the timer.
- K_ULONG [m_ulTimerTolerance](#)

Maximum tolerance (used for timer harmonization)
- Thread * [m_pclOwner](#)

Pointer to the owner thread.
- void * [m_pvData](#)

Pointer to the callback data.

Friends

- class [TimerList](#)

Additional Inherited Members

16.92.1 Detailed Description

[Timer](#) - an event-driven execution context based on a specified time interval.

This inherits from a [LinkListNode](#) for ease of management by a global [TimerList](#) object.

Definition at line 98 of file [timerlist.h](#).

16.92.2 Member Function Documentation

16.92.2.1 void Timer::SetCallback (TimerCallback_t *pfCallback_*) [inline]

Define the callback function to be executed on expiry of the timer.

Parameters

<i>pfCallback_</i>	Pointer to the callback function to call
--------------------	--

Definition at line 159 of file [timerlist.h](#).

16.92.2.2 void Timer::SetData (void * *pvData_*) [inline]

Define a pointer to be sent to the timer callbacak on timer expiry.

Parameters

<i>pvData_</i>	Pointer to data to pass as argument into the callback
----------------	---

Definition at line 168 of file [timerlist.h](#).

16.92.2.3 void Timer::SetFlags (K_UCHAR *ucFlags_*) [inline]

Set the timer's flags based on the bits in the *ucFlags_* argument.

Parameters

<i>ucFlags_</i>	Flags to assign to the timer object. <code>TIMERLIST_FLAG_ONE_SHOT</code> for a one-shot timer, 0 for a continuous timer.
-----------------	---

Definition at line 150 of file [timerlist.h](#).

16.92.2.4 void Timer::SetIntervalMSeconds (K_ULONG *uIMSeconds_*)

Set the timer expiry interval in milliseconds (platform agnostic)

Parameters

<i>uIMSeconds_</i>	Time in milliseconds
--------------------	----------------------

Definition at line 297 of file [timerlist.cpp](#).

16.92.2.5 void Timer::SetIntervalSeconds (K_ULONG *uSeconds_*)

! The next three cost us 330 bytes of flash on AVR...

Set the timer expiry interval in seconds (platform agnostic)

Parameters

<i>uSeconds_</i>	Time in seconds
------------------	-----------------

Definition at line 291 of file [timerlist.cpp](#).

16.92.2.6 void Timer::SetIntervalTicks (K_ULONG *uTicks_*)

Set the timer expiry in system-ticks (platform specific!)

Parameters

<i>ulTicks_</i>	Time in ticks
-----------------	---------------

Definition at line 283 of file [timerlist.cpp](#).

16.92.2.7 void Timer::SetIntervalUSecds (K_ULONG ulUSecds_)

Set the timer expiry interval in microseconds (platform agnostic)

Parameters

<i>ulUSecds_</i>	Time in microseconds
------------------	----------------------

Definition at line 303 of file [timerlist.cpp](#).

16.92.2.8 void Timer::SetOwner (Thread * pOwner_) [inline]

Set the owner-thread of this timer object (all timers must be owned by a thread).

Parameters

<i>pOwner_</i>	Owner thread of this timer object
----------------	-----------------------------------

Definition at line 178 of file [timerlist.h](#).

16.92.2.9 void Timer::SetTolerance (K_ULONG ulTicks_)

Set the timer's maximum tolerance in order to synchronize timer processing with other timers in the system.

Parameters

<i>ulTicks_</i>	Maximum tolerance in ticks
-----------------	----------------------------

Definition at line 309 of file [timerlist.cpp](#).

16.92.2.10 void Timer::Start (K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void * pvData_)

Start a timer using default ownership, using repeats as an option, and millisecond resolution.

Parameters

<i>bRepeat_</i>	0 - timer is one-shot. 1 - timer is repeating.
<i>ulIntervalMs_</i>	- Interval of the timer in milliseconds
<i>pfCallback_</i>	- Function to call on timer expiry
<i>pvData_</i>	- Data to pass into the callback function

Definition at line 252 of file [timerlist.cpp](#).

16.92.2.11 void Timer::Start (K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_, TimerCallback_t pfCallback_, void * pvData_)

Start a timer using default ownership, using repeats as an option, and millisecond resolution.

Parameters

<i>bRepeat_</i>	0 - timer is one-shot. 1 - timer is repeating.
-----------------	--

<i>ulIntervalMs_</i>	- Interval of the timer in milliseconds
<i>ulToleranceMs</i>	- Allow the timer expiry to be delayed by an additional maximum time, in order to have as many timers expire at the same time as possible.
<i>pfCallback_</i>	- Function to call on timer expiry
<i>pvData_</i>	- Data to pass into the callback function

Definition at line 270 of file [timerlist.cpp](#).

16.92.2.12 void Timer::Stop (void)

Stop a timer already in progress.

Has no effect on timers that have already been stopped.

Definition at line 277 of file [timerlist.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/timerlist.h](#)
- [/home/mo/mark3-source/embedded/stage/src/timerlist.cpp](#)

16.93 TimerEvent_t Struct Reference

[Timer](#) UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_USHORT [usTicks](#)
Number of clock ticks (arbitrary) that have elapsed.

16.93.1 Detailed Description

[Timer](#) UI event structure.

Definition at line 177 of file [gui.h](#).

The documentation for this struct was generated from the following file:

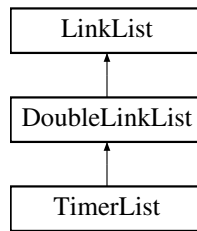
- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.94 TimerList Class Reference

[TimerList](#) class - a doubly-linked-list of timer objects.

```
#include <timerlist.h>
```

Inheritance diagram for TimerList:



Public Member Functions

- void [Init](#) ()
Initialize the [TimerList](#) object.
- void [Add](#) ([Timer](#) *pclListNode_)
Add a timer to the [TimerList](#).
- void [Remove](#) ([Timer](#) *pclListNode_)
Remove a timer from the [TimerList](#), cancelling its expiry.
- void [Process](#) ()
Process all timers in the timerlist as a result of the timer expiring.

Private Attributes

- K_ULONG [m_ulNextWakeup](#)
The time (in system clock ticks) of the next wakeup event.
- K_UCHAR [m_bTimerActive](#)
Whether or not the timer is active.

Additional Inherited Members

16.94.1 Detailed Description

[TimerList](#) class - a doubly-linked-list of timer objects.

Definition at line 260 of file [timerlist.h](#).

16.94.2 Member Function Documentation

16.94.2.1 void [TimerList::Add](#) ([Timer](#) * [pclListNode_](#))

Add a timer to the [TimerList](#).

Parameters

pclListNode_	Pointer to the Timer to Add
------------------------------	---

Definition at line 48 of file [timerlist.cpp](#).

16.94.2.2 void [TimerList::Init](#) (void)

Initialize the [TimerList](#) object.

Must be called before using the object.

Definition at line 41 of file [timerlist.cpp](#).

16.94.2.3 void TimerList::Process (void)

Process all timers in the timerlist as a result of the timer expiring.

This will select a new timer epoch based on the next timer to expire. ToDo - figure out if we need to deal with any overtime here.

Definition at line 113 of file [timerlist.cpp](#).

16.94.2.4 void TimerList::Remove (Timer * pcListNode_)

Remove a timer from the [TimerList](#), cancelling its expiry.

Parameters

pcListNode_	Pointer to the Timer to remove
-----------------------------	--

Definition at line 96 of file [timerlist.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/timerlist.h](#)
- [/home/mo/mark3-source/embedded/stage/src/timerlist.cpp](#)

16.95 TimerScheduler Class Reference

"Static" Class used to interface a global [TimerList](#) with the rest of the kernel.

```
#include <timerlist.h>
```

Static Public Member Functions

- static void [Init](#) ()
Initialize the timer scheduler.
- static void [Add](#) ([Timer](#) *pcListNode_)
Add a timer to the timer scheduler.
- static void [Remove](#) ([Timer](#) *pcListNode_)
Remove a timer from the timer scheduler.
- static void [Process](#) ()
This function must be called on timer expiry (from the timer's ISR context).

Static Private Attributes

- static [TimerList](#) [m_cTimerList](#)
[TimerList](#) object manipulated by the [Timer Scheduler](#).

16.95.1 Detailed Description

"Static" Class used to interface a global [TimerList](#) with the rest of the kernel.

Definition at line 310 of file [timerlist.h](#).

16.95.2 Member Function Documentation

16.95.2.1 `void TimerScheduler::Add (Timer * pcListNode_) [inline],[static]`

Add a timer to the timer scheduler.

Adding a timer implicitly starts the timer as well.

Parameters

<i>pcListNode_</i>	Pointer to the timer list node to add
--------------------	---------------------------------------

Definition at line 329 of file [timerlist.h](#).

16.95.2.2 `void TimerScheduler::Init (void) [inline],[static]`

Initialize the timer scheduler.

Must be called before any timer, or timer-derived functions are used.

Definition at line 319 of file [timerlist.h](#).

16.95.2.3 `void TimerScheduler::Process (void) [inline],[static]`

This function must be called on timer expiry (from the timer's ISR context).

This will result in all timers being updated based on the epoch that just elapsed. New timer epochs are set based on the next timer to expire.

Definition at line 351 of file [timerlist.h](#).

16.95.2.4 `void TimerScheduler::Remove (Timer * pcListNode_) [inline],[static]`

Remove a timer from the timer scheduler.

May implicitly stop the timer if this is the only active timer scheduled.

Parameters

<i>pcListNode_</i>	Pointer to the timer list node to remove
--------------------	--

Definition at line 340 of file [timerlist.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/timerlist.h](#)
- [/home/mo/mark3-source/embedded/stage/src/timerlist.cpp](#)

16.96 Token_t Struct Reference

Token descriptor struct format.

```
#include <memutil.h>
```

Public Attributes

- `const K_CHAR * pcToken`
Pointer to the beginning of the token string.
- `K_UCHAR ucLen`
Length of the token (in bytes)

16.96.1 Detailed Description

Token descriptor struct format.

Definition at line 32 of file [memutil.h](#).

The documentation for this struct was generated from the following file:

- [/home/mo/mark3-source/embedded/stage/src/memutil.h](#)

16.97 TouchEvent_t Struct Reference

Touch UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_USHORT [usX](#)
Absolute touch location (pixels)
- K_USHORT [usY](#)
Absolute touch location (pixels)
- union {
 K_USHORT [ucFlags](#)
 Modifier flags.
 struct {
 unsigned int [bTouch](#):1
 Whether or not touch is up or down.
 }
};

16.97.1 Detailed Description

Touch UI event structure.

Definition at line 125 of file [gui.h](#).

The documentation for this struct was generated from the following file:

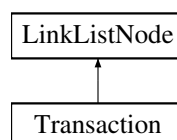
- [/home/mo/mark3-source/embedded/stage/src/gui.h](#)

16.98 Transaction Class Reference

The [Transaction](#) class.

```
#include <transaction.h>
```

Inheritance diagram for Transaction:



Public Member Functions

- void [Set](#) (K_USHORT usCode_, void *pvData_)
Set.
- K_USHORT [GetCode](#) ()
GetCode.
- void * [GetData](#) ()
GetData.

Private Attributes

- K_USHORT [m_usCode](#)
Data code, defined by the blocking object using transactions.
- void * [m_pvData](#)
Abstract data, which is defined by the code.

Additional Inherited Members

16.98.1 Detailed Description

The [Transaction](#) class.

The [Transaction](#) class implements "kernel transaction" functionality used by blocking objects within the kernel.

Each [Transaction](#) object is essentially a FIFO node, which is used to represent an operation that takes place on a blocking object. These operations include things like posting or pending a semaphore, claiming or releasing a mutex, or a thread timeout on a blocking object. Transactions are used exclusively with [TransactionQueue](#)'s to serialize access to blocking objects in order to implement lockless kernel operations with interrupts enabled.

For simplicity, each transaction is implemented as a simple Key/Value pair - the "Code" value is interpreted differently based on the type of blocking object, and the "Data" value is depending on the value held in the code. For examples of how Transactions are used, see the kernel, mutex and event-flag code.

Definition at line 51 of file [transaction.h](#).

16.98.2 Member Function Documentation

16.98.2.1 K_USHORT Transaction::GetCode () [inline]

[GetCode](#).

Return the value held by the Code field

Returns

value of the Code field

Definition at line 75 of file [transaction.h](#).

16.98.2.2 void* Transaction::GetData () [inline]

[GetData](#).

Return the abstract data value held in the object

Returns

Abstract data value held in the object

Definition at line 87 of file [transaction.h](#).

16.98.2.3 void Transaction::Set (K_USHORT *usCode_*, void * *pvData_*) [inline]

Set.

Provide access to set the code/data fields in the object

Parameters

<i>usCode_</i>	Code value to set
<i>pvData_</i>	Abstract data value to set

Definition at line 62 of file [transaction.h](#).

The documentation for this class was generated from the following file:

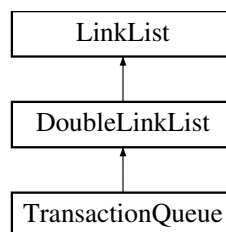
- [/home/mo/mark3-source/embedded/stage/src/transaction.h](#)

16.99 TransactionQueue Class Reference

The [TransactionQueue](#) class.

```
#include <transaction.h>
```

Inheritance diagram for TransactionQueue:

**Public Member Functions**

- void [Enqueue](#) (K_USHORT *usData_*, void **pvData_*)
Enqueue.
- [Transaction *](#) [Dequeue](#) ()
Dequeue.
- void [Finish](#) ([Transaction](#) **pclTransaction_*)
Finish.

Static Public Member Functions

- static void [GlobalQueueInit](#) ()
GlobalQueueInit.

Static Private Attributes

- static [DoubleLinkedList m_clGlobalQueue](#)
List object used to manage all transactions.
- static [Transaction m_aclTransactions \[TRANSACTION_QUEUE_SIZE\]](#)
Static array of objects managed in the above list.

Additional Inherited Members

16.99.1 Detailed Description

The [TransactionQueue](#) class.

A kernel transaction queue is a construct used to build blocking objects which disable interrupts for as short a period of time as possible. Instead of disabling interrupts for the duration of a blocking object operation (i.e. mutex claim or semaphore post), we instead serialize access to the object using a FIFO containing a list of pending actions. Coupled with [Atomic](#) locking operations, the kernel can guarantee that only one thread has permission to process the object's transaction queue, while all other concurrent threads/interrupts (which then fail to claim the object's lock) are only allowed to add transactions to it. In this way, we can keep interrupts enabled for the vast majority of kernel/blocking-object calls, resulting in a much more deterministic, responsive system.

Transactions are very short-lived - i.e. a queue will only have more than 1 pending transaction if pre-empted by interrupts during queue processing within a kernel call. As a result, we maintain a small, global pool of transaction objects which are allocated as-necessary in order to service demand. These [Transaction](#) objects are shared among all blocking objects within the system.

Typical usage of a [TransactionQueue](#) object is as follows:

```
Enqueue(code, data); // Add a new node to the queue to be processed after
```

```
// – somewhere else in the code –
```

```
// Process the queue, one node at a time Transaction *pclTransaction; while ((pclTransaction = Dequeue()) != 0) { //
Do something with the transaction data MyProcessFuntion(pclTransaction);
```

```
// Return the object back to the global queue when done. Finish(pclTransaction); }
```

Definition at line 138 of file [transaction.h](#).

16.99.2 Member Function Documentation

16.99.2.1 [Transaction](#) * [TransactionQueue::Dequeue](#) ()

Dequeue.

Pops the first item in the queue, returning its pointer back to the caller.

Note - [Dequeue\(\)](#) does not return the node back to the global pool. Once the transaction has been processed, it must be returned back by calling the [Finish\(\)](#) method.

Returns

Pointer to the head node in the list, 0 if empty

Definition at line 56 of file [transaction.cpp](#).

16.99.2.2 void [TransactionQueue::Enqueue](#) (K_USHORT *usData_*, void * *pvData_*)

Enqueue.

Enqueue a new entry to the tail of the transaction queue. This pops a node from the global transaction pool, populates it with the data in the fields, and adds the node to the end of this queue.

Parameters

<i>usData_</i>	Data value to encode
<i>pvData_</i>	Abstract data associated with the node

Definition at line 37 of file [transaction.cpp](#).

16.99.2.3 void TransactionQueue::Finish (Transaction * *pclTransaction_*)

Finish.

Return a previously dequeued transaction object back to the global transaction queue. Any Dequeue'd object must be returned by calling this function to avoid leaks.

Parameters

<i>pclTransaction_</i>	Pointer to a transaction object to return back to the queue.
------------------------	--

Definition at line 72 of file [transaction.cpp](#).

16.99.2.4 void TransactionQueue::GlobalQueueInit () [static]

GlobalQueueInit.

This static method is called to initialize the global transaction pool and its included transaction objects.

Definition at line 28 of file [transaction.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/transaction.h](#)
- [/home/mo/mark3-source/embedded/stage/src/transaction.cpp](#)

16.100 UnitTest Class Reference

Class used to implement a simple unit-testing framework.

```
#include <unit_test.h>
```

Public Member Functions

- void [SetName](#) (const K_CHAR *szName_)
Set the name of the test object.
- void [Start](#) ()
Start a new test iteration.
- void [Pass](#) ()
Stop the current iteration (if started), and register that the test was successful.
- void [Fail](#) ()
Stop the current iterations (if started), and register that the current test failed.
- void **ExpectTrue** (bool bExpression_)
- void **ExpectFalse** (bool bExpression_)
- void **ExpectEquals** (bool bVal_, bool bExpression_)
- void **ExpectEquals** (K_UCHAR ucVal_, K_UCHAR ucExpression_)
- void **ExpectEquals** (K_USHORT usVal_, K_USHORT usExpression_)
- void **ExpectEquals** (K_ULONG ulVal_, K_ULONG ulExpression_)
- void **ExpectEquals** (K_CHAR cVal_, K_CHAR cExpression_)
- void **ExpectEquals** (K_SHORT sVal_, K_SHORT sExpression_)

- void **ExpectEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectEquals** (void *pvVal_, void *pvExpression_)
- void **ExpectFailTrue** (bool bExpression_)
- void **ExpectFailFalse** (bool bExpression_)
- void **ExpectFailEquals** (bool bVal_, bool bExpression_)
- void **ExpectFailEquals** (K_UCHAR ucVal_, K_UCHAR ucExpression_)
- void **ExpectFailEquals** (K_USHORT usVal_, K_USHORT usExpression_)
- void **ExpectFailEquals** (K_ULONG ulVal_, K_ULONG ulExpression_)
- void **ExpectFailEquals** (K_CHAR cVal_, K_CHAR cExpression_)
- void **ExpectFailEquals** (K_SHORT sVal_, K_SHORT sExpression_)
- void **ExpectFailEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectFailEquals** (void *pvVal_, void *pvExpression_)
- void **ExpectGreaterThan** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectLessThan** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectGreaterThanEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectLessThanEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectFailGreaterThan** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectFailLessThan** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectFailGreaterThanEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **ExpectFailLessThanEquals** (K_LONG IVal_, K_LONG IExpression_)
- void **Complete** ()
Complete the test.
- const K_CHAR * **GetName** ()
Get the name of the tests associated with this object.
- K_BOOL **GetResult** ()
Return the result of the last test.
- K_USHORT **GetPassed** ()
Return the total number of test points/iterations passed.
- K_USHORT **GetFailed** ()
Return the number of failed test points/iterations.
- K_USHORT **GetTotal** ()
Return the total number of iterations/test-points executed.

Private Attributes

- const K_CHAR * **m_szName**
Name of the tests performed.
- K_BOOL **m_bIsActive**
Whether or not the test is active.
- K_UCHAR **m_bComplete**
Whether or not the test is complete.
- K_BOOL **m_bStatus**
Status of the last-run test.
- K_USHORT **m_usIterations**
Number of iterations executed.
- K_USHORT **m_usPassed**
Number of iterations that have passed.

16.100.1 Detailed Description

Class used to implement a simple unit-testing framework.

Definition at line 28 of file [unit_test.h](#).

16.100.2 Member Function Documentation

16.100.2.1 void UnitTest::Complete () [inline]

Complete the test.

Once a test has been completed, no new iterations can be started (i.e [Start\(\)](#)/[Pass\(\)](#)/[Fail\(\)](#) will have no effect).

Definition at line [157](#) of file [unit_test.h](#).

16.100.2.2 K_USHORT UnitTest::GetFailed () [inline]

Return the number of failed test points/iterations.

Returns

Failed test point/iteration count

Definition at line [193](#) of file [unit_test.h](#).

16.100.2.3 const K_CHAR * UnitTest::GetName () [inline]

Get the name of the tests associated with this object.

Returns

Name of the test

Definition at line [166](#) of file [unit_test.h](#).

16.100.2.4 K_USHORT UnitTest::GetPassed () [inline]

Return the total number of test points/iterations passed.

Returns

Count of all successful test points/iterations

Definition at line [184](#) of file [unit_test.h](#).

16.100.2.5 K_BOOL UnitTest::GetResult () [inline]

Return the result of the last test.

Returns

Status of the last run test (false = fail, true = pass)

Definition at line [175](#) of file [unit_test.h](#).

16.100.2.6 K_USHORT UnitTest::GetTotal () [inline]

Return the total number of iterations/test-points executed.

Returns

Total number of iterations/test-points executed

Definition at line [202](#) of file [unit_test.h](#).

16.100.2.7 void UnitTest::SetName (const K_CHAR * *szName_*) [inline]

Set the name of the test object.

Parameters

<i>szName_</i>	Name of the tests associated with this object
----------------	---

Definition at line 41 of file [unit_test.h](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/unit_test.h](#)
- [/home/mo/mark3-source/embedded/stage/src/unit_test.cpp](#)

16.101 WriteBuffer16 Class Reference

This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

```
#include <writebuf16.h>
```

Public Member Functions

- void [SetBuffers](#) (K_USHORT *pusData_, K_USHORT usSize_)
Assign the data to be used as storage for this circular buffer.
- void [SetCallback](#) ([WriteBufferCallback](#) pfCallback_)
Set the callback function to be called when the buffer hits 50% of its capacity, and again when the buffer rolls over completely.
- void [WriteData](#) (K_USHORT *pusBuf_, K_USHORT usLen_)
Write an array of values to the circular buffer.
- void [WriteVector](#) (K_USHORT **ppusBuf_, K_USHORT *pusLen_, K_UCHAR ucCount_)
Write a multi-part vector to the circular buffer.

Private Attributes

- K_USHORT * [m_pusData](#)
Pointer to the circular buffer data.
- volatile K_USHORT [m_usSize](#)
Size of the buffer.
- volatile K_USHORT [m_usHead](#)
Current head element (where data is written)
- volatile K_USHORT [m_usTail](#)
Current tail element (where data is read)
- [WriteBufferCallback](#) [m_pfCallback](#)
Buffer callback function.

16.101.1 Detailed Description

This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

We use it for implementing a debug print journal.

Definition at line 37 of file [writebuf16.h](#).

16.101.2 Member Function Documentation

16.101.2.1 void WriteBuffer16::SetBuffers (K_USHORT * *pusData_*, K_USHORT *usSize_*) [inline]

Assign the data to be used as storage for this circular buffer.

Parameters

<i>pusData_</i>	Pointer to the array of data to be managed as a circular buffer by this object.
<i>usSize_</i>	Size of the buffer in 16-bit elements

Definition at line 50 of file [writebuf16.h](#).

16.101.2.2 void WriteBuffer16::SetCallback (WriteBufferCallback *pfCallback_*) [inline]

Set the callback function to be called when the buffer hits 50% of its capacity, and again when the buffer rolls over completely.

Parameters

<i>pfCallback_</i>	Function pointer to call whenever the buffer has reached 50% capacity, or has rolled over completely.
--------------------	---

Definition at line 69 of file [writebuf16.h](#).

16.101.2.3 void WriteBuffer16::WriteData (K_USHORT * *pusBuf_*, K_USHORT *usLen_*)

Write an array of values to the circular buffer.

Parameters

<i>pusBuf_</i>	Source data array to write to the circular buffer
<i>usLen_</i>	Length of the source data array in 16-bit elements

Definition at line 25 of file [writebuf16.cpp](#).

16.101.2.4 void WriteBuffer16::WriteVector (K_USHORT ** *ppusBuf_*, K_USHORT * *pusLen_*, K_UCHAR *ucCount_*)

Write a multi-part vector to the circular buffer.

Parameters

<i>ppusBuf_</i>	Pointer to the array of source data pointers
<i>pusLen_</i>	Array of buffer lengths
<i>ucCount_</i>	Number of source-data arrays to write to the buffer

Definition at line 37 of file [writebuf16.cpp](#).

The documentation for this class was generated from the following files:

- [/home/mo/mark3-source/embedded/stage/src/writebuf16.h](#)
- [/home/mo/mark3-source/embedded/stage/src/writebuf16.cpp](#)

File Documentation

Basic Atomic Operations.

Basic Atomic Operations.

```

00001 /*=====*/
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "mark3cfg.h"
00023 #include "atomic.h"
00024 #include "threadport.h"
00025
00026 #if KERNEL_USE_ATOMIC
00027
00028 //-----
00029 K_UCHAR Atomic::Set( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00030 {
00031     K_UCHAR ucRet;
00032     CS_ENTER();
00033     ucRet = *pucSource_;
00034     *pucSource_ = ucVal_;
00035     CS_EXIT();
00036     return ucRet;
00037 }
00038 //-----
00039 K_USHORT Atomic::Set( K_USHORT *pusSource_, K_USHORT usVal_ )
00040 {

```

```

00041     K_USHORT usRet;
00042     CS_ENTER();
00043     usRet = *pusSource_;
00044     *pusSource_ = usVal_;
00045     CS_EXIT();
00046     return usRet;
00047 }
00048 //-----
00049 K_ULONG Atomic::Set( K_ULONG *pulSource_, K_ULONG ulVal_ )
00050 {
00051     K_ULONG ulRet;
00052     CS_ENTER();
00053     ulRet = *pulSource_;
00054     *pulSource_ = ulVal_;
00055     CS_EXIT();
00056     return ulRet;
00057 }
00058
00059 //-----
00060 K_UCHAR Atomic::Add( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00061 {
00062     K_UCHAR ucRet;
00063     CS_ENTER();
00064     ucRet = *pucSource_;
00065     *pucSource_ += ucVal_;
00066     CS_EXIT();
00067     return ucRet;
00068 }
00069
00070 //-----
00071 K_USHORT Atomic::Add( K_USHORT *pusSource_, K_USHORT usVal_ )
00072 {
00073     K_USHORT usRet;
00074     CS_ENTER();
00075     usRet = *pusSource_;
00076     *pusSource_ += usVal_;
00077     CS_EXIT();
00078     return usRet;
00079 }
00080
00081 //-----
00082 K_ULONG Atomic::Add( K_ULONG *pulSource_, K_ULONG ulVal_ )
00083 {
00084     K_ULONG ulRet;
00085     CS_ENTER();
00086     ulRet = *pulSource_;
00087     *pulSource_ += ulVal_;
00088     CS_EXIT();
00089     return ulRet;
00090 }
00091
00092 //-----
00093 K_UCHAR Atomic::Sub( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00094 {
00095     K_UCHAR ucRet;
00096     CS_ENTER();
00097     ucRet = *pucSource_;
00098     *pucSource_ -= ucVal_;
00099     CS_EXIT();
00100     return ucRet;
00101 }
00102
00103 //-----
00104 K_USHORT Atomic::Sub( K_USHORT *pusSource_, K_USHORT usVal_ )
00105 {
00106     K_USHORT usRet;
00107     CS_ENTER();
00108     usRet = *pusSource_;
00109     *pusSource_ -= usVal_;
00110     CS_EXIT();
00111     return usRet;
00112 }
00113
00114 //-----
00115 K_ULONG Atomic::Sub( K_ULONG *pulSource_, K_ULONG ulVal_ )
00116 {
00117     K_ULONG ulRet;
00118     CS_ENTER();
00119     ulRet = *pulSource_;
00120     *pulSource_ -= ulVal_;
00121     CS_EXIT();
00122     return ulRet;
00123 }
00124
00125 //-----
00126 K_BOOL Atomic::TestAndSet( K_BOOL *pbLock_ )
00127 {

```

```

00128     K_UCHAR ucRet;
00129     CS_ENTER();
00130     ucRet = *pbLock_;
00131     if (!ucRet)
00132     {
00133         *pbLock_ = 1;
00134     }
00135     CS_EXIT();
00136     return ucRet;
00137 }
00138
00139 #endif // KERNEL_USE_ATOMIC

```

17.3 /home/mo/mark3-source/embedded/stage/src/atomic.h File Reference

Basic [Atomic](#) Operations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "threadport.h"

```

Classes

- class [Atomic](#)
The [Atomic](#) class.

17.3.1 Detailed Description

Basic [Atomic](#) Operations.

Definition in file [atomic.h](#).

17.4 atomic.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #ifndef __ATOMIC_H__
00022 #define __ATOMIC_H__
00023
00024 #include "kerneltypes.h"
00025 #include "mark3cfg.h"
00026 #include "threadport.h"
00027
00028 #if KERNEL_USE_ATOMIC
00029
00039 class Atomic
00040 {
00041 public:
00048     static K_UCHAR Set( K_UCHAR *pucSource_, K_UCHAR ucVal_ );
00049     static K_USHORT Set( K_USHORT *pusSource_, K_USHORT usVal_ );
00050     static K_ULONG Set( K_ULONG *pulSource_, K_ULONG ulVal_ );
00051
00058     static K_UCHAR Add( K_UCHAR *pucSource_, K_UCHAR ucVal_ );
00059     static K_USHORT Add( K_USHORT *pusSource_, K_USHORT usVal_ );
00060     static K_ULONG Add( K_ULONG *pulSource_, K_ULONG ulVal_ );
00061
00068     static K_UCHAR Sub( K_UCHAR *pucSource_, K_UCHAR ucVal_ );

```

```

00069     static K_USHORT Sub( K_USHORT *pusSource_, K_USHORT usVal_ );
00070     static K_ULONG Sub( K_ULONG *pulSource_, K_ULONG ulVal_ );
00071
00086     static K_BOOL TestAndSet( K_BOOL *pbLock );
00087 };
00088
00089 #endif // KERNEL_USE_ATOMIC
00090
00091 #endif // __ATOMIC_H__

```

17.5 /home/mo/mark3-source/embedded/stage/src/blocking.cpp File Reference

Implementation of base class for blocking objects.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "blocking.h"
#include "thread.h"

```

Macros

- #define `__FILE_ID__` `BLOCKING_CPP`

17.5.1 Detailed Description

Implementation of base class for blocking objects.

Definition in file [blocking.cpp](#).

17.6 blocking.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "mark3cfg.h"
00023 #include "kernel_debug.h"
00024
00025 #include "blocking.h"
00026 #include "thread.h"
00027
00028 //-----
00029 #if defined __FILE_ID__
00030     #undef __FILE_ID__
00031 #endif
00032 #define __FILE_ID__      BLOCKING_CPP
00033
00034 #if KERNEL_USE_SEMAPHORE || KERNEL_USE_MUTEX
00035 //-----
00036 void BlockingObject::Block(Thread *pclThread_)
00037 {
00038     KERNEL_ASSERT( pclThread_ );
00039     KERNEL_TRACE_1( STR_THREAD_BLOCK_1, (K_USHORT)pclThread_->GetID() );
00040
00041     // Remove the thread from its current thread list (the "owner" list)
00042     // ... And add the thread to this object's block list
00043     CS_ENTER();

```

```

00044     Scheduler::Remove(pclThread_);
00045     CS_EXIT();
00046
00047     m_clBlockList.Add(pclThread_);
00048
00049     // Set the "current" list location to the blocklist for this thread
00050     pclThread_>SetCurrent(&m_clBlockList);
00051 }
00052
00053 //-----
00054 void BlockingObject::UnBlock(Thread *pclThread_)
00055 {
00056     KERNEL_ASSERT( pclThread_ );
00057     KERNEL_TRACE_1( STR_THREAD_UNBLOCK_1, (K_USHORT)pclThread_>GetID() );
00058
00059     // Remove the thread from its current thread list (the "owner" list)
00060     pclThread_>GetCurrent()->Remove(pclThread_);
00061
00062     // Put the thread back in its active owner's list. This is usually
00063     // the ready-queue at the thread's original priority.
00064     CS_ENTER();
00065     Scheduler::Add(pclThread_);
00066     CS_EXIT();
00067
00068     // Tag the thread's current list location to its owner
00069     pclThread_>SetCurrent(pclThread_>GetOwner());
00070 }
00071
00072 //-----
00073 K_UCHAR BlockingObject::UnLock()
00074 {
00075     K_UCHAR ucRet;
00076     CS_ENTER();
00077     ucRet = m_ucLocks;
00078     if (m_ucLocks)
00079     {
00080         m_ucLocks--;
00081     }
00082     CS_EXIT();
00083     return ucRet;
00084 }
00085
00086 //-----
00087 K_BOOL BlockingObject::LockAndQueue( K_USHORT usCode_, void *pvData_, K_BOOL *
pbSchedState_)
00088 {
00089     K_UCHAR ucRet;
00090     CS_ENTER();
00091     m_clKTQ.Enqueue(usCode_, pvData_);
00092     if (!m_ucLocks)
00093     {
00094         *pbSchedState_ = Scheduler::SetScheduler(false);
00095     }
00096     ucRet = m_ucLocks;
00097     m_ucLocks++;
00098     CS_EXIT();
00099     return (ucRet);
00100 }
00101
00102 #endif

```

17.7 /home/mo/mark3-source/embedded/stage/src/blocking.h File Reference

Blocking object base class declarations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "threadlist.h"
#include "thread.h"
#include "transaction.h"

```

Classes

- class [BlockingObject](#)

Class implementing thread-blocking primitives.

17.7.1 Detailed Description

Blocking object base class declarations. A Blocking object in Mark3 is essentially a thread list. Any blocking object implementation (being a semaphore, mutex, event flag, etc.) can be built on top of this class, utilizing the provided functions to manipulate thread location within the [Kernel](#).

Blocking a thread results in that thread becoming de-scheduled, placed in the blocking object's own private list of threads which are waiting on the object.

Unblocking a thread results in the reverse: The thread is moved back to its original location from the blocking list.

The only difference between a blocking object based on this class is the logic used to determine what constitutes a Block or Unblock condition.

For instance, a semaphore Pend operation may result in a call to the Block() method with the currently-executing thread in order to make that thread wait for a semaphore Post. That operation would then invoke the Unblock() method, removing the blocking thread from the semaphore's list, and back into the appropriate thread inside the scheduler.

Care must be taken when implementing blocking objects to ensure that critical sections are used judiciously, otherwise asynchronous events like timers and interrupts could result in non-deterministic and often catastrophic behavior.

Definition in file [blocking.h](#).

17.8 blocking.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00047 #ifndef __BLOCKING_H__
00048 #define __BLOCKING_H__
00049
00050 #include "kerneltypes.h"
00051 #include "mark3cfg.h"
00052
00053 #include "ll.h"
00054 #include "threadlist.h"
00055 #include "thread.h"
00056
00057 #include "transaction.h"
00058
00059 #if KERNEL_USE_MUTEX || KERNEL_USE_SEMAPHORE || KERNEL_USE_EVENTFLAG
00060
00061 //-----
00067 class BlockingObject
00068 {
00069 public:
00070     BlockingObject()
00071     {
00072         m_ucLocks = 0;
00073     }
00074
00075 protected:
00096     void Block(Thread *pclThread_);
00097
00109     void Unblock(Thread *pclThread_);
00110
00121     K_UCHAR UnLock();
00122
00123
00140     K_BOOL LockAndQueue( K_USHORT usCode_, void *pvData_, K_BOOL *pbSchedState_);

```



```

00141
00146     ThreadList m_clBlockList;
00147
00152     TransactionQueue m_clKTQ;
00153
00157     K_UCHAR      m_ucLocks;
00158 };
00159
00160 #endif
00161
00162 #endif

```

17.9 /home/mo/mark3-source/embedded/stage/src/control_button.cpp File Reference

GUI Button Control Implementation.

```

#include "control_button.h"
#include "gui.h"

```

17.9.1 Detailed Description

GUI Button Control Implementation. Basic pushbutton control with an up/down state.

Definition in file [control_button.cpp](#).

17.10 control_button.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "control_button.h"
00022 #include "gui.h"
00023
00024
00025 void ButtonControl::Init()
00026 {
00027     m_szCaption = "Button";
00028     m_pstFont = NULL;
00029     m_uBGColor = COLOR_GREY50;
00030     m_uActiveColor = COLOR_GREY25;
00031     m_uLineColor = COLOR_GREY62;
00032     m_uTextColor = COLOR_WHITE;
00033     m_bState = false;
00034     m_pfCallback = NULL;
00035     m_pvCallbackData = NULL;
00036     SetAcceptFocus(true);
00037 }
00038 //-----
00039 void ButtonControl::Draw()
00040 {
00041     DrawText_t stText;
00042     DrawLine_t stLine;
00043
00044     GraphicsDriver *pclDriver = GetParentWindow()->
        GetDriver();
00045
00046     K_USHORT usXOffset = 0;
00047     K_USHORT usHalfWidth = 0;
00048     K_USHORT usYOffset = 0;
00049
00050     // Get the location of the control relative to elements higher in the heirarchy
00051     GetControlOffset(&usXOffset, &usYOffset);
00052

```

```

00053 // Draw the rounded-off rectangle
00054 stLine.usX1 = GetLeft() + usXOffset;
00055 stLine.usX2 = stLine.usX1 + GetWidth() - 1;
00056 stLine.usY1 = GetTop() + usYOffset;
00057 stLine.usY2 = stLine.usY1;
00058 stLine.uColor = m_uLineColor;
00059 pclDriver->Line(&stLine);
00060
00061 stLine.usY1 = GetTop() + GetHeight() + usYOffset - 1;
00062 stLine.usY2 = stLine.usY1;
00063 pclDriver->Line(&stLine);
00064
00065 stLine.usX1 = GetLeft() + usXOffset;
00066 stLine.usX2 = stLine.usX1;
00067 stLine.usY1 = GetTop() + usYOffset + 1;
00068 stLine.usY2 = GetTop() + GetHeight() - 2;
00069 pclDriver->Line(&stLine);
00070
00071 stLine.usX1 = GetLeft() + GetWidth() + usXOffset - 1;
00072 stLine.usX2 = stLine.usX1;
00073 pclDriver->Line(&stLine);
00074
00075 // Draw a rectangle before the text if the BG is specified.
00076 {
00077     DrawRectangle_t stRect;
00078     stRect.usLeft = GetLeft() + usXOffset + 1;
00079     stRect.usRight = GetLeft() + GetWidth() + usXOffset - 2;
00080     stRect.usTop = GetTop() + usYOffset + 1;
00081     stRect.usBottom = GetTop() + GetHeight() + usYOffset - 2;
00082     stRect.bFill = true;
00083
00084     if (m_bState)
00085     {
00086         stRect.uFillColor = m_uActiveColor;
00087     }
00088     else
00089     {
00090         stRect.uFillColor = m_uBGColor;
00091     }
00092
00093     if (GetParentWindow()->IsInFocus(this))
00094     {
00095         stRect.uLineColor = m_uLineColor;
00096     }
00097     else
00098     {
00099         stRect.uLineColor = m_uFillColor;
00100     }
00101
00102     pclDriver->Rectangle(&stRect);
00103 }
00104
00105 // Draw the Text
00106 stText.pstFont = m_pstFont;
00107 stText.pcString = m_szCaption;
00108 stText.uColor = m_uTextColor;
00109 usHalfWidth = pclDriver->TextWidth(&stText);
00110 usHalfWidth >>= 1;
00111 stText.usLeft = GetLeft() + (GetWidth()->>1) - usHalfWidth + usXOffset;
00112 stText.usTop = GetTop() + usYOffset;
00113 pclDriver->Text(&stText);
00114 }
00115
00116 //-----
00117 GuiReturn_t ButtonControl::ProcessEvent(
00118     GuiEvent_t *pstEvent_ )
00119 {
00120     K_USHORT usXOffset, usYOffset;
00121
00122     GetControlOffset(&usXOffset, &usYOffset);
00123
00124     GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");
00125
00126     switch (pstEvent_->ucEventType)
00127     {
00128     case EVENT_TYPE_KEYBOARD:
00129     {
00130         // If this is a space bar or an enter key, behave like a mouse click.
00131         if ((KEYCODE_SPACE == pstEvent_->stKey.ucKeyCode) ||
00132             (KEYCODE_RETURN == pstEvent_->stKey.ucKeyCode))
00133         {
00134             if (pstEvent_->stKey.bKeyState)
00135             {
00136                 m_bState = true;
00137             }
00138             else
00139             {

```

```

00139         m_bState = false;
00140         if (m_pfCallback)
00141         {
00142             m_pfCallback(m_pvCallbackData);
00143         }
00144     }
00145     SetStale();
00146 }
00147 }
00148 break;
00149 case EVENT_TYPE_MOUSE:
00150 {
00151     // Is this control currently in the "active"/pressed state?
00152     if (m_bState)
00153     {
00154         // Check to see if the movement is out-of-bounds based on the coordinates.
00155         // If so, de-activate the control
00156         if (pstEvent_>stMouse.bLeftState)
00157         {
00158             if ((pstEvent_>stMouse.usX < GetLeft() + usXOffset) ||
00159                 (pstEvent_>stMouse.usX >= GetLeft() + usXOffset +
00160                  GetWidth()-1) ||
00161                 (pstEvent_>stMouse.usY < GetTop() + usYOffset) ||
00162                 (pstEvent_>stMouse.usY >= GetTop() + usYOffset +
00163                  GetHeight() - 1))
00164             {
00165                 m_bState = false;
00166                 SetStale();
00167             }
00168             // left button state is now up, and the control was previously active.
00169             // Run the event callback for the mouse, and go from there.
00170             else
00171             {
00172                 if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00173                     (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00174                      GetWidth()-1) &&
00175                     (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00176                     (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00177                      GetHeight() - 1))
00178                 {
00179                     m_bState = false;
00180                     SetStale();
00181                     if (m_pfCallback)
00182                     {
00183                         m_pfCallback(m_pvCallbackData);
00184                     }
00185                 }
00186             }
00187             else if (!m_bState)
00188             {
00189                 // If we registered a down-click in the bounding box, set the state of the
00190                 // control to activated.
00191                 if (pstEvent_>stMouse.bLeftState)
00192                 {
00193                     if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00194                         (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00195                          GetWidth()-1) &&
00196                         (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00197                         (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00198                          GetHeight() - 1))
00199                     {
00200                         m_bState = true;
00201                         SetStale();
00202                     }
00203                 }
00204             }
00205             if (!IsInFocus())
00206             {
00207                 GetParentWindow()->SetFocus(this);
00208                 SetStale();
00209             }
00210             break;
00211         }
00212     }
00213 }
00214 //-----
00215 void ButtonControl::Activate( bool bActivate_ )
00216 {
00217     // When we de-activate the control, simply disarm the control and force
00218     // a redraw
00219     if (!bActivate_)

```



```

00042     void SetLineColor( COLOR eColor_ )      { m_uLineColor = eColor_; }
00043     void SetFillColor( COLOR eColor_ )      { m_uFillColor = eColor_; }
00044     void SetTextColor( COLOR eColor_ )      { m_uTextColor = eColor_; }
00045     void SetActiveColor( COLOR eColor_ )    { m_uActiveColor = eColor_; }
00046
00047     void SetFont( Font_t *pstFont_ )        { m_pstFont = pstFont_; }
00048
00049     void SetCaption( const K_CHAR *szCaption_ ) { m_szCaption = szCaption_; }
00050
00051     void SetCallback( ButtonCallback pfCallback_, void *pvData_ )
00052     { m_pfCallback = pfCallback_; m_pvCallbackData = pvData_; }
00053 private:
00054
00055     const K_CHAR *m_szCaption;
00056     Font_t *m_pstFont;
00057     COLOR      m_uBGColor;
00058     COLOR      m_uActiveColor;
00059     COLOR      m_uLineColor;
00060     COLOR      m_uFillColor;
00061     COLOR      m_uTextColor;
00062     bool       m_bState;
00063
00064     void *m_pvCallbackData;
00065     ButtonCallback m_pfCallback;
00066 };
00067
00068
00069 #endif
00070

```

17.13 /home/mo/mark3-source/embedded/stage/src/control_checkbox.cpp File Reference

Checkbox Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
#include "control_checkbox.h"

```

Macros

- #define **TEXT_X_OFFSET** (13)

Variables

- static const K_UCHAR **aucBox** []
- static const K_UCHAR **aucCheck** []

17.13.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file [control_checkbox.cpp](#).

17.13.2 Variable Documentation

17.13.2.1 **const K_UCHAR aucBox[]** [static]

Initial value:

```
=
{ 0x7E,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x7E }
```

Definition at line 31 of file [control_checkbox.cpp](#).

17.13.2.2 `const K_UCHAR aucCheck[]` `[static]`

Initial value:

```
=
{ 0,
  0,
  0x3C,
  0x3C,
  0x3C,
  0x3C,
  0,
  0 }
```

Definition at line 42 of file [control_checkbox.cpp](#).

17.14 control_checkbox.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "gui.h"
00022 #include "kerneltypes.h"
00023 #include "draw.h"
00024 #include "font.h"
00025 #include "control_checkbox.h"
00026
00027 //-----
00028 #define TEXT_X_OFFSET (13)
00029
00030 //-----
00031 static const K_UCHAR aucBox[] =
00032 { 0x7E,
00033   0x81,
00034   0x81,
00035   0x81,
00036   0x81,
00037   0x81,
00038   0x81,
00039   0x7E };
00040
00041 //-----
00042 static const K_UCHAR aucCheck[] =
00043 { 0,
00044   0,
00045   0x3C,
00046   0x3C,
00047   0x3C,
00048   0x3C,
00049   0,
00050   0 };
00051
00052 //-----
00053 void CheckBoxControl::Init()
```

```

00054 {
00055     SetAcceptFocus(true);
00056 }
00057
00058 //-----
00059 void CheckBoxControl::Draw()
00060 {
00061     GraphicsDriver *pclDriver = GetParentWindow()->
    GetDriver();
00062     K_USHORT usX, usY;
00063     K_USHORT usTextWidth;
00064
00065     GetControlOffset(&usX, &usY);
00066
00067     // Draw the box, (and check, if necessary)
00068     {
00069         DrawRectangle_t stRect;
00070
00071         if (GetParentWindow()->IsInFocus(this))
00072         {
00073             stRect.uLineColor = m_uActiveColor;
00074         }
00075         else
00076         {
00077             stRect.uLineColor = m_uBackColor;
00078         }
00079
00080         stRect.uFillColor = m_uBackColor;
00081         stRect.usTop = usY + GetTop();
00082         stRect.usLeft = usX + GetLeft();
00083         stRect.usRight = stRect.usLeft + GetWidth() - 1;
00084         stRect.usBottom = stRect.usTop + GetHeight() - 1;
00085         stRect.bFill = true;
00086         pclDriver->Rectangle(&stRect);
00087
00088         stRect.uLineColor = m_uBoxBGColor;
00089         stRect.uFillColor = m_uBoxBGColor;
00090         stRect.usTop = usY + GetTop() + ((GetHeight() - 5) >> 1) - 1;
00091         stRect.usLeft = usX + GetLeft() + 2;
00092         stRect.usRight = stRect.usLeft + 7;
00093         stRect.usBottom = stRect.usTop + 7;
00094         stRect.bFill = true;
00095         pclDriver->Rectangle(&stRect);
00096     }
00097
00098     {
00099         DrawStamp_t stStamp;
00100         stStamp.uColor = m_uBoxColor;
00101         stStamp.usY = usY + GetTop() + ((GetHeight() - 5) >> 1) - 1;
00102         stStamp.usX = usX + GetLeft() + 2;
00103         stStamp.usWidth = 8;
00104         stStamp.usHeight = 8;
00105         stStamp.pucData = (K_UCHAR*)aucBox;
00106         pclDriver->Stamp(&stStamp);
00107
00108         if (m_bChecked)
00109         {
00110             stStamp.pucData = (K_UCHAR*)aucCheck;
00111             pclDriver->Stamp(&stStamp);
00112         }
00113     }
00114
00115     // Draw the caption
00116     {
00117         DrawText_t stText;
00118         stText.usLeft = usX + GetLeft() + TEXT_X_OFFSET;
00119         stText.usTop = usY + GetTop();
00120         stText.uColor = m_uFontColor;
00121         stText.pstFont = m_pstFont;
00122         stText.pcString = m_szCaption;
00123
00124         usTextWidth = pclDriver->TextWidth(&stText);
00125         pclDriver->Text(&stText);
00126     }
00127 }
00128
00129 //-----
00130 GuiReturn_t CheckBoxControl::ProcessEvent(
    GuiEvent_t *pstEvent_)
00131 {
00132     K_USHORT usXOffset, usYOffset;
00133
00134     GetControlOffset(&usXOffset, &usYOffset);
00135
00136     GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");
00137
00138     switch (pstEvent_->ucEventType)

```

```

00139     {
00140         case EVENT_TYPE_KEYBOARD:
00141         {
00142             // If this is a space bar or an enter key, behave like a mouse click.
00143             if ((KEYCODE_SPACE == pstEvent->stKey.ucKeyCode) ||
00144                 (KEYCODE_RETURN == pstEvent->stKey.ucKeyCode))
00145             {
00146                 if (pstEvent->stKey.bKeyState)
00147                 {
00148                     m_bChecked = true;
00149                 }
00150                 else
00151                 {
00152                     m_bChecked = false;
00153                 }
00154                 SetStale();
00155             }
00156         }
00157         break;
00158         case EVENT_TYPE_MOUSE:
00159         {
00160             // Is this control currently in the "active"/pressed state?
00161             if (m_bChecked)
00162             {
00163                 // Check to see if the movement is out-of-bounds based on the coordinates.
00164                 // If so, de-activate the control
00165                 if (pstEvent->stMouse.bLeftState)
00166                 {
00167                     if ((pstEvent->stMouse.usX >= GetLeft() + usXOffset) &&
00168                         (pstEvent->stMouse.usX < GetLeft() + usXOffset +
00169                         GetWidth()-1) &&
00170                         (pstEvent->stMouse.usY >= GetTop() + usYOffset) &&
00171                         (pstEvent->stMouse.usY < GetTop() + usYOffset +
00172                         GetHeight() - 1))
00173                     {
00174                         m_bChecked = false;
00175                         SetStale();
00176                     }
00177                 }
00178                 else if (!m_bChecked)
00179                 {
00180                     // If we registered a down-click in the bounding box, set the state of the
00181                     // control to activated.
00182                     if (pstEvent->stMouse.bLeftState)
00183                     {
00184                         if ((pstEvent->stMouse.usX >= GetLeft() + usXOffset) &&
00185                             (pstEvent->stMouse.usX < GetLeft() + usXOffset +
00186                             GetWidth()-1) &&
00187                             (pstEvent->stMouse.usY >= GetTop() + usYOffset) &&
00188                             (pstEvent->stMouse.usY < GetTop() + usYOffset +
00189                             GetHeight() - 1))
00190                         {
00191                             m_bChecked = true;
00192                             SetStale();
00193                         }
00194                     }
00195                 }
00196                 if (!IsInFocus())
00197                 {
00198                     GetParentWindow()->SetFocus(this);
00199                     SetStale();
00200                 }
00201             }
00202             break;

```

17.15 /home/mo/mark3-source/embedded/stage/src/control_checkbox.h File Reference

Checkbox Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

```


Classes

- class [CheckBoxControl](#)

17.15.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file [control_checkbox.h](#).

17.16 control_checkbox.h

```

00001  /*=====
00002
00003  _____|_____|_____|_____|_____|_____|_____
00004  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00005  |  /  \  |  /  \  |  /  \  |  /  \  |  /  \  |  /  \  |
00006  |_____|_____|_____|_____|_____|_____|_____|_____|
00007
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00021  #ifndef __CONTROL_CHECKBOX_H__
00022  #define __CONTROL_CHECKBOX_H__
00023
00024  #include "gui.h"
00025  #include "kerneltypes.h"
00026  #include "draw.h"
00027  #include "font.h"
00028
00029  class CheckBoxControl : public GuiControl
00030  {
00031  public:
00032      virtual void Init();
00033      virtual void Draw();
00034      virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00035      virtual void Activate( bool bActivate_ ) { SetStale(); }
00036
00037      void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00038      void SetCaption( const char *szCaption_ ) { m_szCaption = szCaption_; }
00039      void SetCheck( bool bChecked_ ) { m_bChecked = bChecked_; }
00040      void SetFontColor( COLOR uFontColor_ ) { m_uFontColor = uFontColor_; }
00041      void SetBoxColor( COLOR uBoxColor_ ) { m_uBoxColor = uBoxColor_; }
00042      void SetBackColor( COLOR uBackColor_ ) { m_uBackColor = uBackColor_; }
00043      bool IsChecked( void ) { return m_bChecked; }
00044
00045  private:
00046      const char *m_szCaption;
00047      COLOR m_uBackColor;
00048      COLOR m_uBoxColor;
00049      COLOR m_uFontColor;
00050      Font_t *m_pstFont;
00051      bool m_bChecked;
00052  };
00053
00054  #endif
00055

```

17.17 /home/mo/mark3-source/embedded/stage/src/control_gamepanel.cpp File Reference

GUI Panel Control Implementation with joystick control and tick-based state machine updates.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_gamepanel.h"

```

17.17.1 Detailed Description

GUI Panel Control Implementation with joystick control and tick-based state machine updates.

Definition in file [control_gamepanel.cpp](#).

17.18 control_gamepanel.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "gui.h"
00021 #include "kerneltypes.h"
00022 #include "draw.h"
00023 #include "graphics.h"
00024 #include "control_gamepanel.h"
00025
00026 //-----
00027 void GamePanelControl::Draw()
00028 {
00029     // Game state machine goes here.
00030 }
00031
00032 //-----
00033 GuiReturn_t GamePanelControl::ProcessEvent (
00034     GuiEvent_t *pstEvent_ )
00035 {
00036     K_USHORT usXOffset, usYOffset;
00037
00038     switch (pstEvent_>ucEventType)
00039     {
00040         case EVENT_TYPE_TIMER:
00041             // Every tick, call Draw(). This is used to kick the state
00042             // machine
00043             SetState();
00044             break;
00045         case EVENT_TYPE_KEYBOARD:
00046             break;
00047         case EVENT_TYPE_MOUSE:
00048             break;
00049         case EVENT_TYPE_JOYSTICK:
00050             m_stLastJoy.usRawData = m_stCurrentJoy.usRawData;
00051             m_stCurrentJoy.usRawData = pstEvent_>stJoystick.
00052             usRawData;
00053             break;
00054     }
00055     return GUI_EVENT_OK;
00056 }

```

17.19 /home/mo/mark3-source/embedded/stage/src/control_gamepanel.h File Reference

GUI Game Panel Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"

```

Classes

- class **GamePanelControl**

17.19.1 Detailed Description

GUI Game Panel Control. A game panel is a blank UI element whose dimensions define the dimensions of a gameplay surface. The element triggers a `draw()` call on every tick event (which can be used to kick a game's state machine). The control also responds to joystick events, which can then be used to control the game.

Definition in file [control_gamepanel.h](#).

17.20 control_gamepanel.h

```

00001 /*-----
00002
00003
00004 |-----|-----|-----|-----|-----|-----|
00005 | \      / | \      / | \      / | \      / | \      / |
00006 | /      \ | /      \ | /      \ | /      \ | /      \ |
00007 |-----|-----|-----|-----|-----|-----|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====* /
00014
00025 #ifndef __CONTROL_GAMEPANEL_H__
00026 #define __CONTROL_GAMEPANEL_H__
00027
00028 #include "gui.h"
00029 #include "kerneltypes.h"
00030 #include "draw.h"
00031
00032 class GamePanelControl : public GuiControl
00033 {
00034 public:
00035     virtual void Init() { SetAcceptFocus(false); m_stCurrentJoy.
usRawData = 0; m_stLastJoy.usRawData = 0;}
00036     virtual void Draw();
00037     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00038     virtual void Activate( bool bActivate_ ) {}
00039
00040 private:
00041     JoystickEvent_t m_stLastJoy;
00042     JoystickEvent_t m_stCurrentJoy;
00043
00044 };
00045
00046 #endif
00047

```

17.21	/home/mo/mark3-source/embedded/stage/src/control_groupbox.cpp	File	Reference
-------	---	------	-----------

GUI GroupBox Control Implementation.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_groupbox.h"
```

Macros

- `#define BORDER_OFFSET (4)`
- `#define TEXT_X_OFFSET (8)`
- `#define TEXT_Y_OFFSET (0)`

17.21.1 Detailed Description

GUI GroupBox Control Implementation.

Definition in file [control_groupbox.cpp](#).

17.22 control_groupbox.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "gui.h"
00020 #include "kerneltypes.h"
00021 #include "draw.h"
00022 #include "graphics.h"
00023 #include "control_groupbox.h"
00024
00025 #define BORDER_OFFSET (4)
00026 #define TEXT_X_OFFSET (8)
00027 #define TEXT_Y_OFFSET (0)
00028
00029 //-----
00030 void GroupBoxControl::Draw()
00031 {
00032     GUI_DEBUG_PRINT( "GroupBoxControl::Draw()\n");
00033     GraphicsDriver *pclDriver = GetParentWindow()->
    GetDriver();
00034     K_USHORT usX, usY;
00035     K_USHORT usTextWidth;
00036
00037     GetControlOffset(&usX, &usY);
00038
00039     // Draw the background panel
00040     {
00041         DrawRectangle_t stRectangle;
00042         stRectangle.usTop = GetTop() + usY;
00043         stRectangle.usBottom = stRectangle.usTop + GetHeight() -1;
00044         stRectangle.usLeft = GetLeft() + usX;
00045         stRectangle.usRight = stRectangle.usLeft + GetWidth() -1;
00046         stRectangle.bFill = true;
00047         stRectangle.uLineColor = m_uPanelColor;
00048         stRectangle.uFillColor = m_uPanelColor;
00049
00050         pclDriver->Rectangle(&stRectangle);
00051     }
00052
00053     // Draw the caption
00054     {
00055         DrawText_t stText;
00056         stText.usLeft = usX + TEXT_X_OFFSET;
00057         stText.usTop = usY + TEXT_Y_OFFSET;
00058         stText.uColor = m_uFontColor;
00059         stText.pstFont = m_pstFont;
00060         stText.pcString = m_pcCaption;
00061
00062         usTextWidth = pclDriver->TextWidth(&stText);
00063         pclDriver->Text(&stText);
00064     }
00065
00066     // Draw the lines surrounding the panel
00067     {

```



```

00013 //*****
00022 #ifndef __CONTROL_LABEL_H__
00023 #define __CONTROL_LABEL_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 class LabelControl : public GuiControl
00031 {
00032 public:
00033     virtual void Init() { m_uBackColor = COLOR_BLACK;
00034                         m_uFontColor = COLOR_WHITE;
00035                         m_pstFont = NULL;
00036                         m_pcCaption = "";
00037                         SetAcceptFocus(false); }
00038     virtual void Draw();
00039     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) {}
00040     virtual void Activate( bool bActivate_ ) {}
00041
00042     void SetBackColor( COLOR eColor_ ) { m_uBackColor = eColor_; }
00043     void SetFontColor( COLOR eColor_ ) { m_uFontColor = eColor_; }
00044     void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00045     void SetCaption( const K_CHAR *pcData_ ) { m_pcCaption = pcData_; }
00046
00047 private:
00048     Font_t *m_pstFont;
00049     const K_CHAR *m_pcCaption;
00050     COLOR m_uBackColor;
00051     COLOR m_uFontColor;
00052
00053 };
00054
00055 #endif
00056

```

17.27 /home/mo/mark3-source/embedded/stage/src/control_notification.cpp File Reference

Notification pop-up control.

```
#include "control_notification.h"
#include "kerneltypes.h"
```

17.27.1 Detailed Description

Notification pop-up control. A pop-up control that can be used to present the user with information about system state changes, events, etc.

Definition in file [control_notification.cpp](#).

17.28 control_notification.cpp

```
00001 /*=====
00002 
00003      _____|_____|_____|_____|_____|_____|_____
00004 |   /           \   /       \   /         \   /          \
00005 |  /             \|        \|            \|              \|
00006 |_/               \_       _\          _\_                \_
00007    |_____|_____|_____|_____|_____|_____|_____|_____|
00008 
00009 --[Mark3 Realtime Platform]-----
00010 
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "control_notification.h"
00023 #include "kerneltypes.h"
00024 
00025 //-----
```

```

00026 void NotificationControl::Draw()
00027 {
00028     if (!m_bVisible)
00029     {
00030         return;
00031     }
00032
00033     DrawRectangle_t stRect;
00034     DrawLine_t stLine;
00035     DrawText_t stText;
00036
00037     GraphicsDriver *pclDriver = GetParentWindow()->
GetDriver();
00038
00039     K_USHORT usXOffset = 0;
00040     K_USHORT usHalfWidth = 0;
00041     K_USHORT usYOffset = 0;
00042
00043     // Get the location of the control relative to elements higher in the heirarchy
00044     GetControlOffset(&usXOffset, &usYOffset);
00045
00046     // Draw the rounded-off rectangle
00047     stLine.usX1 = GetLeft() + usXOffset + 1;
00048     stLine.usX2 = stLine.usX1 + GetWidth() - 3;
00049     stLine.usY1 = GetTop() + usYOffset;
00050     stLine.usY2 = stLine.usY1;
00051     stLine.uColor = COLOR_WHITE;
00052     pclDriver->Line(&stLine);
00053
00054     stLine.usY1 = GetTop() + usYOffset + GetHeight() - 1;
00055     stLine.usY2 = stLine.usY1;
00056     pclDriver->Line(&stLine);
00057
00058     // Draw the rounded-off rectangle
00059     stLine.usX1 = GetLeft() + usXOffset;
00060     stLine.usX2 = stLine.usX1;
00061
00062     stLine.usY1 = GetTop() + usYOffset + 1;
00063     stLine.usY2 = stLine.usY1 + GetHeight() - 3;
00064     pclDriver->Line(&stLine);
00065
00066     // Draw the rounded-off rectangle
00067     stLine.usX1 = GetLeft() + usXOffset + GetWidth() - 1;
00068     stLine.usX2 = stLine.usX1;
00069     pclDriver->Line(&stLine);
00070
00071     stRect.usTop = GetTop() + usYOffset + 1;
00072     stRect.usBottom = stRect.usTop + GetHeight() - 3;
00073     stRect.usLeft = GetLeft() + usXOffset + 1;
00074     stRect.usRight = stRect.usLeft + GetWidth() - 3;
00075     stRect.bFill = true;
00076     stRect.uFillColor = COLOR_BLACK;
00077     stRect.uLineColor = COLOR_BLACK;
00078     pclDriver->Rectangle(&stRect);
00079
00080     // Draw the Text
00081     stText.pstFont = m_pstFont;
00082     stText.pcString = m_szCaption;
00083     stText.uColor = COLOR_WHITE;
00084     usHalfWidth = pclDriver->TextWidth(&stText);
00085     usHalfWidth >= 1;
00086     stText.usLeft = GetLeft() + (GetWidth()>>1) - usHalfWidth + usXOffset;
00087     stText.usTop = GetTop() + usYOffset;
00088     pclDriver->Text(&stText);
00089 }
00090
00091 //-----
00092 GuiReturn_t NotificationControl::ProcessEvent(
GuiEvent_t *pstEvent_ )
00093 {
00094
00095     switch (pstEvent_->ucEventType)
00096     {
00097         case EVENT_TYPE_TIMER:
00098         {
00099             if (m_bTrigger && m_usTimeout)
00100             {
00101                 m_usTimeout--;
00102
00103                 if (!m_usTimeout)
00104                 {
00105                     m_bVisible = false;
00106                     m_bTrigger = false;
00107                     SetStale();
00108
00109                     K_USHORT usX, usY;
00110                     GetControlOffset(&usX, &usY);

```


17.29 /home/mo/mark3-source/embedded/stage/src/control_notification.h File Reference

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
```

- class NotificationControl

Definition in file [control_notification.h](#).

```

00001 /*
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022 #ifndef __CONTROL_NOTIFICATION_H__
00023 #define __CONTROL_NOTIFICATION_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028
00029 class NotificationControl : public GuiControl
00030 {
00031 public:
00032     virtual void Init()
00033     {
00034         SetAcceptFocus(false);
00035         m_szCaption = "";
00036         m_pstFont = NULL;
00037         m_bVisible = true;
00038         m_bTrigger = false;
00039     }
00040
00041     virtual void Draw();
00042     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );

```



```

00034
00035     stRectangle.usTop = GetTop() + usY;
00036     stRectangle.usBottom = stRectangle.usTop + GetHeight() -1;
00037     stRectangle.usLeft = GetLeft() + usX;
00038     stRectangle.usRight = stRectangle.usLeft + GetWidth() -1;
00039     stRectangle.bFill = true;
00040     stRectangle.uLineColor = m_uColor;
00041     stRectangle.uFillColor = m_uColor;
00042
00043     pclDriver->Rectangle(&stRectangle);
00044 }

```

17.33 /home/mo/mark3-source/embedded/stage/src/control_panel.h File Reference

GUI Panel Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"

```

Classes

- class [PanelControl](#)

17.33.1 Detailed Description

GUI Panel Control. The "panel" is probably the simplest control that can be implemented in a GUI. It serves as a dock for other controls, and also as an example for implementing more complex controls.

A panel is essentially a flat rectangle, specified by a control's typical top/left/height/width parameters, and a color value.

Definition in file [control_panel.h](#).

17.34 control_panel.h

```

00001 /*=====
00002
00003   _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00006  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00007  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00026 #ifndef __CONTROL_PANEL_H__
00027 #define __CONTROL_PANEL_H__
00028
00029 #include "gui.h"
00030 #include "kerneltypes.h"
00031 #include "draw.h"
00032
00033 class PanelControl : public GuiControl
00034 {
00035 public:
00036     virtual void Init() { m_uColor = COLOR_BLACK; SetAcceptFocus(false); }
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) {};
00039     virtual void Activate( bool bActivate_ ) {}
00040
00041     void SetColor( COLOR eColor_ ) { m_uColor = eColor_; }
00042
00043 private:
00044     COLOR m_uColor;

```

```

00045
00046 };
00047
00048 #endif
00049

```

17.35 /home/mo/mark3-source/embedded/stage/src/control_progress.cpp File Reference

GUI Progress Bar Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "control_progress.h"

```

17.35.1 Detailed Description

GUI Progress Bar Control. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

Definition in file [control_progress.cpp](#).

17.36 control_progress.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "gui.h"
00022 #include "kerneltypes.h"
00023 #include "draw.h"
00024 #include "control_progress.h"
00025
00026 //-----
00027 void ProgressControl::Init()
00028 {
00029     m_uBackColor = COLOR_BLACK;
00030     m_uBorderColor = COLOR_GREY75;
00031     m_uProgressColor = COLOR_GREEN;
00032     SetAcceptFocus(false);
00033 }
00034
00035 //-----
00036 void ProgressControl::Draw()
00037 {
00038     GraphicsDriver *pclDriver = GetParentWindow()->
GetDriver();
00039     DrawRectangle_t stRect;
00040     DrawLine_t stLine;
00041
00042     K_USHORT usX, usY;
00043     K_USHORT usProgressWidth;
00044
00045     GetControlOffset(&usX, &usY);
00046
00047     // Draw the outside of the progress bar region
00048     stLine.uColor = m_uBorderColor;
00049     stLine.usX1 = usX + GetLeft() + 1;
00050     stLine.usX2 = usX + GetLeft() + GetWidth() - 2;
00051     stLine.usY1 = usY + GetTop();
00052     stLine.usY2 = usY + GetTop();
00053     pclDriver->Line(&stLine);

```

```

00054
00055     stLine.usY1 = usY + GetTop() + GetHeight() - 1;
00056     stLine.usY2 = usY + GetTop() + GetHeight() - 1;
00057     pclDriver->Line(&stLine);
00058
00059     stLine.usY1 = usY + GetTop() + 1;
00060     stLine.usY2 = usY + GetTop() + GetHeight() - 2;
00061     stLine.usX1 = usX + GetLeft();
00062     stLine.usX2 = usX + GetLeft();
00063     pclDriver->Line(&stLine);
00064
00065     stLine.usX1 = usX + GetLeft() + GetWidth() - 1;
00066     stLine.usX2 = usX + GetLeft() + GetWidth() - 1;
00067     pclDriver->Line(&stLine);
00068
00069     // Draw the "completed" portion
00070     usProgressWidth = (K_USHORT)( ( ( (K_ULONG)m_ucProgress) * (GetWidth()-2) ) + 50 ) / 100);
00071     stRect.usTop = usY + GetTop() + 1;
00072     stRect.usBottom = usY + GetTop() + GetHeight() - 2;
00073     stRect.usLeft = usX + GetLeft() + 1;
00074     stRect.usRight = stRect.usLeft + usProgressWidth - 1;
00075     stRect.bFill = true;
00076     stRect.uLineColor = m_uProgressColor;
00077     stRect.uFillColor = m_uProgressColor;
00078     pclDriver->Rectangle(&stRect);
00079
00080     // Draw the "incomplete" portion
00081     stRect.usLeft = stRect.usRight + 1;
00082     stRect.usRight = usX + GetLeft() + GetWidth() - 2;
00083     stRect.bFill = true;
00084     stRect.uLineColor = m_uBackColor;
00085     stRect.uFillColor = m_uBackColor;
00086     pclDriver->Rectangle(&stRect);
00087
00088 }
00089
00090 //-----
00091 void ProgressControl::SetProgress( K_UCHAR ucProgress_ )
00092 {
00093     m_ucProgress = ucProgress_;
00094     if (m_ucProgress > 100)
00095     {
00096         m_ucProgress;
00097     }
00098     SetStale();
00099 }
00100
00101 //-----
00102 GuiReturn_t ProgressControl::ProcessEvent (
00103     GuiEvent_t *pstEvent_)
00104 {
00105     return GUI_EVENT_OK;
00106 }

```

17.37 /home/mo/mark3-source/embedded/stage/src/control_progress.h File Reference

GUI Progress Bar Control.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

```

Classes

- class [ProgressControl](#)

17.37.1 Detailed Description

GUI Progress Bar Control. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

17.39.1 Detailed Description

GUI Button Control, with a flare. Basic pushbutton control with an up/down state, and Mark3 visual style

Definition in file [control_slickbutton.h](#).

17.40 control_slickbutton.h

```

00001
00002 /*=====
00003
00004
00005
00006
00007
00008
00009
00010 --[Mark3 Realtime Platform]-----
00011
00012 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00013 See license.txt for more information
00014 =====*/
00022 #ifndef __CONTROL_SLICKBUTTON_H__
00023 #define __CONTROL_SLICKBUTTON_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 typedef void (*ButtonCallback)( void *pvData_ );
00031
00032 class SlickButtonControl : public GuiControl
00033 {
00034 public:
00035
00036     virtual void Init();
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00039     virtual void Activate( bool bActivate_ );
00040
00041     void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00042
00043     void SetCaption( const K_CHAR *szCaption_ ) { m_szCaption = szCaption_; }
00044
00045     void SetCallback( ButtonCallback pfCallback_, void *pvData_ )
00046     { m_pfCallback = pfCallback_; m_pvCallbackData = pvData_; }
00047 private:
00048
00049     const K_CHAR *m_szCaption;
00050     Font_t *m_pstFont;
00051     bool m_bState;
00052     K_UCHAR m_ucTimeout;
00053
00054     void *m_pvCallbackData;
00055     ButtonCallback m_pfCallback;
00056 };
00057
00058
00059 #endif
00060

```

17.41 /home/mo/mark3-source/embedded/stage/src/control_slickprogress.cpp File Reference

GUI Progress Bar Control, with flare.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "control_slickprogress.h"

```



```

00079     stRect.uLineColor = RGB_COLOR( 0, (K_ULONG)(MAX_GREEN * 0.75), (K_ULONG)(MAX_BLUE * 0.20));
00080     stRect.uFillColor = stRect.uLineColor;
00081     pclDriver->Rectangle(&stRect);
00082
00083     // Draw the "incomplete" portion
00084     stRect.usTop = usY + GetTop() + 1;
00085     stRect.usBottom = usY + GetTop() + GetHeight() - 2;
00086     stRect.usLeft = stRect.usRight + 1;
00087     stRect.usRight = usX + GetLeft() + GetWidth() - 2;
00088     stRect.bFill = true;
00089     stRect.uLineColor = RGB_COLOR( (K_ULONG)(MAX_RED * 0.10), (K_ULONG)(MAX_GREEN * 0.10), (
K_ULONG)(MAX_BLUE * 0.10));
00090     stRect.uFillColor = stRect.uLineColor;
00091     pclDriver->Rectangle(&stRect);
00092
00093 }
00094
00095 //-----
00096 void SlickProgressControl::SetProgress( K_UCHAR ucProgress_ )
00097 {
00098     m_ucProgress = ucProgress_;
00099     if (m_ucProgress > 100)
00100     {
00101         m_ucProgress;
00102     }
00103     SetStale();
00104 }
00105
00106 //-----
00107 GuiReturn_t SlickProgressControl::ProcessEvent(
    GuiEvent_t *pstEvent_)
00108 {
00109     return GUI_EVENT_OK;
00110 }

```

17.43 /home/mo/mark3-source/embedded/stage/src/control_slickprogress.h File Reference

GUI Progress Bar Control, with flare.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"

```

Classes

- class [SlickProgressControl](#)

17.43.1 Detailed Description

GUI Progress Bar Control, with flare. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

Definition in file [control_slickprogress.h](#).

17.44 control_slickprogress.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----

```

```

00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #ifndef __CONTROL_SLICKPROGRESS_H__
00023 #define __CONTROL_SLICKPROGRESS_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 class SlickProgressControl : public GuiControl
00031 {
00032 public:
00033     virtual void Init();
00034     virtual void Draw();
00035     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00036     virtual void Activate( bool bActivate_ ) {}
00037
00038     void SetProgress( K_UCHAR ucProgress_ );
00039
00040 private:
00041     K_UCHAR m_ucProgress;
00042 };
00043
00044 #endif
00045

```

17.45 /home/mo/mark3-source/embedded/stage/src/dcpu.cpp File Reference

Portable DCPU-16 CPU emulator.

```

#include "dcpu.h"
#include "kerneltypes.h"
#include "ll.h"

```

Macros

- #define **CORE_DEBUG** 0
- #define **DBG_PRINT**(...)

Variables

- static const K_UCHAR [aucBasicOpcodeCycles](#) []
Define the number of cycles that each "basic" opcode takes to execute.
- static const K_UCHAR [aucExtendedOpcodeCycles](#) []
Define the number of cycles that each "extended" opcode takes to execute.

17.45.1 Detailed Description

Portable DCPU-16 CPU emulator. The DCPU-16 is the in-game CPU used in the upcoming game 0x10^c, from the creators of the wildly successful Minecraft. While the [DCPU](#) is supposed to be part of the game, it has serious potential for use in all sorts of embedded applications.

The fact that [DCPU](#) is a very lightweight VM to implement and contains built-in instructions for accessing hardware peripherals and handling external interrupts lends itself to being used on microcontrollers.

Unlike a lot of embedded CPUs, DCPU-16 assembly is extremely simple to learn, since it has a very limited number of opcodes (37), each of which provide the same register/memory addressing modes for all operands. There are also only 2 opcode formats which make interpreting opcodes very efficient.

The DCPU-16 is extended using a variable number of "external hardware devices" which communicate with the CPU core using interrupts. These devices are enumerated on startup, and since there is no defined format for how

these devices work, we can hijack this interface to provide a way for the DCPU to access resources supplied by the OS (i.e Timers, Drivers), or the hardware directly. This also lends itself to inter-VM communications (multiple DCPUs communicating with eachother in different OS threads). There's an immense amount of flexibility here - applications from debugging to scripting to runtime-configuration are all easily supported by this machine.

But what is a platform without tools support? Fortunately, the hype around 0x10c is building - and a development community for this platform has grown immensely. There are a number of compilers, assemblers, and IDEs, many of which support virtualized hardware extensions. One of the compilers is a CLANG/LLVM backend, which should allow for very good C language support.

I had attempted to do something similar by creating a VM based on the 8051 (see the Funk51 project on sourceforge), but that project was at least four times as large - and the tools support was very spotty. There were C compilers, but there was a lot of shimming required to produce output that was suitable for the VM. Also, the lack of a native host interface (interrupts, hardware bus enumerations, etc.) forced a non-standard approach to triggering native methods by writing commands to a reserved chunk of memory and writing to a special "trigger" address to invoke the native system. Using a DCPU-16 based simulator addresses this in a nice, clean way by providing modern tools, and a VM infrastructure tailored to be interfaced with a host.

Regarding this version of the **DCPU** emulator - it's very simple to use. Program binaries are loaded into buffers in the host CPU's RAM, with the host also providing a separate buffer for **DCPU** RAM. The size of the **DCPU** RAM buffer will contain both the RAM area, as well as the program stack, so care must be taken to ensure that the stack doesn't overflow. The **DCPU** specification allows for 64K words (128KB) of RAM and ROM each, but this implementation allows us to tailor the CPU for more efficient or minimal environments.

In the future, this emulator will be extended to provide a mechanism to allow programs to be run out of flash, EEPROM, or other interfaces via the Mark3 Drivers API.

Once the program has been loaded into the host's address space, the **DCPU** class can be initialized.

```
// Use 16-bit words for 16-bit emulator.
K_USHORT ausRAM[ RAM_SIZE ];
K_USHORT ausROM[ ROM_SIZE ];
{
    class DCPU clMyDCPU;

    // Read program code into ausROM buffer here

    // Initialize the DCPU emulator
    clMyDCPU.Init( ausROM, RAM_SIZE, ausROM, ROM_SIZE );
}
```

Once the emulator has been initialized, the VM can be run one opcode at a time, as in the following example.

```
while (1)
{
    clMyCPU.RunOpcode();
}
```

To inspect the contents of the VM's registers, call the `GetRegisters()` method. This is useful for printing the CPU state on a regular basis, or using the PC value to determine when to end execution, or to provide an offset for disassembling the current opcode.

```
DCPU_Registers *pstRegisters;  
pstRegisters = clMyCPU.GetRegisters();
```

Definition in file [dcpu.cpp](#).

17.46 dcpu.cpp

```

00001  / *-----
00002
00003  |-----|-----|-----|-----|-----|
00004  | \ / | \ / | \ / | \ / | \ / | \ / |
00005  | / \ | / \ | / \ | / \ | / \ | / \ |
00006  |-----|-----|-----|-----|-----|
00007

```

```

00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00114 #include "dcpu.h"
00115 #include "kerneltypes.h"
00116 #include "ll.h"
00117
00118 #define CORE_DEBUG 0
00119
00120 //-----
00121 #if CORE_DEBUG
00122
00123     #include <stdio.h>
00124     #include <string.h>
00125     #include <stdlib.h>
00126
00127     #define DBG_PRINT(...)      printf(__VA_ARGS__)
00128 #else
00129     #define DBG_PRINT(...)
00130 #endif
00131
00132 //-----
00136 static const K_UCHAR aucBasicOpcodeCycles[] =
00137 {
00138     0,          // OP_NON_BASIC = 0
00139     1,          // OP_SET
00140     2,          // OP_ADD
00141     2,          // OP_SUB
00142     2,          // OP_MUL
00143     2,          // OP_MLI
00144     3,          // OP_DIV
00145     3,          // OP_DVI,
00146     3,          // OP_MOD,
00147     3,          // OP_MDI,
00148     1,          // OP_AND,
00149     1,          // OP_BOR,
00150     1,          // OP_XOR,
00151     1,          // OP_SHR,
00152     1,          // OP_ASR,
00153     1,          // OP_SHL,
00154     2,          // OP_IFB,
00155     2,          // OP_IFC,
00156     2,          // OP_IFE,
00157     2,          // OP_IFN,
00158     2,          // OP_IFG,
00159     2,          // OP_IFA,
00160     2,          // OP_IFL,
00161     2,          // OP_IFU,
00162     0,          // OP_18,
00163     0,          // OP_19,
00164     3,          // OP_ADX,
00165     3,          // OP_SBX,
00166     0,          // OP_1C,
00167     0,          // OP_1D,
00168     2,          // OP_STI,
00169     2,          // OP_STD
00170 };
00171
00172 //-----
00176 static const K_UCHAR aucExtendedOpcodeCycles[] =
00177 {
00178     0,          // "RESERVED",
00179     3,          // "JSR",
00180     0,          // "UNDEFINED"
00181     0,          // "UNDEFINED"
00182     0,          // "UNDEFINED"
00183     0,          // "UNDEFINED"
00184     0,          // "UNDEFINED"
00185     0,          // "UNDEFINED"
00186     4,          // "INT",
00187     1,          // "IAG",
00188     1,          // "IAS",
00189     3,          // "RFI",
00190     2,          // "IAQ",
00191     0,          // "UNDEFINED"
00192     0,          // "UNDEFINED"
00193     0,          // "UNDEFINED"
00194     2,          // "HWN",
00195     4,          // "HWQ",
00196     4,          // "HWI",
00197     0,          // "UNDEFINED"
00198     0,          // "UNDEFINED"
00199     0,          // "UNDEFINED"
00200     0,          // "UNDEFINED"

```

```

00201     0,    // "UNDEFINED"
00202     0,    // "UNDEFINED"
00203     0,    // "UNDEFINED"
00204     0,    // "UNDEFINED"
00205     0,    // "UNDEFINED"
00206     0,    // "UNDEFINED"
00207     0,    // "UNDEFINED"
00208     0,    // "UNDEFINED"
00209     0,    // "UNDEFINED"
00210 };
00211
00212 //-----
00213 void DCPU::SET()
00214 {
00215     DBG_PRINT("SET\n");
00216     *b = *a;
00217 }
00218
00219 //-----
00220 void DCPU::ADD()
00221 {
00222     K_ULONG ulTemp;
00223     DBG_PRINT("ADD\n");
00224
00225     ulTemp = (K_ULONG)*a + (K_ULONG)*b;
00226     if (ulTemp >= 65536)
00227     {
00228         m_stRegisters.EX = 0x0001;
00229     }
00230     else
00231     {
00232         m_stRegisters.EX = 0;
00233     }
00234
00235     *b = *b + *a;
00236 }
00237
00238 //-----
00239 void DCPU::SUB()
00240 {
00241     K_LONG lTemp;
00242     DBG_PRINT("SUB\n");
00243
00244     lTemp = (K_LONG)*b - (K_LONG)*a;
00245     if (lTemp < 0)
00246     {
00247         m_stRegisters.EX = 0xFFFF;
00248     }
00249     else
00250     {
00251         m_stRegisters.EX = 0;
00252     }
00253
00254     *b = *b - *a;
00255 }
00256
00257 //-----
00258 void DCPU::MUL()
00259 {
00260     K_ULONG ulTemp;
00261
00262     DBG_PRINT("MUL\n");
00263     ulTemp = ((K_ULONG)*a * (K_ULONG)*b);
00264     m_stRegisters.EX = (K_USHORT)(ulTemp >> 16);
00265     *b = (K_USHORT)(ulTemp & 0x0000FFFF);
00266 }
00267
00268 //-----
00269 void DCPU::MLI()
00270 {
00271     K_LONG lTemp;
00272
00273     DBG_PRINT("MLI\n");
00274     lTemp = ((K_LONG)(*a * (K_SHORT)*b)) * (K_LONG)(*a * (K_SHORT)*b));
00275     m_stRegisters.EX = (K_USHORT)(lTemp >> 16);
00276     *b = (K_USHORT)(lTemp & 0x0000FFFF);
00277 }
00278
00279 //-----
00280 void DCPU::DIV()
00281 {
00282     K_USHORT usTemp;
00283
00284     DBG_PRINT("DIV\n");
00285     if (*a == 0)
00286     {
00287         *b = 0;

```

```

00288     m_stRegisters.EX = 0;
00289 }
00290 else
00291 {
00292     usTemp = (K_USHORT) (((K_ULONG)*b) << 16) / (K_ULONG)*a);
00293     *b = *b / *a;
00294     m_stRegisters.EX = usTemp;
00295 }
00296 }
00297
00298 //-----
00299 void DCPU::DVI()
00300 {
00301     K_USHORT usTemp;
00302
00303     DBG_PRINT("DVI\n");
00304     if (*a == 0)
00305     {
00306         *b = 0;
00307         m_stRegisters.EX = 0;
00308     }
00309     else
00310     {
00311         usTemp = (K_USHORT) (((K_LONG)*(K_SHORT*)b) << 16) / (K_LONG) (*(K_SHORT*)
a));
00312         *b = (K_USHORT) (*(K_SHORT*)b / *(K_SHORT*)a);
00313         m_stRegisters.EX = usTemp;
00314     }
00315 }
00316 }
00317
00318 //-----
00319 void DCPU::MOD()
00320 {
00321     DBG_PRINT("MOD\n");
00322     if (*a == 0)
00323     {
00324         *b = 0;
00325     }
00326     else
00327     {
00328         *b = *b % *a;
00329     }
00330 }
00331
00332 //-----
00333 void DCPU::MDI()
00334 {
00335     DBG_PRINT("MDI\n");
00336     if (*b == 0)
00337     {
00338         *a = 0;
00339     }
00340     else
00341     {
00342         *b = (K_USHORT) (*(K_SHORT*)b % *(K_SHORT*)a);
00343     }
00344 }
00345
00346 //-----
00347 void DCPU::AND()
00348 {
00349     DBG_PRINT("AND\n");
00350     *b = *b & *a;
00351 }
00352
00353 //-----
00354 void DCPU::BOR()
00355 {
00356     DBG_PRINT("BOR\n");
00357     *b = *b | *a;
00358 }
00359
00360 //-----
00361 void DCPU::XOR()
00362 {
00363     DBG_PRINT("XOR\n");
00364     *b = *b ^ *a;
00365 }
00366
00367 //-----
00368 void DCPU::SHR()
00369 {
00370     K_USHORT usTemp = (K_USHORT) (((K_ULONG)*b) << 16) >> (K_ULONG)*a);
00371
00372     DBG_PRINT("SHR\n");
00373     *b = *b >> *a;

```

```

00374     m_stRegisters.EX = usTemp;
00375 }
00376
00377 //-----
00378 void DCPU::ASR()
00379 {
00380     K_USHORT usTemp = (K_USHORT) (((K_LONG)*b) << 16) >> (K_LONG)*a);
00381
00382     DBG_PRINT("ASR\n");
00383     *b = (K_USHORT) (* (K_SHORT*)b >> * (K_SHORT*)a);
00384     m_stRegisters.EX = usTemp;
00385 }
00386 //-----
00387 void DCPU::SHL()
00388 {
00389     K_USHORT usTemp = (K_USHORT) (((K_ULONG)*b) << (K_ULONG)*a) >> 16);
00390
00391     DBG_PRINT("SHL\n");
00392     *b = *b << *a;
00393     m_stRegisters.EX = usTemp;
00394 }
00395
00396 //-----
00397 bool DCPU::IFB()
00398 {
00399     DBG_PRINT("IFB\n");
00400     if ((*b & *a) != 0)
00401     {
00402         return true;
00403     }
00404     return false;
00405 }
00406
00407 //-----
00408 bool DCPU::IFC()
00409 {
00410     DBG_PRINT("IFC\n");
00411     if ((*b & *a) == 0)
00412     {
00413         return true;
00414     }
00415     return false;
00416 }
00417
00418 //-----
00419 bool DCPU::IFE()
00420 {
00421     DBG_PRINT("IFE\n");
00422     if (*b == *a)
00423     {
00424         return true;
00425     }
00426     return false;
00427 }
00428
00429 //-----
00430 bool DCPU::IFN()
00431 {
00432     DBG_PRINT("IFN\n");
00433     if (*b != *a)
00434     {
00435         return true;
00436     }
00437     return false;
00438 }
00439
00440 //-----
00441 bool DCPU::IFG()
00442 {
00443     DBG_PRINT("IFG\n");
00444     if (*b > *a)
00445     {
00446         return true;
00447     }
00448     return false;
00449 }
00450
00451 //-----
00452 bool DCPU::IFA()
00453 {
00454     DBG_PRINT("IFA\n");
00455     if ((* (K_SHORT*)b) > * (K_SHORT*)a)
00456     {
00457         return true;
00458     }
00459     return false;
00460 }

```

```

00461
00462 //-----
00463 bool DCPU::IFL()
00464 {
00465     DBG_PRINT("IFL\n");
00466     if (*b < *a)
00467     {
00468         return true;
00469     }
00470     return false;
00471 }
00472
00473 //-----
00474 bool DCPU::IFU()
00475 {
00476     DBG_PRINT("IFU\n");
00477     if (*(K_SHORT*)b < *(K_SHORT*)a)
00478     {
00479         return true;
00480     }
00481     return false;
00482 }
00483
00484 //-----
00485 void DCPU::ADX()
00486 {
00487     K_ULONG ulTemp;
00488     DBG_PRINT("ADX\n");
00489     ulTemp = (K_ULONG)*b + (K_ULONG)*a + (K_ULONG)m_stRegisters.EX;
00490     if (ulTemp >= 0x10000)
00491     {
00492         m_stRegisters.EX = 1;
00493     }
00494     else
00495     {
00496         m_stRegisters.EX = 0;
00497     }
00498
00499     *b = ((K_USHORT)(ulTemp & 0x0000FFFF));
00500 }
00501
00502 //-----
00503 void DCPU::SBX()
00504 {
00505     K_LONG lTemp;
00506     DBG_PRINT("SBX\n");
00507     lTemp = (K_LONG)*b - (K_LONG)*a + (K_LONG)m_stRegisters.EX;
00508     if (lTemp < 0 )
00509     {
00510         m_stRegisters.EX = 0xFFFF;
00511     }
00512     else
00513     {
00514         m_stRegisters.EX = 0;
00515     }
00516
00517     *b = ((K_USHORT)(lTemp & 0x0000FFFF));
00518 }
00519
00520 //-----
00521 void DCPU::STI()
00522 {
00523     DBG_PRINT("STI\n");
00524     *b = *a;
00525     m_stRegisters.I++;
00526     m_stRegisters.J++;
00527 }
00528
00529 //-----
00530 void DCPU::STD()
00531 {
00532     DBG_PRINT("STD\n");
00533     *b = *a;
00534     m_stRegisters.I--;
00535     m_stRegisters.J--;
00536 }
00537
00538 //-----
00539 void DCPU::JSR()
00540 {
00541     DBG_PRINT("JSR 0x%04X\n", *a);
00542     m_pusRAM[m_stRegisters.SP--] = m_stRegisters.PC;
00543     m_stRegisters.PC = *a;
00544 }
00545
00546 //-----
00547 void DCPU::INT()

```



```

00548 {
00549     DBG_PRINT("INT\n");
00550
00551     if (m_stRegisters.IA == 0)
00552     {
00553         // If IA is not set, return out.
00554         return;
00555     }
00556
00557     // Either acknowledge the interrupt immediately, or queue it.
00558     if (m_bInterruptQueueing == false)
00559     {
00560         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00561         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00562
00563         m_stRegisters.A = *a;
00564         m_stRegisters.PC = m_stRegisters.IA;
00565         m_bInterruptQueueing = true;
00566     }
00567     else
00568     {
00569         // Add interrupt message to the queue
00570         m_ausInterruptQueue[ ++m_ucQueueLevel ] = *
a;
00571     }
00572 }
00573
00574 //-----
00575 void DCPU::ProcessInterruptQueue()
00576 {
00577     // If there's an interrupt address specified, queueing is disabled, and
00578     // the queue isn't empty
00579     if (m_stRegisters.IA && !m_bInterruptQueueing &&
m_ucQueueLevel)
00580     {
00581         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00582         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00583
00584         m_stRegisters.A = m_ausInterruptQueue[
m_ucQueueLevel-- ];
00585         m_stRegisters.PC = m_stRegisters.IA;
00586
00587         m_bInterruptQueueing = true;
00588     }
00589 }
00590
00591 //-----
00592 void DCPU::IAG()
00593 {
00594     DBG_PRINT("IAG\n");
00595
00596     *a = m_stRegisters.IA;
00597 }
00598
00599 //-----
00600 void DCPU::IAS()
00601 {
00602     DBG_PRINT("IAS\n");
00603
00604     m_stRegisters.IA = *a;
00605 }
00606
00607 //-----
00608 void DCPU::RFI()
00609 {
00610     DBG_PRINT("RFI\n");
00611
00612     m_bInterruptQueueing = false;
00613
00614     m_stRegisters.A = m_pusRAM[ ++m_stRegisters.SP ];
00615     m_stRegisters.PC = m_pusRAM[ ++m_stRegisters.SP ];
00616 }
00617
00618 //-----
00619 void DCPU::IAQ()
00620 {
00621     DBG_PRINT("IAQ\n");
00622
00623     if (*a)
00624     {
00625         m_bInterruptQueueing = true;
00626     }
00627     else
00628     {
00629         m_bInterruptQueueing = false;
00630     }
00631 }

```

```

00638     }
00639 }
00640
00641 //-----
00642 void DCPU::HWN()
00643 {
00644     LinkListNode *pclNode;
00645
00646     DBG_PRINT("HWN\n");
00647     m_usTempA = 0;
00648     pclNode = m_clPluginList.GetHead();
00649     while (pclNode)
00650     {
00651         m_usTempA++;
00652         pclNode = pclNode->GetNext();
00653     }
00654
00655     *a = m_usTempA;
00656 }
00657
00658 //-----
00659 void DCPU::HWQ()
00660 {
00661     DBG_PRINT("HWQ\n");
00662     DCPUPugin *pclPlugin;
00663     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead();
00664     while (pclPlugin)
00665     {
00666         if (pclPlugin->GetDeviceNumber() == *a)
00667         {
00668             pclPlugin->Enumerate(&m_stRegisters);
00669             break;
00670         }
00671         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00672     }
00673 }
00674
00675 //-----
00676 void DCPU::HWI()
00677 {
00678     DBG_PRINT("HWI\n");
00679     DCPUPugin *pclPlugin;
00680     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead();
00681     while (pclPlugin)
00682     {
00683         if (pclPlugin->GetDeviceNumber() == *a)
00684         {
00685             pclPlugin->Interrupt(this);
00686             break;
00687         }
00688         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00689     }
00690 }
00691
00692 //-----
00693 void DCPU::Init( K_USHORT *pusRAM_, K_USHORT usRAMSize_,
00694                 const K_USHORT *pusROM_, K_USHORT usROMSize_ )
00695 {
00696     m_stRegisters.PC = 0;
00697     m_stRegisters.SP = usRAMSize_ ;
00698     m_stRegisters.A = 0;
00699     m_stRegisters.B = 0;
00700     m_stRegisters.C = 0;
00701     m_stRegisters.X = 0;
00702     m_stRegisters.Y = 0;
00703     m_stRegisters.Z = 0;
00704     m_stRegisters.I = 0;
00705     m_stRegisters.J = 0;
00706     m_stRegisters.EX = 0;
00707     m_stRegisters.IA = 0;
00708     m_ulCycleCount = 0;
00709
00710     m_pusROM = (K_USHORT*)pusROM_;
00711     m_usROMSize = usROMSize_;
00712
00713     m_pusRAM = pusRAM_;
00714     m_usRAMSize = usRAMSize_;
00715 }
00716
00717 //-----
00718 K_UCHAR DCPU::GetOperand( K_UCHAR ucOpType_, K_USHORT **pusResult_ )
00719 {
00720     K_UCHAR ucRetVal = 0;
00721     switch (ucOpType_)
00722 
```

```

00726     {
00727         case ARG_A:      case ARG_B:      case ARG_C:      case ARG_X:
00728         case ARG_Y:      case ARG_Z:      case ARG_I:      case ARG_J:
00729             *pusResult_ = &m_stRegisters.ausRegisters[ ucOpType_ - ARG_A ];
00730             break;
00731
00732         case ARG_BRACKET_A:      case ARG_BRACKET_B:      case ARG_BRACKET_C:      case ARG_BRACKET_X:
00733
00734         case ARG_BRACKET_Y:      case ARG_BRACKET_Z:      case ARG_BRACKET_I:      case ARG_BRACKET_J:
00735             *pusResult_ = &m_pusRAM[ m_stRegisters.ausRegisters[ ucOpType_ -
ARG_BRACKET_A ] ];
00736             break;
00737
00738         case ARG_WORD_A: case ARG_WORD_B: case ARG_WORD_C: case ARG_WORD_X:
00739         case ARG_WORD_Y: case ARG_WORD_Z: case ARG_WORD_I: case ARG_WORD_J:
00740             {
00741                 K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++ ];
00742                 usTemp += m_stRegisters.ausRegisters[ ucOpType_ - ARG_WORD_A ];
00743                 *pusResult_ = &m_pusRAM[ usTemp ];
00744                 ucRetVal = 1;
00745             }
00746             break;
00747         case ARG_PUSH_POP_SP:
00748             if (*pusResult_ == a)
00749             {
00750                 a = &m_pusRAM[ ++m_stRegisters.SP ];
00751             }
00752             else
00753             {
00754                 b = &m_pusRAM[ m_stRegisters.SP-- ];
00755             }
00756             break;
00757         case ARG_PEEK_SP:
00758             *pusResult_ = &m_pusRAM[ m_stRegisters.SP ];
00759             break;
00760         case ARG_WORD_SP:
00761             {
00762                 K_USHORT usTemp = m_pusROM[ ++m_stRegisters.PC ];
00763                 usTemp += m_stRegisters.SP;
00764                 *pusResult_ = &m_pusRAM[ usTemp ];
00765                 ucRetVal++;
00766             }
00767             break;
00768         case ARG_SP:
00769             *pusResult_ = &(m_stRegisters.SP);
00770             break;
00771         case ARG_PC:
00772             *pusResult_ = &(m_stRegisters.PC);
00773             break;
00774         case ARG_EX:
00775             *pusResult_ = &(m_stRegisters.EX);
00776             break;
00777         case ARG_NEXT_WORD:
00778             *pusResult_ = &m_pusRAM[ m_pusROM[ m_stRegisters.PC++ ] ];
00779             ucRetVal++;
00780             break;
00781         case ARG_NEXT_LITERAL:
00782             *pusResult_ = &m_pusROM[ m_stRegisters.PC++ ];
00783             ucRetVal++;
00784             break;
00785         case ARG_LITERAL_0:
00786             *pusResult_ = &m_usTempA;
00787             m_usTempA = (K_USHORT)(-1);
00788             break;
00789         case ARG_LITERAL_1:      case ARG_LITERAL_2:      case ARG_LITERAL_3:      case ARG_LITERAL_4:
00790         case ARG_LITERAL_5:      case ARG_LITERAL_6:      case ARG_LITERAL_7:      case ARG_LITERAL_8:
00791         case ARG_LITERAL_9:      case ARG_LITERAL_A:      case ARG_LITERAL_B:      case ARG_LITERAL_C:
00792         case ARG_LITERAL_D:      case ARG_LITERAL_E:      case ARG_LITERAL_F:      case ARG_LITERAL_10:
00793         case ARG_LITERAL_11: case ARG_LITERAL_12: case ARG_LITERAL_13: case ARG_LITERAL_14:
00794         case ARG_LITERAL_15: case ARG_LITERAL_16: case ARG_LITERAL_17: case ARG_LITERAL_18:
00795         case ARG_LITERAL_19: case ARG_LITERAL_1A: case ARG_LITERAL_1B: case ARG_LITERAL_1C:
00796         case ARG_LITERAL_1D: case ARG_LITERAL_1E: case ARG_LITERAL_1F:
00797             *pusResult_ = &m_usTempA;
00798             m_usTempA = (K_USHORT)(ucOpType_ - ARG_LITERAL_1);
00799             break;
00800         default:
00801             break;
00802     }
00803     return ucRetVal;
00804 }
00805
00806 //-----
00807 void DCPU::RunOpcode()
00808 {
00809     // Fetch the opcode @ the current program counter
00810     K_USHORT usWord = m_pusROM[ m_stRegisters.PC++ ];

```

```

00811     K_UCHAR ucOp = (K_UCHAR)DCPU_NORMAL_OPCODE_MASK(usWord);
00812     K_UCHAR ucA = (K_UCHAR)DCPU_A_MASK(usWord);
00813     K_UCHAR ucB = (K_UCHAR)DCPU_B_MASK(usWord);
00814     K_UCHAR ucSize = 1;
00815
00816     DBG_PRINT("0x%04X: %04X\n", m_stRegisters.PC - 1, usWord );
00817
00818     // Decode the opcode
00819     if (ucOp)
00820     {
00821         bool bRunNext = true;
00822
00823         a = &m_usTempA;
00824         b = 0;
00825
00826         // If this is a "basic" opcode, decode "a" and "b"
00827         ucSize += GetOperand( ucA , &a );
00828         ucSize += GetOperand( ucB, &b );
00829
00830         // Add the cycles to the runtime clock
00831         m_ulCycleCount += (K_ULONG)aucBasicOpcodeCycles[ ucOp ];
00832         m_ulCycleCount += (ucSize - 1);
00833
00834         // Execute the instruction once we've decoded the opcode and
00835         // processed the arguments.
00836         switch (DCPU_NORMAL_OPCODE_MASK(usWord))
00837         {
00838             case OP_SET:      SET();          break;
00839             case OP_ADD:      ADD();          break;
00840             case OP_SUB:      SUB();          break;
00841
00842             case OP_MUL:      MUL();          break;
00843             case OP_MLI:      MLI();          break;
00844             case OP_DIV:      DIV();          break;
00845             case OP_DVI:      DVI();          break;
00846             case OP_MOD:      MOD();          break;
00847             case OP_MDI:      MDI();          break;
00848             case OP_AND:      AND();          break;
00849             case OP_BOR:      BOR();          break;
00850             case OP_XOR:      XOR();          break;
00851             case OP_SHR:      SHR();          break;
00852             case OP_ASR:      ASR();          break;
00853             case OP_SHL:      SHL();          break;
00854             case OP_IFB:      bRunNext = IFB(); break;
00855             case OP_IFC:      bRunNext = IFC(); break;
00856             case OP_IFE:      bRunNext = IFE(); break;
00857             case OP_IFN:      bRunNext = IFN(); break;
00858             case OP_IFG:      bRunNext = IFG(); break;
00859             case OP_IFA:      bRunNext = IFA(); break;
00860             case OP_IFL:      bRunNext = IFL(); break;
00861             case OP_IFU:      bRunNext = IFU(); break;
00862             case OP_ADX:      ADX();          break;
00863             case OP_SBX:      SBX();          break;
00864             case OP_STI:      STI();          break;
00865             case OP_STD:      STD();          break;
00866             default:          break;
00867         }
00868
00869         // If we're not supposed to run the next instruction (i.e. skip it
00870         // due to failed condition), adjust the PC.
00871         if (!bRunNext)
00872         {
00873             // Skipped branches take an extra cycle
00874             m_ulCycleCount++;
00875
00876             // Skip the next opcode
00877             usWord = m_pusROM[ m_stRegisters.PC++ ];
00878             if (DCPU_NORMAL_OPCODE_MASK(usWord))
00879             {
00880                 DBG_PRINT( "Skipping Basic Opcode: %X\n",
DCPU_NORMAL_OPCODE_MASK(usWord));
00881                 // If this is a "basic" opcode, decode "a" and "b" - we do this to make sure our
00882                 // PC gets adjusted properly.
00883                 GetOperand( DCPU_A_MASK(usWord), &a );
00884                 GetOperand( DCPU_B_MASK(usWord), &b );
00885             }
00886             else
00887             {
00888                 DBG_PRINT( "Skipping Extended Opcode: %X\n", DCPU_EXTENDED_OPCODE_MASK(usWord));
00889                 GetOperand( DCPU_A_MASK(usWord), &a );
00890             }
00891         }
00892     }
00893     else
00894     {
00895         // Extended opcode. These only have a single argument, stored in the
00896         // "a" field.

```

```

00897         GetOperand( ucA, &a );
00898         m_ulCycleCount++;
00899
00900         // Execute the "extended" instruction now that the opcode has been
00901         // decoded, and the arguments processed.
00902         switch (ucB)
00903         {
00904             case OP_EX_JSR:      JSR();      break;
00905             case OP_EX_INT:      INT();      break;
00906             case OP_EX_IAG:      IAG();      break;
00907             case OP_EX_IAS:      IAS();      break;
00908             case OP_EX_RFI:      RFI();      break;
00909             case OP_EX_IAQ:      IAQ();      break;
00910             case OP_EX_HWN:      HWN();      break;
00911             case OP_EX_HWQ:      HWQ();      break;
00912             case OP_EX_HWI:      HWI();      break;
00913             default:             break;
00914         }
00915     }
00916
00917     // Process an interrupt from the queue (if there is one)
00918     ProcessInterruptQueue();
00919 }
00920
00921 //-----
00922 void DCPU::SendInterrupt( K_USHORT usMessage_ )
00923 {
00924     if (m_stRegisters.IA == 0)
00925     {
00926         // If IA is not set, return out.
00927         return;
00928     }
00929
00930     // Either acknowledge the interrupt immediately, or queue it.
00931     if (m_bInterruptQueueing == false)
00932     {
00933         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00934         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00935
00936         m_stRegisters.A = usMessage_;
00937         m_stRegisters.PC = m_stRegisters.IA;
00938         m_bInterruptQueueing = true;
00939     }
00940     else
00941     {
00942         // Add interrupt message to the queue
00943         m_ausInterruptQueue[ ++m_ucQueueLevel ] = usMessage_;
00944     }
00945 }
00946
00947 //-----
00948 void DCPU::AddPlugin( DCPUPlugin *pclPlugin_ )
00949 {
00950     m_clPluginList.Add( (LinkListNode*)pclPlugin_ );
00951 }

```

17.47 /home/mo/mark3-source/embedded/stage/src/dcpu.h File Reference

DCPU-16 emulator.

```

#include "kerneltypes.h"
#include "ll.h"

```

Classes

- struct [DCPU_Registers](#)
Structure defining the [DCPU](#) hardware registers.
- class [DCPUPlugin](#)
Class used to provide the hardware device abstraction between the [DCPU-16](#) emulator/VM and the host system.
- class [DCPU](#)
[DCPU](#) emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Macros

- `#define DCPU_NORMAL_OPCODE_MASK(x) ((K_USHORT)(x & 0x001F))`
DCPU v1.7 CPU emulator.
- `#define DCPU_EXTENDED_OPCODE_MASK(x) ((K_USHORT)((x >> 5) & 0x001F))`
- `#define DCPU_A_MASK(x) ((K_USHORT)((x >> 10) & 0x003F))`
- `#define DCPU_B_MASK(x) ((K_USHORT)((x >> 5) & 0x001F))`
- `#define DCPU_BUILD_NORMAL(x, y, z) (((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 | ((K_USHORT)(z) & 0x003F) << 10)`
- `#define DCPU_BUILD_EXTENDED(x, y) (((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10))`

Typedefs

- `typedef void(* DCPU_Callback)(DCPU *pcIVM_)`
Callback function type used to implement HWI for VM->Host communications.

Enumerations

- `enum DCPU_OpBasic {`
`OP_NON_BASIC = 0, OP_SET, OP_ADD, OP_SUB,`
`OP_MUL, OP_MLI, OP_DIV, OP_DVI,`
`OP_MOD, OP_MDI, OP_AND, OP_BOR,`
`OP_XOR, OP_SHR, OP_ASR, OP_SHL,`
`OP_IFB, OP_IFC, OP_IFE, OP_IFN,`
`OP_IFG, OP_IFA, OP_IFL, OP_IFU,`
`OP_18, OP_19, OP_ADX, OP_SBX,`
`OP_1C, OP_1D, OP_STI, OP_STD }`
DCPU Basic Opcodes.
- `enum DCPU_OpExtended {`
`OP_EX_RESERVED = 0, OP_EX_JSR, OP_EX_2, OP_EX_3,`
`OP_EX_4, OP_EX_5, OP_EX_6, OP_EX_7,`
`OP_EX_INT, OP_EX_IAG, OP_EX_IAS, OP_EX_RFI,`
`OP_EX_IAQ, OP_EX_D, OP_EX_E, OP_EX_F,`
`OP_EX_HWN, OP_EX_HWQ, OP_EX_HWI, OP_EX_13,`
`OP_EX_14, OP_EX_15, OP_EX_16, OP_EX_17,`
`OP_EX_18, OP_EX_19, OP_EX_1A, OP_EX_1B,`
`OP_EX_1C, OP_EX_1D, OP_EX_1E, OP_EX_1F }`
DCPU Extended opcodes.
- `enum DCPU_Argument {`
`ARG_A = 0, ARG_B, ARG_C, ARG_X,`
`ARG_Y, ARG_Z, ARG_I, ARG_J,`
`ARG_BRACKET_A, ARG_BRACKET_B, ARG_BRACKET_C, ARG_BRACKET_X,`
`ARG_BRACKET_Y, ARG_BRACKET_Z, ARG_BRACKET_I, ARG_BRACKET_J,`
`ARG_WORD_A, ARG_WORD_B, ARG_WORD_C, ARG_WORD_X,`
`ARG_WORD_Y, ARG_WORD_Z, ARG_WORD_I, ARG_WORD_J,`
`ARG_PUSH_POP_SP, ARG_PEEK_SP, ARG_WORD_SP, ARG_SP,`
`ARG_PC, ARG_EX, ARG_NEXT_WORD, ARG_NEXT_LITERAL,`
`ARG_LITERAL_0, ARG_LITERAL_1, ARG_LITERAL_2, ARG_LITERAL_3,`
`ARG_LITERAL_4, ARG_LITERAL_5, ARG_LITERAL_6, ARG_LITERAL_7,`
`ARG_LITERAL_8, ARG_LITERAL_9, ARG_LITERAL_A, ARG_LITERAL_B,`
`ARG_LITERAL_C, ARG_LITERAL_D, ARG_LITERAL_E, ARG_LITERAL_F,`
`ARG_LITERAL_10, ARG_LITERAL_11, ARG_LITERAL_12, ARG_LITERAL_13,`
`ARG_LITERAL_14, ARG_LITERAL_15, ARG_LITERAL_16, ARG_LITERAL_17,`
`ARG_LITERAL_18, ARG_LITERAL_19, ARG_LITERAL_1A, ARG_LITERAL_1B,`
`ARG_LITERAL_1C, ARG_LITERAL_1D, ARG_LITERAL_1E, ARG_LITERAL_1F }`

Argument formats.

17.47.1 Detailed Description

DCPU-16 emulator.

Definition in file [dcpu.h](#).

17.47.2 Macro Definition Documentation

17.47.2.1 `#define DCPU_NORMAL_OPCODE_MASK(x) ((K_USHORT)(x & 0x001F))`

[DCPU](#) v1.7 CPU emulator.

Basic opcode format: [aaaaaabbbbbbooooo]

Where: - aaaaaa 6-bit source argument

- bbbbb 5-bit destination argument
- o is the opcode itself in a

If oooo = 0, then it's an "extended" opcode

Extended opcode format: [aaaaaaoooooxxxxx]

Where:

- xxxxx = all 0's - (basic opcode)
- ooooo = an extended opcode
- aaaaaa = the argument

Definition at line 48 of file [dcpu.h](#).

17.47.3 Enumeration Type Documentation

17.47.3.1 `enum DCPU_OpBasic`

[DCPU](#) Basic Opcodes.

Enumerator

OP_NON_BASIC special instruction - see below

OP_SET b, a | sets b to a

OP_ADD b, a | sets b to b+a, sets EX to 0x0001 if there's an overflow, 0x0 otherwise

OP_SUB b, a | sets b to b-a, sets EX to 0xffff if there's an underflow, 0x0 otherwise

OP_MUL b, a | sets b to b*a, sets EX to ((b*a)>>16)&0xffff (treats b, a as unsigned)

OP_MLI b, a | like MUL, but treat b, a as signed

OP_DIV b, a | sets b to b/a, sets EX to ((b<<16)/a)&0xffff. if a==0, sets b and EX to 0 instead. (treats b, a as unsigned)

OP_DVI b, a | like DIV, but treat b, a as signed. Rounds towards 0

OP_MOD b, a | sets b to ba. if a==0, sets b to 0 instead.

OP_MDI b, a | like MOD, but treat b, a as signed. (MDI -7, 16 == -7)

OP_AND b, a | sets b to b&a

OP_BOR b, a | sets b to b|a
OP_XOR b, a | sets b to b^a
OP_SHR b, a | sets b to b>>a, sets EX to ((b<<16)>>a)&0xffff (logical shift)
OP_ASR b, a | sets b to b>>a, sets EX to ((b<<16)>>a)&0xffff (arithmetic shift) (treats b as signed)
OP_SHL b, a | sets b to b<<a, sets EX to ((b<<a)>>16)&0xffff
OP_IFB b, a | performs next instruction only if (b&a)!=0
OP_IFC b, a | performs next instruction only if (b&a)==0
OP_IFE b, a | performs next instruction only if b==a
OP_IFN b, a | performs next instruction only if b!=a
OP_IFG b, a | performs next instruction only if b>a
OP_IFA b, a | performs next instruction only if b>a (signed)
OP_IFL b, a | performs next instruction only if b<a
OP_IFU b, a | performs next instruction only if b<a (signed)
OP_18 UNDEFINED
OP_19 UNDEFINED
OP_ADX b, a | sets b to b+a+EX, sets EX to 0x0001 if there is an over-flow, 0x0 otherwise
OP_SBX b, a | sets b to b-a+EX, sets EX to 0xFFFF if there is an under-flow, 0x0 otherwise
OP_1C UNDEFINED
OP_1D UNDEFINED
OP_STI b, a | sets b to a, then increases I and J by 1
OP_STD b, a | sets b to a, then decreases I and J by 1

Definition at line 99 of file [dcpu.h](#).

17.47.3.2 enum DCPU_OpExtended

[DCPU](#) Extended opcodes.

Enumerator

OP_EX_JSR a - pushes the address of the next instruction to the stack, then sets PC to a
OP_EX_2 UNDEFINED
OP_EX_3 UNDEFINED
OP_EX_4 UNDEFINED
OP_EX_5 UNDEFINED
OP_EX_6 UNDEFINED
OP_EX_7 UNDEFINED
OP_EX_INT Invoke software interrupt "a".
OP_EX_IAG Get interrupt address in "a".
OP_EX_IAS Set interrupt address from "a".
OP_EX_RFI Disables interrupt queueing, pops A from the stack, then pops PC from the stack.
OP_EX_IAQ if a is nonzero, interrupts will be added to the queue instead of triggered. if a is zero, interrupts will be triggered as normal again
OP_EX_D UNDEFINED
OP_EX_E UNDEFINED
OP_EX_F UNDEFINED
OP_EX_HWN Sets "a" to number of connected HW devices.

OP_EX_HWQ Set registers with information about hardware at index "a".

OP_EX_HWI Send an interrupt to hardware interface "a".

OP_EX_13 UNDEFINED

OP_EX_14 UNDEFINED

OP_EX_15 UNDEFINED

OP_EX_16 UNDEFINED

OP_EX_17 UNDEFINED

OP_EX_18 UNDEFINED

OP_EX_19 UNDEFINED

OP_EX_1A UNDEFINED

OP_EX_1B UNDEFINED

OP_EX_1C UNDEFINED

OP_EX_1D UNDEFINED

OP_EX_1E UNDEFINED

OP_EX_1F UNDEFINED

Definition at line 139 of file [dcpu.h](#).

17.48 dcpu.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00018 #ifndef __DCPU_H__
00019 #define __DCPU_H__
00020
00021 #include "kerneltypes.h"
00022 #include "ll.h"
00023
00024 //-----
00046 //-----
00047 // Macros to access individual elements from within an opcode
00048 #define DCPU_NORMAL_OPCODE_MASK( x ) \
00049     ((K_USHORT)(x & 0x001F))
00050
00051 #define DCPU_EXTENDED_OPCODE_MASK( x ) \
00052     ((K_USHORT)((x >> 5) & 0x001F))
00053
00054 #define DCPU_A_MASK( x ) \
00055     ((K_USHORT)((x >> 10) & 0x003F))
00056
00057 #define DCPU_B_MASK( x ) \
00058     ((K_USHORT)((x >> 5) & 0x001F))
00059
00060 //-----
00061 // Macros to emit opcodes in the normal/extended formats
00062 #define DCPU_BUILD_NORMAL( x, y, z ) \
00063     ( ((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 | ((K_USHORT)(z) & 0x003F) << 10 )
00064
00065 #define DCPU_BUILD_EXTENDED( x, y ) \
00066     ( ((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10) )
00067
00068 //-----
00072 typedef struct
00073 {
00074     union
00075     {
00076         struct

```

```

00077     {
00078         K_USHORT A;
00079         K_USHORT B;
00080         K_USHORT C;
00081         K_USHORT X;
00082         K_USHORT Y;
00083         K_USHORT Z;
00084         K_USHORT I;
00085         K_USHORT J;
00086         K_USHORT PC;
00087         K_USHORT SP;
00088         K_USHORT EX;
00089         K_USHORT IA;
00090     };
00091     K_USHORT ausRegisters[12];
00092 };
00093 } DCPU_Registers;
00094
00095 //-----
00099 typedef enum
00100 {
00101     OP_NON_BASIC = 0,
00102     OP_SET,
00103     OP_ADD,
00104     OP_SUB,
00105     OP_MUL,
00106     OP_MLI,
00107     OP_DIV,
00108     OP_DVI,
00109     OP_MOD,
00110     OP_MDI,
00111     OP_AND,
00112     OP_BOR,
00113     OP_XOR,
00114     OP_SHR,
00115     OP_ASR,
00116     OP_SHL,
00117     OP_IFB,
00118     OP_IFC,
00119     OP_IFE,
00120     OP_IFN,
00121     OP_IFG,
00122     OP_IFA,
00123     OP_IFL,
00124     OP_IFU,
00125     OP_18,
00126     OP_19,
00127     OP_ADX,
00128     OP_SBX,
00129     OP_1C,
00130     OP_1D,
00131     OP_STI,
00132     OP_STD
00133 } DCPU_OpBasic;
00134
00135 //-----
00139 typedef enum
00140 {
00141     OP_EX_RESERVED = 0,
00142     OP_EX_JSR,
00143     OP_EX_2,
00144     OP_EX_3,
00145     OP_EX_4,
00146     OP_EX_5,
00147     OP_EX_6,
00148     OP_EX_7,
00149     OP_EX_INT,
00150     OP_EX_IAG,
00151     OP_EX_IAS,
00152     OP_EX_RFI,
00153     OP_EX_IAQ,
00154     OP_EX_D,
00155     OP_EX_E,
00156     OP_EX_F,
00157     OP_EX_HWN,
00158     OP_EX_HWQ,
00159     OP_EX_HWI,
00160     OP_EX_13,
00161     OP_EX_14,
00162     OP_EX_15,
00163     OP_EX_16,
00164     OP_EX_17,
00165     OP_EX_18,
00166     OP_EX_19,
00167     OP_EX_1A,
00168     OP_EX_1B,
00169     OP_EX_1C,

```

```

00170     OP_EX_1D,
00171     OP_EX_1E,
00172     OP_EX_1F
00173 } DCPU_OpExtended;
00174
00175 //-----
00180 typedef enum
00181 {
00182     ARG_A = 0,
00183     ARG_B,
00184     ARG_C,
00185     ARG_X,
00186     ARG_Y,
00187     ARG_Z,
00188     ARG_I,
00189     ARG_J,
00190
00191     ARG_BRACKET_A,
00192     ARG_BRACKET_B,
00193     ARG_BRACKET_C,
00194     ARG_BRACKET_X,
00195     ARG_BRACKET_Y,
00196     ARG_BRACKET_Z,
00197     ARG_BRACKET_I,
00198     ARG_BRACKET_J,
00199
00200     ARG_WORD_A,
00201     ARG_WORD_B,
00202     ARG_WORD_C,
00203     ARG_WORD_X,
00204     ARG_WORD_Y,
00205     ARG_WORD_Z,
00206     ARG_WORD_I,
00207     ARG_WORD_J,
00208
00209     ARG_PUSH_POP_SP,
00210     ARG_PEEK_SP,
00211     ARG_WORD_SP,
00212     ARG_SP,
00213     ARG_PC,
00214     ARG_EX,
00215     ARG_NEXT_WORD,
00216     ARG_NEXT_LITERAL,
00217
00218     ARG_LITERAL_0,
00219     ARG_LITERAL_1,
00220     ARG_LITERAL_2,
00221     ARG_LITERAL_3,
00222     ARG_LITERAL_4,
00223     ARG_LITERAL_5,
00224     ARG_LITERAL_6,
00225     ARG_LITERAL_7,
00226     ARG_LITERAL_8,
00227     ARG_LITERAL_9,
00228     ARG_LITERAL_A,
00229     ARG_LITERAL_B,
00230     ARG_LITERAL_C,
00231     ARG_LITERAL_D,
00232     ARG_LITERAL_E,
00233     ARG_LITERAL_F,
00234     ARG_LITERAL_10,
00235     ARG_LITERAL_11,
00236     ARG_LITERAL_12,
00237     ARG_LITERAL_13,
00238     ARG_LITERAL_14,
00239     ARG_LITERAL_15,
00240     ARG_LITERAL_16,
00241     ARG_LITERAL_17,
00242     ARG_LITERAL_18,
00243     ARG_LITERAL_19,
00244     ARG_LITERAL_1A,
00245     ARG_LITERAL_1B,
00246     ARG_LITERAL_1C,
00247     ARG_LITERAL_1D,
00248     ARG_LITERAL_1E,
00249     ARG_LITERAL_1F
00250
00251 } DCPU_Argument;
00252
00253 //-----
00254 class DCPU;    // Forward declaration - required by the plugin class
00255
00256 //-----
00260 typedef void (*DCPU_Callback) (DCPU *pclVM_);
00261
00262 //-----
00267 class DCPUPlugin : public LinkListNode

```

```

00268 {
00269 public:
00288     void Init(      K_USHORT usDeviceNumber_,
00289                 K_ULONG ulHWID_,
00290                 K_ULONG ulVID_,
00291                 K_USHORT usVersion_,
00292                 DCPU_Callback pfCallback_)
00293     {
00294         m_ulHWID = ulHWID_;
00295         m_ulVID = ulVID_;
00296         m_usDeviceNumber = usDeviceNumber_;
00297         m_usVersion = usVersion_;
00298         m_pfCallback = pfCallback_;
00299     }
00300
00311     void Enumerate( DCPU_Registers *pstRegisters_ )
00312     {
00313         pstRegisters_>A = (K_USHORT) (m_ulHWID & 0x0000FFFF);
00314         pstRegisters_>B = (K_USHORT) ((m_ulHWID >> 16) & 0x0000FFFF);
00315         pstRegisters_>C = m_usVersion;
00316         pstRegisters_>X = (K_USHORT) (m_ulVID & 0x0000FFFF);
00317         pstRegisters_>Y = (K_USHORT) ((m_ulVID >> 16) & 0x0000FFFF);
00318     }
00319
00327     void Interrupt( DCPU *pclCPU_ )
00328     {
00329         m_pfCallback(pclCPU_);
00330     }
00331
00339     K_USHORT GetDeviceNumber()
00340     {
00341         return m_usDeviceNumber;
00342     }
00343
00344     friend class DCPUPuginList;
00345 private:
00346     K_USHORT      m_usDeviceNumber;
00347     K_ULONG       m_ulHWID;
00348     K_ULONG       m_ulVID;
00349     K_USHORT      m_usVersion;
00350
00351     DCPU_Callback m_pfCallback;
00352 };
00353
00354 //-----
00359 class DCPU
00360 {
00361 public:
00375     void Init( K_USHORT *pusRAM_, K_USHORT usRAMSize_, const K_USHORT *pusROM_, K_USHORT usROMSize_ );
00376
00382     void RunOpcode();
00383
00391     DCPU_Registers *GetRegisters() { return &
m_stRegisters; }
00392
00400     void SendInterrupt( K_USHORT usMessage_ );
00401
00409     void AddPlugin( DCPUPugin *pclPlugin_ );
00410
00411 private:
00412
00413     // Basic opcodes
00414     void SET();
00415     void ADD();
00416     void SUB();
00417     void MUL();
00418     void MLI();
00419     void DIV();
00420     void DVI();
00421     void MOD();
00422     void MDI();
00423     void AND();
00424     void BOR();
00425     void XOR();
00426     void SHR();
00427     void ASR();
00428     void SHL();
00429     bool IFB();
00430     bool IFC();
00431     bool IFE();
00432     bool IFN();
00433     bool IFG();
00434     bool IFA();
00435     bool IFL();
00436     bool IFU();
00437     void ADX();
00438     void SBX();

```

```

00439     void STI();
00440     void STD();
00441
00442     // Extended opcodes
00443     void JSR();
00444     void INT();
00445     void IAG();
00446     void IAS();
00447     void RFI();
00448     void IAQ();
00449     void HWN();
00450     void HWQ();
00451     void HWI();
00452
00460     K_UCHAR GetOperand( K_UCHAR ucOpType_, K_USHORT **pusResult_ );
00461
00462
00468     void ProcessInterruptQueue();
00469
00470     DCPU_Registers m_stRegisters;
00471
00472     K_USHORT *a;
00473     K_USHORT *b;
00474
00475     K_USHORT m_usTempA;
00476
00477     K_USHORT *m_pusRAM;
00478     K_USHORT m_usRAMSize;
00479
00480     K_USHORT *m_pusROM;
00481     K_USHORT m_usROMSize;
00482
00483     K_ULONG m_ulCycleCount;
00484
00485     K_BOOL m_bInterruptQueueing;
00486     K_UCHAR m_ucQueueLevel;
00487     K_USHORT m_ausInterruptQueue[ 8 ];
00488
00489     DoubleLinkedList m_clPluginList;
00490 };
00491
00492 #endif

```

17.49 /home/mo/mark3-source/embedded/stage/src/debug_tokens.h File Reference

Hex codes/translation tables used for efficient string tokenization.

Macros

- #define **BLOCKING_CPP** 0x0001 /* SUBSTITUTE="blocking.cpp" */
Source file names start at 0x0000.
- #define **DRIVER_CPP** 0x0002 /* SUBSTITUTE="driver.cpp" */
- #define **KERNEL_CPP** 0x0003 /* SUBSTITUTE="kernel.cpp" */
- #define **LL_CPP** 0x0004 /* SUBSTITUTE="ll.cpp" */
- #define **MESSAGE_CPP** 0x0005 /* SUBSTITUTE="message.cpp" */
- #define **MUTEX_CPP** 0x0006 /* SUBSTITUTE="mutex.cpp" */
- #define **PROFILE_CPP** 0x0007 /* SUBSTITUTE="profile.cpp" */
- #define **QUANTUM_CPP** 0x0008 /* SUBSTITUTE="quantum.cpp" */
- #define **SCHEDULER_CPP** 0x0009 /* SUBSTITUTE="scheduler.cpp" */
- #define **SEMAPHORE_CPP** 0x000A /* SUBSTITUTE="semaphore.cpp" */
- #define **THREAD_CPP** 0x000B /* SUBSTITUTE="thread.cpp" */
- #define **THREADLIST_CPP** 0x000C /* SUBSTITUTE="threadlist.cpp" */
- #define **TIMERLIST_CPP** 0x000D /* SUBSTITUTE="timerlist.cpp" */
- #define **KERNELSWI_CPP** 0x000E /* SUBSTITUTE="kernelswi.cpp" */
- #define **KERNELTIMER_CPP** 0x000F /* SUBSTITUTE="kerneltimer.cpp" */
- #define **KPROFILE_CPP** 0x0010 /* SUBSTITUTE="kprofile.cpp" */
- #define **THREADPORT_CPP** 0x0011 /* SUBSTITUTE="threadport.cpp" */
- #define **BLOCKING_H** 0x1000 /* SUBSTITUTE="blocking.h" */

Header file names start at 0x1000.

- #define **DRIVER_H** 0x1001 /* SUBSTITUTE="driver.h" */
- #define **KERNEL_H** 0x1002 /* SUBSTITUTE="kernel.h" */
- #define **KERNELTYPES_H** 0x1003 /* SUBSTITUTE="kerneltypes.h" */
- #define **LL_H** 0x1004 /* SUBSTITUTE="ll.h" */
- #define **MANUAL_H** 0x1005 /* SUBSTITUTE="manual.h" */
- #define **MARK3CFG_H** 0x1006 /* SUBSTITUTE="mark3cfg.h" */
- #define **MESSAGE_H** 0x1007 /* SUBSTITUTE="message.h" */
- #define **MUTEX_H** 0x1008 /* SUBSTITUTE="mutex.h" */
- #define **PROFILE_H** 0x1009 /* SUBSTITUTE="profile.h" */
- #define **PROFILING_RESULTS_H** 0x100A /* SUBSTITUTE="profiling_results.h" */
- #define **QUANTUM_H** 0x100B /* SUBSTITUTE="quantum.h" */
- #define **SCHEDULER_H** 0x100C /* SUBSTITUTE="scheduler.h" */
- #define **SEMAPHORE_H** 0x100D /* SUBSTITUTE="ksemaphore.h" */
- #define **THREAD_H** 0x100E /* SUBSTITUTE="thread.h" */
- #define **THREADLIST_H** 0x100F /* SUBSTITUTE="threadlist.h" */
- #define **TIMERLIST_H** 0x1010 /* SUBSTITUTE="timerlist.h" */
- #define **KERNELSWI_H** 0x1011 /* SUBSTITUTE="kernelswi.h" */
- #define **KERNELTIMER_H** 0x1012 /* SUBSTITUTE="kerneltimer.h" */
- #define **KPROFILE_H** 0x1013 /* SUBSTITUTE="kprofile.h" */
- #define **THREADPORT_H** 0x1014 /* SUBSTITUTE="threadport.h" */
- #define **STR_PANIC** 0x2000 /* SUBSTITUTE="!Panic!" */

Indexed strings start at 0x2000.

- #define **STR_MARK3_INIT** 0x2001 /* SUBSTITUTE="Initializing Kernel Objects" */
- #define **STR_KERNEL_ENTER** 0x2002 /* SUBSTITUTE="Starting Kernel" */
- #define **STR_THREAD_START** 0x2003 /* SUBSTITUTE="Switching to First Thread" */
- #define **STR_START_ERROR** 0x2004 /* SUBSTITUTE="Error starting kernel - function should never return" */
- #define **STR_THREAD_CREATE** 0x2005 /* SUBSTITUTE="Creating Thread" */
- #define **STR_STACK_SIZE_1** 0x2006 /* SUBSTITUTE=" Stack Size: %1" */
- #define **STR_PRIORITY_1** 0x2007 /* SUBSTITUTE=" Priority: %1" */
- #define **STR_THREAD_ID_1** 0x2008 /* SUBSTITUTE=" Thread ID: %1" */
- #define **STR_ENTRYPOINT_1** 0x2009 /* SUBSTITUTE=" EntryPoint: %1" */
- #define **STR_CONTEXT_SWITCH_1** 0x200A /* SUBSTITUTE="Context Switch To Thread: %1" */
- #define **STR_IDLING** 0x200B /* SUBSTITUTE="Idling CPU" */
- #define **STR_WAKEUP** 0x200C /* SUBSTITUTE="Waking up" */
- #define **STR_SEMAPHORE_PEND_1** 0x200D /* SUBSTITUTE="Semaphore Pend: %1" */
- #define **STR_SEMAPHORE_POST_1** 0x200E /* SUBSTITUTE="Semaphore Post: %1" */
- #define **STR_MUTEX_CLAIM_1** 0x200F /* SUBSTITUTE="Mutex Claim: %1" */
- #define **STR_MUTEX_RELEASE_1** 0x2010 /* SUBSTITUTE="Mutex Release: %1" */
- #define **STR_THREAD_BLOCK_1** 0x2011 /* SUBSTITUTE="Thread %1 Blocked" */
- #define **STR_THREAD_UNBLOCK_1** 0x2012 /* SUBSTITUTE="Thread %1 Unblocked" */
- #define **STR_ASSERT_FAILED** 0x2013 /* SUBSTITUTE="Assertion Failed" */
- #define **STR_SCHEDULE_1** 0x2014 /* SUBSTITUTE="Scheduler chose %1" */
- #define **STR_THREAD_START_1** 0x2015 /* SUBSTITUTE="Thread Start: %1" */
- #define **STR_THREAD_EXIT_1** 0x2016 /* SUBSTITUTE="Thread Exit: %1" */
- #define **STR_UNDEFINED** 0xFFFF /* SUBSTITUTE="UNDEFINED" */

17.49.1 Detailed Description

Hex codes/translation tables used for efficient string tokenization. We use this for efficiently encoding strings used for kernel traces, debug prints, etc. The upside - this is really fast and efficient for encoding strings and data. Downside? The tools need to parse this header file in order to convert the enumerated data into actual strings, decoding them.

Definition in file [debug_tokens.h](#).

17.50 debug_tokens.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00025 #ifndef __DEBUG_TOKENS_H__
00026 #define __DEBUG_TOKENS_H__
00027 //-----
00029 #define BLOCKING_CPP          0x0001      /* SUBSTITUTE="blocking.cpp" */
00030 #define DRIVER_CPP           0x0002      /* SUBSTITUTE="driver.cpp" */
00031 #define KERNEL_CPP           0x0003      /* SUBSTITUTE="kernel.cpp" */
00032 #define LL_CPP               0x0004      /* SUBSTITUTE="ll.cpp" */
00033 #define MESSAGE_CPP          0x0005      /* SUBSTITUTE="message.cpp" */
00034 #define MUTEX_CPP            0x0006      /* SUBSTITUTE="mutex.cpp" */
00035 #define PROFILE_CPP          0x0007      /* SUBSTITUTE="profile.cpp" */
00036 #define QUANTUM_CPP          0x0008      /* SUBSTITUTE="quantum.cpp" */
00037 #define SCHEDULER_CPP        0x0009      /* SUBSTITUTE="scheduler.cpp" */
00038 #define SEMAPHORE_CPP        0x000A      /* SUBSTITUTE="semaphore.cpp" */
00039 #define THREAD_CPP           0x000B      /* SUBSTITUTE="thread.cpp" */
00040 #define THREADLIST_CPP       0x000C      /* SUBSTITUTE="threadlist.cpp" */
00041 #define TIMERLIST_CPP        0x000D      /* SUBSTITUTE="timerlist.cpp" */
00042 #define KERNELSWI_CPP        0x000E      /* SUBSTITUTE="kernelswi.cpp" */
00043 #define KERNELTIMER_CPP      0x000F      /* SUBSTITUTE="kerneltimer.cpp" */
00044 #define KPROFILE_CPP         0x0010      /* SUBSTITUTE="kprofile.cpp" */
00045 #define THREADPORT_CPP       0x0011      /* SUBSTITUTE="threadport.cpp" */
00046
00047 //-----
00049 #define BLOCKING_H           0x1000      /* SUBSTITUTE="blocking.h" */
00050 #define DRIVER_H            0x1001      /* SUBSTITUTE="driver.h" */
00051 #define KERNEL_H            0x1002      /* SUBSTITUTE="kernel.h" */
00052 #define KERNELTYPES_H       0x1003      /* SUBSTITUTE="kerneltypes.h" */
00053 #define LL_H                0x1004      /* SUBSTITUTE="ll.h" */
00054 #define MANUAL_H            0x1005      /* SUBSTITUTE="manual.h" */
00055 #define MARK3CFG_H          0x1006      /* SUBSTITUTE="mark3cfg.h" */
00056 #define MESSAGE_H           0x1007      /* SUBSTITUTE="message.h" */
00057 #define MUTEX_H             0x1008      /* SUBSTITUTE="mutex.h" */
00058 #define PROFILE_H           0x1009      /* SUBSTITUTE="profile.h" */
00059 #define PROFILING_RESULTS_H 0x100A      /* SUBSTITUTE="profiling_results.h" */
00060 #define QUANTUM_H           0x100B      /* SUBSTITUTE="quantum.h" */
00061 #define SCHEDULER_H         0x100C      /* SUBSTITUTE="scheduler.h" */
00062 #define SEMAPHORE_H         0x100D      /* SUBSTITUTE="ksemaphore.h" */
00063 #define THREAD_H            0x100E      /* SUBSTITUTE="thread.h" */
00064 #define THREADLIST_H        0x100F      /* SUBSTITUTE="threadlist.h" */
00065 #define TIMERLIST_H         0x1010      /* SUBSTITUTE="timerlist.h" */
00066 #define KERNELSWI_H         0x1011      /* SUBSTITUTE="kernelswi.h" */
00067 #define KERNELTIMER_H       0x1012      /* SUBSTITUTE="kerneltimer.h" */
00068 #define KPROFILE_H          0x1013      /* SUBSTITUTE="kprofile.h" */
00069 #define THREADPORT_H        0x1014      /* SUBSTITUTE="threadport.h" */
00070
00071 //-----
00073 #define STR_PANIC            0x2000      /* SUBSTITUTE="!Panic!" */
00074 #define STR_MARK3_INIT       0x2001      /* SUBSTITUTE="Initializing Kernel Objects" */
00075 #define STR_KERNEL_ENTER     0x2002      /* SUBSTITUTE="Starting Kernel" */
00076 #define STR_THREAD_START     0x2003      /* SUBSTITUTE="Switching to First Thread" */
00077 #define STR_START_ERROR      0x2004      /* SUBSTITUTE="Error starting kernel - function should never
return" */
00078 #define STR_THREAD_CREATE    0x2005      /* SUBSTITUTE="Creating Thread" */
00079 #define STR_STACK_SIZE_1     0x2006      /* SUBSTITUTE=" Stack Size: %1" */
00080 #define STR_PRIORITY_1       0x2007      /* SUBSTITUTE=" Priority: %1" */
00081 #define STR_THREAD_ID_1      0x2008      /* SUBSTITUTE=" Thread ID: %1" */
00082 #define STR_ENTRYPOINT_1     0x2009      /* SUBSTITUTE=" EntryPoint: %1" */
00083 #define STR_CONTEXT_SWITCH_1 0x200A      /* SUBSTITUTE="Context Switch To Thread: %1" */
00084 #define STR_IDLING           0x200B      /* SUBSTITUTE="Idling CPU" */
00085 #define STR_WAKEUP           0x200C      /* SUBSTITUTE="Waking up" */
00086 #define STR_SEMAPHORE_PEND_1 0x200D      /* SUBSTITUTE="Semaphore Pend: %1" */
00087 #define STR_SEMAPHORE_POST_1 0x200E      /* SUBSTITUTE="Semaphore Post: %1" */
00088 #define STR_MUTEX_CLAIM_1    0x200F      /* SUBSTITUTE="Mutex Claim: %1" */
00089 #define STR_MUTEX_RELEASE_1  0x2010      /* SUBSTITUTE="Mutex Release: %1" */
00090 #define STR_THREAD_BLOCK_1   0x2011      /* SUBSTITUTE="Thread %1 Blocked" */
00091 #define STR_THREAD_UNBLOCK_1 0x2012      /* SUBSTITUTE="Thread %1 Unblocked" */
00092 #define STR_ASSERT_FAILED    0x2013      /* SUBSTITUTE="Assertion Failed" */
00093 #define STR_SCHEDULE_1       0x2014      /* SUBSTITUTE="Scheduler chose %1" */
00094 #define STR_THREAD_START_1   0x2015      /* SUBSTITUTE="Thread Start: %1" */
00095 #define STR_THREAD_EXIT_1    0x2016      /* SUBSTITUTE="Thread Exit: %1" */
00096
00097 //-----

```

```
00098 #define STR_UNDEFINED          0xFFFF      /* SUBSTITUTE="UNDEFINED" */
00099 #endif
```

17.51 /home/mo/mark3-source/embedded/stage/src/draw.h File Reference

Raster graphics APIs Description: Implements basic drawing functionality.

```
#include "kerneltypes.h"
#include "font.h"
#include "colorspace.h"
```

Classes

- struct [DrawPoint_t](#)
Defines a pixel.
- struct [DrawLine_t](#)
Defines a simple line.
- struct [DrawRectangle_t](#)
Defines a rectangle.
- struct [DrawCircle_t](#)
Defines a circle.
- struct [DrawEllipse_t](#)
Defines an ellipse.
- struct [DrawBitmap_t](#)
Defines a bitmap.
- struct [DrawStamp_t](#)
Defines a 1-bit 2D bitmap of arbitrary resolution.
- struct [DrawText_t](#)
Defines a bitmap-rendered string.
- struct [TextFX_t](#)
- struct [DrawWindow_t](#)
Defines the active window - establishes boundaries for drawing on the current display.
- struct [DrawMove_t](#)
Simple 2D copy/paste.
- struct [DrawVector_t](#)
Specifies a single 2D point.
- struct [DrawPoly_t](#)
Defines the structure of an arbitrary polygon.

Macros

- #define [TEXTFX_FLAG_OPAQUE_BG](#) (0x01)
Use an opaque BG.
- #define [TEXTFX_FLAG_ROTATE](#) (0x02)
Apply text rotation.
- #define [TEXTFX_FLAG_SCALE_X](#) (0x04)
Scale the text horizontally.
- #define [TEXTFX_FLAG_SCALE_Y](#) (0x08)
Scale the text vertically.

Enumerations

- enum **DisplayEvent_t** {
 DISPLAY_EVENT_SET_PIXEL = 0x00, **DISPLAY_EVENT_GET_PIXEL**, **DISPLAY_EVENT_CLEAR**, **DISPLAY_EVENT_LINE**,
 DISPLAY_EVENT_RECTANGLE, **DISPLAY_EVENT_CIRCLE**, **DISPLAY_EVENT_ELLIPSE**, **DISPLAY_EVENT_BITMAP**,
 DISPLAY_EVENT_STAMP, **DISPLAY_EVENT_TEXT**, **DISPLAY_EVENT_MOVE**, **DISPLAY_EVENT_POLY** }

17.51.1 Detailed Description

Raster graphics APIs Description: Implements basic drawing functionality. This forms a hardware abstraction layer which requires a backend for rendering.

Definition in file [draw.h](#).

17.52 draw.h

```

00001 /*-----
00002
00003 |-----|-----|-----|-----|-----|-----|
00004 | \ / | | | \ / | | | \ / | | | \ / | | | \ / |
00005 |  V  | | |  V  | | |  V  | | |  V  | | |  V  |
00006 | / \ | | | / \ | | | / \ | | | / \ | | | / \ |
00007 |_____|_____|_____|_____|_____|_____|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*
00021 //-----
00022
00023 #ifndef __DRAW_H_
00024 #define __DRAW_H_
00025
00026 #include "kerneltypes.h"
00027 #include "font.h"
00028 #include "colorspace.h"
00029
00030 //-----
00031 // Event definitions for 2D hardware accelerated graphics functions
00032 typedef enum
00033 {
00034     ///--[Mandatory for a display driver]-----
00035     DISPLAY_EVENT_SET_PIXEL = 0x00,
00036     DISPLAY_EVENT_GET_PIXEL,
00037
00038     ///--[Optional if supported in hardware]-----
00039     DISPLAY_EVENT_CLEAR,
00040     DISPLAY_EVENT_LINE,
00041     DISPLAY_EVENT_RECTANGLE,
00042     DISPLAY_EVENT_CIRCLE,
00043     DISPLAY_EVENT_ELLIPSE,
00044     DISPLAY_EVENT_BITMAP,
00045     DISPLAY_EVENT_STAMP,
00046     DISPLAY_EVENT_TEXT,
00047     DISPLAY_EVENT_MOVE,
00048     DISPLAY_EVENT_POLY
00049 } DisplayEvent_t;
00050
00051 //-----
00055 typedef struct
00056 {
00057     K_USHORT usX;
00058     K_USHORT usY;
00059     COLOR uColor;
00060 } DrawPoint_t;
00061
00062 //-----
00066 typedef struct
00067 {
00068     K_USHORT usX1;
00069     K_USHORT usX2;

```

```

00070     K_USHORT  usY1;
00071     K_USHORT  usY2;
00072     COLOR    uColor;
00073 } DrawLine_t;
00074 //-----
00078 typedef struct
00079 {
00080     K_USHORT  usLeft;
00081     K_USHORT  usTop;
00082     K_USHORT  usRight;
00083     K_USHORT  usBottom;
00084     COLOR    uLineColor;
00085     K_BOOL    bFill;
00086     COLOR    uFillColor;
00087 } DrawRectangle_t;
00088 //-----
00092 typedef struct
00093 {
00094     K_USHORT  usX;
00095     K_USHORT  usY;
00096     K_USHORT  usRadius;
00097     COLOR    uLineColor;
00098     K_BOOL    bFill;
00099     COLOR    uFillColor;
00100 } DrawCircle_t;
00101 //-----
00105 typedef struct
00106 {
00107     K_USHORT  usX;
00108     K_USHORT  usY;
00109     K_USHORT  usHeight;
00110     K_USHORT  usWidth;
00111     COLOR    uColor;
00112 } DrawEllipse_t;
00113 //-----
00117 typedef struct
00118 {
00119     K_USHORT  usX;
00120     K_USHORT  usY;
00121     K_USHORT  usWidth;
00122     K_USHORT  usHeight;
00123     K_UCHAR   ucBPP;
00124     K_UCHAR   *pucData;
00125 } DrawBitmap_t;
00126 //-----
00130 typedef struct
00131 {
00132     K_USHORT  usX;
00133     K_USHORT  usY;
00134     K_USHORT  usWidth;
00135     K_USHORT  usHeight;
00136     COLOR    uColor;
00137     K_UCHAR   *pucData;
00138 } DrawStamp_t;    // monochrome stamp, bitpacked 8bpp
00139
00140 //-----
00144 typedef struct
00145 {
00146     K_USHORT  usLeft;
00147     K_USHORT  usTop;
00148     COLOR    uColor;
00149     Font_t   *pstFont;
00150     const K_CHAR *pcString;
00151 } DrawText_t;
00152
00153 //-----
00154 #define TEXTFX_FLAG_OPAQUE_BG    (0x01)
00155 #define TEXTFX_FLAG_ROTATE      (0x02)
00156 #define TEXTFX_FLAG_SCALE_X     (0x04)
00157 #define TEXTFX_FLAG_SCALE_Y     (0x08)
00158
00159 //-----
00160 typedef struct
00161 {
00162     K_UCHAR   ucFlags;
00163     COLOR    uBGColor;
00164     K_USHORT  usRotateDeg;
00165     K_USHORT  usScaleX100;
00166     K_USHORT  usScaleY100;
00167 } TextFX_t;
00168
00169 //-----
00175 typedef struct
00176 {
00177     K_USHORT  usLeft;
00178     K_USHORT  usRight;
00179     K_USHORT  usTop;

```

```

00180     K_USHORT  usBottom;
00181 } DrawWindow_t;
00182
00183 //-----
00188 typedef struct
00189 {
00190     K_USHORT  usSrcX;
00191     K_USHORT  usSrcY;
00192     K_USHORT  usDstX;
00193     K_USHORT  usDstY;
00194     K_USHORT  usCopyHeight;
00195     K_USHORT  usCopyWidth;
00196 } DrawMove_t;
00197
00198 //-----
00204 typedef struct
00205 {
00206     K_USHORT  usX;
00207     K_USHORT  usY;
00208 } DrawVector_t;
00209
00210 //-----
00215 typedef struct
00216 {
00217     K_USHORT  usNumPoints;
00218     COLOR     uColor;
00219     K_BOOL    bFill;
00220     DrawVector_t *pstVector;
00221 } DrawPoly_t;
00222
00223 #endif //__DRAW_H_

```

17.53 /home/mo/mark3-source/embedded/stage/src/driver.cpp File Reference

Device driver/hardware abstraction layer.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "driver.h"

```

Classes

- class [DevNull](#)
This class implements the "default" driver (/dev/null)

Macros

- #define `__FILE_ID__` DRIVER_CPP

Functions

- static K_UCHAR **DrvCmp** (const K_CHAR *szStr1_, const K_CHAR *szStr2_)

Variables

- static [DevNull](#) **clDevNull**

17.53.1 Detailed Description

Device driver/hardware abstraction layer.

Definition in file [driver.cpp](#).


```

00094 Driver *DriverList::FindByPath( const K_CHAR *m_pcPath )
00095 {
00096     KERNEL_ASSERT( m_pcPath );
00097     Driver *pclTemp = static_cast<Driver*>(m_clDriverList.
    GetHead());
00098
00099     while (pclTemp)
00100     {
00101         if(DrvCmp(m_pcPath, pclTemp->GetPath()))
00102         {
00103             return pclTemp;
00104         }
00105         pclTemp = static_cast<Driver*>(pclTemp->GetNext());
00106     }
00107     return &clDevNull;
00108 }
00109
00110 #endif

```

17.55 /home/mo/mark3-source/embedded/stage/src/driver.h File Reference

[Driver](#) abstraction framework.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"

```

Classes

- class [Driver](#)
Base device-driver class used in hardware abstraction.
- class [DriverList](#)
List of [Driver](#) objects used to keep track of all device drivers in the system.

17.55.1 Detailed Description

[Driver](#) abstraction framework.

17.55.2 Intro

This is the basis of the driver framework. In the context of Mark3, drivers don't necessarily have to be based on physical hardware peripherals. They can be used to represent algorithms (such as random number generators), files, or protocol stacks. Unlike FunkOS, where driver IO is protected automatically by a mutex, we do not use this kind of protection - we leave it up to the driver implementor to do what's right in its own context. This also frees up the driver to implement all sorts of other neat stuff, like sending messages to threads associated with the driver. Drivers are implemented as character devices, with the standard array of posix-style accessor methods for reading, writing, and general driver control.

A global driver list is provided as a convenient and minimal "filesystem" structure, in which devices can be accessed by name.

17.55.3 Driver Design

A device driver needs to be able to perform the following operations: -Initialize a peripheral -Start/stop a peripheral -Handle I/O control operations -Perform various read/write operations

At the end of the day, that's pretty much all a device driver has to do, and all of the functionality that needs to be presented to the developer.

We abstract all device drivers using a base-class which implements the following methods: -Start/Open -Stop/Close -Control -Read -Write

A basic driver framework and API can thus be implemented in five function calls - that's it! You could even reduce that further by handling the initialize, start, and stop operations inside the "control" operation.

17.55.4 Driver API

In C++, we can implement this as a class to abstract these event handlers, with virtual void functions in the base class overridden by the inherited objects.

To add and remove device drivers from the global table, we use the following methods:

```
void DriverList::Add( Driver *pclDriver_ );
void DriverList::Remove( Driver *pclDriver_ );
```

`DriverList::Add()/Remove()` takes a single arguments the pointer to the object to operate on.

Once a driver has been added to the table, drivers are opened by NAME using `DriverList::FindByName("/dev/name")`. This function returns a pointer to the specified driver if successful, or to a built in `/dev/null` device if the path name is invalid. After a driver is open, that pointer is used for all other driver access functions.

This abstraction is incredibly useful any peripheral or service can be accessed through a consistent set of APIs, that make it easy to substitute implementations from one platform to another. Portability is ensured, the overhead is negligible, and it emphasizes the reuse of both driver and application code as separate entities.

Consider a system with drivers for I2C, SPI, and UART peripherals - under our driver framework, an application can initialize these peripherals and write a greeting to each using the same simple API functions for all drivers:

```
pciI2C = DriverList::FindByName("/dev/i2c");
pciUART = DriverList::FindByName("/dev/tty0");
pciSPI = DriverList::FindByName("/dev/spi");

pciI2C->Write(12, "Hello World!");
pciUART->Write(12, "Hello World!");
pciSPI->Write(12, "Hello World!");
```

Definition in file [driver.h](#).

17.56 driver.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00105 #include "kerneltypes.h"
00106 #include "mark3cfg.h"
00107
00108 #include "ll.h"
00109
00110 #ifndef __DRIVER_H__
00111 #define __DRIVER_H__
00112
00113 #if KERNEL_USE_DRIVER
00114
00115 class DriverList;
00116 //-----
00121 class Driver : public LinkListNode
00122 {
00123 public:
00129     virtual void Init() = 0;
```

```

00130
00138     virtual K_UCHAR Open() = 0;
00139
00147     virtual K_UCHAR Close() = 0;
00148
00164     virtual K_USHORT Read( K_USHORT usBytes_,
00165                             K_UCHAR *pucData_) = 0;
00166
00183     virtual K_USHORT Write( K_USHORT usBytes_,
00184                             K_UCHAR *pucData_) = 0;
00185
00208     virtual K_USHORT Control( K_USHORT usEvent_,
00209                               void *pvDataIn_,
00210                               K_USHORT usSizeIn_,
00211                               void *pvDataOut_,
00212                               K_USHORT usSizeOut_ ) = 0;
00213
00222     void SetName( const K_CHAR *pcName_ ) { m_pcPath = pcName_; }
00223
00231     const K_CHAR *GetPath() { return m_pcPath; }
00232
00233 private:
00234
00236     const K_CHAR *m_pcPath;
00237 };
00238
00239 //-----
00244 class DriverList
00245 {
00246 public:
00254     static void Init();
00255
00264     static void Add( Driver *pclDriver_ ) { m_clDriverList.
Add(pclDriver_); }
00265
00274     static void Remove( Driver *pclDriver_ ) { m_clDriverList.
Remove(pclDriver_); }
00275
00282     static Driver *FindByPath( const K_CHAR *m_pcPath );
00283
00284 private:
00285
00287     static DoubleLinkedList m_clDriverList;
00288 };
00289
00290 #endif //KERNEL_USE_DRIVER
00291
00292 #endif

```

17.57 /home/mo/mark3-source/embedded/stage/src/eventflag.cpp File Reference

Event Flag Blocking Object/IPC-Object implementation.

```

#include "mark3cfg.h"
#include "blocking.h"
#include "kernel.h"
#include "thread.h"
#include "eventflag.h"
#include "timerlist.h"

```

Macros

- #define **EVENT_TRANSACTION_WAIT** (0)
- #define **EVENT_TRANSACTION_SET** (1)
- #define **EVENT_TRANSACTION_CLEAR** (2)
- #define **EVENT_TRANSACTION_TIMEOUT** (3)

Functions

- void **TimedEventFlag_Callback** (Thread *pclOwner_, void *pvData_)


```

00080 #endif
00081 {
00082     // Claim the lock (we know only one thread can hold the lock, only one thread can
00083     // execute at a time, and only threads can call wait)
00084     K_BOOL bSchedState;
00085     if (LockAndQueue(EVENT_TRANSACTION_WAIT, (void*)((K_ADDR)usMask_), &bSchedState))
00086     {
00087         // This should never be able to happen with the logic implemented above
00088         Kernel::Panic( PANIC_EVENT_LOCK_VIOLATION );
00089     }
00090
00091     // Set data on the current thread that needs to be passed into the transaction
00092     // handler (and can't be queued in the simple key-value pair in the transaction
00093     // object)
00094     Scheduler::GetCurrentThread()->SetEventFlagMode(eMode_);
00095 #if KERNEL_USE_TIMERS
00096     Scheduler::GetCurrentThread()->GetTimer()->
00097     SetIntervalTicks(ulTimeMS_);
00098     Scheduler::GetCurrentThread()->SetExpired(false);
00099 #endif
00100     // Drain the FIFO of all queued events and trigger a context switch if necessary
00101     if (ProcessQueue())
00102     {
00103         Thread::Yield();
00104     }
00105
00106     // Re-enable the scheduler
00107     Scheduler::SetScheduler(bSchedState);
00108
00109 #if KERNEL_USE_TIMERS
00110     if (ulTimeMS_)
00111     {
00112         Scheduler::GetCurrentThread()->GetTimer()->
00113         Stop();
00114     }
00115 #endif
00116     return Scheduler::GetCurrentThread()->
00117     GetEventFlagMask();
00118 }
00119
00120 K_BOOL EventFlag::ProcessQueue()
00121 {
00122     Transaction *pclTRX;
00123     K_BOOL bReschedule = false;
00124
00125     do
00126     {
00127         pclTRX = m_clKTQ.Dequeue();
00128         KERNEL_ASSERT(pclTRX);
00129
00130         switch (pclTRX->GetCode())
00131         {
00132             case EVENT_TRANSACTION_WAIT:
00133                 WaitTransaction(pclTRX, &bReschedule);
00134                 break;
00135             case EVENT_TRANSACTION_SET:
00136                 SetTransaction(pclTRX, &bReschedule);
00137                 break;
00138             case EVENT_TRANSACTION_CLEAR:
00139                 ClearTransaction(pclTRX, &bReschedule);
00140                 break;
00141             case EVENT_TRANSACTION_TIMEOUT:
00142                 TimeoutTransaction(pclTRX, &bReschedule);
00143                 break;
00144             default:
00145                 break;
00146         }
00147         m_clKTQ.Finish(pclTRX);
00148     } while (UnLock() > 1);
00149
00150     return bReschedule;
00151 }
00152
00153 void EventFlag::WaitTransaction( Transaction *pclTRX_, K_BOOL *
00154 pbReschedule_ )
00155 {
00156     bool bMatch = false;
00157     Thread *pclThread = Scheduler::GetCurrentThread();
00158     K_USHORT usMask = (K_USHORT)((K_ADDR)pclTRX_->GetData());
00159

```

```

00166 #if KERNEL_USE_TIMERS
00167     Timer *pclTimer = pclThread->GetTimer();
00168     pclThread->SetExpired(false);
00169 #endif
00170
00171     // Check to see whether or not the current mask matches any of the
00172     // desired bits.
00173
00174     EventFlagOperation_t eMode = pclThread->GetEventFlagMode();
00175     if ((eMode == EVENT_FLAG_ALL) || (eMode == EVENT_FLAG_ALL_CLEAR))
00176     {
00177         // Check to see if the flags in their current state match all of
00178         // the set flags in the event flag group, with this mask.
00179         if ((m_usSetMask & usMask) == usMask)
00180         {
00181             bMatch = true;
00182             pclThread->SetEventFlagMask(usMask);
00183         }
00184     }
00185     else if ((eMode == EVENT_FLAG_ANY) || (eMode == EVENT_FLAG_ANY_CLEAR))
00186     {
00187         // Check to see if the existing flags match any of the set flags in
00188         // the event flag group with this mask
00189         if (m_usSetMask & usMask)
00190         {
00191             bMatch = true;
00192             pclThread->SetEventFlagMask(m_usSetMask & usMask);
00193         }
00194     }
00195
00196     // We're unable to match this pattern as-is, so we must block.
00197     if (!bMatch)
00198     {
00199         // Reset the current thread's event flag mask & mode
00200         pclThread->SetEventFlagMask(usMask);
00201         pclThread->SetEventFlagMode(eMode);
00202     }
00203 #if KERNEL_USE_TIMERS
00204     K_ULONG ulTimeMS = pclTimer->GetInterval();
00205     if (ulTimeMS)
00206     {
00207         pclTimer->Start(0, ulTimeMS, TimedEventFlag_Callback, (void*)this);
00208     }
00209 #endif
00210
00211     // Add the thread to the object's block-list.
00212     Block(pclThread);
00213
00214     *pbReschedule_ = true;
00215 }
00216 }
00217
00218 //-----
00219 void EventFlag::SetTransaction( Transaction *pclTRX_, K_BOOL *
pbReschedule_ )
00220 {
00221     Thread *pclPrev;
00222     Thread *pclCurrent;
00223
00224     K_USHORT usNewMask;
00225     K_USHORT usMask = (K_USHORT)((K_ADDR)pclTRX_->GetData());
00226     // Walk through the whole block list, checking to see whether or not
00227     // the current flag set now matches any/all of the masks and modes of
00228     // the threads involved.
00229
00230     m_usSetMask |= usMask;
00231     usNewMask = m_usSetMask;
00232
00233     // Start at the head of the list, and iterate through until we hit the
00234     // "head" element in the list again. Ensure that we handle the case where
00235     // we remove the first or last elements in the list, or if there's only
00236     // one element in the list.
00237     pclCurrent = static_cast<Thread*>(m_clBlockList.GetHead());
00238
00239     // Do nothing when there are no objects blocking.
00240     if (pclCurrent)
00241     {
00242         // First loop - process every thread in the block-list and check to
00243         // see whether or not the current flags match the event-flag conditions
00244         // on the thread.
00245         do
00246         {
00247             pclPrev = pclCurrent;
00248             pclCurrent = static_cast<Thread*>(pclCurrent->GetNext());
00249
00250             // Read the thread's event mask/mode
00251             K_USHORT usThreadMask = pclPrev->GetEventFlagMask();

```

```

00252         EventFlagOperation_t eThreadMode = pclPrev->GetEventFlagMode();
00253
00254         // For the "any" mode - unblock the blocked threads if one or more bits
00255         // in the thread's bitmask match the object's bitmask
00256         if ((EVENT_FLAG_ANY == eThreadMode) || (EVENT_FLAG_ANY_CLEAR == eThreadMode))
00257         {
00258             if (usThreadMask & m_usSetMask)
00259             {
00260                 pclPrev->SetEventFlagMode(EVENT_FLAG_PENDING_UNBLOCK);
00261                 pclPrev->SetEventFlagMask(m_usSetMask & usThreadMask);
00262                 *pbReschedule_ = true;
00263
00264                 // If the "clear" variant is set, then clear the bits in the mask
00265                 // that caused the thread to unblock.
00266                 if (EVENT_FLAG_ANY_CLEAR == eThreadMode)
00267                 {
00268                     usNewMask &=~ (usThreadMask & usMask);
00269                 }
00270             }
00271         }
00272         // For the "all" mode, every set bit in the thread's requested bitmask must
00273         // match the object's flag mask.
00274         else if ((EVENT_FLAG_ALL == eThreadMode) || (EVENT_FLAG_ALL_CLEAR == eThreadMode))
00275         {
00276             if ((usThreadMask & m_usSetMask) == usThreadMask)
00277             {
00278                 pclPrev->SetEventFlagMode(EVENT_FLAG_PENDING_UNBLOCK);
00279                 pclPrev->SetEventFlagMask(usThreadMask);
00280                 *pbReschedule_ = true;
00281
00282                 // If the "clear" variant is set, then clear the bits in the mask
00283                 // that caused the thread to unblock.
00284                 if (EVENT_FLAG_ALL_CLEAR == eThreadMode)
00285                 {
00286                     usNewMask &=~ (usThreadMask & usMask);
00287                 }
00288             }
00289         }
00290     }
00291     // To keep looping, ensure that there's something in the list, and
00292     // that the next item isn't the head of the list.
00293     while (pclPrev != m_clBlockList.GetTail());
00294
00295     // Second loop - go through and unblock all of the threads that
00296     // were tagged for unblocking.
00297     pclCurrent = static_cast<Thread*>(m_clBlockList.
GetHead());
00298     bool bIsTail = false;
00299     do
00300     {
00301         pclPrev = pclCurrent;
00302         pclCurrent = static_cast<Thread*>(pclCurrent->GetNext());
00303
00304         // Check to see if this is the condition to terminate the loop
00305         if (pclPrev == m_clBlockList.GetTail())
00306         {
00307             bIsTail = true;
00308         }
00309
00310         // If the first pass indicated that this thread should be
00311         // unblocked, then unblock the thread
00312         if (pclPrev->GetEventFlagMode() == EVENT_FLAG_PENDING_UNBLOCK)
00313         {
00314             Unblock(pclPrev);
00315         }
00316     }
00317     while (!bIsTail);
00318 }
00319
00320 // Update the bitmask based on any "clear" operations performed along
00321 // the way
00322 m_usSetMask = usNewMask;
00323 }
00324
00325 //-----
00326 void EventFlag::ClearTransaction( Transaction *pclTRX_, K_BOOL *
pbReschedule_ )
00327 {
00328     m_usSetMask &= ~( (K_USHORT) ((K_ADDR) pclTRX_->GetData()) );
00329 }
00330
00331 #if KERNEL_USE_TIMERS
00332 //-----
00333 void EventFlag::TimeoutTransaction( Transaction *pclTRX_, K_BOOL *
pbReschedule_ )
00334 {
00335     Thread *pclChosenOne = static_cast<Thread*>(pclTRX_->GetData());

```

```

00336
00337     Unblock(pclChosenOne);
00338
00339     pclChosenOne->SetExpired(true);
00340     pclChosenOne->SetEventFlagMask(0);
00341
00342     if (pclChosenOne->GetPriority() > Scheduler::GetCurrentThread()->
        GetPriority())
00343     {
00344         *pbReschedule_ = true;
00345     }
00346 }
00347 #endif
00348
00349 //-----
00350 void EventFlag::Set(K_USHORT usMask_)
00351 {
00352     // This function follows the signature of Wait() and Timeout()
00353     K_BOOL bSchedState;
00354     if (LockAndQueue( EVENT_TRANSACTION_SET, (void*)((K_ADDR)usMask_), &bSchedState))
00355     {
00356         return;
00357     }
00358
00359     if (ProcessQueue())
00360     {
00361         Thread::Yield();
00362     }
00363
00364     Scheduler::SetScheduler(bSchedState);
00365 }
00366
00367 //-----
00368 void EventFlag::Clear(K_USHORT usMask_)
00369 {
00370     // This function follows the signature of Wait() and Timeout()
00371     K_BOOL bSchedState;
00372     if (LockAndQueue( EVENT_TRANSACTION_CLEAR, (void*)((K_ADDR)usMask_), &bSchedState))
00373     {
00374         return;
00375     }
00376
00377     if (ProcessQueue())
00378     {
00379         Thread::Yield();
00380     }
00381
00382     Scheduler::SetScheduler(bSchedState);
00383 }
00384
00385 //-----
00386 K_USHORT EventFlag::GetMask()
00387 {
00388     // Return the presently held event flag values in this object. Ensure
00389     // we get this within a critical section to guarantee atomicity.
00390     K_USHORT usReturn;
00391     CS_ENTER();
00392     usReturn = m_usSetMask;
00393     CS_EXIT();
00394     return usReturn;
00395 }
00396
00397 #endif // KERNEL_USE_EVENTFLAG

```

17.59 /home/mo/mark3-source/embedded/stage/src/eventflag.h File Reference

Event Flag Blocking Object/IPC-Object definition.

```

#include "mark3cfg.h"
#include "kernel.h"
#include "kerneltypes.h"
#include "blocking.h"
#include "thread.h"
#include "transaction.h"

```

Classes

- class [EventFlag](#)

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

17.59.1 Detailed Description

Event Flag Blocking Object/IPC-Object definition.

Definition in file [eventflag.h](#).

17.60 eventflag.h

```

00001  /*=====
00002
00003
00004
00005
00006
00007
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00019  #ifndef __EVENTFLAG_H__
00020  #define __EVENTFLAG_H__
00021
00022  #include "mark3cfg.h"
00023  #include "kernel.h"
00024  #include "kerneltypes.h"
00025  #include "blocking.h"
00026  #include "thread.h"
00027  #include "transaction.h"
00028
00029  #if KERNEL_USE_EVENTFLAG
00030
00031  //-----
00047  class EventFlag : public BlockingObject
00048  {
00049  public:
00053      void Init() { m_usSetMask = 0; m_clBlockList.
          Init(); }
00054
00062      K_USHORT Wait(K_USHORT usMask_, EventFlagOperation_t eMode_);
00063
00064  #if KERNEL_USE_TIMERS
00065
00073      K_USHORT Wait(K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_);
00074
00075      void Timeout(Thread *pclOwner_);
00076
00077  #endif
00078
00084      void Set(K_USHORT usMask_);
00085
00090      void Clear(K_USHORT usMask_);
00091
00096      K_USHORT GetMask();
00097
00098  private:
00099
00111      K_BOOL ProcessQueue();
00112
00123      void WaitTransaction( Transaction *pclTRX_, K_BOOL *pbReschedule_ );
00124
00135      void SetTransaction( Transaction *pclTRX_, K_BOOL *pbReschedule_ );
00136
00147      void ClearTransaction( Transaction *pclTRX_, K_BOOL *pbReschedule_ );
00148
00149  #if KERNEL_USE_TIMERS
00150
00160      void TimeoutTransaction( Transaction *pclTRX_, K_BOOL *pbReschedule_ );
00161  #endif
00162

```



```

00055
00056 // Create a heap (linked-list nodes + byte pool) in the middle of
00057 // the data blob
00058 for (K_USHORT i = 0; i < usNodeCount; i++)
00059 {
00060     // Create a pointer back to the source list.
00061     BlockHeap **pclTemp = (BlockHeap**) (adNode + sizeof(
LinkListNode));
00062     *pclTemp = (BlockHeap*) (this);
00063
00064     // Add the node to the block list
00065     m_clList.Add( (LinkListNode*) adNode );
00066
00067     // Move the pointer in the pool to point to the next block to allocate
00068     adNode += (usBlockSize_ + sizeof(LinkListNode) + sizeof(
BlockHeap*));
00069
00070     // Bail if we would be going past the end of the allocated space...
00071     if ((K_ULONG) adNode >= (K_ULONG) adMaxNode)
00072     {
00073         break;
00074     }
00075 }
00076 m_usBlocksFree = usNodeCount;
00077
00078 // Return pointer to end of heap (used for heap-chaining)
00079 return (void*) adNode;
00080 }
00081
00082 //-----
00083 void *BlockHeap::Alloc()
00084 {
00085     LinkListNode *pclNode = m_clList.GetHead();
00086
00087     // Return the first node from the head of the list
00088     if (pclNode)
00089     {
00090         m_clList.Remove( pclNode );
00091         m_usBlocksFree--;
00092
00093         // Account for block-management metadata
00094         return (void*) ((K_ADDR) pclNode + sizeof(LinkListNode) + sizeof(void*));
00095     }
00096
00097     // Or null, if the heap is empty.
00098     return 0;
00099 }
00100
00101 //-----
00102 void BlockHeap::Free( void* pvData_ )
00103 {
00104     // Compute the address of the original object (class metadata included)
00105     LinkListNode *pclNode = (LinkListNode*) ((K_ADDR) pvData_ - sizeof(
LinkListNode) - sizeof(void*));
00106
00107     // Add the object back to the block data pool
00108     m_clList.Add(pclNode);
00109     m_usBlocksFree++;
00110 }
00111
00112 //-----
00113 void FixedHeap::Create( void *pvHeap_, HeapConfig *pclHeapConfig_ )
00114 {
00115     K_USHORT i = 0;
00116     void *pvTemp = pvHeap_;
00117     while( pclHeapConfig_[i].m_usBlockSize != 0)
00118     {
00119         pvTemp = pclHeapConfig_[i].m_clHeap.Create
00120             (pvTemp,
00121              (pclHeapConfig_[i].m_usBlockSize + sizeof(LinkListNode) + sizeof(void*)) *
00122               pclHeapConfig_[i].m_usBlockCount,
00123              pclHeapConfig_[i].m_usBlockSize );
00124         i++;
00125     }
00126     m_paclHeaps = pclHeapConfig_;
00127 }
00128
00129 //-----
00130 void *FixedHeap::Alloc( K_USHORT usSize_ )
00131 {
00132     void *pvRet = 0;
00133     K_USHORT i = 0;
00134
00135     // Go through all heaps, trying to find the smallest one that
00136     // has a free item to satisfy the allocation
00137     while (m_paclHeaps[i].m_usBlockSize != 0)
00138     {

```



```

00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __FIXED_HEAP_H__
00020 #define __FIXED_HEAP_H__
00021
00022 #include "kerneltypes.h"
00023 #include "ll.h"
00024
00025 //-----
00029 class BlockHeap
00030 {
00031 public:
00046     void *Create( void *pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_ );
00047
00055     void *Alloc();
00056
00065     void Free( void* pvData_ );
00066
00074     K_BOOL IsFree() { return m_usBlocksFree != 0; }
00075
00076 protected:
00077     K_USHORT m_usBlocksFree;
00078
00079 private:
00080     DoubleLinkedList m_clList;
00081 };
00082
00083
00084 class FixedHeap;
00085
00086 //-----
00090 class HeapConfig
00091 {
00092 public:
00093     K_USHORT m_usBlockSize;
00094     K_USHORT m_usBlockCount;
00095     friend class FixedHeap;
00096 protected:
00097     BlockHeap m_clHeap;
00098 };
00099
00100 //-----
00104 class FixedHeap
00105 {
00106 public:
00122     void Create( void *pvHeap_, HeapConfig *pclHeapConfig_ );
00123
00135     void *Alloc( K_USHORT usSize_ );
00136
00148     static void Free( void *pvNode_ );
00149
00150 private:
00151     HeapConfig *m_paclHeaps;
00152 };
00153
00154 #endif
00155

```

17.65 /home/mo/mark3-source/embedded/stage/src/font.h File Reference

Font structure definitions.

```

#include "kerneltypes.h"
#include "fontport.h"

```

Classes

- struct [Glyph_t](#)
- struct [Font_t](#)

Macros

- `#define GLYPH_SIZE(x) (((K_USHORT)((x->ucWidth + 7) >> 3) * (K_USHORT)(x->ucHeight)) + sizeof(Glyph_t) - 1)`

*The size of the glyph is the width*height (in bytes), plus the overhead of the struct parameters.*

17.65.1 Detailed Description

Font structure definitions.

Definition in file [font.h](#).

17.66 font.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __FONT_H__
00020 #define __FONT_H__
00021
00022 #include "kerneltypes.h"
00023 #include "fontport.h"
00024
00025 //-----
00026 typedef struct
00027 {
00028     K_UCHAR ucWidth;
00029     K_UCHAR ucHeight;
00030     K_UCHAR ucVOffset;
00031     K_UCHAR aucData[1];
00032 } Glyph_t;
00033
00034 //-----
00039 #define GLYPH_SIZE(x) \
00040     (((K_USHORT)((x->ucWidth + 7) >> 3) * (K_USHORT)(x->ucHeight)) + sizeof(Glyph_t) - 1)
00041
00042 //-----
00043 typedef struct
00044 {
00045     K_UCHAR ucSize;
00046     K_UCHAR ucFlags;
00047     K_UCHAR ucStartChar;
00048     K_UCHAR ucMaxChar;
00049     const K_CHAR *szName;
00050     const FONT_STORAGE_TYPE *pucFontData;
00051 } Font_t;
00052
00053 #endif
00054
```

17.67 /home/mo/mark3-source/embedded/stage/src/graphics.cpp File Reference

Generic graphics driver implementation.

```
#include "kerneltypes.h"
#include "graphics.h"
#include "draw.h"
#include "driver.h"
#include "colorspace.h"
#include "font.h"
#include <stdio.h>
```

17.67.1 Detailed Description

Generic graphics driver implementation.

Definition in file [graphics.cpp](#).

17.68 graphics.cpp

```
00001 /*=====
00002
00003
00004 | | | | | | | | | | | | | | | | | | | | | |
00005 | | | | | | | | | | | | | | | | | | | | | |
00006 | | | | | | | | | | | | | | | | | | | | | |
00007 | | | | | | | | | | | | | | | | | | | | | |
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "kerneltypes.h"
00020 #include "graphics.h"
00021 #include "draw.h"
00022 #include "driver.h"
00023 #include "colorspace.h"
00024 #include "font.h"
00025
00026 #include <stdio.h>
00027
00028 //-----
00029 void GraphicsDriver::ClearScreen()
00030 {
00031     DrawPoint_t stPoint;
00032     stPoint.uColor = COLOR_BLACK;
00033
00034     for (stPoint.usX = 0; stPoint.usX < m_usResX; stPoint.usX++)
00035     {
00036         for (stPoint.usY = 0; stPoint.usY < m_usResY; stPoint.usY++)
00037         {
00038             // Pixel Write
00039             DrawPixel(&stPoint);
00040         }
00041     }
00042 }
00043 //-----
00044 void GraphicsDriver::Point(DrawPoint_t *pstPoint_)
00045 {
00046     DrawPixel(pstPoint_);
00047 }
00048
00049 //-----
00050 void GraphicsDriver::Line(DrawLine_t *pstLine_)
00051 {
00052     // Bresenham Line drawing algorithm, adapted from:
00053     // www.cs.unc.edu/~mcmillan/compl36/Lecture6/Lines.html
00054
00055     DrawPoint_t stPoint;
00056     K_SHORT sX1 = (K_SHORT)pstLine_>usX1;
00057     K_SHORT sX2 = (K_SHORT)pstLine_>usX2;
00058     K_SHORT sY1 = (K_SHORT)pstLine_>usY1;
00059     K_SHORT sY2 = (K_SHORT)pstLine_>usY2;
00060     K_SHORT sDeltaY = sY2 - sY1;
00061     K_SHORT sDeltaX = sX2 - sX1;
00062     K_CHAR cStepx, cStepy;
00063     stPoint.uColor = pstLine_>uColor;
00064 }
```

```

00065     if (sDeltaY < 0)
00066     {
00067         sDeltaY = -sDeltaY;
00068         cStepy = -1;
00069     }
00070     else
00071     {
00072         cStepy = 1;
00073     }
00074
00075     if (sDeltaX < 0)
00076     {
00077         sDeltaX = -sDeltaX;
00078         cStepx = -1;
00079     }
00080     else
00081     {
00082         cStepx = 1;
00083     }
00084
00085     // Scale by a factor of 2 in each direction
00086     sDeltaY <<= 1;
00087     sDeltaX <<= 1;
00088
00089     stPoint.usX = sX1;
00090     stPoint.usY = sY1;
00091     DrawPixel(&stPoint);
00092
00093     if (sDeltaX > sDeltaY)
00094     {
00095         K_SHORT sFraction = sDeltaY - (sDeltaX >> 1);
00096
00097         while (sX1 != sX2)
00098         {
00099             if (sFraction >= 0)
00100             {
00101                 sY1 += cStepy;
00102                 sFraction -= sDeltaX;
00103             }
00104             sX1 += cStepx;
00105             sFraction += sDeltaY;
00106
00107             stPoint.usX = sX1;
00108             stPoint.usY = sY1;
00109             DrawPixel(&stPoint);
00110         }
00111     }
00112     else
00113     {
00114         K_SHORT sFraction = sDeltaX - (sDeltaY >> 1);
00115         while (sY1 != sY2)
00116         {
00117             if (sFraction >= 0)
00118             {
00119                 sX1 += cStepx;
00120                 sFraction -= sDeltaY;
00121             }
00122             sY1 += cStepy;
00123             sFraction += sDeltaX;
00124
00125             stPoint.usX = sX1;
00126             stPoint.usY = sY1;
00127             DrawPixel(&stPoint);
00128         }
00129     }
00130 }
00131
00132 //-----
00133 void GraphicsDriver::Rectangle(DrawRectangle_t *pstRectangle_)
00134 {
00135     DrawPoint_t stPoint;
00136
00137     // if drawing a background fill color (optional)
00138     if (pstRectangle_>bFill == true)
00139     {
00140         stPoint.uColor = pstRectangle_>uFillColor;
00141         for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00142         {
00143             for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00144             {
00145                 DrawPixel(&stPoint);
00146             }
00147         }
00148     }
00149 }

```

```

00150     // Draw four orthogonal lines...
00151     stPoint.uColor = pstRectangle_>uLineColor;
00152     stPoint.usY = pstRectangle_>usTop;
00153     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00154     {
00155         DrawPixel(&stPoint);
00156     }
00157
00158     stPoint.usY = pstRectangle_>usBottom;
00159     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00160     {
00161         DrawPixel(&stPoint);
00162     }
00163
00164     stPoint.usX = pstRectangle_>usLeft;
00165     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00166     {
00167         DrawPixel(&stPoint);
00168     }
00169
00170     stPoint.usX = pstRectangle_>usRight;
00171     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00172     {
00173         DrawPixel(&stPoint);
00174     }
00175 }
00176
00177 //-----
00178 void GraphicsDriver::Circle(DrawCircle_t *pstCircle_)
00179 {
00180     DrawPoint_t stPoint;
00181     K_SHORT sX;
00182     K_SHORT sY;
00183     K_ULONG ulRadSquare;
00184
00185     K_ULONG ulXSquare;
00186     K_ULONG ulYSquare;
00187
00188     // Get the radius squared value...
00189     ulRadSquare = (K_ULONG)pstCircle_>usRadius;
00190     ulRadSquare *= ulRadSquare;
00191
00192     // Look at the upper-right quarter of the circle
00193     for (sX = 0; sX <= (K_SHORT)pstCircle_>usRadius; sX++)
00194     {
00195         ulXSquare = (K_ULONG)sX;
00196         ulXSquare *= ulXSquare;
00197         for (sY = 0; sY <= (K_SHORT)pstCircle_>usRadius; sY++)
00198         {
00199             ulYSquare = (K_ULONG)sY;
00200             ulYSquare *= ulYSquare;
00201
00202             // if filled...
00203             if (pstCircle_>bFill == true)
00204             {
00205                 stPoint.uColor = pstCircle_>uFillColor;
00206                 if (ulXSquare + ulYSquare <= ulRadSquare)
00207                 {
00208                     // Draw the fill color at the appropriate locations (quadrature...)
00209                     stPoint.usX = pstCircle_>usX + sX;
00210                     stPoint.usY = pstCircle_>usY + sY;
00211                     DrawPixel(&stPoint);
00212                     stPoint.usX = pstCircle_>usX - sX;
00213                     stPoint.usY = pstCircle_>usY + sY;
00214                     DrawPixel(&stPoint);
00215                     stPoint.usX = pstCircle_>usX + sX;
00216                     stPoint.usY = pstCircle_>usY - sY;
00217                     DrawPixel(&stPoint);
00218                     stPoint.usX = pstCircle_>usX - sX;
00219                     stPoint.usY = pstCircle_>usY - sY;
00220                     DrawPixel(&stPoint);
00221                 }
00222             }
00223             // Check for edge...
00224             if (
00225                 ((ulXSquare + ulYSquare) >= (ulRadSquare-pstCircle_>usRadius)) &&
00226                 ((ulXSquare + ulYSquare) <= (ulRadSquare+pstCircle_>usRadius))
00227             )
00228             {
00229                 stPoint.uColor = pstCircle_>uLineColor;
00230
00231                 // Draw the fill color at the appropriate locations (quadrature...)
00232                 stPoint.usX = pstCircle_>usX + sX;

```

```

00233         stPoint.usY = pstCircle->usY + sY;
00234         DrawPixel(&stPoint);
00235         stPoint.usX = pstCircle->usX - sX;
00236         stPoint.usY = pstCircle->usY + sY;
00237         DrawPixel(&stPoint);
00238         stPoint.usX = pstCircle->usX + sX;
00239         stPoint.usY = pstCircle->usY - sY;
00240         DrawPixel(&stPoint);
00241         stPoint.usX = pstCircle->usX - sX;
00242         stPoint.usY = pstCircle->usY - sY;
00243         DrawPixel(&stPoint);
00244     }
00245 }
00246 }
00247 }
00248
00249 //-----
00250 void GraphicsDriver::Ellipse(DrawEllipse_t *pstEllipse_)
00251 {
00252     DrawPoint_t stPoint;
00253     K_SHORT sX;
00254     K_SHORT sY;
00255     K_ULONG ulRadius;
00256     K_ULONG ulHSquare;
00257     K_ULONG ulVSquare;
00258     K_ULONG ulXSquare;
00259     K_ULONG ulYSquare;
00260
00261     ulHSquare = (K_ULONG)pstEllipse->usWidth;
00262     ulHSquare *= ulHSquare;
00263
00264     ulVSquare = (K_ULONG)pstEllipse->usHeight;
00265     ulVSquare *= ulVSquare;
00266
00267     ulRadius = ulHSquare * ulVSquare;
00268
00269     for (sX = 0; sX <= (K_SHORT)pstEllipse->usWidth; sX++)
00270     {
00271         ulXSquare = (K_ULONG)sX;
00272         ulXSquare *= ulXSquare;
00273         ulXSquare *= ulHSquare;
00274
00275         for (sY = 0; sY <= (K_SHORT)pstEllipse->usHeight; sY++)
00276         {
00277             ulYSquare = (K_ULONG)sY;
00278             ulYSquare *= ulYSquare;
00279             ulYSquare *= ulVSquare;
00280
00281             if ((ulXSquare + ulYSquare) <= ulRadius)
00282             {
00283                 // Draw the fill color at the appropriate locations (quadrature...)
00284                 stPoint.usX = pstEllipse->usX + sX;
00285                 stPoint.usY = pstEllipse->usY + sY;
00286                 DrawPixel(&stPoint);
00287                 stPoint.usX = pstEllipse->usX - sX;
00288                 stPoint.usY = pstEllipse->usY + sY;
00289                 DrawPixel(&stPoint);
00290                 stPoint.usX = pstEllipse->usX + sX;
00291                 stPoint.usY = pstEllipse->usY - sY;
00292                 DrawPixel(&stPoint);
00293                 stPoint.usX = pstEllipse->usX - sX;
00294                 stPoint.usY = pstEllipse->usY - sY;
00295                 DrawPixel(&stPoint);
00296             }
00297         }
00298     }
00299 }
00300
00301 //-----
00302 void GraphicsDriver::Bitmap(DrawBitmap_t *pstBitmap_)
00303 {
00304     K_USHORT usRow;
00305     K_USHORT usCol;
00306
00307     K_USHORT usIndex;
00308
00309     K_UCHAR ucRed = 0;
00310     K_UCHAR ucBlue = 0;
00311     K_UCHAR ucGreen = 0;
00312
00313     DrawPoint_t stPoint;
00314
00315     usIndex = 0;
00316     for (usRow = pstBitmap->usY; usRow < (pstBitmap->usY + pstBitmap->
00317         usHeight); usRow++)
00318     {
00319         for (usCol = pstBitmap->usX; usCol < (pstBitmap->usX + pstBitmap->

```

```

        usWidth); usCol++;
    {
00319
00320
00321         stPoint.usX = usCol;
00322         stPoint.usY = usRow;
00323
00324         // Build the color based on the bitmap value... This algorithm
00325         // is slow, but it automatically converts any 8/16/24 bit bitmap into the
00326         // current colorspace defined...
00327         switch(pstBitmap->ucBPP)
00328         {
00329             case 1:
00330             {
00331                 // 3:2:3, RGB
00332                 ucRed   = ((pstBitmap->pucData[usIndex]) & 0xE0) << 1;
00333                 ucGreen = ((pstBitmap->pucData[usIndex]) & 0x18) << 3;
00334                 ucBlue  = ((pstBitmap->pucData[usIndex]) & 0x07) << 5;
00335             }
00336             break;
00337             case 2:
00338             {
00339                 K_USHORT usTemp;
00340                 usTemp = pstBitmap->pucData[usIndex];
00341                 usTemp <= 8;
00342                 usTemp |= pstBitmap->pucData[usIndex + 1];
00343
00344                 // 5:6:5, RGB
00345                 ucRed   = (K_UCHAR)((usTemp >> 11) & 0x001F) << 3;
00346                 ucGreen = (K_UCHAR)((usTemp >> 5) & 0x003F) << 2;
00347                 ucBlue  = (K_UCHAR)(usTemp & 0x001F) << 3;
00348             }
00349             break;
00350             case 3:
00351             {
00352                 K_ULONG ulTemp;
00353                 ulTemp = pstBitmap->pucData[usIndex];
00354                 ulTemp <= 8;
00355                 ulTemp |= pstBitmap->pucData[usIndex + 1];
00356                 ulTemp <= 8;
00357                 ulTemp |= pstBitmap->pucData[usIndex + 2];
00358
00359                 // 8:8:8 RGB
00360                 ucRed   = (K_UCHAR)((ulTemp & 0x00FF0000) >> 16);
00361                 ucGreen = (K_UCHAR)((ulTemp & 0x0000FF00) >> 8);
00362                 ucBlue  = (K_UCHAR)((ulTemp & 0x000000FF));
00363             }
00364             break;
00365             default:
00366                 break;
00367         }
00368
00369         // Convert the R,G,B values into the correct colorspace for display
00370         #if DRAW_COLOR_2BIT
00371             //1-bit
00372             ucRed >>= 7;
00373             ucGreen >>= 7;
00374             ucBlue >>= 7;
00375         #elif DRAW_COLOR_8BIT
00376             //3:2:3 R:G:B
00377             ucRed >>= 5;
00378             ucGreen >>= 6;
00379             ucBlue >>= 5;
00380         #elif DRAW_COLOR_16BIT
00381             //5:6:5 R:G:B
00382             ucRed >>= 3;
00383             ucGreen >>= 2;
00384             ucBlue >>= 3;
00385         #elif DRAW_COLOR_24BIT
00386             // No conversion required
00387         #endif
00388         // Build the color.
00389         stPoint.uColor = RGB_COLOR(ucRed,ucGreen,ucBlue);
00390
00391         // Draw the point.
00392         DrawPixel(&stPoint);
00393
00394         // Stamps are opaque, don't fill in the BG
00395         usIndex += m_ucBPP / 8;
00396     }
00397 }
00398 }
00399
00400 //-----
00401 void GraphicsDriver::Stamp(DrawStamp_t *pstStamp_)
00402 {
00403     K_USHORT usRow;
00404     K_USHORT usCol;

```

```

00405     K_USHORT usShift;
00406     K_USHORT usIndex;
00407     DrawPoint_t stPoint;
00408
00409     usIndex = 0;
00410     for (usRow = pstStamp_>usY; usRow < (pstStamp_>usY + pstStamp_>
usHeight); usRow++)
00411     {
00412         usShift = 0x80;
00413         for (usCol = pstStamp_>usX; usCol < (pstStamp_>usX + pstStamp_>
usWidth); usCol++)
00414         {
00415             // If the packed bit in the bitmap is a "1", draw the color.
00416             if (pstStamp_>pucData[usIndex] & usShift)
00417             {
00418                 stPoint.usX = usCol;
00419                 stPoint.usY = usRow;
00420                 stPoint.uColor = pstStamp_>uColor;
00421                 DrawPixel(&stPoint);
00422             }
00423             // Stamps are opaque, don't fill in the BG
00424
00425             // Shift to the next bit in the field
00426             usShift >>= 1;
00427
00428             // Rollover - next bit in the bitmap.
00429             // This obviously works best for stamps that are multiples of 8x8
00430             if (usShift == 0)
00431             {
00432                 usShift = 0x80;
00433                 usIndex++;
00434             }
00435         }
00436     }
00437 }
00438
00439 //-----
00440 void GraphicsDriver::Move( DrawMove_t *pstMove_ )
00441 {
00442     DrawPoint_t stPoint;
00443     K_LONG sX;
00444     K_LONG sY;
00445     K_LONG sXInc = 0;
00446     K_LONG sYInc = 0;
00447
00448     K_BOOL bLeftToRight = false;
00449     K_BOOL bTopToBottom = false;
00450
00451     if (pstMove_>usSrcX > pstMove_>usDstX)
00452     {
00453         bLeftToRight = true;
00454     }
00455     if (pstMove_>usSrcY > pstMove_>usDstY)
00456     {
00457         bTopToBottom = true;
00458     }
00459
00460     if (bLeftToRight)
00461     {
00462         sXInc++;
00463     }
00464     else
00465     {
00466         sXInc--;
00467         pstMove_>usSrcX += pstMove_>usCopyWidth - 1;
00468         pstMove_>usDstX += pstMove_>usCopyWidth - 1;
00469     }
00470
00471     if (bTopToBottom)
00472     {
00473         sYInc++;
00474     }
00475     else
00476     {
00477         sYInc--;
00478         pstMove_>usSrcY += pstMove_>usCopyHeight - 1;
00479         pstMove_>usDstY += pstMove_>usCopyHeight - 1;
00480     }
00481
00482     // Hideously inefficient memory move...
00483     for (sX = 0; sX < pstMove_>usCopyWidth; sX++)
00484     {
00485         for (sY = 0; sY < pstMove_>usCopyHeight; sY++)
00486         {
00487             // Read from source (value read into the point struct)
00488             stPoint.usY = (K_USHORT) ((K_LONG)pstMove_>usSrcY + ((K_LONG)sY * sYInc));
00489             stPoint.usX = (K_USHORT) ((K_LONG)pstMove_>usSrcX + ((K_LONG)sX * sXInc));

```



```

00490         ReadPixel(&stPoint);
00491
00492         // Copy to dest
00493         stPoint.usY = (K_USHORT)((K_LONG)pstMove_>usDstY + ((K_LONG)sY * sYInc));
00494         stPoint.usX = (K_USHORT)((K_LONG)pstMove_>usDstX + ((K_LONG)sX * sXInc));
00495         DrawPixel(&stPoint);
00496     }
00497 }
00498 }
00499
00500 //-----
00501 void GraphicsDriver::Text(DrawText_t *pstText_)
00502 {
00503     K_USHORT usX, usY;
00504     K_USHORT usStartX;
00505     K_USHORT usStartY;
00506     K_USHORT usCharOffsetX;
00507     K_USHORT usCharIndex = 0;
00508     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00509     DrawPoint_t stPoint;
00510
00511     // set the color for this element.
00512     stPoint.uColor = pstText_>uColor;
00513
00514     usCharOffsetX = 0;
00515
00516     // Draw every character in the string, one at a time
00517     while (pstText_>pcString[usCharIndex] != 0)
00518     {
00519         K_USHORT usOffset = 0;
00520
00521         K_UCHAR ucWidth;
00522         K_UCHAR ucHeight;
00523         K_UCHAR ucVOffset;
00524         K_UCHAR ucBitmask;
00525
00526         // Read the glyphs from memory until we arrive at the one we wish to print
00527         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00528         {
00529             // Glyphs are variable-sized for efficiency - to look up a particular
00530             // glyph, we must traverse all preceding glyphs in the list
00531             ucWidth = Font_ReadByte(usOffset, pucData);
00532             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00533
00534             // Adjust the offset to point to the next glyph
00535             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00536                 + (sizeof(Glyph_t) - 1);
00537         }
00538
00539         // Header information: glyph size and vertical offset
00540         ucWidth = Font_ReadByte(usOffset++, pucData);
00541         ucHeight = Font_ReadByte(usOffset++, pucData);
00542         ucVOffset = Font_ReadByte(usOffset++, pucData);
00543
00544         usStartY = pstText_>usTop + (K_USHORT)ucVOffset;
00545         usStartX = pstText_>usLeft;
00546
00547         // Draw the font from left->right, top->bottom
00548         for (usY = usStartY;
00549              usY < usStartY + (K_USHORT)ucHeight;
00550              usY++)
00551         {
00552             K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
00553             ucBitmask = 0x80;
00554
00555             for (usX = usCharOffsetX + usStartX;
00556                  usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
00557                  usX++)
00558             {
00559                 if (!ucBitmask)
00560                 {
00561                     ucBitmask = 0x80;
00562                     usOffset++;
00563                     ucTempChar = Font_ReadByte(usOffset, pucData);
00564                 }
00565
00566                 if (ucTempChar & ucBitmask)
00567                 {
00568                     // Update the location
00569                     stPoint.usX = usX;
00570                     stPoint.usY = usY;
00571
00572                     // Draw the point.
00573                     DrawPixel(&stPoint);
00574                 }
00575             }
00576             ucBitmask >>= 1;

```

```

00577         }
00578
00579         usOffset++;
00580     }
00581
00582     // Next character
00583     usCharIndex++;
00584     usCharOffsetX += (K_USHORT)ucWidth + 1;
00585 }
00586 }
00587
00588 //-----
00589 void GraphicsDriver::TextFX(DrawText_t *pstText_,
    TextFX_t *pstFX_ )
00590 {
00591     K_USHORT usX, usY;
00592     K_USHORT usPartialX = 0;
00593     K_USHORT usPartialY = 0;
00594     K_USHORT usStartX;
00595     K_USHORT usStartY;
00596     K_USHORT usCharOffsetX;
00597     K_USHORT usCharIndex = 0;
00598     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00599     DrawPoint_t stPoint;
00600
00601     // set the color for this element.
00602     stPoint.uColor = pstText_>uColor;
00603
00604     usCharOffsetX = 0;
00605
00606     // Draw every character in the string, one at a time
00607     while (pstText_>pcString[usCharIndex] != 0)
00608     {
00609         K_USHORT usOffset = 0;
00610
00611         K_UCHAR ucWidth;
00612         K_UCHAR ucHeight;
00613         K_UCHAR ucVOffset;
00614         K_UCHAR ucBitmask;
00615
00616         // Read the glyphs from memory until we arrive at the one we wish to print
00617         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00618         {
00619             // Glyphs are variable-sized for efficiency - to look up a particular
00620             // glyph, we must traverse all preceding glyphs in the list
00621             ucWidth = Font_ReadByte(usOffset, pucData);
00622             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00623
00624             // Adjust the offset to point to the next glyph
00625             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00626                 + (sizeof(Glyph_t) - 1);
00627         }
00628
00629         // Header information: glyph size and vertical offset
00630         ucWidth = Font_ReadByte(usOffset++, pucData);
00631         ucHeight = Font_ReadByte(usOffset++, pucData);
00632         ucVOffset = Font_ReadByte(usOffset++, pucData);
00633
00634         usStartY = pstText_>usTop + (K_USHORT)ucVOffset;
00635         usStartX = pstText_>usLeft;
00636
00637         // Draw the font from left->right, top->bottom
00638         for ( usY = usStartY;
00639             usY < usStartY + (K_USHORT)ucHeight;
00640             usY++ )
00641         {
00642             K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
00643             ucBitmask = 0x80;
00644             usPartialY = 0;
00645             usPartialX = 0;
00646
00647             K_USHORT usTempPartialX = 0;
00648
00649             for ( usX = usCharOffsetX + usStartX;
00650                 usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
00651                 usX++ )
00652             {
00653                 K_USHORT usTempPartialY = 0;
00654                 usPartialY = 0;
00655                 if (!ucBitmask)
00656                 {
00657                     ucBitmask = 0x80;
00658                     usOffset++;
00659                     ucTempChar = Font_ReadByte(usOffset, pucData);
00660                 }
00661
00662                 if ((ucTempChar & ucBitmask) || (pstFX_>ucFlags &

```

```

TEXTFX_FLAG_OPAQUE_BG))
00663     {
00664         // usX and usY represent the untransformed data...
00665         // we need usStartX, usStartY, usDeltaX, usDeltaY to proceed.
00666         K_USHORT usDeltaX = (usX - pstText_>usLeft);
00667         K_USHORT usDeltaY = (usY - pstText_>usTop);
00668
00669         // Compute "unadjusted" pixels for normal or scaled
00670         K_USHORT usRawX, usRawY;
00671
00672         if (pstFX_>ucFlags & TEXTFX_FLAG_SCALE_X)
00673         {
00674             usRawX = usStartX + (((usDeltaX * pstFX_>usScaleX100))/100);
00675             usTempPartialX = pstFX_>usScaleX100;
00676         }
00677         else
00678         {
00679             usRawX = usX;
00680             usTempPartialX = 100;
00681         }
00682         usTempPartialX += usPartialX;
00683
00684         if (pstFX_>ucFlags & TEXTFX_FLAG_SCALE_Y)
00685         {
00686             usRawY = usStartY + (((usDeltaY * pstFX_>usScaleY100))/100);
00687             usTempPartialY = pstFX_>usScaleY100;
00688         }
00689         else
00690         {
00691             usRawY = usY;
00692             usTempPartialY = 100;
00693         }
00694         usTempPartialY += usPartialY;
00695
00696         K_USHORT usBLAH = usTempPartialX;
00697
00698         if (!(ucTempChar & ucBitmask))
00699         {
00700             stPoint.uColor = pstFX_>uBGColor;
00701         }
00702         else
00703         {
00704             stPoint.uColor = pstText_>uColor;
00705         }
00706
00707         stPoint.usX = usRawX;
00708         while (usTempPartialX >= 50)
00709         {
00710             stPoint.usY = usRawY;
00711             usBLAH = usTempPartialY;
00712             while (usBLAH >= 50)
00713             {
00714                 DrawPixel(&stPoint);
00715                 stPoint.usY++;
00716                 if (usBLAH >= 100)
00717                 {
00718                     usBLAH -= 100;
00719                 }
00720                 else
00721                 {
00722                     usBLAH = 0;
00723                 }
00724             }
00725             stPoint.usX++;
00726             if (usTempPartialX >= 100)
00727             {
00728                 usTempPartialX -= 100;
00729             }
00730             else
00731             {
00732                 usTempPartialX = 0;
00733             }
00734         }
00735         usPartialX = (usTempPartialX % 100);
00736         usPartialY = (usTempPartialY % 100);
00737     }
00738
00739     ucBitmask >>= 1;
00740 }
00741
00742     usOffset++;
00743 }
00744
00745 // Next character
00746 usCharIndex++;
00747

```

```

00750         usCharOffsetX += (K_USHORT)ucWidth + 1;
00751     }
00752 }
00753
00754 //-----
00755 K_USHORT GraphicsDriver::TextWidth(DrawText_t *pstText_)
00756 {
00757     K_USHORT usCharOffsetX;
00758     K_USHORT usCharIndex = 0;
00759     K_USHORT usX;
00760     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00761
00762     usCharOffsetX = 0;
00763
00764     // Draw every character in the string, one at a time
00765     while (pstText_>pcString[usCharIndex] != 0)
00766     {
00767         K_USHORT usOffset = 0;
00768
00769         K_UCHAR ucWidth;
00770         K_UCHAR ucHeight;
00771
00772         // Read the glyphs from memory until we arrive at the one we wish to print
00773         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00774         {
00775             // Glyphs are variable-sized for efficiency - to look up a particular
00776             // glyph, we must traverse all preceding glyphs in the list
00777             ucWidth = Font_ReadByte(usOffset, pucData);
00778             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00779
00780             // Adjust the offset to point to the next glyph
00781             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00782                 + (sizeof(Glyph_t) - 1);
00783         }
00784
00785         // Header information: glyph size and vertical offset
00786         ucWidth = Font_ReadByte(usOffset, pucData);
00787         usOffset += (sizeof(Glyph_t) - 1);
00788
00789         // Next character
00790         usCharIndex++;
00791         usCharOffsetX += (K_USHORT)ucWidth + 1;
00792     }
00793
00794     return usCharOffsetX;
00795 }
00796
00797 //-----
00798 void GraphicsDriver::TriangleWire(DrawPoly_t *pstPoly_)
00799 {
00800     DrawLine_t stLine;
00801
00802     stLine.uColor = pstPoly_>uColor;
00803
00804     stLine.usX1 = pstPoly_>pstVector[0].usX;
00805     stLine.usY1 = pstPoly_>pstVector[0].usY;
00806     stLine.usX2 = pstPoly_>pstVector[1].usX;
00807     stLine.usY2 = pstPoly_>pstVector[1].usY;
00808     Line(&stLine);
00809
00810     stLine.usX1 = pstPoly_>pstVector[1].usX;
00811     stLine.usY1 = pstPoly_>pstVector[1].usY;
00812     stLine.usX2 = pstPoly_>pstVector[2].usX;
00813     stLine.usY2 = pstPoly_>pstVector[2].usY;
00814     Line(&stLine);
00815
00816     stLine.usX1 = pstPoly_>pstVector[2].usX;
00817     stLine.usY1 = pstPoly_>pstVector[2].usY;
00818     stLine.usX2 = pstPoly_>pstVector[0].usX;
00819     stLine.usY2 = pstPoly_>pstVector[0].usY;
00820     Line(&stLine);
00821 }
00822 //-----
00823 void GraphicsDriver::TriangleFill(DrawPoly_t *pstPoly_)
00824 {
00825     // Drawing a raster-filled triangle:
00826     K_UCHAR ucMaxEdge = 0;
00827     K_UCHAR ucMinEdge1 = 0, ucMinEdge2 = 0;
00828     K_SHORT sMax = 0;
00829     K_SHORT sTemp;
00830
00831     K_SHORT sDeltaX1, sDeltaX2;
00832     K_SHORT sDeltaY1, sDeltaY2;
00833     K_CHAR cStepX1, cStepX2;
00834     K_CHAR cStepY;
00835     K_SHORT sX1, sX2, sX3, sY1, sY2, sY3;
00836     K_SHORT sTempX1, sTempY1, sTempX2, sTempY2;

```

```

00837     K_SHORT sFraction1;
00838     K_SHORT sFraction2;
00839     K_SHORT i;
00840     DrawPoint_t stPoint;
00841
00842     // Figure out which line segment is the longest
00843     sTemp = (K_SHORT)pstPoly->pstVector[0].usY - (K_SHORT)pstPoly->
pstVector[1].usY;
00844     if( sTemp < 0 )      { sTemp = -sTemp; }
00845     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 0; ucMinEdge1 = 1; ucMinEdge2 = 2; }
00846
00847     sTemp = (K_SHORT)pstPoly->pstVector[1].usY - (K_SHORT)pstPoly->
pstVector[2].usY;
00848     if( sTemp < 0 )      { sTemp = -sTemp; }
00849     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 1; ucMinEdge1 = 2; ucMinEdge2 = 0; }
00850
00851     sTemp = (K_SHORT)pstPoly->pstVector[2].usY - (K_SHORT)pstPoly->
pstVector[0].usY;
00852     if( sTemp < 0 )      { sTemp = -sTemp; }
00853     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 2; ucMinEdge1 = 0; ucMinEdge2 = 1; }
00854
00855     // Label the vectors and copy into temporary signed buffers
00856     sX1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usX;
00857     sX2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usX;
00858     sX3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usX;
00859
00860     sY1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usY;
00861     sY2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usY;
00862     sY3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usY;
00863
00864     // Figure out whether or not we're drawing up-down or down-up
00865     sDeltaY1 = sY1 - sY2;
00866     if (sDeltaY1 < 0) { cStepY = -1; sDeltaY1 = -sDeltaY1; } else { cStepY = 1; }
00867
00868     sDeltaX1 = sX1 - sX2;
00869     if (sDeltaX1 < 0) { cStepX1 = -1; sDeltaX1 = -sDeltaX1; } else { cStepX1 = 1; }
00870
00871     sDeltaY2 = sY1 - sY3;
00872     if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1; }
00873
00874     sDeltaX2 = sX1 - sX3;
00875     if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 = 1; }
00876
00877     sDeltaX1 <=&=1;
00878     sDeltaX2 <=&=1;
00879     sDeltaY1 <=&=1;
00880     sDeltaY2 <=&=1;
00881
00882     sFraction1 = sDeltaX1;// - (sDeltaY1 >> 1);
00883     sFraction2 = sDeltaX2;// - (sDeltaY2 >> 1);
00884
00885     sTempY1 = sY1;
00886     sTempY2 = sY1;
00887     sTempX1 = sX1;
00888     sTempX2 = sX1;
00889
00890     stPoint.uColor = pstPoly->uColor;
00891
00892     if( sDeltaY2 != 0 )
00893     {
00894         while (sTempY2 != sY3)
00895         {
00896             stPoint.usY = sTempY2;
00897             if( sTempX1 < sTempX2 ) {
00898                 for( i = sTempX1; i <= sTempX2; i++) {
00899                     stPoint.usX = i;
00900                     Point(&stPoint);
00901                 }
00902             } else {
00903                 for( i = sTempX2; i <= sTempX1; i++ ) {
00904                     stPoint.usX = i;
00905                     Point(&stPoint);
00906                 }
00907             }
00908
00909             while (sFraction2 >= sDeltaY2)
00910             {
00911                 sTempX2 -= cStepX2;
00912                 sFraction2 -= sDeltaY2;
00913             }
00914             sTempY2 -= cStepY;
00915             sFraction2 += sDeltaX2;
00916
00917             while (sFraction1 >= sDeltaY1)
00918             {
00919                 sTempX1 -= cStepX1;
00920                 sFraction1 -= sDeltaY1;

```

```

00921         }
00922         sTempY1 -= cStepY;
00923         sFraction1 += sDeltaX1;
00924     }
00925 }
00926
00927 sDeltaY2 = sY3 - sY2;
00928 sDeltaX2 = sX3 - sX2;
00929
00930 if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 = 1; }
00931 if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1; }
00932
00933 sDeltaX2 <= 1;
00934 sDeltaY2 <= 1;
00935
00936 sFraction2 = sDeltaX2; // - (sDeltaY2 >> 1);
00937
00938 sTempY2 = sY3;
00939 sTempX2 = sX3;
00940
00941 if( sDeltaY2 != 0)
00942 {
00943     while (sTempY2 != sY2)
00944     {
00945         stPoint.usY = sTempY2;
00946         if( sTempX1 < sTempX2 ) {
00947             for( i = sTempX1; i <= sTempX2; i++) {
00948                 stPoint.usX = i;
00949                 Point(&stPoint);
00950             }
00951         } else {
00952             for( i = sTempX2; i <= sTempX1; i++ ) {
00953                 stPoint.usX = i;
00954                 Point(&stPoint);
00955             }
00956         }
00957         while (sFraction2 >= sDeltaY2)
00958         {
00959             sTempX2 -= cStepX2;
00960             sFraction2 -= sDeltaY2;
00961         }
00962         sTempY2 -= cStepY;
00963         sFraction2 += sDeltaX2;
00964         while (sFraction1 >= sDeltaY1)
00965         {
00966             sTempX1 -= cStepX1;
00967             sFraction1 -= sDeltaY1;
00968         }
00969         sTempY1 -= cStepY;
00970         sFraction1 += sDeltaX1;
00971     }
00972 }
00973 }
00974 }
00975 }
00976
00977 //-----
00978 void GraphicsDriver::Polygon(DrawPoly_t *pstPoly_)
00979 {
00980     K_USHORT i,j,k;
00981     K_BOOL bState = false;
00982
00983     DrawPoly_t stTempPoly;
00984     DrawVector_t astTempVec[3];
00985
00986     if (pstPoly_>usNumPoints < 3)
00987     {
00988         return;
00989     }
00990
00991     stTempPoly.uColor = pstPoly_>uColor;
00992     stTempPoly.bFill = pstPoly_>bFill;
00993     stTempPoly.pstVector = astTempVec;
00994     stTempPoly.usNumPoints = 3;
00995
00996     astTempVec[0].usX = pstPoly_>pstVector[0].usX;
00997     astTempVec[1].usX = pstPoly_>pstVector[1].usX;
00998     astTempVec[0].usY = pstPoly_>pstVector[0].usY;
00999     astTempVec[1].usY = pstPoly_>pstVector[1].usY;
01000
01001     j = 2;
01002     astTempVec[2].usX = pstPoly_>pstVector[pstPoly_>usNumPoints - 1].usX;
01003     astTempVec[2].usY = pstPoly_>pstVector[pstPoly_>usNumPoints - 1].usY;
01004
01005     k = pstPoly_>usNumPoints - 2;
01006
01007     if( pstPoly_>bFill )

```

```

01008     {
01009         TriangleFill(&stTempPoly);
01010     }
01011     else
01012     {
01013         TriangleWire(&stTempPoly);
01014     }
01015
01016     // Filled polygon/wireframe polygon using triangle decomp.
01017     for(i = 0; i < pstPoly->usNumPoints - 3; i++)
01018     {
01019         astTempVec[0].usX = astTempVec[1].usX;
01020         astTempVec[1].usX = astTempVec[2].usX;
01021         astTempVec[0].usY = astTempVec[1].usY;
01022         astTempVec[1].usY = astTempVec[2].usY;
01023
01024         if( !bState )
01025         {
01026             bState = true;
01027             astTempVec[2].usX = pstPoly->pstVector[j].usX;
01028             astTempVec[2].usY = pstPoly->pstVector[j].usY;
01029             j++;
01030         }
01031         else
01032         {
01033             bState = false;
01034             astTempVec[2].usX = pstPoly->pstVector[k].usX;
01035             astTempVec[2].usY = pstPoly->pstVector[k].usY;
01036             k--;
01037         }
01038         if( pstPoly->bFill )
01039         {
01040             TriangleFill(&stTempPoly);
01041         }
01042         else
01043         {
01044             TriangleWire(&stTempPoly);
01045         }
01046     }
01047 }
01048
01049 //-----
01050 void GraphicsDriver::SetWindow(DrawWindow_t *pstWindow_)
01051 {
01052     if ((pstWindow_->usLeft <= pstWindow_->usRight) &&
01053         (pstWindow_->usRight < m_usResX) &&
01054         (pstWindow_->usLeft < m_usResX))
01055     {
01056         m_usLeft = pstWindow_->usLeft;
01057         m_usRight = pstWindow_->usRight;
01058     }
01059
01060     if ((pstWindow_->usTop <= pstWindow_->usBottom) &&
01061         (pstWindow_->usTop < m_usTop) &&
01062         (pstWindow_->usBottom < m_usBottom))
01063     {
01064         m_usTop = pstWindow_->usTop;
01065         m_usBottom = pstWindow_->usBottom;
01066     }
01067 }
01068
01069 //-----
01070 void GraphicsDriver::ClearWindow()
01071 {
01072     {
01073         m_usLeft = 0;
01074         m_usTop = 0;
01075         m_usRight = m_usResX - 1;
01076         m_usBottom = m_usResY - 1;
01077     }

```

17.69 /home/mo/mark3-source/embedded/stage/src/graphics.h File Reference

Graphics driver class declaration.

```

#include "driver.h"
#include "draw.h"

```



```

00159     virtual void Text(DrawText_t *pstText_);
00160
00169     void TextFX(DrawText_t *pstText_, TextFX_t *pstFX_);
00170
00177     virtual K_USHORT TextWidth(DrawText_t *pstText_);
00178
00184     void SetWindow( DrawWindow_t *pstWindow_ );
00185
00191     void ClearWindow();
00192 protected:
00193
00194     K_USHORT m_usResX;
00195     K_USHORT m_usResY;
00196
00197     K_USHORT m_usLeft;
00198     K_USHORT m_usTop;
00199     K_USHORT m_usRight;
00200     K_USHORT m_usBottom;
00201
00202     K_UCHAR m_ucBPP;
00203 };
00204
00205 #endif
00206

```

17.71 /home/mo/mark3-source/embedded/stage/src/gui.cpp File Reference

Graphical User Interface classes and data structure definitions.

```
#include "message.h"
#include "kerneltypes.h"
#include "gui.h"
#include "system_heap.h"
#include "fixed_heap.h"
#include "memutil.h"
```

17.71.1 Detailed Description

Graphical User Interface classes and data structure definitions.

Definition in file [gui.cpp](#).

17.72 gui.cpp

```

00001 /*-----
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00014 #include "message.h"
00020 #include "kerneltypes.h"
00021 #include "gui.h"
00022 #include "system_heap.h"
00023 #include "fixed_heap.h"
00024 #include "memutil.h"
00025
00026 //-----
00027 void GuiWindow::AddControl( GuiControl *pclControl_,
GuiControl *pclParent_ )
00028 {
00029     GUI_DEBUG_PRINT("GuiWindow::AddControl\n");
00030
00031     m_clControlList.Add(static_cast<LinkListNode*>(pclControl_));

```

```

00032     m_pclInFocus = pclControl_;
00033     m_ucControlCount++;
00034
00035     pclControl_->SetParentWindow(this);
00036     pclControl_->SetParentControl(pclParent_);
00037 }
00038
00039 //-----
00040 void GuiWindow::RemoveControl( GuiControl *pclControl_ )
00041 {
00042     GUI_DEBUG_PRINT("GuiWindow::RemoveControl\n");
00043     if (pclControl_->GetPrev())
00044     {
00045         m_pclInFocus = static_cast<GuiControl*>(pclControl_->
00046         GetPrev());
00047     }
00048     else if (pclControl_->GetNext())
00049     {
00050         m_pclInFocus = static_cast<GuiControl*>(pclControl_->
00051         GetNext());
00052     }
00053     {
00054         m_pclInFocus = NULL;
00055     }
00056     m_clControlList.Remove(static_cast<LinkListNode*>(pclControl_));
00057     m_ucControlCount--;
00058 }
00059
00060 //-----
00061 K_UCHAR GuiWindow::GetMaxZOrder()
00062 {
00063     GUI_DEBUG_PRINT("GuiWindow::GetMaxZOrder\n");
00064
00065     LinkListNode *pclTempNode;
00066     K_UCHAR ucZ = 0;
00067     K_UCHAR ucTempZ;
00068
00069     pclTempNode = m_clControlList.GetHead();
00070
00071     while (pclTempNode)
00072     {
00073         ucTempZ = (static_cast<GuiControl*>(pclTempNode))->GetZOrder();
00074         if (ucTempZ > ucZ)
00075         {
00076             ucZ = ucTempZ;
00077         }
00078         pclTempNode = pclTempNode->GetNext();
00079     }
00080
00081     return ucZ;
00082 }
00083
00084 //-----
00085 void GuiWindow::Redraw( K_BOOL bRedrawAll_ )
00086 {
00087     GUI_DEBUG_PRINT("GuiWindow::Redraw\n");
00088
00089     K_UCHAR ucControlsLeft = m_ucControlCount;
00090     K_UCHAR ucCurrentZ = 0;
00091     K_UCHAR ucMaxZ;
00092
00093     ucMaxZ = GetMaxZOrder();
00094
00095     // While there are still controls left to process (and we're less than
00096     // the maximum Z-order, just a sanity check.), redraw each object that
00097     // has its stale flag set, or all controls if the bRedrawAll_ parameter
00098     // is true.
00099     while (ucControlsLeft && (ucCurrentZ <= ucMaxZ))
00100     {
00101         LinkListNode *pclTempNode;
00102
00103         pclTempNode = m_clControlList.GetHead();
00104         while (pclTempNode)
00105         {
00106             GuiControl* pclTempControl = static_cast<GuiControl*>(pclTempNode);
00107             if (pclTempControl->GetZOrder() == ucCurrentZ)
00108             {
00109                 if ((bRedrawAll_) || (pclTempControl->IsStale()))
00110                 {
00111                     pclTempControl->Draw();
00112                     pclTempControl->ClearStale();
00113                 }
00114
00115                 ucControlsLeft--;
00116             }

```

```

00117
00118         pclTempNode = pclTempNode->GetNext();
00119     }
00120     ucCurrentZ++;
00121 }
00122 GUI_DEBUG_PRINT(" Current Z: %d\n", ucCurrentZ);
00123 GUI_DEBUG_PRINT(" Controls Left: %d\n", ucControlsLeft);
00124 }
00125
00126 //-----
00127 void GuiWindow::InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT
usWidth_, K_USHORT usHeight_ )
00128 {
00129     LinkListNode *pclTempNode;
00130     K_USHORT usLeft1, usLeft2, usRight1, usRight2, usTop1, usTop2, usBottom1, usBottom2;
00131
00132     pclTempNode = m_clControlList.GetHead();
00133
00134     usLeft1 = usLeft_;
00135     usRight1 = usLeft_ + usWidth_ - 1;
00136     usTop1 = usTop_;
00137     usBottom1 = usTop_ + usHeight_ - 1;
00138
00139     while (pclTempNode)
00140     {
00141         GuiControl *pclControl = static_cast<GuiControl*>(pclTempNode);
00142         K_USHORT usX, usY;
00143
00144         bool bMatch = false;
00145
00146         // Get the absolute display coordinates
00147         pclControl->GetControlOffset(&usX, &usY);
00148
00149
00150         usLeft2 = pclControl->GetLeft() + usX;
00151         usRight2 = usLeft2 + pclControl->GetWidth() - 1;
00152         usTop2 = pclControl->GetTop() + usY;
00153         usBottom2 = usTop2 + pclControl->GetHeight() - 1;
00154
00155         // If the control has any pixels in the bounding box.
00156         if (
00157             (
00158                 (
00159                     (usLeft1 >= usLeft2) &&
00160                     (usLeft1 <= usRight2)
00161                 ) ||
00162                 (
00163                     (usRight1 >= usLeft2) &&
00164                     (usRight1 <= usRight2)
00165                 ) ||
00166                 ((usLeft1 <= usLeft2) && (usRight1 >= usRight2))
00167             ) &&
00168             (
00169                 (
00170                     (usTop1 >= usTop2) &&
00171                     (usTop1 <= usBottom2)
00172                 ) ||
00173                 (
00174                     (usBottom1 >= usTop2) &&
00175                     (usBottom1 <= usBottom2)
00176                 ) ||
00177                 ((usTop1 <= usTop2) && (usBottom1 >= usBottom2))
00178             )
00179         )
00180         {
00181             bMatch = true;
00182         }
00183         else if(
00184             (
00185                 (
00186                     (usLeft2 >= usLeft1) &&
00187                     (usLeft2 <= usRight1)
00188                 ) ||
00189                 (
00190                     (usRight2 >= usLeft1) &&
00191                     (usRight2 <= usRight1)
00192                 ) ||
00193                 ((usLeft2 <= usLeft1) && (usRight2 >= usRight1))
00194             ) &&
00195             (
00196                 (
00197                     (usTop2 >= usTop1) &&
00198                     (usTop2 <= usBottom1)
00199                 ) ||
00200                 (
00201                     (usBottom2 >= usTop1) &&
00202                     (usBottom2 <= usBottom1)

```

```

00203         ) ||
00204         ((usTop2 <= usTop1) && (usBottom2 >= usBottom1))
00205     )
00206 )
00207 {
00208     bMatch = true;
00209 }
00210
00211
00212 if (bMatch)
00213 {
00214     pclControl->SetStale();
00215
00216     // Invalidate all child controls as well (since redrawing a parent could cause them to
disappear)
00217     GuiControl *pclChild = static_cast<GuiControl*>(
m_clControlList.GetHead());
00218
00219     // Go through all controls and check for parental ancestry
00220     while (pclChild)
00221     {
00222         GuiControl *pclParent = static_cast<GuiControl*>(pclChild->
GetParentControl());
00223
00224         // If this control is a descendant of the current control at some level
00225         while (pclParent)
00226         {
00227             if (pclParent == pclControl)
00228             {
00229                 // Set the control as stale
00230                 pclChild->SetStale();
00231                 break;
00232             }
00233             pclParent = pclParent->GetParentControl();
00234         }
00235
00236         pclChild = static_cast<GuiControl*>((static_cast<
LinkListNode*>(pclChild))->GetNext());
00237     }
00238 }
00239
00240     pclTempNode = pclTempNode->GetNext();
00241 }
00242 }
00243
00244 //-----
00245 void GuiWindow::ProcessEvent( GuiEvent_t *pstEvent_ )
00246 {
00247     GUI_DEBUG_PRINT("GuiWindow::ProcessEvent\n");
00248
00249     // If the event is for broadcast - send it to all controls,
00250     // without regard to order.
00251     if ((TARGET_ID_BROADCAST == pstEvent_->ucTargetID)
00252         || (TARGET_ID_BROADCAST_Z == pstEvent_->ucTargetID))
00253     {
00254         GUI_DEBUG_PRINT(" TARGET_ID_BROADCAST(_Z)\n");
00255
00256         LinkListNode *pclTempNode;
00257         pclTempNode = m_clControlList.GetHead();
00258
00259         while (pclTempNode)
00260         {
00261             GuiReturn_t eRet;
00262             eRet = (static_cast<GuiControl*>(pclTempNode))->ProcessEvent(pstEvent_);
00263             if (GUI_EVENT_CONSUMED == eRet)
00264             {
00265                 break;
00266             }
00267             pclTempNode = pclTempNode->GetNext();
00268         }
00269     }
00270     // Send the event only to the currently-selected object.
00271     else if (TARGET_ID_FOCUS == pstEvent_->ucTargetID)
00272     {
00273         GUI_DEBUG_PRINT(" TARGET_ID_FOCUS\n");
00274         GuiReturn_t eReturn = GUI_EVENT_OK;
00275
00276         // Try to let the control process the event on its own
00277         if (m_pclInFocus)
00278         {
00279             eReturn = m_pclInFocus->ProcessEvent(pstEvent_);
00280         }
00281
00282         // If the event was not consumed, use default logic to process the event
00283         if (GUI_EVENT_CONSUMED != eReturn)
00284         {
00285             if (EVENT_TYPE_KEYBOARD == pstEvent_->ucEventType)

```

```

00286         {
00287             if (KEYCODE_TAB == pstEvent_>stKey.ucKeyCode)
00288             {
00289                 if (pstEvent_>stKey.bKeyState)
00290                 {
00291                     CycleFocus(true);
00292                 }
00293             }
00294         }
00295         else if (EVENT_TYPE_JOYSTICK == pstEvent_>
ucEventType)
00296         {
00297             if (pstEvent_>stJoystick.bUp || pstEvent_>
stJoystick.bLeft)
00298             {
00299                 // Cycle focus *backwards*
00300                 CycleFocus(false);
00301             }
00302             else if (pstEvent_>stJoystick.bRight || pstEvent_>
stJoystick.bDown)
00303             {
00304                 // Cycle focus *forewards*
00305                 CycleFocus(true);
00306             }
00307         }
00308     }
00309 }
00310 else if (TARGET_ID_HIGH_Z == pstEvent_>ucTargetID)
00311 {
00312     GUI_DEBUG_PRINT("  TARGET_ID_HIGH_Z\n");
00313     K_USHORT usTargetX, usTargetY;
00314     K_USHORT usOffsetX, usOffsetY;
00315     K_UCHAR ucMaxZ = 0;
00316     LinkListNode *pclTempNode;
00317     pclTempNode = m_clControlList.GetHead();
00318     switch (pstEvent_>ucEventType)
00319     {
00320     case EVENT_TYPE_MOUSE:
00321     case EVENT_TYPE_TOUCH:
00322     {
00323         GuiControl *pclTargetControl = NULL;
00324         // Read the target X/Y coordinates out of the event struct
00325         if (EVENT_TYPE_TOUCH == pstEvent_>ucEventType)
00326         {
00327             usTargetX = pstEvent_>stTouch.usX;
00328             usTargetY = pstEvent_>stTouch.usY;
00329         }
00330         else
00331         {
00332             usTargetX = pstEvent_>stMouse.usX;
00333             usTargetY = pstEvent_>stMouse.usY;
00334         }
00335         // Go through every control on the window, checking to see if the
00336         // event falls within the bounding box
00337         while (pclTempNode)
00338         {
00339             GuiControl *pclControl = (static_cast<GuiControl*>)(pclTempNode);
00340             pclControl->GetControlOffset(&usOffsetX, &usOffsetY);
00341             // Compare event coordinates to bounding box (with offsets)
00342             if ( ((usTargetX >= (usOffsetX + pclControl->GetLeft()) &&
(usTargetX <= (usOffsetX + pclControl->GetLeft() + pclControl->
GetWidth() - 1))) &&
00343             ((usTargetY >= (usOffsetY + pclControl->GetTop()) &&
(usTargetY <= (usOffsetY + pclControl->GetTop() + pclControl->
GetHeight() - 1))) )
00344             {
00345                 // If this control is higher in Z-Order, set this as the newest
00346                 // candidate control to accept the event
00347                 if (pclControl->GetZOrder() >= ucMaxZ)
00348                 {
00349                     pclTargetControl = pclControl;
00350                     ucMaxZ = pclControl->GetZOrder();
00351                 }
00352             }
00353             pclTempNode = pclTempNode->GetNext();
00354         }
00355         // If a suitable control was found on the event surface, pass the event off
00356         // for processing.
00357     }
00358 }

```

```

00368         if (pclTargetControl)
00369         {
00370             // If the selected control is different from the current in-focus
00371             // control, then deactivate that control.
00372             if (m_pclInFocus && (m_pclInFocus != pclTargetControl))
00373             {
00374                 m_pclInFocus->Activate(false);
00375                 m_pclInFocus = NULL;
00376             }
00377             (static_cast<GuiControl*>(pclTargetControl))->ProcessEvent(pstEvent_);
00378         }
00379     }
00380     break;
00381 default:
00382     break;
00383 }
00384 }
00385 }
00386 //-----
00387 void GuiWindow::SetFocus( GuiControl *pclControl_ )
00388 {
00389     GUI_DEBUG_PRINT("GuiWindow::SetFocus\n");
00390     m_pclInFocus = pclControl_;
00391 }
00392 }
00393 //-----
00394 void GuiWindow::CycleFocus( bool bForward_ )
00395 {
00396     GUI_DEBUG_PRINT("GuiWindow::CycleFocus\n");
00397     // Set starting point and cached copy of current nodes
00398     LinkListNode *pclTempNode = static_cast<GuiControl*>(
00399         m_clControlList.GetHead());
00400     LinkListNode *pclStartNode = m_pclInFocus;
00401     if (bForward_)
00402     {
00403         // If there isn't a current focus node, set the focus to the beginning
00404         // of the list
00405         if (!m_pclInFocus)
00406         {
00407             m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00408             if (!m_pclInFocus)
00409             {
00410                 return;
00411             }
00412             pclTempNode = static_cast<GuiControl*>(m_pclInFocus);
00413             pclStartNode = NULL;
00414         }
00415         else
00416         {
00417             // Deactivate the control that's losing focus
00418             static_cast<GuiControl*>(m_pclInFocus)->Activate(false);
00419             // Otherwise start with the next node
00420             pclStartNode = pclStartNode->GetNext();
00421         }
00422         // Go through the whole control list and find the next one to accept
00423         // the focus
00424         while (pclTempNode && pclTempNode != pclStartNode)
00425         {
00426             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00427             {
00428                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00429                 m_pclInFocus->Activate(true);
00430                 SetFocus(m_pclInFocus);
00431                 return;
00432             }
00433             pclTempNode = pclTempNode->GetNext();
00434         }
00435         pclTempNode = static_cast<GuiControl*>(m_clControlList.
00436             GetHead());
00437         while (pclTempNode && pclTempNode != pclStartNode)
00438         {
00439             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00440             {
00441                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00442                 m_pclInFocus->Activate(true);
00443                 SetFocus(m_pclInFocus);
00444                 return;
00445             }
00446             pclTempNode = pclTempNode->GetNext();
00447         }
00448     }
00449 }

```

```

00453     else
00454     {
00455         pclTempNode = static_cast<GuiControl*>(m_clControllist.
GetTail());
00456         pclStartNode = m_pclInFocus;
00457
00458         // If there isn't a current focus node, set the focus to the end
00459         // of the list
00460         if (!m_pclInFocus)
00461         {
00462             m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00463             if (!m_pclInFocus)
00464             {
00465                 return;
00466             }
00467             pclTempNode = static_cast<GuiControl*>(m_pclInFocus);
00468             pclStartNode = NULL;
00469         }
00470         else
00471         {
00472             // Deactivate the control that's losing focus
00473             static_cast<GuiControl*>(m_pclInFocus)->Activate(false);
00474
00475             // Otherwise start with the previous node
00476             pclStartNode = pclStartNode->GetPrev();
00477         }
00478
00479         // Go through the whole control list and find the next one to accept
00480         // the focus
00481         while (pclTempNode && pclTempNode != pclStartNode)
00482         {
00483             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00484             {
00485                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00486                 m_pclInFocus->Activate(true);
00487                 SetFocus(m_pclInFocus);
00488                 return;
00489             }
00490             pclTempNode = pclTempNode->GetPrev();
00491         }
00492
00493         pclTempNode = static_cast<GuiControl*>(m_clControllist.
GetTail());
00494         while (pclTempNode && pclTempNode != pclStartNode)
00495         {
00496             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00497             {
00498                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00499                 m_pclInFocus->Activate(true);
00500                 SetFocus(m_pclInFocus);
00501                 return;
00502             }
00503             pclTempNode = pclTempNode->GetPrev();
00504         }
00505     }
00506 }
00507 //-----
00508 GuiWindow *GuiEventSurface::FindWindowByName( const K_CHAR *
szName_ )
00509 {
00510     LinkListNode *pclTempNode = static_cast<LinkListNode*>(
m_clWindowList.GetHead());
00511
00512     while (pclTempNode)
00513     {
00514         if (MemUtil::CompareStrings( szName_, static_cast<GuiWindow*>(pclTempNode)->
GetName()))
00515         {
00516             return static_cast<GuiWindow*>(pclTempNode);
00517         }
00518         pclTempNode = pclTempNode->GetNext();
00519     }
00520
00521     return NULL;
00522 }
00523
00524 //-----
00525 void GuiEventSurface::AddWindow( GuiWindow *pclWindow_ )
00526 {
00527     GUI_DEBUG_PRINT("GuiEventSurface::AddWindow\n");
00528     m_clWindowList.Add(static_cast<LinkListNode*>(pclWindow_));
00529 }
00530
00531 //-----
00532 void GuiEventSurface::RemoveWindow( GuiWindow *pclWindow_ )
00533 {
00534

```

```

00535     GUI_DEBUG_PRINT("GuiEventSurface::RemoveWindow\n");
00536
00537     m_clWindowList.Remove(static_cast<LinkListNode*>(pclWindow_));
00538 }
00539
00540 //-----
00541 K_BOOL GuiEventSurface::SendEvent( GuiEvent_t *pstEvent_ )
00542 {
00543     GUI_DEBUG_PRINT("GuiEventSurface::SendEvent\n");
00544
00545     // Allocate a message from the global message pool
00546     Message *pclMessage = GlobalMessagePool::Pop();
00547
00548     // No messages available? Return a failure
00549     if (!pclMessage)
00550     {
00551         return false;
00552     }
00553
00554     // Allocate a copy of the event from the heap
00555     GuiEvent_t *pstEventCopy = static_cast<GuiEvent_t*>(
00556         SystemHeap::Alloc(sizeof(GuiEvent_t)));
00557
00558     // If the allocation fails, push the message back to the global pool and bail
00559     if (!pstEventCopy)
00560     {
00561         GlobalMessagePool::Push(pclMessage);
00562         return false;
00563     }
00564
00565     // Copy the source event into the destination event buffer
00566     CopyEvent(pstEventCopy, pstEvent_);
00567
00568     // Set the new event as the message payload
00569     pclMessage->SetData(static_cast<void*>(pstEventCopy));
00570
00571     // Send the event to the message queue
00572     m_clMessageQueue.Send(pclMessage);
00573
00574     return true;
00575 }
00576 //-----
00577 K_BOOL GuiEventSurface::ProcessEvent()
00578 {
00579     GUI_DEBUG_PRINT("GuiEventSurface::ProcessEvent\n");
00580
00581     // read the event from the queue (blocking call)
00582     Message *pclMessage = m_clMessageQueue.Receive();
00583     GuiEvent_t stLocalEvent;
00584
00585     // If we failed to get something from the queue,
00586     // bail out
00587     if (!pclMessage)
00588     {
00589         return false;
00590     }
00591
00592     // Copy the event data from the message into a local copy
00593     CopyEvent(&stLocalEvent,
00594         static_cast<GuiEvent_t*>(pclMessage->GetData()));
00595
00596     // Free the message and event as soon as possible, since
00597     // they are shared system resources
00598     SystemHeap::Free(pclMessage->GetData());
00599     GlobalMessagePool::Push(pclMessage);
00600
00601     // Special case check - target ID is the highest Z-ordered window(s) ONLY.
00602     if (stLocalEvent.ucTargetID == TARGET_ID_BROADCAST_Z)
00603     {
00604         LinkListNode* pclTempNode = m_clWindowList.
00605         GetHead();
00606         K_UCHAR ucMaxZ = 0;
00607
00608         while (pclTempNode)
00609         {
00610             if (ucMaxZ < (static_cast<GuiWindow*>(pclTempNode))->GetZOrder() )
00611             {
00612                 ucMaxZ = static_cast<GuiWindow*>(pclTempNode)->GetZOrder();
00613             }
00614             pclTempNode = pclTempNode->GetNext();
00615         }
00616
00617         // Iterate through all windows again - may have multiple windows
00618         // at the same z-order.
00619         pclTempNode = m_clWindowList.GetHead();
00620         while (pclTempNode)

```



```

00620     {
00621         if (ucMaxZ == (static_cast<GuiWindow*>(pclTempNode))->GetZOrder())
00622         {
00623             (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&stLocalEvent);
00624         }
00625         pclTempNode = pclTempNode->GetNext();
00626     }
00627 }
00628 // Broadcast the event - sending it to *all* windows. Let the individual
00629 // windows figure out what to do with the events.
00630 else
00631 {
00632     LinkListNode* pclTempNode = m_clWindowList.
00633     GetHead();
00634     while (pclTempNode)
00635     {
00636         (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&stLocalEvent);
00637         pclTempNode = pclTempNode->GetNext();
00638     }
00639 }
00640 // Return out
00641 return true;
00642 }
00643
00644 //-----
00645 void GuiEventSurface::CopyEvent( GuiEvent_t *pstDst_,
00646     GuiEvent_t *pstSrc_ )
00647 {
00648     GUI_DEBUG_PRINT("GuiEventSurface::CopyEvent\n");
00649     K_UCHAR *pucDst_ = (K_UCHAR*)pstDst_;
00650     K_UCHAR *pucSrc_ = (K_UCHAR*)pstSrc_;
00651     K_UCHAR i;
00652     for (i = 0; i < sizeof(GuiEvent_t); i++)
00653     {
00654         *pucDst_++ = *pucSrc_++;
00655     }
00656 }
00657 //-----
00658 void GuiEventSurface::InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_,
00659     K_USHORT usWidth_, K_USHORT usHeight_ )
00660 {
00661     LinkListNode* pclTempNode = m_clWindowList.GetHead();
00662     while (pclTempNode)
00663     {
00664         (static_cast<GuiWindow*>(pclTempNode))->InvalidateRegion(usLeft_, usTop_, usWidth_,
00665             usHeight_);
00666         pclTempNode = pclTempNode->GetNext();
00667     }
00668 }
00669 //-----
00670 void GuiControl::GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ )
00671 {
00672     GUI_DEBUG_PRINT("GuiControl::GetControlOffset\n");
00673     GuiControl *pclTempControl = m_pclParentControl;
00674     *pusX_ = 0;
00675     *pusY_ = 0;
00676     while (pclTempControl)
00677     {
00678         *pusX_ += pclTempControl->GetLeft();
00679         *pusY_ += pclTempControl->GetTop();
00680         pclTempControl = pclTempControl->GetParentControl();
00681     }
00682     if (m_pclParentWindow)
00683     {
00684         *pusX_ += m_pclParentWindow->GetLeft();
00685         *pusY_ += m_pclParentWindow->GetTop();
00686     }
00687 }

```

17.73 /home/mo/mark3-source/embedded/stage/src/gui.h File Reference

Graphical User Interface classes and data structure declarations.

```
#include "kerneltypes.h"
#include "ll.h"
#include "driver.h"
#include "graphics.h"
#include "message.h"
#include "keycodes.h"
```

Classes

- struct [KeyEvent_t](#)
Keyboard UI event structure definition.
- struct [MouseEvent_t](#)
Mouse UI event structure.
- struct [TouchEvent_t](#)
Touch UI event structure.
- struct [JoystickEvent_t](#)
Joystick UI event structure.
- struct [TimerEvent_t](#)
Timer UI event structure.
- struct [GuiEvent_t](#)
Composite UI event structure.
- class [GuiWindow](#)
Basic Window Class.
- class [GuiEventSurface](#)
GUI Event Surface Object.
- class [GuiControl](#)
GUI Control Base Class.
- class [StubControl](#)
Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Macros

- #define [GUI_DEBUG](#) (0)
- #define [GUI_DEBUG_PRINT](#)(...)
- #define [EVENT_STATE_UP](#) (0)
Event state definitions, used for determining whether or not a button or key is in the "up" or "down" contact state.
- #define [EVENT_STATE_DOWN](#) (1)
- #define [MAX_WINDOW_CONTROLS](#) (251)
Maximum number of controls per window.
- #define [TARGET_ID_BROADCAST_Z](#) (252)
Broadcast event to all controls in the topmost window.
- #define [TARGET_ID_BROADCAST](#) (253)
Send event to all controls in all windows.
- #define [TARGET_ID_FOCUS](#) (254)
Send event to the in-focus control.
- #define [TARGET_ID_HIGH_Z](#) (255)
Send event to the highest Z-order control.

Enumerations

- enum `GuiEventType_t` {
 `EVENT_TYPE_KEYBOARD`, `EVENT_TYPE_MOUSE`, `EVENT_TYPE_TOUCH`, `EVENT_TYPE_JOYSTICK`,
 `EVENT_TYPE_TIMER`, `EVENT_TYPE_COUNT` }

 Enumeration defining the various UI event codes.
- enum `GuiReturn_t` {
 `GUI_EVENT_OK` = 0, `GUI_EVENT_CONSUMED`, `GUI_EVENT_CANCEL`, `GUI_EVENT_RETRY`,
 `GUI_EVENT_COUNT` }

17.73.1 Detailed Description

Graphical User Interface classes and data structure declarations.

Definition in file [gui.h](#).

17.73.2 Enumeration Type Documentation

17.73.2.1 enum GuiEventType_t

Enumeration defining the various UI event codes.

Enumerator

EVENT_TYPE_KEYBOARD Keypress event.

EVENT_TYPE_MOUSE Mouse movement or click event.

EVENT_TYPE_TOUCH Touchscreen movement event.

EVENT_TYPE_JOYSTICK Joystick event.

EVENT_TYPE_TIMER [Timer](#) event.

EVENT_TYPE_COUNT Count of different event types supported.

Definition at line 65 of file gui.h.

17.73.2.2 enum GuiReturn_t

Enumerator

GUI_EVENT_OK No problem.

GUI_EVENT_CONSUMED Event was consumed.

GUI_EVENT_CANCEL Event processing canceled.

GUI_EVENT_RETRY Retry processing the event.

Definition at line 203 of file gui.h.

17.74 gui.h

```

00001 /*=====
00002
00003  _____
00004  |         |         |         |         |         |
00005  |   \     /   \     /   \     /   \     /   \     /
00006  |  / \   / \   / \   / \   / \   / \   / \   / \
00007  |_/___/_/___/_/___/_/___/_/___/_/___/_/___/_/___/_/
00008
00009  --[Mark3 Realtime Platform]-----
00010

```

```

00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __GUI_H__
00020 #define __GUI_H__
00021
00022 #include "kerneltypes.h"
00023 #include "ll.h"
00024 #include "driver.h"
00025 #include "graphics.h"
00026
00027 #include "message.h"
00028
00029 #include "keycodes.h"
00030
00031 #define GUI_DEBUG                (0)
00032
00033 #if GUI_DEBUG
00034     #include <stdio.h>
00035     #include <stdlib.h>
00036     #include <string.h>
00037
00038     #define GUI_DEBUG_PRINT        printf
00039 #else
00040 #define GUI_DEBUG_PRINT(...)
00041 #endif
00042
00043
00044 //-----
00049 #define EVENT_STATE_UP            (0)
00050 #define EVENT_STATE_DOWN        (1)
00051
00052 //-----
00053 #define MAX_WINDOW_CONTROLS      (251)
00054
00055 #define TARGET_ID_BROADCAST_Z    (252)
00056 #define TARGET_ID_BROADCAST      (253)
00057 #define TARGET_ID_FOCUS          (254)
00058 #define TARGET_ID_HIGH_Z        (255)
00059
00060
00061 //-----
00065 typedef enum
00066 {
00067     EVENT_TYPE_KEYBOARD,
00068     EVENT_TYPE_MOUSE,
00069     EVENT_TYPE_TOUCH,
00070     EVENT_TYPE_JOYSTICK,
00071     EVENT_TYPE_TIMER,
00072 }
00073     EVENT_TYPE_COUNT
00074 } GuiEventType_t;
00075
00076 //-----
00080 typedef struct
00081 {
00082     K_UCHAR ucKeyCode;
00083     union
00084     {
00085         K_UCHAR ucFlags;
00086         struct
00087         {
00088             unsigned int bKeyState:1;
00089             unsigned int bShiftState:1;
00090             unsigned int bCtrlState:1;
00091             unsigned int bAltState:1;
00092             unsigned int bWinState:1;
00093             unsigned int bFnState:1;
00094         };
00095     };
00096 } KeyEvent_t;
00097
00098 //-----
00102 typedef struct
00103 {
00104     K_USHORT usX;
00105     K_USHORT usY;
00106     union
00107     {
00108         K_UCHAR ucFlags;
00109         struct
00110         {
00111             unsigned int bLeftState:1;
00112             unsigned int bRightState:1;
00113             unsigned int bMiddleState:1;
00114             unsigned int bScrollUp:1;
00115

```

```

00116         unsigned int bScrollDown:1;
00117     };
00118 };
00119 } MouseEvent_t;
00120
00121 //-----
00125 typedef struct
00126 {
00127     K_USHORT usX;
00128     K_USHORT usY;
00129
00130     union
00131     {
00132         K_USHORT ucFlags;
00133         struct
00134         {
00135             unsigned int bTouch:1;
00136         };
00137     };
00138 } TouchEvent_t;
00139
00140 //-----
00144 typedef struct
00145 {
00146     union
00147     {
00148         K_USHORT usRawData;
00149         struct
00150         {
00151             unsigned int bUp:1;
00152             unsigned int bDown:1;
00153             unsigned int bLeft:1;
00154             unsigned int bRight:1;
00155
00156             unsigned int bButton1:1;
00157             unsigned int bButton2:1;
00158             unsigned int bButton3:1;
00159             unsigned int bButton4:1;
00160             unsigned int bButton5:1;
00161             unsigned int bButton6:1;
00162             unsigned int bButton7:1;
00163             unsigned int bButton8:1;
00164             unsigned int bButton9:1;
00165             unsigned int bButton10:1;
00166
00167             unsigned int bSelect:1;
00168             unsigned int bStart:1;
00169         };
00170     };
00171 } JoystickEvent_t;
00172
00173 //-----
00177 typedef struct
00178 {
00179     K_USHORT usTicks;
00180 } TimerEvent_t;
00181
00182 //-----
00187 typedef struct
00188 {
00189     K_UCHAR ucEventType;
00190     K_UCHAR ucTargetID;
00191     union
00192     {
00193         KeyEvent_t      stKey;
00194         MouseEvent_t     stMouse;
00195         TouchEvent_t     stTouch;
00196         JoystickEvent_t  stJoystick;
00197         TimerEvent_t     stTimer;
00198     };
00199
00200 } GuiEvent_t;
00201
00202 //-----
00203 typedef enum
00204 {
00205     GUI_EVENT_OK = 0,
00206     GUI_EVENT_CONSUMED,
00207     GUI_EVENT_CANCEL,
00208     GUI_EVENT_RETRY,
00209 } GUI_EVENT_COUNT
00210 } GuiReturn_t;
00211
00212 class GuiControl;
00213
00214 //-----

```

```

00223 class GuiWindow : public LinkListNode
00224 {
00225
00226 public:
00231     void Init()
00232     {
00233         m_ucControlCount = 0;
00234         m_pclDriver = NULL;
00235         m_szName = "";
00236     }
00237
00244     void SetDriver( GraphicsDriver *pclDriver_ ) {
m_pclDriver = pclDriver_; }
00245
00252     GraphicsDriver *GetDriver() { return m_pclDriver; }
00253
00265     void AddControl( GuiControl *pclControl_, GuiControl *pclParent_ );
00266
00274     void RemoveControl( GuiControl *pclControl_ );
00275
00283     K_UCHAR GetMaxZOrder();
00284
00293     void Redraw( K_BOOL bRedrawAll_ );
00294
00301     void ProcessEvent( GuiEvent_t *pstEvent_ );
00302
00311     void SetFocus( GuiControl *pclControl_ );
00312
00323     K_BOOL IsInFocus( GuiControl *pclControl_ )
00324     {
00325         if (m_pclInFocus == pclControl_)
00326         {
00327             return true;
00328         }
00329         return false;
00330     }
00331
00337     void SetTop( K_USHORT usTop_ )          { m_usTop = usTop_; }
00338
00344     void SetLeft( K_USHORT usLeft_ )        { m_usLeft = usLeft_; }
00345
00351     void SetHeight( K_USHORT usHeight_ ) { m_usHeight = usHeight_; }
00352
00358     void SetWidth( K_USHORT usWidth_ )      { m_usWidth = usWidth_; }
00359
00365     K_USHORT GetTop()                       { return m_usTop; }
00366
00372     K_USHORT GetLeft()                     { return m_usLeft; }
00373
00379     K_USHORT GetHeight()                   { return m_usHeight; }
00380
00386     K_USHORT GetWidth()                    { return m_usWidth; }
00387
00391     K_UCHAR GetZOrder()                    { return m_ucZ; }
00392
00396     void SetZOrder( K_UCHAR ucZ_ ) { m_ucZ = ucZ_; }
00397
00405     void CycleFocus( bool bForward_ );
00406
00410     void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }
00411
00415     const K_CHAR *GetName() { return m_szName; }
00416
00422     void InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT
usHeight_ );
00423
00424 private:
00425     K_USHORT m_usTop;
00426     K_USHORT m_usLeft;
00427     K_USHORT m_usHeight;
00428     K_USHORT m_usWidth;
00429
00430     K_UCHAR m_ucZ;
00431     const K_CHAR *m_szName;
00432
00433     DoubleLinkedList m_clControlList;
00434     GuiControl *m_pclInFocus;
00435     K_UCHAR m_ucControlCount;
00436     GraphicsDriver *m_pclDriver;
00437 };
00438
00439 //-----
00452 class GuiEventSurface
00453 {
00454 public:
00459     void Init() { m_clMessageQueue.Init(); }
00460

```

```

00466     void AddWindow( GuiWindow *pclWindow_ );
00467
00473     void RemoveWindow( GuiWindow *pclWindow_ );
00474
00482     K_BOOL SendEvent( GuiEvent_t *pstEvent_ );
00483
00488     K_BOOL ProcessEvent();
00489
00493     K_UCHAR GetEventCount() { return m_clMessageQueue.
GetCount(); }
00494
00498     GuiWindow *FindWindowByName( const K_CHAR *szName_ );
00499
00505     void InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT
usHeight_ );
00506
00507 private:
00514     void CopyEvent( GuiEvent_t *pstDst_, GuiEvent_t *pstSrc_ );
00515
00516 private:
00520     DoubleLinkedList m_clWindowList;
00521
00525     MessageQueue m_clMessageQueue;
00526 };
00527
00528 //-----
00538 class GuiControl : public LinkListNode
00539 {
00540 public:
00547     virtual void Init() = 0;
00548
00554     virtual void Draw() = 0;
00555
00563     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) = 0;
00564
00570     void SetTop( K_USHORT usTop_ ) { m_usTop = usTop_; }
00571
00577     void SetLeft( K_USHORT usLeft_ ) { m_usLeft = usLeft_; }
00578
00584     void SetHeight( K_USHORT usHeight_ ) { m_usHeight = usHeight_; }
00585
00591     void SetWidth( K_USHORT usWidth_ ) { m_usWidth = usWidth_; }
00592
00598     void SetZOrder( K_UCHAR ucZ_ ) { m_ucZOrder = ucZ_; }
00599
00606     void SetControlIndex( K_UCHAR ucIdx_ ) { m_ucControlIndex = ucIdx_; }
00607
00613     K_USHORT GetTop() { return m_usTop; }
00614
00620     K_USHORT GetLeft() { return m_usLeft; }
00621
00627     K_USHORT GetHeight() { return m_usHeight; }
00628
00634     K_USHORT GetWidth() { return m_usWidth; }
00635
00641     K_UCHAR GetZOrder() { return m_ucZOrder; }
00642
00648     K_UCHAR GetControlIndex() { return m_ucControlIndex; }
00649
00655     K_BOOL IsStale() { return m_bStale; }
00656
00668     void GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ );
00669
00677     K_BOOL IsInFocus()
00678     {
00679         return m_pclParentWindow->IsInFocus(this);
00680     }
00681
00689     virtual void Activate( bool bActivate_ ) = 0;
00690
00691 protected:
00692     friend class GuiWindow;
00693     friend class GuiEventSurface;
00694
00706     void SetParentControl( GuiControl *pclParent_ ) {
m_pclParentControl = pclParent_; }
00707
00717     void SetParentWindow( GuiWindow *pclWindow_ ) {
m_pclParentWindow = pclWindow_; }
00718
00725     GuiControl *GetParentControl() { return
m_pclParentControl; }
00726
00733     GuiWindow *GetParentWindow() { return
m_pclParentWindow; }
00734
00741     void ClearStale() { m_bStale = false; }

```

```

00742
00746     void SetStale()                                     { m_bStale = true; }
00747
00751     void SetAcceptFocus( bool bFocus_ )                 {
m_bAcceptsFocus = bFocus_; }
00752
00756     bool AcceptsFocus()                                 { return
m_bAcceptsFocus; }
00757 private:
00759     K_BOOL      m_bStale;
00760
00762     K_BOOL      m_bAcceptsFocus;
00763
00766     K_UCHAR     m_ucZOrder;
00767
00770     K_UCHAR     m_ucControlIndex;
00771
00773     K_USHORT    m_usTop;
00774
00776     K_USHORT    m_usLeft;
00777
00779     K_USHORT    m_usWidth;
00780
00782     K_USHORT    m_usHeight;
00783
00785     GuiControl *m_pclParentControl;
00786
00788     GuiWindow   *m_pclParentWindow;
00789 };
00790
00791 //-----
00796 class StubControl : public GuiControl
00797 {
00798 public:
00799     virtual void Init() { }
00800     virtual void Draw() { }
00801     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) { return
GUI_EVENT_OK; }
00802     virtual void Activate( bool bActivate_ ) { }
00803 };
00804
00805 #endif
00806

```

17.75 /home/mo/mark3-source/embedded/stage/src/kernel.cpp File Reference

[Kernel](#) initialization and startup code.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel.h"
#include "scheduler.h"
#include "thread.h"
#include "threadport.h"
#include "timerlist.h"
#include "message.h"
#include "driver.h"
#include "profile.h"
#include "kprofile.h"
#include "tracebuffer.h"
#include "kernel_debug.h"
#include "transaction.h"

```

Macros

- `#define __FILE_ID__ KERNEL_CPP`

17.75.1 Detailed Description

[Kernel](#) initialization and startup code.

Definition in file [kernel.cpp](#).

17.76 kernel.cpp

```

00001 /*=====
00002
00003
00004 | | | | | | | | | | | | | | | | | | | | | |
00005 | | | | | | | | | | | | | | | | | | | | | |
00006 | | | | | | | | | | | | | | | | | | | | | |
00007 | | | | | | | | | | | | | | | | | | | | | |
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "mark3cfg.h"
00023
00024 #include "kernel.h"
00025 #include "scheduler.h"
00026 #include "thread.h"
00027 #include "threadport.h"
00028 #include "timerlist.h"
00029 #include "message.h"
00030 #include "driver.h"
00031 #include "profile.h"
00032 #include "kprofile.h"
00033 #include "tracebuffer.h"
00034 #include "kernel_debug.h"
00035 #include "transaction.h"
00036
00037 bool Kernel::m_bIsStarted;
00038 bool Kernel::m_bIsPanic;
00039 panic_func_t Kernel::m_pfPanic;
00040
00041 //-----
00042 #if defined __FILE_ID__
00043 #undef __FILE_ID__
00044 #endif
00045 #define __FILE_ID__ KERNEL_CPP
00046
00047 //-----
00048 void Kernel::Init(void)
00049 {
00050     m_bIsStarted = false;
00051     m_bIsPanic = false;
00052     m_pfPanic = 0;
00053
00054 #if KERNEL_USE_DEBUG
00055     TraceBuffer::Init();
00056 #endif
00057     KERNEL_TRACE( STR_MARK3_INIT );
00058
00059     // Initialize the global kernel data - scheduler, timer-scheduler, and
00060     // the global message pool.
00061     Scheduler::Init();
00062 #if KERNEL_USE_DRIVER
00063     DriverList::Init();
00064 #endif
00065 #if KERNEL_USE_TIMERS
00066     TimerScheduler::Init();
00067 #endif
00068 #if KERNEL_USE_MESSAGE
00069     GlobalMessagePool::Init();
00070 #endif
00071 #if KERNEL_USE_PROFILER
00072     Profiler::Init();
00073 #endif
00074     TransactionQueue::GlobalQueueInit();
00075 }
00076
00077 //-----
00078 void Kernel::Start(void)
00079 {
00080     KERNEL_TRACE( STR_THREAD_START );
00081     m_bIsStarted = true;

```



```
00043 {
00044     public:
00053         static void Init(void);
00054
00067         static void Start(void);
00068
00074         static bool IsStarted()    {    return m_bIsStarted;    }
00075
00083         static void SetPanic( panic_func_t pfPanic_ ) { m_pfPanic = pfPanic_; }
00084
00089         static bool IsPanic()      {    return m_bIsPanic;    }
00090
00095         static void Panic(K_USHORT usCause_);
00096
00097     private:
00098         static bool m_bIsStarted;
00099         static bool m_bIsPanic;
00100         static panic_func_t m_pfPanic;
00101 };
00102
00103 #endif
00104
```

17.79 /home/mo/mark3-source/embedded/stage/src/kernel_debug.h File Reference

Macros and functions used for assertions, kernel traces, etc.

```
#include "debug_tokens.h"
#include "mark3cfg.h"
#include "tracebuffer.h"
#include "kernel.h"
```

Macros

- #define **__FILE_ID__** 0
- #define **KERNEL_TRACE(x)**
- #define **KERNEL_TRACE_1(x, arg1)**
- #define **KERNEL_TRACE_2(x, arg1, arg2)**
- #define **KERNEL_ASSERT(x)**

17.79.1 Detailed Description

Macros and functions used for assertions, kernel traces, etc.

Definition in file [kernel debug.h](#).

17.80 kernel_debug.h

```
00001 /*-----
00002
00003
00004 |-----|-----|-----|-----|-----|-----|
00005 | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / |
00006 | / \ | / \ | / \ | / \ | / \ | / \ | / \ | / \ |
00007 |-----|-----|-----|-----|-----|-----|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*
00020 #ifndef __KERNEL_DEBUG_H__
00021 #define __KERNEL_DEBUG_H__
00022
00023 #include "debug_tokens.h"
00024 #include "mark3cfg.h"
```

```

00025 #include "tracebuffer.h"
00026 #include "kernel.h"
00027
00028 //-----
00029 #if KERNEL_USE_DEBUG
00030
00031 //-----
00032 #define __FILE_ID__          STR_UNDEFINED
00033
00034 //-----
00035 #define KERNEL_TRACE( x )    \
00036 {                               \
00037     K_USHORT ausMsg__[5]; \
00038     ausMsg__[0] = 0xACDC; \
00039     ausMsg__[1] = __FILE_ID__; \
00040     ausMsg__[2] = __LINE__; \
00041     ausMsg__[3] = TraceBuffer::Increment(); \
00042     ausMsg__[4] = (K_USHORT)(x); \
00043     TraceBuffer::Write(ausMsg__, 5); \
00044 };
00045
00046 //-----
00047 #define KERNEL_TRACE_1( x, arg1 ) \
00048 {                               \
00049     K_USHORT ausMsg__[6]; \
00050     ausMsg__[0] = 0xACDC; \
00051     ausMsg__[1] = __FILE_ID__; \
00052     ausMsg__[2] = __LINE__; \
00053     ausMsg__[3] = TraceBuffer::Increment(); \
00054     ausMsg__[4] = (K_USHORT)(x); \
00055     ausMsg__[5] = arg1; \
00056     TraceBuffer::Write(ausMsg__, 6); \
00057 }
00058
00059 //-----
00060 #define KERNEL_TRACE_2( x, arg1, arg2 ) \
00061 {                               \
00062     K_USHORT ausMsg__[7]; \
00063     ausMsg__[0] = 0xACDC; \
00064     ausMsg__[1] = __FILE_ID__; \
00065     ausMsg__[2] = __LINE__; \
00066     ausMsg__[3] = TraceBuffer::Increment(); \
00067     ausMsg__[4] = (K_USHORT)(x); \
00068     ausMsg__[5] = arg1; \
00069     ausMsg__[6] = arg2; \
00070     TraceBuffer::Write(ausMsg__, 7); \
00071 }
00072
00073 //-----
00074 #define KERNEL_ASSERT( x ) \
00075 {                               \
00076     if( ( x ) == false ) \
00077     {                               \
00078         K_USHORT ausMsg__[5]; \
00079         ausMsg__[0] = 0xACDC; \
00080         ausMsg__[1] = __FILE_ID__; \
00081         ausMsg__[2] = __LINE__; \
00082         ausMsg__[3] = TraceBuffer::Increment(); \
00083         ausMsg__[4] = STR_ASSERT_FAILED; \
00084         TraceBuffer::Write(ausMsg__, 5); \
00085         Kernel::Panic(PANIC_ASSERT_FAILED); \
00086     } \
00087 }
00088
00089 #else
00090 //-----
00091 #define __FILE_ID__          0
00092 //-----
00093 #define KERNEL_TRACE( x )
00094 //-----
00095 #define KERNEL_TRACE_1( x, arg1 )
00096 //-----
00097 #define KERNEL_TRACE_2( x, arg1, arg2 )
00098 //-----
00099 #define KERNEL_ASSERT( x )
00100
00101 #endif // KERNEL_USE_DEBUG
00102
00103 #endif

```

17.81 /home/mo/mark3-source/embedded/stage/src/kernelswi.cpp File Reference

[Kernel](#) Software interrupt implementation for ATmega328p.

```
#include "kerneltypes.h"
#include "kernelswi.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

17.81.1 Detailed Description

Kernel Software interrupt implementation for ATmega328p.

Definition in file [kernelswi.cpp](#).

17.82 kernelswi.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "kernelswi.h"
00024
00025 #include <avr/io.h>
00026 #include <avr/interrupt.h>
00027
00028 //-----
00029 void KernelSWI::Config(void)
00030 {
00031     PORTD &= ~0x04; // Clear INT0
00032     DDRD |= 0x04;    // Set PortD, bit 2 (INT0) As Output
00033     EICRA |= (1 << ISC00) | (1 << ISC01); // Rising edge on INT0
00034 }
00035
00036 //-----
00037 void KernelSWI::Start(void)
00038 {
00039     EIFR &= ~(1 << INTF0); // Clear any pending interrupts on INT0
00040     EIMSK |= (1 << INT0);   // Enable INT0 interrupt (as K_LONG as I-bit is set)
00041 }
00042
00043 //-----
00044 void KernelSWI::Stop(void)
00045 {
00046     EIMSK &= ~(1 << INT0); // Disable INT0 interrupts
00047 }
00048
00049 //-----
00050 K_UCHAR KernelSWI::DI()
00051 {
00052     K_UCHAR bEnabled = ((EIMSK & (1 << INT0)) != 0);
00053     EIMSK &= ~(1 << INT0);
00054     return bEnabled;
00055 }
00056
00057 //-----
00058 void KernelSWI::RI(K_UCHAR bEnable_)
00059 {
00060     if (bEnable_)
00061     {
00062         EIMSK |= (1 << INT0);
00063     }
00064     else
00065     {
00066         EIMSK &= ~(1 << INT0);
00067     }
00068 }
00069
00070 //-----
00071 void KernelSWI::Clear(void)
```



```
00090
00091 #endif // __KERNELSIW_H_
```

17.85 /home/mo/mark3-source/embedded/stage/src/kerneltimer.cpp File Reference

[Kernel Timer](#) Implementation for ATmega328p.

```
#include "kerneltypes.h"
#include "kerneltimer.h"
#include "mark3cfg.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Macros

- `#define TCCR1B_INIT ((1 << WGM12) | (1 << CS12))`
- `#define TIMER_IMSK (1 << OCIE1A)`
- `#define TIMER_IFR (1 << OCF1A)`

17.85.1 Detailed Description

[Kernel Timer](#) Implementation for ATmega328p.

Definition in file [kerneltimer.cpp](#).

17.86 kerneltimer.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "kerneltimer.h"
00023 #include "mark3cfg.h"
00024
00025 #include <avr/io.h>
00026 #include <avr/interrupt.h>
00027
00028 #define TCCR1B_INIT      ((1 << WGM12) | (1 << CS12))
00029 #define TIMER_IMSK      (1 << OCIE1A)
00030 #define TIMER_IFR       (1 << OCF1A)
00031
00032 //-----
00033 void KernelTimer::Config(void)
00034 {
00035     TCCR1B = TCCR1B_INIT;
00036 }
00037
00038 //-----
00039 void KernelTimer::Start(void)
00040 {
00041     #if !KERNEL_TIMERS_TICKLESS
00042         TCCR1B = ((1 << WGM12) | (1 << CS11) | (1 << CS10));
00043         OCR1A = ((SYSTEM_FREQ / 1000) / 64);
00044     #else
00045         TCCR1B |= (1 << CS12);
00046     #endif
00047 }
```

```

00048     TCNT1 = 0;
00049     TIFR1 &= ~TIMER_IFR;
00050     TIMSK1 |= TIMER_IMSK;
00051 }
00052
00053 //-----
00054 void KernelTimer::Stop(void)
00055 {
00056     #if KERNEL_TIMERS_TICKLESS
00057         TIFR1 &= ~TIMER_IFR;
00058         TIMSK1 &= ~TIMER_IMSK;
00059         TCCR1B &= ~(1 << CS12);    // Disable count...
00060         TCNT1 = 0;
00061         OCR1A = 0;
00062     #endif
00063 }
00064
00065 //-----
00066 K_USHORT KernelTimer::Read(void)
00067 {
00068     #if KERNEL_TIMERS_TICKLESS
00069         volatile K_USHORT usRead1;
00070         volatile K_USHORT usRead2;
00071
00072         do {
00073             usRead1 = TCNT1;
00074             usRead2 = TCNT1;
00075         } while (usRead1 != usRead2);
00076
00077         return usRead1;
00078     #else
00079         return 0;
00080     #endif
00081 }
00082
00083 //-----
00084 K_ULONG KernelTimer::SubtractExpiry(K_ULONG ulInterval_)
00085 {
00086     #if KERNEL_TIMERS_TICKLESS
00087         OCR1A -= (K_USHORT)ulInterval_;
00088         return (K_ULONG)OCR1A;
00089     #else
00090         return 0;
00091     #endif
00092 }
00093
00094 //-----
00095 K_ULONG KernelTimer::TimeToExpiry(void)
00096 {
00097     #if KERNEL_TIMERS_TICKLESS
00098         K_USHORT usRead = KernelTimer::Read();
00099         K_USHORT usOCR1A = OCR1A;
00100
00101         if (usRead >= usOCR1A)
00102         {
00103             return 0;
00104         }
00105         else
00106         {
00107             return (K_ULONG)(usOCR1A - usRead);
00108         }
00109     #else
00110         return 0;
00111     #endif
00112 }
00113
00114 //-----
00115 K_ULONG KernelTimer::GetOvertime(void)
00116 {
00117     return KernelTimer::Read();
00118 }
00119
00120 //-----
00121 K_ULONG KernelTimer::SetExpiry(K_ULONG ulInterval_)
00122 {
00123     #if KERNEL_TIMERS_TICKLESS
00124         K_USHORT usSetInterval;
00125         if (ulInterval_ > 65535)
00126         {
00127             usSetInterval = 65535;
00128         }
00129         else
00130         {
00131             usSetInterval = (K_USHORT)ulInterval_ ;
00132         }
00133         OCR1A = usSetInterval;
00134         return (K_ULONG)usSetInterval;

```



```

00135 #else
00136     return 0;
00137 #endif
00138 }
00139
00140 //-----
00141 void KernelTimer::ClearExpiry(void)
00142 {
00143     #if KERNEL_TIMERS_TICKLESS
00144         OCR1A = 65535;           // Clear the compare value
00145     #endif
00146 }
00147
00148 //-----
00149 K_UCHAR KernelTimer::DI(void)
00150 {
00151     #if KERNEL_TIMERS_TICKLESS
00152         K_UCHAR bEnabled = ((TIMSK1 & (TIMER_IMSK)) != 0);
00153         TIFR1 &= ~TIMER_IFR;    // Clear interrupt flags
00154         TIMSK1 &= ~TIMER_IMSK;  // Disable interrupt
00155         return bEnabled;
00156     #else
00157         return 0;
00158     #endif
00159 }
00160
00161 //-----
00162 void KernelTimer::EI(void)
00163 {
00164     KernelTimer::RI(0);
00165 }
00166
00167 //-----
00168 void KernelTimer::RI(K_UCHAR bEnable_)
00169 {
00170     #if KERNEL_TIMERS_TICKLESS
00171         if (bEnable_)
00172         {
00173             TIMSK1 |= (1 << OCIE1A);    // Enable interrupt
00174         }
00175         else
00176         {
00177             TIMSK1 &= ~(1 << OCIE1A);
00178         }
00179     #endif
00180 }

```

17.87 /home/mo/mark3-source/embedded/stage/src/kerneltimer.h File Reference

[Kernel Timer](#) Class declaration.

```
#include "kerneltypes.h"
```

Classes

- class [KernelTimer](#)
Hardware timer interface, used by all scheduling/timer subsystems.

Macros

- #define **SYSTEM_FREQ** ((K_ULONG)16000000)
- #define **TIMER_FREQ** ((K_ULONG)(SYSTEM_FREQ / 256))

17.87.1 Detailed Description

[Kernel Timer](#) Class declaration.

Definition in file [kerneltimer.h](#).

17.88 kerneltimer.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #ifndef __KERNELTIMER_H_
00023 #define __KERNELTIMER_H_
00024
00025 //-----
00026 #define SYSTEM_FREQ ((K_ULONG)16000000)
00027 #define TIMER_FREQ ((K_ULONG)(SYSTEM_FREQ / 256)) // Timer ticks per second...
00028
00029 //-----
00033 class KernelTimer
00034 {
00035 public:
00041     static void Config(void);
00042
00048     static void Start(void);
00049
00055     static void Stop(void);
00056
00062     static K_UCHAR DI(void);
00063
00071     static void RI(K_UCHAR bEnable_);
00072
00078     static void EI(void);
00079
00090     static K_ULONG SubtractExpiry(K_ULONG ulInterval_);
00091
00100     static K_ULONG TimeToExpiry(void);
00101
00110     static K_ULONG SetExpiry(K_ULONG ulInterval_);
00111
00120     static K_ULONG GetOvertime(void);
00121
00127     static void ClearExpiry(void);
00128
00129 private:
00137     static K_USHORT Read(void);
00138
00139 };
00140
00141 #endif //__KERNELTIMER_H_

```

17.89 /home/mo/mark3-source/embedded/stage/src/kerneltypes.h File Reference

Basic data type primitives used throughout the OS.

```
#include <stdint.h>
```

Macros

- #define **K_BOOL** uint8_t
- #define **K_CHAR** char
- #define **K_UCHAR** uint8_t
- #define **K_USHORT** uint16_t
- #define **K_SHORT** int16_t
- #define **K_ULONG** uint32_t
- #define **K_LONG** int32_t
- #define **K_ADDR** uint32_t
- #define **K_WORD** uint32_t

Typedefs

- typedef void(* **panic_func_t**)(K_USHORT usPanicCode_)

Enumerations

- enum **EventFlagOperation_t** {
EVENT_FLAG_ALL, **EVENT_FLAG_ANY**, **EVENT_FLAG_ALL_CLEAR**, **EVENT_FLAG_ANY_CLEAR**,
EVENT_FLAG_MODES, **EVENT_FLAG_PENDING_UNBLOCK** }

17.89.1 Detailed Description

Basic data type primitives used throughout the OS.

Definition in file [kerneltypes.h](#).

17.90 kerneltypes.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include <stdint.h>
00020
00021 #ifndef __KERNELTYPES_H__
00022 #define __KERNELTYPES_H__
00023
00024 #if defined(bool)
00025     #define K_BOOL          bool
00026 #else
00027     #define K_BOOL          uint8_t
00028 #endif
00029
00030 #define K_CHAR              char
00031 #define K_UCHAR             uint8_t
00032 #define K_USHORT            uint16_t
00033 #define K_SHORT             int16_t
00034 #define K_ULONG             uint32_t
00035 #define K_LONG              int32_t
00036
00037 #if !defined(K_ADDR)
00038     #define K_ADDR          uint32_t
00039 #endif
00040 #if !defined(K_WORD)
00041     #define K_WORD          uint32_t
00042 #endif
00043
00044 //-----
00045 typedef void (*panic_func_t) ( K_USHORT usPanicCode_ );
00046
00047 //-----
00048 typedef enum
00049 {
00050     EVENT_FLAG_ALL,
00051     EVENT_FLAG_ANY,
00052     EVENT_FLAG_ALL_CLEAR,
00053     EVENT_FLAG_ANY_CLEAR,
00054     EVENT_FLAG_MODES,
00055     EVENT_FLAG_PENDING_UNBLOCK
00056 } EventFlagOperation_t;
00057
00058
00059 #endif

```



```
00012 See license.txt for more information
00013 =====*/
00020 #ifndef __KEYCODES_H_
00021 #define __KEYCODES_H_
00022
00023 #include "kerneltypes.h"
00024
00025 typedef enum
00026 {
00027     KEYCODE_LBUTTON = 0x01,
00028     KEYCODE_RBUTTON,
00029     KEYCODE_CANCEL,
00030     KEYCODE_MBUTTON,
00031     KEYCODE_BACK = 0x08,
00032     KEYCODE_TAB,
00033     KEYCODE_CLEAR = 0x0C,
00034     KEYCODE_RETURN,
00035     KEYCODE_SHIFT = 0x10,
00036     KEYCODE_CONTROL,
00037     KEYCODE_MENU,
00038     KEYCODE_PAUSE,
00039     KEYCODE_CAPITAL,
00040     KEYCODE_ESCAPE = 0x1B,
00041     KEYCODE_SPACE,
00042     KEYCODE_PRIOR,
00043     KEYCODE_NEXT,
00044     KEYCODE_END,
00045     KEYCODE_HOME,
00046     KEYCODE_LEFT,
00047     KEYCODE_UP,
00048     KEYCODE_RIGHT,
00049     KEYCODE_DOWN,
00050     KEYCODE_SELECT,
00051     KEYCODE_PRINT,
00052     KEYCODE_EXECUTE,
00053     KEYCODE_SNAPSHOT,
00054     KEYCODE_INSERT,
00055     KEYCODE_DELETE,
00056     KEYCODE_HELP = 0x2F,
00057     KEYCODE_0,
00058     KEYCODE_1,
00059     KEYCODE_2,
00060     KEYCODE_3,
00061     KEYCODE_4,
00062     KEYCODE_5,
00063     KEYCODE_6,
00064     KEYCODE_7,
00065     KEYCODE_8,
00066     KEYCODE_9,
00067     KEYCODE_A,
00068     KEYCODE_B,
00069     KEYCODE_C,
00070     KEYCODE_D,
00071     KEYCODE_E,
00072     KEYCODE_F,
00073     KEYCODE_G,
00074     KEYCODE_H,
00075     KEYCODE_I,
00076     KEYCODE_J,
00077     KEYCODE_K,
00078     KEYCODE_L,
00079     KEYCODE_M,
00080     KEYCODE_N,
00081     KEYCODE_O,
00082     KEYCODE_P,
00083     KEYCODE_Q,
00084     KEYCODE_R,
00085     KEYCODE_S,
00086     KEYCODE_T,
00087     KEYCODE_U,
00088     KEYCODE_V,
00089     KEYCODE_W,
00090     KEYCODE_X,
00091     KEYCODE_Y,
00092     KEYCODE_Z,
00093     KEYCODE_NUMPAD0 = 0x60,
00094     KEYCODE_NUMPAD1,
00095     KEYCODE_NUMPAD2,
00096     KEYCODE_NUMPAD3,
00097     KEYCODE_NUMPAD4,
00098     KEYCODE_NUMPAD5,
00099     KEYCODE_NUMPAD6,
00100     KEYCODE_NUMPAD7,
00101     KEYCODE_NUMPAD8,
00102     KEYCODE_NUMPAD9,
00103     KEYCODE_SEPARATOR = 0x6C,
00104     KEYCODE_SUBTRACT,
```


17.95 /home/mo/mark3-source/embedded/stage/src/kprofile.h File Reference

Generated on Sat Jan 25 2014 18:56:55 for Mark3 Realtime Kernel by Doxygen

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
```

Classes

- class [Profiler](#)
System profiling timer interface.

Macros

- `#define TICKS_PER_OVERFLOW (256)`
- `#define CLOCK_DIVIDE (8)`

17.95.1 Detailed Description

Profiling timer hardware interface.

Definition in file [kprofile.h](#).

17.96 kprofile.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "kerneltypes.h"
00021 #include "mark3cfg.h"
00022 #include "ll.h"
00023
00024 #ifndef __KPROFILE_H__
00025 #define __KPROFILE_H__
00026
00027 #if KERNEL_USE_PROFILER
00028
00029 //-----
00030 #define TICKS_PER_OVERFLOW (256)
00031 #define CLOCK_DIVIDE (8)
00032
00033 //-----
00037 class Profiler
00038 {
00039 public:
00046     static void Init();
00047
00053     static void Start();
00054
00060     static void Stop();
00061
00067     static K_USHORT Read();
00068
00072     static void Process();
00073
00077     static K_ULONG GetEpoch() { return m_ulEpoch; }
00078 private:
00079     static K_ULONG m_ulEpoch;
00081 };
00082
00083 #endif //KERNEL_USE_PROFILER
```


17.97 /home/mo/mark3-source/embedded/stage/src/ksemaphore.cpp File Reference

Generated on Sat Jan 25 2014 18:56:55 for Mark3 Realtime Kernel by Doxygen

```

00033     #undef __FILE_ID__
00034 #endif
00035 #define __FILE_ID__      SEMAPHORE_CPP
00036
00037 #if KERNEL_USE_SEMAPHORE
00038
00039 //-----
00040 #define SEMAPHORE_TRANSACTION_POST      (0)
00041 #define SEMAPHORE_TRANSACTION_PEND      (1)
00042 #define SEMAPHORE_TRANSACTION_UNBLOCK   (2)
00043
00044 #if KERNEL_USE_TIMERS
00045 #include "timerlist.h"
00046
00047 //-----
00048 void TimedSemaphore_Callback(Thread *pclOwner_, void *pvData_)
00049 {
00050     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_);
00051
00052     // Indicate that the semaphore has expired on the thread
00053     pclOwner_->SetExpired(true);
00054
00055     // Wake up the thread that was blocked on this semaphore.
00056     pclSemaphore->Timeout(pclOwner_);
00057 }
00058
00059 //-----
00060 void Semaphore::Timeout(Thread *pclChosenOne_)
00061 {
00062     K_BOOL bSchedState;
00063     if (LockAndQueue(SEMAPHORE_TRANSACTION_UNBLOCK, pclChosenOne_, &bSchedState))
00064     {
00065         return;
00066     }
00067
00068     if (ProcessQueue()) {
00069         Thread::Yield();
00070     }
00071
00072     Scheduler::SetScheduler(bSchedState);
00073 }
00074
00075 #endif // KERNEL_USE_TIMERS
00076
00077 //-----
00078 K_BOOL Semaphore::ProcessQueue()
00079 {
00080     Transaction *pclTRX;
00081     K_BOOL bReschedule = false;
00082
00083     do
00084     {
00085         pclTRX = m_clKTQ.Dequeue();
00086         KERNEL_ASSERT(pclTRX);
00087
00088         switch (pclTRX->GetCode())
00089         {
00090             case SEMAPHORE_TRANSACTION_POST:
00091                 PostTransaction(pclTRX, &bReschedule);
00092                 break;
00093             case SEMAPHORE_TRANSACTION_PEND:
00094                 PendTransaction(pclTRX, &bReschedule);
00095                 break;
00096             case SEMAPHORE_TRANSACTION_UNBLOCK:
00097                 TimeoutTransaction(pclTRX, &bReschedule);
00098                 break;
00099             default:
00100                 break;
00101         }
00102         m_clKTQ.Finish(pclTRX);
00103     } while (Unlock() > 1);
00104
00105     return bReschedule;
00106 }
00107
00108 //-----
00109 void Semaphore::PostTransaction(Transaction *pclTRX_, K_BOOL *
pbReschedule_)
00110 {
00111     // If nothing is waiting for the semaphore
00112     if (m_clBlockList.GetHead() == NULL)
00113     {
00114         // Check so see if we've reached the maximum value in the semaphore
00115         if (m_usValue < m_usMaxValue)
00116         {
00117             // Increment the count value
00118             m_usValue++;

```

```

00119     }
00120 }
00121 else
00122 {
00123     // Otherwise, there are threads waiting for the semaphore to be
00124     // posted, so wake the next one (highest priority goes first).
00125     *pbReschedule_ = WakeNext();
00126 }
00127 }
00128
00129 //-----
00130 void Semaphore::PendTransaction(Transaction *pclTRX_, K_BOOL *
pbReschedule_)
00131 {
00132     // Decrement-and-set the semaphore value
00133     if (0 == m_usValue)
00134     {
00135         // The semaphore count is zero - we need to block the current thread
00136         // and wait until the semaphore is posted from elsewhere.
00137         *pbReschedule_ = true;
00138
00139         // Get the current thread pointer.
00140         Thread *pclThread = static_cast<Thread*>(pclTRX_->GetData());
00141
00142 #if KERNEL_USE_TIMERS
00143         Timer *pclSemTimer = pclThread->GetTimer();
00144         pclThread->SetExpired(false);
00145         K_ULONG ulWaitTimeMS = pclSemTimer->GetInterval();
00146
00147         if (ulWaitTimeMS)
00148         {
00149             pclSemTimer->Start(0, ulWaitTimeMS, TimedSemaphore_Callback, (void*)this);
00150         }
00151 #endif
00152         Block(pclThread);
00153     }
00154     else
00155     {
00156         m_usValue--;
00157     }
00158 }
00159
00160 //-----
00161 void Semaphore::TimeoutTransaction(Transaction *pclTRX_, K_BOOL *
pbReschedule_)
00162 {
00163     Thread *pclChosenOne = static_cast<Thread*>(pclTRX_->GetData());
00164
00165     Unblock(pclChosenOne);
00166
00167     // Call a task switch only if higher priority thread
00168     if (pclChosenOne->GetPriority() > Scheduler::GetCurrentThread()->
GetPriority())
00169     {
00170         *pbReschedule_ = true;
00171     }
00172 }
00173
00174 //-----
00175 K_BOOL Semaphore::WakeNext()
00176 {
00177     Thread *pclChosenOne;
00178
00179     pclChosenOne = m_clBlockList.HighestWaiter();
00180
00181     // Remove from the semaphore waitlist and back to its ready list.
00182     Unblock(pclChosenOne);
00183
00184     // Call a task switch only if higher priority thread
00185     if (pclChosenOne->GetPriority() > Scheduler::GetCurrentThread()->
GetPriority())
00186     {
00187         return true;
00188     }
00189
00190     return false;
00191 }
00192
00193 //-----
00194 void Semaphore::Init(K_USHORT usInitVal_, K_USHORT usMaxVal_)
00195 {
00196     // Copy the paramters into the object - set the maximum value for this
00197     // semaphore to implement either binary or counting semaphores, and set
00198     // the initial count. Clear the wait list for this object.
00199     m_usValue = usInitVal_;
00200     m_usMaxValue = usMaxVal_;
00201 }

```

```

00202     m_clBlockList.Init();
00203 }
00204
00205 //-----
00206 void Semaphore::Post()
00207 {
00208     KERNEL_TRACE_1( STR_SEMAPHORE_POST_1, (K_USHORT)Scheduler::GetCurrentThread(
00209 )->GetID() );
00210
00211     K_BOOL bSchedState;
00212     if (LockAndQueue(SEMAPHORE_TRANSACTION_POST, 0, &bSchedState))
00213     {
00214         return;
00215     }
00216
00217     if (ProcessQueue()) {
00218         Thread::Yield();
00219     }
00220
00221     Scheduler::SetScheduler(bSchedState);
00222
00223     return;
00224 }
00225
00226 #if !KERNEL_USE_TIMERS
00227 //-----
00228 // No timers, no timed pend
00229 void Semaphore::Pend()
00230 #else
00231 //-----
00232 // Redirect the untimed pend API to the timed pend, with a null timeout.
00233 void Semaphore::Pend()
00234 {
00235     Pend(0);
00236 }
00237 //-----
00238 bool Semaphore::Pend( K_ULONG ulWaitTimeMS_ )
00239 #endif
00240 {
00241     KERNEL_TRACE_1( STR_SEMAPHORE_PEND_1, (K_USHORT)Scheduler::GetCurrentThread(
00242 )->GetID() );
00243
00244     // By locking the queue, we ensure that any post/unblock operations on this
00245     // semaphore that interrupt our normal execution wind up being queued flushed
00246     // before we exit.
00247
00248     K_BOOL bSchedState;
00249     if (LockAndQueue(SEMAPHORE_TRANSACTION_PEND, (void*)
00250 Scheduler::GetCurrentThread(), &bSchedState))
00251     {
00252         // This should never happen - kernel panic if we do.
00253         Kernel::Panic( PANIC_PEND_LOCK_VIOLATION );
00254     }
00255
00256     // Set data on the current thread that needs to be passed into the transaction
00257     // handler (and can't be queued in the simple key-value pair in the transaction
00258     // object)
00259
00260     #if KERNEL_USE_TIMERS
00261         // Pre-set the interval, since we can't cache it in the transaction
00262         Scheduler::GetCurrentThread()->GetTimer()->
00263         SetIntervalTicks(ulWaitTimeMS_);
00264         Scheduler::GetCurrentThread()->SetExpired(false);
00265     #endif
00266
00267     if (ProcessQueue())
00268     {
00269         // Switch Threads immediately
00270         Thread::Yield();
00271     }
00272
00273     Scheduler::SetScheduler(bSchedState);
00274
00275     #if KERNEL_USE_TIMERS
00276         if (ulWaitTimeMS_)
00277         {
00278             Scheduler::GetCurrentThread()->GetTimer()->
00279             Stop();
00280         }
00281         K_BOOL retVal = (Scheduler::GetCurrentThread()->GetExpired() == false);
00282         return retVal;
00283     #endif
00284 }
00285
00286 //-----
00287 K_USHORT Semaphore::GetCount()

```

17.99 /home/mo/mark3-source/embedded/stage/src/ksemaphore.h File Reference

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "threadlist.h"
#include "transaction.h"
#include "atomic.h"
```

Classes

- class `Semaphore`
Counting semaphore, based on `BlockingObject` base class.

17.99.1 Detailed Description

Semaphore Blocking Object class declarations.

Definition in file [ksemaphore.h](#).

17.100 ksemaphore.h

```

00001 /*
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*
00022 #ifndef __KSEMAPHORE_H__
00023 #define __KSEMAPHORE_H__
00024
00025 #include "kerneltypes.h"
00026 #include "mark3cfg.h"
00027
00028 #include "blocking.h"
00029 #include "threadlist.h"
00030 #include "transaction.h"
00031 #include "atomic.h"
00032
00033 #if KERNEL_USE_SEMAPHORE
00034
00035 //-----
00039 class Semaphore : public BlockingObject
00040 {
00041 public:
00051     void Init(K_USHORT usInitVal_, K_USHORT usMaxVal_);
00052
00061     void Post();

```



```

00023 #include "kernel.h"
00024 #include "ll.h"
00025 #include "kernel_debug.h"
00026
00027 //-----
00028 #if defined __FILE_ID__
00029     #undef __FILE_ID__
00030 #endif
00031 #define __FILE_ID__      LL_CPP
00032
00033 //-----
00034 void LinkListNode::ClearNode()
00035 {
00036     next = NULL;
00037     prev = NULL;
00038 }
00039
00040 //-----
00041 void DoubleLinkedList::Add(LinkListNode *node_)
00042 {
00043     KERNEL_ASSERT( node_ );
00044
00045     // Add a node to the end of the linked list.
00046     if (!m_pstHead)
00047     {
00048         // If the list is empty, initilize the nodes
00049         m_pstHead = node_;
00050         m_pstTail = node_;
00051
00052         m_pstHead->prev = NULL;
00053         m_pstTail->next = NULL;
00054         return;
00055     }
00056
00057     // Move the tail node, and assign it to the new node just passed in
00058     m_pstTail->next = node_;
00059     node_->prev = m_pstTail;
00060     node_->next = NULL;
00061     m_pstTail = node_;
00062 }
00063
00064 //-----
00065 void DoubleLinkedList::Remove(LinkListNode *node_)
00066 {
00067     KERNEL_ASSERT( node_ );
00068
00069     if (node_->prev)
00070     {
00071         #if SAFE_UNLINK
00072             if (node_->prev->next != node_)
00073             {
00074                 Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00075             }
00076         #endif
00077         node_->prev->next = node_->next;
00078     }
00079     if (node_->next)
00080     {
00081         #if SAFE_UNLINK
00082             if (node_->next->prev != node_)
00083             {
00084                 Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00085             }
00086         #endif
00087         node_->next->prev = node_->prev;
00088     }
00089     if (node_ == m_pstHead)
00090     {
00091         m_pstHead = node_->next;
00092     }
00093     if (node_ == m_pstTail)
00094     {
00095         m_pstTail = node_->prev;
00096     }
00097     node_->ClearNode();
00098 }
00099
00100 //-----
00101 void CircularLinkedList::Add(LinkListNode *node_)
00102 {
00103     KERNEL_ASSERT( node_ );
00104
00105     // Add a node to the end of the linked list.
00106     if (!m_pstHead)
00107     {
00108         // If the list is empty, initilize the nodes

```

```

00110         m_pstHead = node_;
00111         m_pstTail = node_;
00112
00113         m_pstHead->prev = m_pstHead;
00114         m_pstHead->next = m_pstHead;
00115         return;
00116     }
00117
00118     // Move the tail node, and assign it to the new node just passed in
00119     m_pstTail->next = node_;
00120     node_->prev = m_pstTail;
00121     node_->next = m_pstHead;
00122     m_pstTail = node_;
00123     m_pstHead->prev = node_;
00124 }
00125
00126 //-----
00127 void CircularLinkedList::Remove(LinkListNode *node_)
00128 {
00129     KERNEL_ASSERT( node_ );
00130
00131     // Check to see if this is the head of the list...
00132     if ((node_ == m_pstHead) && (m_pstHead == m_pstTail))
00133     {
00134         // Clear the head and tail pointers - nothing else left.
00135         m_pstHead = NULL;
00136         m_pstTail = NULL;
00137         return;
00138     }
00139
00140 #if SAFE_UNLINK
00141     // Verify that all nodes are properly connected
00142     if ((node_->prev->next != node_) || (node_->next->prev != node_))
00143     {
00144         Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00145     }
00146 #endif
00147
00148     // This is a circularly linked list - no need to check for connection,
00149     // just remove the node.
00150     node_->next->prev = node_->prev;
00151     node_->prev->next = node_->next;
00152
00153     if (node_ == m_pstHead)
00154     {
00155         m_pstHead = m_pstHead->next;
00156     }
00157     if (node_ == m_pstTail)
00158     {
00159         m_pstTail = m_pstTail->prev;
00160     }
00161     node_->ClearNode();
00162 }
00163
00164 //-----
00165 void CircularLinkedList::PivotForward()
00166 {
00167     if (m_pstHead)
00168     {
00169         m_pstHead = m_pstHead->next;
00170         m_pstTail = m_pstTail->next;
00171     }
00172 }
00173
00174 //-----
00175 void CircularLinkedList::PivotBackward()
00176 {
00177     if (m_pstHead)
00178     {
00179         m_pstHead = m_pstHead->prev;
00180         m_pstTail = m_pstTail->prev;
00181     }
00182 }

```

17.103 /home/mo/mark3-source/embedded/stage/src/ll.h File Reference

Core linked-list declarations, used by all kernel list types.

```

#include "kerneltypes.h"
#include "mark3cfg.h"

```


Classes

- class [LinkedListNode](#)
Basic linked-list node data structure.
- class [LinkedList](#)
Abstract-data-type from which all other linked-lists are derived.
- class [DoubleLinkedList](#)
Doubly-linked-list data type, inherited from the base [LinkedList](#) type.
- class [CircularLinkedList](#)
Circular-linked-list data type, inherited from the base [LinkedList](#) type.

Macros

- `#define NULL (0)`

17.103.1 Detailed Description

Core linked-list declarations, used by all kernel list types. At the heart of RTOS data structures are linked lists. Having a robust and efficient set of linked-list types that we can use as a foundation for building the rest of our kernel types allows us to keep our RTOS code efficient and logically-separated.

So what data types rely on these linked-list classes?

-Threads -ThreadLists -The [Scheduler](#) -Timers, -The [Timer Scheduler](#) -Blocking objects (Semaphores, Mutexes, etc...)

Pretty much everything in the kernel uses these linked lists. By having objects inherit from the base linked-list node type, we're able to leverage the double and circular linked-list classes to manager virtually every object type in the system without duplicating code. These functions are very efficient as well, allowing for very deterministic behavior in our code.

Definition in file [ll.h](#).

17.104 ll.h

```

00001  /*=====
00002
00003  _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00006  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00007  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00008  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00043  #ifndef __LL_H__
00044  #define __LL_H__
00045
00046  #include "kerneltypes.h"
00047  #include "mark3cfg.h"
00048
00049  //-----
00050  #ifndef NULL
00051  #define NULL (0)
00052  #endif
00053
00054  //-----
00060  class LinkedList;
00061  class DoubleLinkedList;
00062  class CircularLinkedList;
00063
00064  //-----
00069  class LinkedListNode

```

```

00070 {
00071     protected:
00072
00073         LinkListNode *next;
00074         LinkListNode *prev;
00075
00076         LinkListNode() { ClearNode(); }
00077
00083         void ClearNode();
00084
00085     public:
00093         LinkListNode *GetNext(void) { return next; }
00094
00102         LinkListNode *GetPrev(void) { return prev; }
00103
00104         friend class LinkList;
00105         friend class DoubleLinkList;
00106         friend class CircularLinkList;
00107 };
00108
00109 //-----
00113 class LinkList
00114 {
00115     protected:
00116         LinkListNode *m_pstHead;
00117         LinkListNode *m_pstTail;
00118
00119     public:
00123         void Init() { m_pstHead = NULL; m_pstTail = NULL; }
00124
00132         virtual void Add(LinkListNode *node_) = 0;
00133
00141         virtual void Remove(LinkListNode *node_) = 0;
00142
00150         LinkListNode *GetHead() { return m_pstHead; }
00151
00159         LinkListNode *GetTail() { return m_pstTail; }
00160 };
00161
00162 //-----
00166 class DoubleLinkList : public LinkList
00167 {
00168     public:
00172         DoubleLinkList() { m_pstHead = NULL; m_pstTail = NULL; }
00173
00181         virtual void Add(LinkListNode *node_);
00182
00190         virtual void Remove(LinkListNode *node_);
00191 };
00192
00193 //-----
00197 class CircularLinkList : public LinkList
00198 {
00199     public:
00200         CircularLinkList() { m_pstHead = NULL; m_pstTail = NULL; }
00201
00209         virtual void Add(LinkListNode *node_);
00210
00218         virtual void Remove(LinkListNode *node_);
00219
00226         void PivotForward();
00227
00234         void PivotBackward();
00235 };
00236
00237 #endif

```

17.105 /home/mo/mark3-source/embedded/stage/src/manual.h File Reference

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

17.105.1 Detailed Description

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

Definition in file [manual.h](#).

17.106 manual.h

```

00001 /*=====
00002
00003      |-----|-----|-----|-----|-----|
00004      |  \    /  |  \    /  |  \    /  |  \    /  |  \    /  |
00005      |   \  /   |   \  /   |   \  /   |   \  /   |   \  /   |
00006      |__/\___/|__/\___/|__/\___/|__/\___/|__/\___/|
00007      |_____|   |_____|   |_____|   |_____|   |_____|
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
02337
02349

```

17.107 /home/mo/mark3-source/embedded/stage/src/mark3cfg.h File Reference

Mark3 Kernel Configuration.

Macros

- #define `KERNEL_USE_TIMERS` (1)
The following options is related to all kernel time-tracking.
- #define `KERNEL_TIMERS_TICKLESS` (1)
If you've opted to use the kernel timers module, you have an option as to which timer implementation to use: Tick-based or Tick-less.
- #define `KERNEL_USE_QUANTUM` (1)
Do you want to enable time quanta? This is useful when you want to have tasks in the same priority group share time in a controlled way.
- #define `THREAD_QUANTUM_DEFAULT` (4)
This value defines the default thread quantum when `KERNEL_USE_QUANTUM` is enabled.
- #define `KERNEL_USE_SEMAPHORE` (1)
Do you want the ability to use counting/binary semaphores for thread synchronization? Enabling this features provides fully-blocking semaphores and enables all API functions declared in `semaphore.h`.
- #define `KERNEL_USE_MUTEX` (1)
Do you want the ability to use mutual exclusion semaphores (mutex) for resource/block protection? Enabling this feature provides mutexes, with priority inheritance, as declared in `mutex.h`.
- #define `KERNEL_USE_EVENTFLAG` (1)
Provides additional event-flag based blocking.
- #define `KERNEL_USE_MESSAGE` (1)
Enable inter-thread messaging using message queues.
- #define `GLOBAL_MESSAGE_POOL_SIZE` (8)
If Messages are enabled, define the size of the default kernel message pool.
- #define `KERNEL_USE_SLEEP` (1)
Do you want to be able to set threads to sleep for a specified time? This enables the `Thread::Sleep()` API.
- #define `KERNEL_USE_DRIVER` (1)
Enabling device drivers provides a posix-like filesystem interface for peripheral device drivers.
- #define `KERNEL_USE_THREADNAME` (1)
Provide `Thread` method to allow the user to set a name for each thread in the system.
- #define `KERNEL_USE_DYNAMIC_THREADS` (1)
Provide extra `Thread` methods to allow the application to create (and more importantly destroy) threads at runtime.
- #define `KERNEL_USE_PROFILER` (1)
Provides extra classes for profiling the performance of code.

- `#define KERNEL_USE_DEBUG (0)`
Provides extra logic for kernel debugging, and instruments the kernel with extra asserts, and kernel trace functionality.
- `#define KERNEL_USE_ATOMIC (1)`
Provides support for atomic operations, including addition, subtraction, set, and test-and-set.
- `#define SAFE_UNLINK (1)`
"Safe unlinking" performs extra checks on data to make sure that there are no consistencies when performing operations on linked lists.
- `#define TRANSACTION_QUEUE_SIZE (3)`
Defines the size of the kernel transaction queue.

17.107.1 Detailed Description

Mark3 [Kernel](#) Configuration. This file is used to configure the kernel for your specific application in order to provide the optimal set of features for a given use case.

Since you only pay the price (code space/RAM) for the features you use, you can usually find a sweet spot between features and resource usage by picking and choosing features a-la-carte. This config file is written in an "interactive" way, in order to minimize confusion about what each option provides, and to make dependencies obvious.

Definition in file [mark3cfg.h](#).

17.107.2 Macro Definition Documentation

17.107.2.1 `#define GLOBAL_MESSAGE_POOL_SIZE (8)`

If Messages are enabled, define the size of the default kernel message pool.

Messages can be manually added to the message pool, but this mechanism is more convenient and automatic. All message queues share their message objects from this global pool to maximize efficiency and simplify data management.

Definition at line 127 of file [mark3cfg.h](#).

17.107.2.2 `#define KERNEL_TIMERS_TICKLESS (1)`

If you've opted to use the kernel timers module, you have an option as to which timer implementation to use: Tick-based or Tick-less.

Tick-based timers provide a "traditional" RTOS timer implementation based on a fixed-frequency timer interrupt. While this provides very accurate, reliable timing, it also means that the CPU is being interrupted far more often than may be necessary (as not all timer ticks result in "real work" being done).

Tick-less timers still rely on a hardware timer interrupt, but uses a dynamic expiry interval to ensure that the interrupt is only called when the next timer expires. This increases the complexity of the timer interrupt handler, but reduces the number and frequency.

Note that the CPU port ([kerneltimer.cpp](#)) must be implemented for the particular timer variant desired.

Definition at line 62 of file [mark3cfg.h](#).

17.107.2.3 `#define KERNEL_USE_ATOMIC (1)`

Provides support for atomic operations, including addition, subtraction, set, and test-and-set.

Add/Sub/Set contain 8, 16, and 32-bit variants.

Definition at line 177 of file [mark3cfg.h](#).

17.107.2.4 #define KERNEL_USE_DYNAMIC_THREADS (1)

Provide extra [Thread](#) methods to allow the application to create (and more importantly destroy) threads at runtime. Useful for designs implementing worker threads, or threads that can be restarted after encountering error conditions. Definition at line 159 of file [mark3cfg.h](#).

17.107.2.5 #define KERNEL_USE_EVENTFLAG (1)

Provides additional event-flag based blocking.

This relies on an additional per-thread flag-mask to be allocated, which adds 2 bytes to the size of each thread object.

Definition at line 106 of file [mark3cfg.h](#).

17.107.2.6 #define KERNEL_USE_MESSAGE (1)

Enable inter-thread messaging using message queues.

This is the preferred mechanism for IPC for serious multi-threaded communications; generally anywhere a semaphore or event-flag is insufficient.

Definition at line 114 of file [mark3cfg.h](#).

17.107.2.7 #define KERNEL_USE_PROFILER (1)

Provides extra classes for profiling the performance of code.

Useful for debugging and development, but uses an additional hardware timer.

Definition at line 165 of file [mark3cfg.h](#).

17.107.2.8 #define KERNEL_USE_QUANTUM (1)

Do you want to enable time quanta? This is useful when you want to have tasks in the same priority group share time in a controlled way.

This allows equal tasks to use unequal amounts of the CPU, which is a great way to set up CPU budgets per thread in a round-robin scheduling system. If enabled, you can specify a number of ticks that serves as the default time period (quantum). Unless otherwise specified, every thread in a priority will get the default quantum.

Definition at line 75 of file [mark3cfg.h](#).

17.107.2.9 #define KERNEL_USE_SEMAPHORE (1)

Do you want the ability to use counting/binary semaphores for thread synchronization? Enabling this features provides fully-blocking semaphores and enables all API functions declared in semaphore.h.

If you have to pick one blocking mechanism, this is the one to choose.

Definition at line 92 of file [mark3cfg.h](#).

17.107.2.10 #define KERNEL_USE_THREADNAME (1)

Provide [Thread](#) method to allow the user to set a name for each thread in the system.

Adds a const K_CHAR* pointer to the size of the thread object.

Definition at line 151 of file [mark3cfg.h](#).


```

00079
00084 #define THREAD_QUANTUM_DEFAULT (4)
00085
00092 #define KERNEL_USE_SEMAPHORE (1)
00093
00099 #define KERNEL_USE_MUTEX (1)
00100
00106 #define KERNEL_USE_EVENTFLAG (1)
00107
00113 #if KERNEL_USE_SEMAPHORE
00114     #define KERNEL_USE_MESSAGE (1)
00115 #else
00116     #define KERNEL_USE_MESSAGE (0)
00117 #endif
00118
00126 #if KERNEL_USE_MESSAGE
00127     #define GLOBAL_MESSAGE_POOL_SIZE (8)
00128 #endif
00129
00134 #if KERNEL_USE_TIMERS && KERNEL_USE_SEMAPHORE
00135     #define KERNEL_USE_SLEEP (1)
00136 #else
00137     #define KERNEL_USE_SLEEP (0)
00138 #endif
00139
00144 #define KERNEL_USE_DRIVER (1)
00145
00151 #define KERNEL_USE_THREADNAME (1)
00152
00159 #define KERNEL_USE_DYNAMIC_THREADS (1)
00160
00165 #define KERNEL_USE_PROFILER (1)
00166
00171 #define KERNEL_USE_DEBUG (0)
00172
00177 #define KERNEL_USE_ATOMIC (1)
00178
00185 #define SAFE_UNLINK (1)
00186
00194 #define TRANSACTION_QUEUE_SIZE (3)
00195
00196 #endif

```

17.109 /home/mo/mark3-source/embedded/stage/src/memutil.cpp File Reference

Implementation of memory, string, and conversion routines.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
#include "memutil.h"
```

17.109.1 Detailed Description

Implementation of memory, string, and conversion routines.

Definition in file [memutil.cpp](#).

17.110 memutil.cpp

```
00001 /*=====
```

```
00002
```

```
00003      _____
```

```
00004      |   \ /   |   |   \ /   |   |   \ /   |   |   \ /   |   |   \ /   |
```

```
00005      |___/\___|___|\___|___|\___|___|\___|___|\___|___|\___|___|\___|
```

```
00006          |       |       |       |       |
```

```
00007
```

```
00008
```

```
00009 --[Mark3 Realtime Platform]-----
```

```
00010
```

```
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
```

```
00012 See license.txt for more information
```

```

00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024 #include "kernel_debug.h"
00025 #include "memutil.h"
00026
00027 //-----
00028 void MemUtil::DecimalToHex( K_UCHAR ucData_, char *szText_ )
00029 {
00030     K_UCHAR ucTmp = ucData_;
00031     K_UCHAR ucMax;
00032
00033     KERNEL_ASSERT( szText_ );
00034
00035     if (ucTmp >= 0x10)
00036     {
00037         ucMax = 2;
00038     }
00039     else
00040     {
00041         ucMax = 1;
00042     }
00043
00044     ucTmp = ucData_;
00045     szText_[ucMax] = 0;
00046     while (ucMax--)
00047     {
00048         if ((ucTmp & 0x0F) <= 9)
00049         {
00050             szText_[ucMax] = '0' + (ucTmp & 0x0F);
00051         }
00052         else
00053         {
00054             szText_[ucMax] = 'A' + ((ucTmp & 0x0F) - 10);
00055         }
00056         ucTmp>>=4;
00057     }
00058 }
00059
00060 //-----
00061 void MemUtil::DecimalToHex( K_USHORT usData_, char *szText_ )
00062 {
00063     K_USHORT usTmp = usData_;
00064     K_USHORT usMax = 1;
00065     K_USHORT usCompare = 0x0010;
00066
00067     KERNEL_ASSERT( szText_ );
00068
00069     while (usData_ > usCompare && usMax < 4)
00070     {
00071         usMax++;
00072         usCompare <= 4;
00073     }
00074
00075     usTmp = usData_;
00076     szText_[usMax] = 0;
00077     while (usMax--)
00078     {
00079         if ((usTmp & 0x0F) <= 9)
00080         {
00081             szText_[usMax] = '0' + (usTmp & 0x0F);
00082         }
00083         else
00084         {
00085             szText_[usMax] = 'A' + ((usTmp & 0x0F) - 10);
00086         }
00087         usTmp>>=4;
00088     }
00089 }
00090
00091 //-----
00092 void MemUtil::DecimalToHex( K_ULONG ulData_, char *szText_ )
00093 {
00094     K_ULONG ulTmp = ulData_;
00095     K_ULONG ulMax = 1;
00096     K_ULONG ulCompare = 0x0010;
00097
00098     KERNEL_ASSERT( szText_ );
00099
00100     while (ulData_ > ulCompare && ulMax < 8)
00101     {
00102         ulMax++;
00103         ulCompare <= 4;
00104     }
00105
00106     ulTmp = ulData_;
00107     szText_[ulMax] = 0;

```



```

00108     while (ulMax--)
00109     {
00110         if ((ulTmp & 0x0F) <= 9)
00111         {
00112             szText_[ulMax] = '0' + (ulTmp & 0x0F);
00113         }
00114         else
00115         {
00116             szText_[ulMax] = 'A' + ((ulTmp & 0x0F) - 10);
00117         }
00118         ulTmp>>=4;
00119     }
00120 }
00121 //-----
00122 void MemUtil::DecimalToString( K_UCHAR ucData_, char *szText_ )
00123 {
00124     K_UCHAR ucTmp = ucData_;
00125     K_UCHAR ucMax;
00126
00127     KERNEL_ASSERT(szText_);
00128
00129     // Find max index to print...
00130     if (ucData_ >= 100)
00131     {
00132         ucMax = 3;
00133     }
00134     else if (ucData_ >= 10)
00135     {
00136         ucMax = 2;
00137     }
00138     else
00139     {
00140         ucMax = 1;
00141     }
00142
00143     szText_[ucMax] = 0;
00144     while (ucMax--)
00145     {
00146         szText_[ucMax] = '0' + (ucTmp % 10);
00147         ucTmp/=10;
00148     }
00149 }
00150
00151 //-----
00152 void MemUtil::DecimalToString( K_USHORT usData_, char *szText_ )
00153 {
00154     K_USHORT usTmp = usData_;
00155     K_USHORT usMax = 1;
00156     K_USHORT usCompare = 10;
00157
00158     KERNEL_ASSERT(szText_);
00159
00160     while (usData_ >= usCompare && usMax < 5)
00161     {
00162         usCompare *= 10;
00163         usMax++;
00164     }
00165
00166     szText_[usMax] = 0;
00167     while (usMax--)
00168     {
00169         szText_[usMax] = '0' + (usTmp % 10);
00170         usTmp/=10;
00171     }
00172 }
00173
00174 //-----
00175 void MemUtil::DecimalToString( K_ULONG ulData_, char *szText_ )
00176 {
00177     K_ULONG ulTmp = ulData_;
00178     K_ULONG ulMax = 1;
00179     K_ULONG ulCompare = 10;
00180
00181     KERNEL_ASSERT(szText_);
00182
00183     while (ulData_ >= ulCompare && ulMax < 12)
00184     {
00185         ulCompare *= 10;
00186         ulMax++;
00187     }
00188
00189     szText_[ulMax] = 0;
00190     while (ulMax--)
00191     {
00192         szText_[ulMax] = '0' + (ulTmp % 10);
00193         ulTmp/=10;
00194     }

```

```

00195 }
00196
00197 //-----
00198 // Basic checksum routines
00199 K_UCHAR MemUtil::Checksum8( const void *pvSrc_, K_USHORT usLen_ )
00200 {
00201     K_UCHAR ucRet = 0;
00202     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00203
00204     KERNEL_ASSERT(pvSrc_);
00205
00206     // 8-bit CRC, computed byte at a time
00207     while (usLen_--)
00208     {
00209         ucRet += *pcData++;
00210     }
00211     return ucRet;
00212 }
00213
00214 //-----
00215 K_USHORT MemUtil::Checksum16( const void *pvSrc_, K_USHORT usLen_ )
00216 {
00217     K_USHORT usRet = 0;
00218     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00219
00220     KERNEL_ASSERT(pvSrc_);
00221
00222     // 16-bit CRC, computed byte at a time
00223     while (usLen_--)
00224     {
00225         usRet += *pcData++;
00226     }
00227     return usRet;
00228 }
00229
00230 //-----
00231 // Basic string routines
00232 K_USHORT MemUtil::StringLength( const char *szStr_ )
00233 {
00234     K_UCHAR *pcData = (K_UCHAR*)szStr_;
00235     K_USHORT usLen = 0;
00236
00237     KERNEL_ASSERT(szStr_);
00238
00239     while (*pcData++)
00240     {
00241         usLen++;
00242     }
00243     return usLen;
00244 }
00245
00246 //-----
00247 bool MemUtil::CompareStrings( const char *szStr1_, const char *szStr2_ )
00248 {
00249     char *szTmp1 = (char*) szStr1_;
00250     char *szTmp2 = (char*) szStr2_;
00251
00252     KERNEL_ASSERT(szStr1_);
00253     KERNEL_ASSERT(szStr2_);
00254
00255     while (*szTmp1 && *szTmp2)
00256     {
00257         if (*szTmp1++ != *szTmp2++)
00258         {
00259             return false;
00260         }
00261     }
00262
00263     // Both terminate at the same length
00264     if (!(*szTmp1) && !(*szTmp2))
00265     {
00266         return true;
00267     }
00268
00269     return false;
00270 }
00271
00272 //-----
00273 void MemUtil::CopyMemory( void *pvDst_, const void *pvSrc_, K_USHORT usLen_ )
00274 {
00275     char *szDst = (char*) pvDst_;
00276     char *szSrc = (char*) pvSrc_;
00277
00278     KERNEL_ASSERT(pvDst_);
00279     KERNEL_ASSERT(pvSrc_);
00280
00281     // Run through the strings verifying that each character matches

```

```

00282     // and the lengths are the same.
00283     while (usLen_--)
00284     {
00285         *szDst++ = *szSrc++;
00286     }
00287 }
00288
00289 //-----
00290 void MemUtil::CopyString( char *szDst_, const char *szSrc_ )
00291 {
00292     char *szDst = (char*) szDst_;
00293     char *szSrc = (char*) szSrc_;
00294
00295     KERNEL_ASSERT(szDst_);
00296     KERNEL_ASSERT(szSrc_);
00297
00298     // Run through the strings verifying that each character matches
00299     // and the lengths are the same.
00300     while (*szSrc)
00301     {
00302         *szDst++ = *szSrc++;
00303     }
00304 }
00305
00306 //-----
00307 K_SHORT MemUtil::StringSearch( const char *szBuffer_, const char *szPattern_ )
00308 {
00309     char *szTmpPat = (char*)szPattern_;
00310     K_SHORT il6Idx = 0;
00311     K_SHORT il6Start;
00312     KERNEL_ASSERT( szBuffer_ );
00313     KERNEL_ASSERT( szPattern_ );
00314
00315     // Run through the big buffer looking for a match of the pattern
00316     while (szBuffer_[il6Idx])
00317     {
00318         // Reload the pattern
00319         il6Start = il6Idx;
00320         szTmpPat = (char*)szPattern_;
00321         while (*szTmpPat && szBuffer_[il6Idx])
00322         {
00323             if (*szTmpPat != szBuffer_[il6Idx])
00324             {
00325                 break;
00326             }
00327             szTmpPat++;
00328             il6Idx++;
00329         }
00330         // Made it to the end of the pattern, it's a match.
00331         if (*szTmpPat == '\0')
00332         {
00333             return il6Start;
00334         }
00335         il6Idx++;
00336     }
00337
00338     return -1;
00339 }
00340
00341 //-----
00342 bool MemUtil::CompareMemory( const void *pvMem1_, const void *pvMem2_, K_USHORT
usLen_ )
00343 {
00344     char *szTmp1 = (char*) pvMem1_;
00345     char *szTmp2 = (char*) pvMem2_;
00346
00347     KERNEL_ASSERT(pvMem1_);
00348     KERNEL_ASSERT(pvMem2_);
00349
00350     // Run through the strings verifying that each character matches
00351     // and the lengths are the same.
00352     while (usLen_--)
00353     {
00354         if (*szTmp1++ != *szTmp2++)
00355         {
00356             return false;
00357         }
00358     }
00359     return true;
00360 }
00361
00362 //-----
00363 void MemUtil::SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_ )
00364 {
00365     char *szDst = (char*)pvDst_;
00366
00367     KERNEL_ASSERT(pvDst_);

```

```

00368
00369     while (usLen_--)
00370     {
00371         *szDst++ = ucVal_;
00372     }
00373 }
00374
00375 //-----
00376 K_UCHAR MemUtil::Tokenize( const K_CHAR *szBuffer_, Token_t *pastTokens_, K_UCHAR
ucMaxTokens_)
00377 {
00378     K_UCHAR ucCurrArg = 0;
00379     K_UCHAR ucLastArg = 0;
00380     K_UCHAR i = 0;
00381
00382     K_UCHAR bEscape = false;
00383
00384     KERNEL_ASSERT(szBuffer_);
00385     KERNEL_ASSERT(pastTokens_);
00386
00387     while (szBuffer_[i])
00388     {
00389         //-- Handle unescaped quotes
00390         if (szBuffer_[i] == '\"')
00391         {
00392             if (bEscape)
00393             {
00394                 bEscape = false;
00395             }
00396             else
00397             {
00398                 bEscape = true;
00399             }
00400             i++;
00401             continue;
00402         }
00403
00404         //-- Handle all escaped chars - by ignoring them
00405         if (szBuffer_[i] == '\\')
00406         {
00407             i++;
00408             if (szBuffer_[i])
00409             {
00410                 i++;
00411             }
00412             continue;
00413         }
00414
00415         //-- Process chars based on current escape characters
00416         if (bEscape)
00417         {
00418             // Everything within the quote is treated as literal, but escaped chars are still treated the
same
00419             i++;
00420             continue;
00421         }
00422
00423         //-- Non-escaped case
00424         if (szBuffer_[i] != ' ' )
00425         {
00426             i++;
00427             continue;
00428         }
00429
00430         pastTokens_[ucCurrArg].pcToken = &(szBuffer_[ucLastArg]);
00431         pastTokens_[ucCurrArg].ucLen = i - ucLastArg;
00432         ucCurrArg++;
00433         if (ucCurrArg >= ucMaxTokens_)
00434         {
00435             return ucMaxTokens_;
00436         }
00437
00438         i++;
00439         while (szBuffer_[i] && szBuffer_[i] == ' ')
00440         {
00441             i++;
00442         }
00443
00444         ucLastArg = i;
00445     }
00446     if (i && !szBuffer_[i] && (i - ucLastArg))
00447     {
00448         pastTokens_[ucCurrArg].pcToken = &(szBuffer_[ucLastArg]);
00449         pastTokens_[ucCurrArg].ucLen = i - ucLastArg;
00450         ucCurrArg++;
00451     }
00452     return ucCurrArg;

```

```
00453 }
00454
00455
```

17.111 /home/mo/mark3-source/embedded/stage/src/memutil.h File Reference

Utility class containing memory, string, and conversion routines.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"
```

Classes

- struct [Token_t](#)
Token descriptor struct format.
- class [MemUtil](#)
String and Memory manipulation class.

17.111.1 Detailed Description

Utility class containing memory, string, and conversion routines.

Definition in file [memutil.h](#).

17.112 memutil.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #ifndef __MEMUTIL_H__
00022 #define __MEMUTIL_H__
00023
00024 #include "kerneltypes.h"
00025 #include "mark3cfg.h"
00026 #include "kernel_debug.h"
00027
00028 //-----
00032 typedef struct
00033 {
00034     const K_CHAR *pcToken;
00035     K_UCHAR ucLen;
00036 } Token_t;
00037
00038 //-----
00047 class MemUtil
00048 {
00049 public:
00050
00051
00052 //-----
00061 static void DecimalToHex( K_UCHAR ucData_, char *szText_ );
00062 static void DecimalToHex( K_USHORT usData_, char *szText_ );
00063 static void DecimalToHex( K_ULONG ulData_, char *szText_ );
00064
00065 //-----
00074 static void DecimalToString( K_UCHAR ucData_, char *szText_ );
```

```

00075     static void DecimalToString( K_USHORT usData_, char *szText_ );
00076     static void DecimalToString( K_ULONG ulData_, char *szText_ );
00077
00078     //-----
00088     static K_UCHAR Checksum8( const void *pvSrc_, K_USHORT usLen_ );
00089
00090     //-----
00100     static K_USHORT Checksum16( const void *pvSrc_, K_USHORT usLen_ );
00101
00102     //-----
00112     static K_USHORT StringLength( const char *szStr_ );
00113
00114     //-----
00124     static bool CompareStrings( const char *szStr1_, const char *szStr2_ );
00125
00126     //-----
00136     static void CopyMemory( void *pvDst_, const void *pvSrc_, K_USHORT usLen_ );
00137
00138     //-----
00147     static void CopyString( char *szDst_, const char *szSrc_ );
00148
00149     //-----
00159     static K_SHORT StringSearch( const char *szBuffer_, const char *szPattern_ );
00160
00161     //-----
00173     static bool CompareMemory( const void *pvMem1_, const void *pvMem2_, K_USHORT usLen_ );
00174
00175     //-----
00185     static void SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_ );
00186
00187     //-----
00197     static K_UCHAR Tokenize( const char *szBuffer_, Token_t *pastTokens_, K_UCHAR
ucMaxTokens_ );
00198 };
00199
00200
00201 #endif //__MEMUTIL_H__
00202
00203
00204
00205

```

17.113 /home/mo/mark3-source/embedded/stage/src/message.cpp File Reference

Inter-thread communications via message passing.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "message.h"
#include "threadport.h"
#include "kernel_debug.h"
#include "timerlist.h"

```

Macros

- #define `__FILE_ID__` MESSAGE_CPP

17.113.1 Detailed Description

Inter-thread communications via message passing.

Definition in file [message.cpp](#).

17.114 message.cpp

```

00001 /*=====
00002     _____

```

Generated on Sat Jan 25 2014 18:56:55 for Mark3 Realtime Kernel by Doxygen

```

00098     CS_ENTER();
00099
00100     // Pop the head of the message queue and return it
00101     pclRet = static_cast<Message*>( m_clLinkedList.GetHead() );
00102     m_clLinkedList.Remove( static_cast<Message*>(pclRet) );
00103
00104     CS_EXIT();
00105
00106     return pclRet;
00107 }
00108
00109 #if KERNEL_USE_TIMERS
00110 //-----
00111 Message *MessageQueue::Receive( K_ULONG ulTimeWaitMS_ )
00112 {
00113     Message *pclRet;
00114
00115     // Block the current thread on the counting semaphore
00116     if (!m_clSemaphore.Pend(ulTimeWaitMS_))
00117     {
00118         return NULL;
00119     }
00120
00121     CS_ENTER();
00122
00123     // Pop the head of the message queue and return it
00124     pclRet = static_cast<Message*>( m_clLinkedList.GetHead() );
00125     m_clLinkedList.Remove( static_cast<Message*>(pclRet) );
00126
00127     CS_EXIT();
00128
00129     return pclRet;
00130 }
00131 #endif
00132 //-----
00133 void MessageQueue::Send( Message *pclSrc_ )
00134 {
00135     KERNEL_ASSERT( pclSrc_ );
00136
00137     CS_ENTER();
00138
00139     // Add the message to the head of the linked list
00140     m_clLinkedList.Add( pclSrc_ );
00141
00142     // Post the semaphore, waking the blocking thread for the queue.
00143     m_clSemaphore.Post();
00144
00145     CS_EXIT();
00146 }
00147
00148 //-----
00149 K_USHORT MessageQueue::GetCount()
00150 {
00151     return m_clSemaphore.GetCount();
00152 }
00153 #endif //KERNEL_USE_MESSAGE

```

17.115 /home/mo/mark3-source/embedded/stage/src/message.h File Reference

Inter-thread communication via message-passing.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "ksemaphore.h"
#include "timerlist.h"

```

Classes

- class [Message](#)
Class to provide message-based IPC services in the kernel.
- class [GlobalMessagePool](#)
Implements a list of message objects shared between all threads.

- class [MessageQueue](#)

List of messages, used as the channel for sending and receiving messages between threads.

17.115.1 Detailed Description

Inter-thread communication via message-passing. Embedded systems guru Jack Ganssle once said that without a robust form of interprocess communications (IPC), an RTOS is just a toy. Mark3 implements a form of IPC to provide safe and flexible messaging between threads.

Using kernel-managed IPC offers significant benefits over other forms of data sharing (i.e. Global variables) in that it avoids synchronization issues and race conditions common to the practice. Using IPC also enforces a more disciplined coding style that keeps threads decoupled from one another and minimizes global data preventing careless and hard-to-debug errors.

17.115.2 Using Messages, Queues, and the Global Message Pool

```
// Declare a message queue shared between two threads
MessageQueue my_queue;

int main()
{
    ...
    // Initialize the message queue
    my_queue.init();
    ...
}

void Thread1()
{
    // Example TX thread - sends a message every 10ms
    while(1)
    {
        // Grab a message from the global message pool
        Message *tx_message = GlobalMessagePool::Pop();

        // Set the message data/parameters
        tx_message->SetCode( 1234 );
        tx_message->SetData( NULL );

        // Send the message on the queue.
        my_queue.Send( tx_message );
        Thread::Sleep(10);
    }
}

void Thread2()
{
    while()
    {
        // Blocking receive - wait until we have messages to process
        Message *rx_message = my_queue.Recv();

        // Do something with the message data...

        // Return back into the pool when done
        GlobalMessagePool::Push(rx_message);
    }
}
```

Definition in file [message.h](#).

17.116 message.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
```

```

00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00080 #ifndef __MESSAGE_H__
00081 #define __MESSAGE_H__
00082
00083 #include "kerneltypes.h"
00084 #include "mark3cfg.h"
00085
00086 #include "ll.h"
00087 #include "ksemaphore.h"
00088
00089 #if KERNEL_USE_MESSAGE
00090
00091 #if KERNEL_USE_TIMERS
00092     #include "timerlist.h"
00093 #endif
00094
00095 //-----
00099 class Message : public LinkListNode
00100 {
00101 public:
00107     void Init() { m_pvData = NULL; m_usCode = 0; }
00108
00116     void SetData( void *pvData_ ) { m_pvData = pvData_; }
00117
00125     void *GetData() { return m_pvData; }
00126
00134     void SetCode( K_USHORT usCode_ ) { m_usCode = usCode_; }
00135
00143     K_USHORT GetCode() { return m_usCode; }
00144 private:
00145
00147     void *m_pvData;
00148
00150     K_USHORT m_usCode;
00151 };
00152
00153 //-----
00157 class GlobalMessagePool
00158 {
00159 public:
00165     static void Init();
00166
00176     static void Push( Message *pclMessage_ );
00177
00186     static Message *Pop();
00187
00188 private:
00190     static Message m_aclMessagePool[
        GLOBAL_MESSAGE_POOL_SIZE];
00191
00193     static DoubleLinkList m_clList;
00194 };
00195
00196 //-----
00201 class MessageQueue
00202 {
00203 public:
00209     void Init();
00210
00219     Message *Receive();
00220
00221 #if KERNEL_USE_TIMERS
00222
00236     Message *Receive( K_ULONG ulTimeWaitMS_ );
00237 #endif
00238
00247     void Send( Message *pclSrc_ );
00248
00249
00257     K_USHORT GetCount();
00258 private:
00259
00261     Semaphore m_clSemaphore;
00262
00264     DoubleLinkList m_clLinkList;
00265 };
00266
00267 #endif //KERNEL_USE_MESSAGE
00268
00269 #endif

```



```

00041 #define MUTEX_TRANSACTION_TIMEOUT    (2)
00042
00043 #if KERNEL_USE_TIMERS
00044
00045 //-----
00046 void TimedMutex_Callback(Thread *pclOwner_, void *pvData_)
00047 {
00048     Mutex *pclMutex = static_cast<Mutex*>(pvData_);
00049
00050     // Wake up the thread that was blocked on this semaphore.
00051     pclMutex->Timeout(pclOwner_);
00052 }
00053
00054 //-----
00055 void Mutex::Timeout(Thread *pclOwner_)
00056 {
00057     // Take a lock on the object - if the object is already locked, it means
00058     // that another context is currently operating within the locked context.
00059     // In that case, queue an event in the kernel transaction queue, and
00060     // return out immediately. The operation will be executed on the
00061     // thread currently holding the lock.
00062
00063     K_BOOL bSchedState;
00064     if (LockAndQueue( MUTEX_TRANSACTION_TIMEOUT, (void*)pclOwner_, &bSchedState))
00065     {
00066         return;
00067     }
00068
00069     // Drain the FIFO - this will ensure that the operation above is executed,
00070     // as well as any other queued operations that occur as a result of
00071     // processing through interrupts.
00072     if (ProcessQueue()) {
00073         Thread::Yield();
00074     }
00075
00076     // Re-enable the scheduler to its previous state.
00077     Scheduler::SetScheduler(bSchedState);
00078 }
00079
00080 #endif
00081
00082 //-----
00083 K_BOOL Mutex::ProcessQueue()
00084 {
00085     Transaction *pclTRX;
00086     K_BOOL bReschedule = false;
00087
00088     do
00089     {
00090         pclTRX = m_clKTQ.Dequeue();
00091         KERNEL_ASSERT(pclTRX);
00092
00093         switch (pclTRX->GetCode())
00094         {
00095             case MUTEX_TRANSACTION_CLAIM:
00096                 ClaimTransaction(pclTRX, &bReschedule);
00097                 break;
00098             case MUTEX_TRANSACTION_RELEASE:
00099                 ReleaseTransaction(pclTRX, &bReschedule);
00100                 break;
00101             #if KERNEL_USE_TIMERS
00102             case MUTEX_TRANSACTION_TIMEOUT:
00103                 TimeoutTransaction(pclTRX, &bReschedule);
00104                 break;
00105             #endif
00106             default:
00107                 break;
00108         }
00109         m_clKTQ.Finish(pclTRX);
00110     } while (UnLock() > 1);
00111
00112     return bReschedule;
00113 }
00114 //-----
00115 void Mutex::ClaimTransaction(Transaction *pclTRX_, K_BOOL *pbReschedule_)
00116 {
00117     Thread *pclThread = static_cast<Thread*>(pclTRX_->GetData());
00118
00119     // Check to see if the mutex is claimed or not
00120     if (m_bReady != 0)
00121     {
00122         // Mutex isn't claimed, claim it.
00123         m_bReady = 0;
00124         m_ucRecurse = 0;
00125         m_ucMaxPri = pclThread->GetPriority();
00126         m_pclOwner = pclThread;
00127     }

```

```

00128     else
00129     {
00130         // If the mutex is already claimed, check to see if this is the owner thread,
00131         // since we allow the mutex to be claimed recursively.
00132         if (pclThread == m_pclOwner)
00133         {
00134             // Ensure that we haven't exceeded the maximum recursive-lock count
00135             KERNEL_ASSERT( (m_ucRecurse < 255) );
00136             m_ucRecurse++;
00137             return;
00138         }
00139
00140         // The mutex is claimed already - we have to block now. Move the
00141         // current thread to the list of threads waiting on the mutex.
00142 #if KERNEL_USE_TIMERS
00143         K_ULONG ulWaitTimeMS = pclThread->GetTimer()->GetInterval();
00144         pclThread->SetExpired(false);
00145         if (ulWaitTimeMS)
00146         {
00147             pclThread->GetTimer()->Start(0, ulWaitTimeMS, (TimerCallback_t)TimedMutex_Callback,
00148             (void*)this);
00149         }
00150 #endif
00151         Block(pclThread);
00152
00153         // Check if priority inheritance is necessary. We do this in order
00154         // to ensure that we don't end up with priority inversions in case
00155         // multiple threads are waiting on the same resource.
00156
00157         // We can get away with doing this outside of a critical section, as all
00158         // transactions are serialized by the transaction queue, and the scheduler
00159         // is disabled.
00160
00161         if (m_ucMaxPri <= pclThread->GetPriority())
00162         {
00163             m_ucMaxPri = pclThread->GetPriority();
00164
00165             {
00166                 Thread *pclTemp = static_cast<Thread*>(m_clBlockList.
00167                 GetHead());
00168                 while(pclTemp)
00169                 {
00170                     pclTemp->InheritPriority(m_ucMaxPri);
00171                     if(pclTemp == static_cast<Thread*>(m_clBlockList.
00172                     GetTail()))
00173                     {
00174                         break;
00175                     }
00176                     pclTemp = static_cast<Thread*>(pclTemp->GetNext());
00177                 }
00178                 m_pclOwner->InheritPriority(m_ucMaxPri);
00179             }
00180             *pbReschedule_ = true;
00181         }
00182     }
00183
00184 //-----
00185 void Mutex::ReleaseTransaction(Transaction *pclTRX_, K_BOOL *
pbReschedule_)
00186 {
00187     Thread *pclThread;
00188
00189     // Disable the scheduler while we deal with internal data structures.
00190     pclThread = Scheduler::GetCurrentThread();
00191
00192     // This thread had better be the one that owns the mutex currently...
00193     KERNEL_ASSERT((pclThread == m_pclOwner));
00194
00195     // If the owner had claimed the lock multiple times, decrease the lock
00196     // count and return immediately.
00197     if (m_ucRecurse)
00198     {
00199         m_ucRecurse--;
00200         return;
00201     }
00202
00203     // Restore the thread's original priority
00204     if (pclThread->GetCurPriority() != pclThread->GetPriority())
00205     {
00206         pclThread->SetPriority(pclThread->GetPriority());
00207
00208         // In this case, we want to reschedule
00209         *pbReschedule_ = true;
00210     }

```

```

00211
00212 // No threads are waiting on this semaphore?
00213 if (m_clBlockList.GetHead() == NULL)
00214 {
00215     // Re-initialize the mutex to its default values
00216     m_bReady = 1;
00217     m_ucMaxPri = 0;
00218     m_pclOwner = NULL;
00219 }
00220 else
00221 {
00222     // Wake the highest priority Thread pending on the mutex
00223     if(WakeNext())
00224     {
00225         // Switch threads if it's higher or equal priority than the current thread
00226         *pbReschedule_ = true;
00227     }
00228 }
00229 }
00230
00231 #if KERNEL_USE_TIMERS
00232 //-----
00233 void Mutex::TimeoutTransaction(Transaction *pclTRX_, K_BOOL *
pbReschedule_)
00234 {
00235     Thread *pclChosenOne = static_cast<Thread*>(pclTRX_->GetData());
00236     Unblock(pclChosenOne);
00237     pclChosenOne->SetExpired(true);
00238     if (pclChosenOne->GetPriority() > Scheduler::GetCurrentThread()->
GetPriority())
00239     {
00240         *pbReschedule_ = true;
00241     }
00242 }
00243 #endif
00244 //-----
00245 K_UCHAR Mutex::WakeNext()
00246 {
00247     Thread *pclChosenOne = NULL;
00248     // Get the highest priority waiter thread
00249     pclChosenOne = m_clBlockList.HighestWaiter();
00250     // Unblock the thread
00251     Unblock(pclChosenOne);
00252     // The chosen one now owns the mutex
00253     m_pclOwner = pclChosenOne;
00254     // Signal a context switch if it's a greater than or equal to the current priority
00255     if (pclChosenOne->GetPriority() >= Scheduler::GetCurrentThread()->
GetPriority())
00256     {
00257         return 1;
00258     }
00259     return 0;
00260 }
00261 //-----
00262 void Mutex::Init()
00263 {
00264     // Reset the data in the mutex
00265     m_bReady = 1; // The mutex is free.
00266     m_ucMaxPri = 0; // Set the maximum priority inheritance state
00267     m_pclOwner = NULL; // Clear the mutex owner
00268     m_ucRecurse = 0; // Reset recurse count
00269 }
00270 //-----
00271 #if KERNEL_USE_TIMERS
00272 void Mutex::Claim()
00273 {
00274     Claim(0);
00275 }
00276 bool Mutex::Claim(K_ULONG ulWaitTimeMS_)
00277 #else
00278 void Mutex::Claim()
00279 #endif
00280 {
00281     KERNEL_TRACE_1( STR_MUTEX_CLAIM_1, (K_USHORT)Scheduler::GetCurrentThread()->
GetID() );
00282     // Claim the lock (we know only one thread can hold the lock, only one thread can

```

```

00294     // execute at a time, and only threads can call wait)
00295     K_BOOL bSchedState;
00296     if (LockAndQueue( MUTEX_TRANSACTION_CLAIM, (void*)
Scheduler::GetCurrentThread(), &bSchedState))
00297     {
00298         Kernel::Panic( PANIC_MUTEX_LOCK_VIOLATION );
00299     }
00300
00301     // Set data on the current thread that needs to be passed into the transaction
00302     // handler (and can't be queued in the simple key-value pair in the transaciton
00303     // object)
00304 #if KERNEL_USE_TIMERS
00305     Scheduler::GetCurrentThread()->GetTimer()->
SetIntervalTicks(ulWaitTimeMS_);
00306     Scheduler::GetCurrentThread()->SetExpired(false);
00307 #endif
00308
00309     if (ProcessQueue()) {
00310         Thread::Yield();
00311     }
00312
00313     Scheduler::SetScheduler(bSchedState);
00314
00315 #if KERNEL_USE_TIMERS
00316     if (ulWaitTimeMS_)
00317     {
00318         Scheduler::GetCurrentThread()->GetTimer()->
Stop();
00319     }
00320     return (Scheduler::GetCurrentThread()->GetExpired() == false);
00321 #endif
00322 }
00323
00324 //-----
00325 void Mutex::Release()
00326 {
00327     KERNEL_TRACE_1( STR_MUTEX_RELEASE_1, (K_USHORT)Scheduler::GetCurrentThread()
->GetID() );
00328
00329     K_BOOL bSchedState;
00330     if (LockAndQueue( MUTEX_TRANSACTION_RELEASE, (void*)
Scheduler::GetCurrentThread(), &bSchedState))
00331     {
00332         return;
00333     }
00334
00335     if (ProcessQueue()) {
00336         Thread::Yield();
00337     }
00338
00339     Scheduler::SetScheduler(bSchedState);
00340 }
00341
00342 #endif //KERNEL_USE_MUTEX

```

17.119 /home/mo/mark3-source/embedded/stage/src/mutex.h File Reference

Mutual exclusion class declaration.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "transaction.h"
#include "timerlist.h"

```

Classes

- class [Mutex](#)

Mutual-exclusion locks, based on [BlockingObject](#).

17.119.1 Detailed Description

Mutual exclusion class declaration. Resource locks are implemented using mutual exclusion semaphores (Mutex_t). Protected blocks can be placed around any resource that may only be accessed by one thread at a time. If additional threads attempt to access the protected resource, they will be placed in a wait queue until the resource becomes available. When the resource becomes available, the thread with the highest original priority claims the resource and is activated. Priority inheritance is included in the implementation to prevent priority inversion. Always ensure that you claim and release your mutex objects consistently, otherwise you may end up with a deadlock scenario that's hard to debug.

17.119.2 Initializing

Initializing a mutex object by calling:

```
clMutex.Init();
```

17.119.3 Resource protection example

```
clMutex.Claim();
...
<resource protected block>
...
clMutex.Release();
```

Definition in file [mutex.h](#).

17.120 mutex.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00050 #ifndef __MUTEX_H_
00051 #define __MUTEX_H_
00052
00053 #include "kerneltypes.h"
00054 #include "mark3cfg.h"
00055
00056 #include "blocking.h"
00057 #include "transaction.h"
00058
00059 #if KERNEL_USE_MUTEX
00060
00061 #if KERNEL_USE_TIMERS
00062 #include "timerlist.h"
00063 #endif
00064
00065 //-----
00069 class Mutex : public BlockingObject
00070 {
00071 public:
00072     void Init();
00073
00074     void Claim();
00075
00076
00077 #if KERNEL_USE_TIMERS
00078     bool Claim(K_ULONG ulWaitTimeMS_);
00079
00080     void Timeout( Thread *pclOwner_ );
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
```


17.121 /home/mo/mark3-source/embedded/stage/src/nlfs.cpp File Reference

```
#include "kerneltypes.h"
#include "nlfs.h"
#include "nlfs_file.h"
#include "memutil.h"
#include "nlfs_config.h"
```

Definition in file [nlfs.cpp](#).

```

00001 /*-----
00002
00003
00004 |-----|-----|-----|-----|-----|-----|
00005 | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / |
00006 | / \ | / \ | / \ | / \ | / \ | / \ | / \ | / \ |
00007 |-----|-----|-----|-----|-----|-----|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*
00019 #include "kerneltypes.h"
00020 #include "nlfs.h"
00021 #include "nlfs_file.h"
00022 #include "memutil.h"
00023 #include "nlfs_config.h"
00024
00025 //-----
00026 K_CHAR NLFS::Find_Last_Slash( const char *szPath_ )
00027 {

```

```

00028     K_UCHAR ucLastSlash = 0;
00029     K_UCHAR i = 0;
00030     while (szPath_[i])
00031     {
00032         if (szPath_[i] == '/')
00033         {
00034             ucLastSlash = i;
00035         }
00036         i++;
00037     }
00038     return ucLastSlash;
00039 }
00040
00041 //-----
00042 K_BOOL NLFS::File_Names_Match( const K_CHAR *szPath_,
    NLFS_Node_t *pstNode_)
00043 {
00044     K_UCHAR ucLastSlash = Find_Last_Slash( szPath_ );
00045     K_UCHAR i;
00046     ucLastSlash++;
00047     for (i = 0; i < FILE_NAME_LENGTH; i++)
00048     {
00049         if (!szPath_[ucLastSlash+i] || !pstNode_>stFileNode.
acFileName[i])
00050         {
00051             break;
00052         }
00053         if (szPath_[ucLastSlash+i] != pstNode_>stFileNode.acFileName[i])
00054         {
00055             return false;
00056         }
00057     }
00058     if (szPath_[ucLastSlash+i] != pstNode_>stFileNode.acFileName[i])
00059     {
00060         return false;
00061     }
00062     return true;
00063 }
00064
00065 //-----
00066 void NLFS::Print_File_Details( K_USHORT usNode_ )
00067 {
00068     NLFS_Node_t stFileNode;
00069     Read_Node(usNode_, &stFileNode);
00070     DEBUG_PRINT(" Name      : %16s\n" , stFileNode.stFileNode.
acFileName);
00071     DEBUG_PRINT(" Next Peer : %d\n" , stFileNode.stFileNode.
usNextPeer);
00072     DEBUG_PRINT(" Prev Peer : %d\n" , stFileNode.stFileNode.
usPrevPeer);
00073     DEBUG_PRINT(" User|Group : %d|%d\n" , stFileNode.stFileNode.ucUser,
stFileNode.stFileNode.ucGroup);
00074     DEBUG_PRINT(" Permissions: %04X\n" , stFileNode.stFileNode.usPerms);
00075     DEBUG_PRINT(" Parent      : %d\n" , stFileNode.stFileNode.
usParent);
00076     DEBUG_PRINT(" First Child: %d\n" , stFileNode.stFileNode.usChild);
00077     DEBUG_PRINT(" Alloc Size : %d\n" , stFileNode.stFileNode.
ulAllocSize);
00078     DEBUG_PRINT(" File Size : %d\n" , stFileNode.stFileNode.
ulFileSize);
00079     DEBUG_PRINT(" First Block: %d\n" , stFileNode.stFileNode.
ulFirstBlock);
00080     DEBUG_PRINT(" Last Block : %d\n" , stFileNode.stFileNode.
ulLastBlock);
00081 }
00082
00083 //-----
00084 void NLFS::Print_Dir_Details( K_USHORT usNode_ )
00085 {
00086     NLFS_Node_t stFileNode;
00087     Read_Node(usNode_, &stFileNode);
00088     DEBUG_PRINT(" Name      : %16s\n" , stFileNode.stFileNode.
acFileName);
00089     DEBUG_PRINT(" Next Peer : %d\n" , stFileNode.stFileNode.
usNextPeer);
00090     DEBUG_PRINT(" Prev Peer : %d\n" , stFileNode.stFileNode.
usPrevPeer);
00091     DEBUG_PRINT(" User|Group : %d|%d\n" , stFileNode.stFileNode.ucUser,
stFileNode.stFileNode.ucGroup);
00092     DEBUG_PRINT(" Permissions: %04X\n" , stFileNode.stFileNode.
usPerms);

```

```

00101     DEBUG_PRINT(" Parent      : %d\n"      , stFileNode.stFileNode.
00102     usParent);
00102     DEBUG_PRINT(" First Child: %d\n"      , stFileNode.stFileNode.usChild);
00103 }
00104
00105 //-----
00106 void NLFS::Print_Free_Details( K_USHORT usNode_ )
00107 {
00108     NLFS_Node_t stFileNode;
00109     Read_Node(usNode_, &stFileNode);
00110
00111     DEBUG_PRINT(" Next Free  : %d\n"      , stFileNode.stFileNode.
00112     usNextPeer );
00113 }
00114 //-----
00115 void NLFS::Print_Node_Details( K_USHORT usNode_ )
00116 {
00117     NLFS_Node_t stTempNode;
00118     Read_Node(usNode_, &stTempNode);
00119
00120     DEBUG_PRINT("\nNode: %d\n"
00121     " Node Type: ", usNode_);
00122     switch (stTempNode.eBlockType)
00123     {
00124     case NLFS_NODE_FREE:
00125         DEBUG_PRINT( "Free\n" );
00126         Print_Free_Details(usNode_);
00127         break;
00128     case NLFS_NODE_ROOT:
00129         DEBUG_PRINT( "Root Block\n" );
00130         break;
00131     case NLFS_NODE_FILE:
00132         DEBUG_PRINT( "File\n" );
00133         Print_File_Details(usNode_);
00134         break;
00135     case NLFS_NODE_DIR:
00136         DEBUG_PRINT( "Directory\n" );
00137         Print_Dir_Details(usNode_);
00138         break;
00139     default:
00140         break;
00141     }
00142 }
00143 //-----
00144 K_USHORT NLFS::Pop_Free_Node(void)
00145 {
00146     K_USHORT usRetVal = m_stLocalRoot.usNextFreeNode;
00147     NLFS_Node_t stFileNode;
00148
00149     if (INVALID_NODE == usRetVal)
00150     {
00151         return 0;
00152     }
00153
00154     // Update Claimed node
00155     Read_Node(usRetVal, &stFileNode);
00156     m_stLocalRoot.usNextFreeNode = stFileNode.
00157     stFileNode.usNextPeer;
00158     stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00159     DEBUG_PRINT("Node %d allocated, next free %d\n", usRetVal, m_stLocalRoot.
00160     usNextFreeNode);
00161     Write_Node(usRetVal, &stFileNode);
00162
00163     //Update root node
00164     Read_Node(FS_CONFIG_BLOCK , &stFileNode);
00165     stFileNode.stRootNode.usNextFreeNode = m_stLocalRoot.
00166     usNextFreeNode;
00167     stFileNode.stRootNode.usNumFilesFree--;
00168     Write_Node(FS_CONFIG_BLOCK, &stFileNode);
00169     return usRetVal;
00170 }
00171 //-----
00172 void NLFS::Push_Free_Node(K_USHORT usNode_)
00173 {
00174     NLFS_Node_t stFileNode;
00175
00176     Read_Node(usNode_, &stFileNode);
00177     stFileNode.stFileNode.usNextPeer = m_stLocalRoot.
00178     usNextFreeNode;
00179     m_stLocalRoot.usNextFreeNode = usNode_;
00180     Write_Node(usNode_, &stFileNode);
00181 }

```

```

00182     DEBUG_PRINT("Node %d freed\n", usNode_);
00183
00184     //Update root node
00185     Read_Node(FS_CONFIG_BLOCK , &stFileNode);
00186     stFileNode.stRootNode.usNextFreeNode = m_stLocalRoot.
usNextFreeNode;
00187     stFileNode.stRootNode.usNumFilesFree++;
00188     Write_Node(FS_CONFIG_BLOCK , &stFileNode);
00189 }
00190
00191 //-----
00192 K_ULONG NLFS::Pop_Free_Block(void)
00193 {
00194     K_ULONG ulRetVal = m_stLocalRoot.ulNextFreeBlock;
00195     NLFS_Block_t stFileBlock;
00196     NLFS_Node_t stFileNode;
00197
00198     if ((INVALID_BLOCK == ulRetVal) || (0 == m_stLocalRoot.
ulNumBlocksFree))
00199     {
00200         DEBUG_PRINT("Out of data blocks\n");
00201         return 0;
00202     }
00203
00204     Read_Block_Header(ulRetVal, &stFileBlock);
00205
00206     m_stLocalRoot.ulNextFreeBlock = stFileBlock.
ulNextBlock;
00207     m_stLocalRoot.ulNumBlocksFree--;
00208     stFileBlock.ulNextBlock = INVALID_BLOCK;
00209
00210     Write_Block_Header(ulRetVal, &stFileBlock);
00211
00212     Read_Node(FS_CONFIG_BLOCK , &stFileNode);
00213
00214     stFileNode.stRootNode.ulNextFreeBlock =
m_stLocalRoot.ulNextFreeBlock;
00215     stFileNode.stRootNode.ulNumBlocksFree--;
00216
00217     Write_Node(FS_CONFIG_BLOCK , &stFileNode);
00218
00219     DEBUG_PRINT("Allocated block %d, next free %d\n", ulRetVal, m_stLocalRoot.
ulNextFreeBlock);
00220     return ulRetVal;
00221 }
00222
00223 //-----
00224 void NLFS::Push_Free_Block(K_ULONG ulBlock_ )
00225 {
00226     NLFS_Block_t stFileBlock;
00227     NLFS_Node_t stFileNode;
00228
00229     Read_Block_Header(ulBlock_, &stFileBlock);
00230
00231     stFileBlock.ulNextBlock = m_stLocalRoot.
ulNextFreeBlock;
00232     m_stLocalRoot.ulNextFreeBlock = ulBlock_;
00233
00234     Write_Block_Header(ulBlock_, &stFileBlock);
00235
00236     Read_Node(FS_CONFIG_BLOCK , &stFileNode);
00237     stFileNode.stRootNode.ulNextFreeBlock =
m_stLocalRoot.ulNextFreeBlock;
00238     stFileNode.stRootNode.ulNumBlocksFree++;
00239     Write_Node(FS_CONFIG_BLOCK , &stFileNode);
00240
00241     DEBUG_PRINT("Block %d freed\n", ulBlock_);
00242 }
00243
00244 //-----
00245 K_ULONG NLFS::Append_Block_To_Node(NLFS_Node_t *pstFile_ )
00246 {
00247     K_ULONG ulBlock;
00248     NLFS_Block_t stFileBlock;
00249
00250     // Allocate a new block
00251     ulBlock = Pop_Free_Block();
00252     if (ulBlock == INVALID_BLOCK)
00253     {
00254         return -1;
00255     }
00256
00257     // Initialize the block
00258     DEBUG_PRINT("reading block header\n");
00259     Read_Block_Header(ulBlock, &stFileBlock);
00260     stFileBlock.ulNextBlock = INVALID_BLOCK;
00261     stFileBlock.uAllocated = 1;

```

```

00262
00263     DEBUG_PRINT("writing block header\n");
00264     Write_Block_Header(ulBlock, &stFileBlock);
00265
00266     // Update the previous last-block links (if there is one)
00267     DEBUG_PRINT("updating previous block %d\n", pstFile_>stFileNode.
ulLastBlock);
00268     if (pstFile_>stFileNode.ulLastBlock != INVALID_BLOCK)
00269     {
00270         Read_Block_Header(pstFile_>stFileNode.
ulLastBlock, &stFileBlock);
00271         stFileBlock.ulNextBlock = ulBlock;
00272         Write_Block_Header(pstFile_>stFileNode.
ulLastBlock, &stFileBlock);
00273     }
00274     else
00275     {
00276         DEBUG_PRINT(" previous block is invalid, setting as first\n");
00277         pstFile_>stFileNode.ulFirstBlock = ulBlock;
00278     }
00279
00280     pstFile_>stFileNode.ulLastBlock = ulBlock;
00281     pstFile_>stFileNode.ulAllocSize += m_stLocalRoot.
ulBlockSize;
00282
00283     RootSync();
00284
00285     return ulBlock;
00286 }
00287
00288 //-----
00289 K_USHORT NLFS::Find_Parent_Dir(const K_CHAR *szPath_)
00290 {
00291     int i, j;
00292     K_UCHAR ucLastSlash = 0;
00293     K_USHORT usRetVal;
00294     K_CHAR szTempName[FILE_NAME_LENGTH];
00295     NLFS_Node_t stFileNode;
00296     K_USHORT usTempPeer;
00297
00298     Read_Node(FS_ROOT_BLOCK, &stFileNode );
00299
00300     usRetVal = FS_ROOT_BLOCK;
00301
00302     if (szPath_[0] != '/')
00303     {
00304         DEBUG_PRINT("Only fully-qualified paths are supported.  Bailing\n");
00305         return -1;
00306     }
00307
00308     // Starting from the root fs_block (which is the mount point...)
00309     ucLastSlash = Find_Last_Slash(szPath_);
00310
00311     // a) Search for each "/" if we've got more than one...
00312     if (0 == ucLastSlash)
00313     {
00314         return usRetVal;
00315     }
00316
00317     usTempPeer = stFileNode.stFileNode.usChild;
00318     Read_Node(usTempPeer, &stFileNode );
00319
00320     i = 1;
00321     while (szPath_[i] && i < ucLastSlash)
00322     {
00323         NLFS_Node_t stTempNode;
00324         K_BOOL bMatch = false;
00325
00326         j = 0;
00327         MemUtil::SetMemory(szTempName, 0, FILE_NAME_LENGTH);
00328
00329         while (szPath_[i] && (szPath_[i] != '/') && j < FILE_NAME_LENGTH)
00330         {
00331             szTempName[j] = szPath_[i];
00332             i++;
00333             j++;
00334         }
00335         DEBUG_PRINT("Checking %s\n", szTempName );
00336         if (j == FILE_NAME_LENGTH && szPath_[i] != '/')
00337         {
00338             DEBUG_PRINT("Directory name too long, invalid\n");
00339             return -1;
00340         }
00341         else if (szPath_[i] != '/')
00342         {
00343             i++;
00344             continue;

```

```

00345     }
00346
00347     // Check to see if there's a valid peer with this name...
00348     while (INVALID_NODE != usTempPeer)
00349     {
00350         Read_Node(usTempPeer, &stTempNode);
00351         if (NLFS_NODE_DIR == stTempNode.eBlockType)
00352         {
00353             if (true == MemUtil::CompareStrings(stTempNode.
stFileName.acFileName, szTempName))
00354             {
00355                 bMatch = true;
00356                 break;
00357             }
00358             usTempPeer = stTempNode.stFileName.usNextPeer;
00359         }
00360     }
00361
00362     // Matched the folder name descend into the folder
00363     if (bMatch)
00364     {
00365         DEBUG_PRINT("Matched folder: %s, node %d\n", szTempName, usTempPeer);
00366
00367         usRetVal = usTempPeer;
00368
00369         usTempPeer = stTempNode.stFileName.usChild;
00370         if (INVALID_NODE != usTempPeer)
00371         {
00372             DEBUG_PRINT("Entering subdirectory %d\n", usTempPeer);
00373             Read_Node(usTempPeer, &stTempNode);
00374         }
00375         else
00376         {
00377             break;
00378         }
00379     }
00380     // Failed to match the folder name, bail
00381     else
00382     {
00383         DEBUG_PRINT("Could not match folder name, bailing\n");
00384         usRetVal = -1;
00385         break;
00386     }
00387
00388     if (i >= ucLastSlash)
00389     {
00390         break;
00391     }
00392     i++;
00393 }
00394
00395 if (i == ucLastSlash)
00396 {
00397     // No more folders to traverse - we're successful.
00398     DEBUG_PRINT("Found root path for %s\n with node %d\n", szPath_, usRetVal);
00399     return usRetVal;
00400 }
00401 return INVALID_NODE;
00402 }
00403
00404 //-----
00405 K_USHORT NLFS::Find_File(const K_CHAR *szPath_)
00406 {
00407     NLFS_Node_t stTempNode;
00408     NLFS_Node_t stTempDir;
00409
00410     K_USHORT usTempNode;
00411
00412     K_USHORT usParentDir = Find_Parent_Dir(szPath_);
00413
00414     if (INVALID_NODE == usParentDir)
00415     {
00416         DEBUG_PRINT("invalid root dir\n");
00417         return INVALID_NODE;
00418     }
00419
00420     Read_Node(usParentDir, &stTempDir);
00421
00422     if (INVALID_NODE == stTempDir.stFileName.usChild)
00423     {
00424         return INVALID_NODE;
00425     }
00426
00427     usTempNode = stTempDir.stFileName.usChild;
00428
00429     // See if there are matching child nodes
00430     while (INVALID_NODE != usTempNode)

```

```

00431     {
00432         Read_Node(usTempNode, &stTempNode);
00433
00434         if (true == File_Names_Match(szPath_, &stTempNode ))
00435         {
00436             DEBUG_PRINT("matched file: %16s, node %d\n",
00437                 stTempNode.stFileNode.acFileName, usTempNode);
00438             return usTempNode;
00439         }
00440
00441         usTempNode = stTempNode.stFileNode.usNextPeer;
00442     }
00443     DEBUG_PRINT("couldn't match file: %s\n", szPath_);
00444     return INVALID_NODE;
00445 }
00446
00447 //-----
00448 void NLFS::Print(void)
00449 {
00450     K_USHORT i;
00451     for (i = 0; i < m_stLocalRoot.usNumFiles; i++)
00452     {
00453         Print_Node_Details(i);
00454     }
00455 }
00456
00457 //-----
00458 void NLFS::Set_Node_Name( NLFS_Node_t *pstFileNode_, const char *szPath_ )
00459 {
00460     K_UCHAR i, j;
00461     K_UCHAR ucLastSlash = 0;
00462
00463     // Search for the last "/", that's where we stop looking.
00464     i = 0;
00465     while (szPath_[i])
00466     {
00467         if (szPath_[i] == '/')
00468         {
00469             ucLastSlash = i;
00470         }
00471         i++;
00472     }
00473
00474     // Parse out filename
00475     i = ucLastSlash + 1;
00476     j = 0;
00477     while (szPath_[i] && j < FILE_NAME_LENGTH)
00478     {
00479         pstFileNode_>stFileNode.acFileName[j] = szPath_[i];
00480         j++;
00481         i++;
00482     }
00483     if (!szPath_[i]) // if no extension, we're done.
00484     {
00485         return;
00486     }
00487 }
00488
00489 //-----
00490 K_USHORT NLFS::Create_File_i(const K_CHAR *szPath_,
    NLFS_Type_t eType_ )
00491 {
00492     K_USHORT usNode;
00493     K_USHORT usRootNodes;
00494
00495     NLFS_Node_t stFileNode;
00496     NLFS_Node_t stParentNode;
00497     NLFS_Node_t stPeerNode;
00498
00499     // Tricky part - directory traversal
00500     usRootNodes = Find_Parent_Dir(szPath_);
00501
00502     if (INVALID_NODE == usRootNodes)
00503     {
00504         DEBUG_PRINT("Unable to find path - bailing\n");
00505         return INVALID_NODE;
00506     }
00507
00508     usNode = Pop_Free_Node();
00509     if (!usNode)
00510     {
00511         DEBUG_PRINT("Unable to allocate node. Failing\n");
00512         return INVALID_NODE;
00513     }
00514     DEBUG_PRINT("New file using node %d\n", usNode);
00515
00516     // File node allocated, do something with it...

```

```

00517 // Set the file's name and extension
00518
00519 Read_Node(usNode, &stFileNode);
00520
00521 // Set the file path
00522 Set_Node_Name(&stFileNode, szPath_);
00523
00524 // Set block as in-use as a file
00525 stFileNode.eBlockType = eType_;
00526
00527 // Zero-out the file
00528 stFileNode.stFileNode.ulFileSize = 0;
00529
00530 // Set the default user and group, as well as perms
00531 stFileNode.stFileNode.ucUser = 0;
00532 stFileNode.stFileNode.ucGroup = 0;
00533 stFileNode.stFileNode.usPerms = PERM_U_ALL | PERM_G_ALL | PERM_O_ALL;
00534
00535 stFileNode.stFileNode.usChild = INVALID_NODE;
00536 stFileNode.stFileNode.usParent = usRootNodes;
00537
00538 // Update the parent node.
00539 Read_Node(usRootNodes, &stParentNode);
00540
00541 DEBUG_PRINT( "Parent's root child: %d\n", stParentNode.stFileNode.
usChild );
00542 // Insert node at the beginning of the peer list
00543 if (INVALID_NODE != stParentNode.stFileNode.usChild)
00544 {
00545     stFileNode.stFileNode.usNextPeer = stParentNode.
stFileNode.usChild;
00546     stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00547
00548     // Update the peer node.
00549     Read_Node(stFileNode.stFileNode.usNextPeer, &stPeerNode);
00550
00551     stPeerNode.stFileNode.usPrevPeer = usNode;
00552     stParentNode.stFileNode.usChild = usNode;
00553
00554     DEBUG_PRINT("updating peer's prev: %d\n", stPeerNode.stFileNode.
usPrevPeer);
00555     Write_Node(stFileNode.stFileNode.usNextPeer, &stPeerNode);
00556 }
00557 else
00558 {
00559     stParentNode.stFileNode.usChild = usNode;
00560     stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00561     stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00562 }
00563
00564 Write_Node(usNode, &stFileNode);
00565 Write_Node(usRootNodes, &stParentNode);
00566
00567 RootSync();
00568
00569 return usNode;
00570 }
00571
00572 //-----
00573 K_USHORT NLFS::Create_File( const K_CHAR *szPath_ )
00574 {
00575
00576     if (INVALID_NODE != Find_File(szPath_))
00577     {
00578         DEBUG_PRINT("Create_File: File already exists\n");
00579         return INVALID_NODE;
00580     }
00581
00582     return Create_File_i( szPath_, NLFS_NODE_FILE );
00583 }
00584
00585 //-----
00586 K_USHORT NLFS::Create_Dir( const K_CHAR *szPath_ )
00587 {
00588
00589     if (INVALID_NODE != Find_File(szPath_))
00590     {
00591         DEBUG_PRINT("Create_Dir: Dir already exists!\n");
00592         return INVALID_NODE;
00593     }
00594
00595     return Create_File_i(szPath_, NLFS_NODE_DIR );
00596 }
00597 //-----
00598 void NLFS::Cleanup_Node_Links(K_USHORT usNode_,
NLFS_Node_t *pstNode_)
00599 {

```



```

00600     DEBUG_PRINT("Cleanup_Node_Links: Entering\n");
00601
00602     if (INVALID_NODE != pstNode->stFileNode.usParent)
00603     {
00604         NLFS_Node_t stParent;
00605         DEBUG_PRINT("Cleanup_Node_Links: Parent Node: %d\n", pstNode->
stFileNode.usParent);
00606         Read_Node(pstNode->stFileNode.usParent, &stParent);
00607
00608         DEBUG_PRINT("0\n");
00609         if (stParent.stFileNode.usChild == usNode_)
00610         {
00611             DEBUG_PRINT("1\n");
00612             stParent.stFileNode.usChild = pstNode->stFileNode.
usNextPeer;
00613             Write_Node(pstNode->stFileNode.usParent, &stParent);
00614             DEBUG_PRINT("2\n");
00615         }
00616     }
00617
00618     DEBUG_PRINT("a\n");
00619     if ( (INVALID_NODE != pstNode->stFileNode.usNextPeer) ||
(INVALID_NODE != pstNode->stFileNode.usPrevPeer) )
00620     {
00621         NLFS_Node_t stNextPeer;
00622         NLFS_Node_t stPrevPeer;
00623
00624         DEBUG_PRINT("b\n");
00625         if (INVALID_NODE != pstNode->stFileNode.usNextPeer)
00626         {
00627             DEBUG_PRINT("c\n");
00628             Read_Node(pstNode->stFileNode.usNextPeer, &stNextPeer);
00629             DEBUG_PRINT("d\n");
00630         }
00631
00632         if (INVALID_NODE != pstNode->stFileNode.usPrevPeer)
00633         {
00634             DEBUG_PRINT("e\n");
00635             Read_Node(pstNode->stFileNode.usPrevPeer, &stPrevPeer);
00636             DEBUG_PRINT("f\n");
00637         }
00638
00639         if (INVALID_NODE != pstNode->stFileNode.usNextPeer)
00640         {
00641             DEBUG_PRINT("g\n");
00642             stNextPeer.stFileNode.usPrevPeer = pstNode->
stFileNode.usPrevPeer;
00643             Write_Node(pstNode->stFileNode.usNextPeer, &stNextPeer);
00644             DEBUG_PRINT("h\n");
00645         }
00646
00647         if (INVALID_NODE != pstNode->stFileNode.usPrevPeer)
00648         {
00649             DEBUG_PRINT("i\n");
00650             stPrevPeer.stFileNode.usNextPeer = pstNode->
stFileNode.usNextPeer;
00651             Write_Node(pstNode->stFileNode.usPrevPeer, &stPrevPeer);
00652             DEBUG_PRINT("j\n");
00653         }
00654     }
00655     pstNode->stFileNode.usParent = INVALID_NODE;
00656     pstNode->stFileNode.usPrevPeer = INVALID_NODE;
00657     pstNode->stFileNode.usNextPeer = INVALID_NODE;
00658 }
00659
00660
00661 //-----
00662 K_USHORT NLFS::Delete_Folder(const K_CHAR *szPath_)
00663 {
00664     K_USHORT usNode = Find_File(szPath_);
00665     NLFS_Node_t stNode;
00666
00667     if (INVALID_NODE == usNode)
00668     {
00669         DEBUG_PRINT("Delete_Folder: File not found!\n");
00670         return INVALID_NODE;
00671     }
00672     if (FS_ROOT_BLOCK == usNode || FS_CONFIG_BLOCK == usNode)
00673     {
00674         DEBUG_PRINT("Delete_Folder: Cannot delete root!\n");
00675         return INVALID_NODE;
00676     }
00677
00678     Read_Node(usNode, &stNode);
00679
00680     if (NLFS_NODE_FILE == stNode.eBlockType)
00681     {
00682         DEBUG_PRINT("Delete_Folder: Path is not a Folder (is it a file?)");

```

```

00683         return INVALID_NODE;
00684     }
00685
00686     if (INVALID_NODE != stNode.stFileNode.usChild)
00687     {
00688         DEBUG_PRINT("Delete_Folder: Folder is not empty!");
00689         return INVALID_NODE;
00690     }
00691
00692     Cleanup_Node_Links(usNode, &stNode);
00693
00694     stNode.eBlockType = NLFS_NODE_FREE;
00695
00696     Write_Node(usNode, &stNode);
00697     Push_Free_Node(usNode);
00698
00699     RootSync();
00700
00701     return usNode;
00702 }
00703
00704 //-----
00705 K_USHORT NLFS::Delete_File( const K_CHAR *szPath_)
00706 {
00707     K_USHORT usNode = Find_File(szPath_);
00708     K_ULONG ulCurr;
00709     K_ULONG ulPrev;
00710     NLFS_Node_t stNode;
00711     NLFS_Block_t stBlock;
00712
00713     if (INVALID_NODE == usNode)
00714     {
00715         DEBUG_PRINT("Delete_File: File not found!\n");
00716         return INVALID_NODE;
00717     }
00718     if (FS_ROOT_BLOCK == usNode || FS_CONFIG_BLOCK == usNode)
00719     {
00720         DEBUG_PRINT("Delete_File: Cannot delete root!\n");
00721         return INVALID_NODE;
00722     }
00723
00724     Read_Node(usNode, &stNode);
00725
00726     if (NLFS_NODE_DIR == stNode.eBlockType)
00727     {
00728         DEBUG_PRINT("Delete_File: Path is not a file (is it a directory?)");
00729         return INVALID_NODE;
00730     }
00731
00732     Cleanup_Node_Links(usNode, &stNode);
00733     ulCurr = stNode.stFileNode.ulFirstBlock;
00734
00735     while (INVALID_BLOCK != ulCurr)
00736     {
00737         Read_Block_Header(ulCurr, &stBlock);
00738
00739         ulPrev = ulCurr;
00740         ulCurr = stBlock.ulNextBlock;
00741
00742         Push_Free_Block(ulPrev);
00743     }
00744
00745     stNode.eBlockType = NLFS_NODE_FREE;
00746
00747     Write_Node(usNode, &stNode);
00748     Push_Free_Node(usNode);
00749
00750     RootSync();
00751
00752     return usNode;
00753 }
00754
00755 //-----
00756 void NLFS::Format(NLFS_Host_t *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_,
00757 K_USHORT usDataBlockSize_)
00758 {
00759     K_ULONG i;
00760     K_ULONG ulNumBlocks;
00761
00762     NLFS_Node_t stFileNode;
00763     NLFS_Block_t stFileBlock;
00764
00765     // Compute number of data blocks (based on FS Size and the number of file blocks)
00766     ulTotalSize_ -= ((K_ULONG)usNumFiles_) * sizeof(stFileNode);
00767     ulNumBlocks = ulTotalSize_ / (((K_ULONG)usDataBlockSize_) + (sizeof(stFileBlock) - 1) + 3) & ~3);
00768
00769     DEBUG_PRINT("Number of blocks %d\n", ulNumBlocks);

```

```

00769
00770 // Set up the local_pointer -> this is used for the low-level, platform-specific
00771 // bits, allowing the FS to be used on RAM buffers, EEPROM's, networks, etc.
00772 m_puHost = puHost_;
00773
00774 // Set the local copies of the data block byte-offset, as well as the data-block size
00775 m_stLocalRoot.usNumFiles = usNumFiles_;
00776 m_stLocalRoot.usNumFilesFree = m_stLocalRoot.
    usNumFiles - 2;
00777 m_stLocalRoot.usNextFreeNode = 2;
00778
00779 m_stLocalRoot.ulNumBlocks = ulNumBlocks;
00780 m_stLocalRoot.ulNumBlocksFree = ulNumBlocks;
00781 m_stLocalRoot.ulNextFreeBlock = 0;
00782
00783 m_stLocalRoot.ulBlockSize = (((K_ULONG)usDataBlockSize_) + 3) & ~3;
00784 m_stLocalRoot.ulBlockOffset = (((K_ULONG)usNumFiles_) * sizeof(
    NLFS_Node_t));
00785 m_stLocalRoot.ulDataOffset = m_stLocalRoot.
    ulBlockOffset
00786 + (((K_ULONG)ulNumBlocks) * sizeof(
    NLFS_Block_t));
00787
00788 // Create root data block node
00789 MemUtil::CopyMemory(&(stFileNode.stRootNode), &
    m_stLocalRoot, sizeof(m_stLocalRoot));
00790 stFileNode.eBlockType = NLFS_NODE_ROOT;
00791
00792 DEBUG_PRINT("Writing root node\n");
00793 Write_Node(0, &stFileNode);
00794 DEBUG_PRINT("Done\n");
00795
00796 // Create root mount point (directory)
00797 MemUtil::SetMemory(&stFileNode, 0, sizeof(stFileNode));
00798 stFileNode.eBlockType = NLFS_NODE_DIR;
00799
00800 stFileNode.stFileNode.acFileName[0] = '/';
00801
00802 stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00803 stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00804 stFileNode.stFileNode.ucGroup = 0;
00805 stFileNode.stFileNode.ucUser = 0;
00806 stFileNode.stFileNode.usPerms = PERM_U_ALL | PERM_G_ALL | PERM_O_ALL;
00807
00808 stFileNode.stFileNode.usParent = INVALID_NODE;
00809 stFileNode.stFileNode.usChild = INVALID_NODE;
00810
00811 stFileNode.stFileNode.ulAllocSize = 0;
00812 stFileNode.stFileNode.ulFileSize = 0;
00813
00814 stFileNode.stFileNode.ulFirstBlock = INVALID_BLOCK;
00815 stFileNode.stFileNode.ulLastBlock = INVALID_BLOCK;
00816
00817 DEBUG_PRINT("Writing mount point\n");
00818 Write_Node(1, &stFileNode);
00819 DEBUG_PRINT("Done\n");
00820
00821 stFileNode.stFileNode.acFileName[0] = 0;
00822 // Format nodes
00823 for (i = 2; i < usNumFiles_; i++)
00824 {
00825     stFileNode.eBlockType = NLFS_NODE_FREE;
00826     if (i != usNumFiles_ - 1)
00827     {
00828         stFileNode.stFileNode.usNextPeer = (K_USHORT)(i + 1);
00829     }
00830     else
00831     {
00832         stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00833     }
00834     Write_Node(i, &stFileNode);
00835 }
00836
00837 DEBUG_PRINT("File nodes formatted\n");
00838
00839 // Format file blocks
00840 MemUtil::SetMemory(&stFileBlock, 0, sizeof(stFileBlock));
00841
00842 DEBUG_PRINT("Writing file blocks\n");
00843 for (i = 0; i < ulNumBlocks; i++)
00844 {
00845     if (i == ulNumBlocks - 1)
00846     {
00847         stFileBlock.ulNextBlock = INVALID_BLOCK;
00848     }
00849     else
00850     {

```

```

00851         stFileBlock.ulNextBlock = i + 1;
00852     }
00853
00854     Write_Block_Header(i, &stFileBlock);
00855 }
00856 }
00857
00858 //-----
00859 void NLFS::Mount(NLFS_Host_t *puHost_)
00860 {
00861     NLFS_Node_t stRootNode;
00862
00863     m_puHost = puHost_;
00864     DEBUG_PRINT("Remounting FS %X - reading config node\n", puHost_);
00865
00866     // Reload the root block into the local cache
00867     Read_Node(FS_CONFIG_BLOCK, &stRootNode);
00868
00869     DEBUG_PRINT("Copying config node\n");
00870     MemUtil::CopyMemory(&m_stLocalRoot, &(stRootNode.
00871 stRootNode), sizeof(m_stLocalRoot));
00872
00873     DEBUG_PRINT("Block Size", m_stLocalRoot.ulBlockSize );
00874     DEBUG_PRINT("Data Offset", m_stLocalRoot.ulDataOffset );
00875     DEBUG_PRINT("Block Offset", m_stLocalRoot.ulBlockOffset );
00876 }
00877
00878 //-----
00879 void NLFS::RootSync()
00880 {
00881     NLFS_Node_t stRootNode;
00882
00883     MemUtil::CopyMemory(&(stRootNode.stRootNode), &
00884 m_stLocalRoot, sizeof(m_stLocalRoot));
00885     stRootNode.eBlockType = NLFS_NODE_ROOT;
00886     Write_Node(FS_CONFIG_BLOCK, &stRootNode);
00887 }
00888
00889 //-----
00890 K_USHORT NLFS::GetFirstChild( K_USHORT usNode_ )
00891 {
00892     NLFS_Node_t stTemp;
00893     if (!usNode_ || INVALID_NODE == usNode_)
00894     {
00895         return INVALID_NODE;
00896     }
00897     Read_Node(usNode_, &stTemp);
00898
00899     if (stTemp.eBlockType != NLFS_NODE_DIR)
00900     {
00901         return INVALID_NODE;
00902     }
00903
00904     return stTemp.stFileNode.usChild;
00905 }
00906
00907 //-----
00908 K_USHORT NLFS::GetNextPeer( K_USHORT usNode_ )
00909 {
00910     NLFS_Node_t stTemp;
00911     if (!usNode_ || INVALID_NODE == usNode_)
00912     {
00913         return INVALID_NODE;
00914     }
00915     Read_Node(usNode_, &stTemp);
00916     return stTemp.stFileNode.usNextPeer;
00917 }
00918
00919 //-----
00920 K_BOOL NLFS::GetStat( K_USHORT usNode_, NLFS_File_Stat_t *pstStat_)
00921 {
00922     NLFS_Node_t stTemp;
00923     if (!usNode_ || INVALID_NODE == usNode_)
00924     {
00925         return false;
00926     }
00927     Read_Node(usNode_, &stTemp);
00928     pstStat_>ulAllocSize = stTemp.stFileNode.ulAllocSize;
00929     pstStat_>ulFileSize = stTemp.stFileNode.ulFileSize;
00930     pstStat_>ucGroup = stTemp.stFileNode.ucGroup;
00931     pstStat_>ucUser = stTemp.stFileNode.ucUser;
00932     pstStat_>usPerms = stTemp.stFileNode.usPerms;
00933     MemUtil::CopyMemory(pstStat_>acFileName, stTemp.
00934 stFileNode.acFileName, 16);
00935     return true;
00936 }

```

00936

17.123 /home/mo/mark3-source/embedded/stage/src/nlfs.h File Reference

Nice Little Filesystem ([NLFS](#)) - a simple, embeddable filesystem.

```
#include "kerneltypes.h"
#include <stdint.h>
```

Classes

- struct [NLFS_File_Node_t](#)
Data structure for the "file" FS-node type.
- struct [NLFS_Root_Node_t](#)
Data structure for the Root-configuration FS-node type.
- struct [NLFS_Node_t](#)
Filesystem node data structure.
- struct [NLFS_Block_t](#)
Block data structure.
- union [NLFS_Host_t](#)
Union used for managing host-specific pointers/data-types.
- struct [NLFS_File_Stat_t](#)
Structure used to report the status of a given file.
- class [NLFS](#)
Nice Little File System class.

Macros

- #define [PERM_UX](#) (0x0001)
Permission bit definitions.
- #define [PERM_UW](#) (0x0002)
- #define [PERM_UR](#) (0x0004)
- #define [PERM_U_ALL](#) ([PERM_UX](#) | [PERM_UW](#) | [PERM_UR](#))
- #define [PERM_GX](#) (0x0008)
- #define [PERM_GW](#) (0x0010)
- #define [PERM_GR](#) (0x0020)
- #define [PERM_G_ALL](#) ([PERM_GX](#) | [PERM_GW](#) | [PERM_GR](#))
- #define [PERM_OX](#) (0x0040)
- #define [PERM_OW](#) (0x0080)
- #define [PERM_OR](#) (0x0100)
- #define [PERM_O_ALL](#) ([PERM_OX](#) | [PERM_OW](#) | [PERM_OR](#))
- #define [INVALID_BLOCK](#) (0xFFFFFFFF)
- #define [INVALID_NODE](#) (0xFFFF)
- #define [FILE_NAME_LENGTH](#) (16)
- #define [FS_CONFIG_BLOCK](#) (0)
- #define [FS_ROOT_BLOCK](#) (1)

Enumerations

- enum `NLFS_Type_t` {
`NLFS_NODE_FREE`, `NLFS_NODE_ROOT`, `NLFS_NODE_FILE`, `NLFS_NODE_DIR`,
`FILE_BLOCK_COUNTS` }

Enumeration describing the various types of filesystem nodes used by [NLFS](#).

17.123.1 Detailed Description

Nice Little Filesystem ([NLFS](#)) - a simple, embeddable filesystem. Introduction to the Nice-Little-Filesystem ([NLFS](#))
[NLFS](#) is yet-another filesystem intended for use in embedded applications.

It is intended to be portable, lightweight, and flexible in terms of supporting different types of physical storage media. In order to ensure that it's easily embeddable, there are no external library dependencies, aside from library code provided elsewhere in Mark3 (namely the [MemUtil](#) utility class). Balancing code-size with features and functionality is also a tradeoff - [NLFS](#) supports basic operations (create file, create directory, read, write, seek, and delete), without a lot of other bells and whistles. One other feature built into the filesystem is posix-style user-group permissions. While the APIs in the [NLFS](#) classes do not enforce permissions explicitly, application-specific implementations of [NLFS](#) can enforce permissions based on facilities based on the security mechanisms built into the host OS.

The original purpose of this filesystem was to provide a flexible way of packaging files for read-only use within Mark3 (such as scripts and compiled DCPU-16 objects). However, there are all sorts of purposes for this type of filesystem - essentially, any application where a built-in file manifest or resource container format.

[NLFS](#) is a block-based filesystem, composed of three separate regions of data structures within a linearly-addressed blob of storage. These regions are represented on the physical storage in the following order:

[File Nodes][Data Block Headers][Block Data]

The individual regions are as follows:

1) File Nodes

This region is composed of a linear array of equally-sized file-node ([NLFS_Node_t](#)) structures, starting at byte offset 0 in the underlying media.

Each node defines a particular file or directory within the filesystem. Because of the linear layout of the filesystem, the file nodes are all pre-allocated during the time of filesystem creation. As a result, care should be taken to ensure enough file nodes are allocated to meet the needs of your application, without wasting space in the filesystem for nodes that will never be needed.

The first two nodes (node 0 and node 1) are special in the [NLFS](#) implementation.

Node 0 is also known as the root filesystem node. This block contains a different internal data structure from other file nodes, and stores the configuration information for the particular filesystem, such as the number of file nodes, file blocks, block sizes, as well as indexes of the first free file and block nodes in the filesystem. With this information, it is possible to re-mount a filesystem created once in another location.

Node 1 is the mount-point for the filesystem, and is the root directory under which all other files and directories are found. By default Node 1 is simply named "/".

2) Block Headers

The block header region of the system comes after the file node region, and consists of a linear array of block node data structures. All storage in a filesystem not allocated towards file nodes is automatically allocated towards data blocks, and for each data block allocated, there is a block node data structure allocated within the block node region.

The [NLFS_Block_t](#) data structure contains a link to the next node in a block chain. If the block is free, the link points to the index of the next free block in the filesystem. If allocated, the link points to the index of the next block in the file. This structure also contains flags which indicate whether or not a block is free or allocated, and other flags used for filesystem continuity checks.

3) Block Data

The block data region is the last linear range in the filesystem, and consists of equally-sized blocks in the filesystem. Each block consists of a region of raw physical storage, without any additional metadata.

The contents of any files read or written to the filesystem is stored within the blocks in this region.

The [NLFS](#) Class has a number of virtual methods, which require that a user provides an implementaiton appropriate for the underlying physical storage medium from within a class inheriting [NLFS.s](#)

An example implementation for a RAM-based filesystem is provided in the [NLFS_RAM](#) class located within [nlfs_ram.-cpp](#).

Definition in file [nlfs.h](#).

17.123.2 Enumeration Type Documentation

17.123.2.1 enum NLFS_Type_t

Enumeration describing the various types of filesystem nodes used by [NLFS](#).

A fileysstem node is a fixed-sized data structure consisting of a type specifier, and a union of the data structures representing each possible block type.

Enumerator

NLFS_NODE_FREE File node is free.

NLFS_NODE_ROOT Root filesystem descriptor.

NLFS_NODE_FILE File node.

NLFS_NODE_DIR Directory node.

Definition at line [152](#) of file [nlfs.h](#).

17.124 nlfs.h

```

00001  /*=====
00002
00003  _____
00004  |   \   |   \   |   \   |   \   |   \   |   \   |
00005  |   \   |   \   |   \   |   \   |   \   |   \   |
00006  |   \   |   \   |   \   |   \   |   \   |   \   |
00007  |   \   |   \   |   \   |   \   |   \   |   \   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00108  #ifndef __NLFS_H__
00109  #define __NLFS_H__
00110
00111  #include "kerneltypes.h"
00112  #include <stdint.h>
00113
00114  class NLFS_File;
00115
00116  //-----
00120  #define PERM_UX      (0x0001)
00121  #define PERM_UW      (0x0002)
00122  #define PERM_UR      (0x0004)
00123  #define PERM_U_ALL   ( PERM_UX | PERM_UW | PERM_UR )
00124
00125  #define PERM_GX      (0x0008)
00126  #define PERM_GW      (0x0010)
00127  #define PERM_GR      (0x0020)
00128  #define PERM_G_ALL   ( PERM_GX | PERM_GW | PERM_GR )
00129
00130  #define PERM_OX      (0x0040)
00131  #define PERM_OW      (0x0080)
00132  #define PERM_OR      (0x0100)
00133  #define PERM_O_ALL   ( PERM_OX | PERM_OW | PERM_OR )
00134
00135  //-----

```

```

00136 #define INVALID_BLOCK    (0xFFFFFFFF)
00137 #define INVALID_NODE      (0xFFFF)
00138
00139 //-----
00140 #define FILE_NAME_LENGTH  (16)
00141
00142 #define FS_CONFIG_BLOCK    (0)
00143 #define FS_ROOT_BLOCK     (1)
00144
00145 //-----
00152 typedef enum
00153 {
00154     NLFS_NODE_FREE,
00155     NLFS_NODE_ROOT,
00156     NLFS_NODE_FILE,
00157     NLFS_NODE_DIR,
00158 } --
00159 FILE_BLOCK_COUNTS
00160 } NLFS_Type_t;
00161
00162 //-----
00168 typedef struct
00169 {
00170     K_CHAR      acFileName[16];
00171
00172     K_USHORT    usNextPeer;
00173     K_USHORT    usPrevPeer;
00174
00175     K_UCHAR     ucGroup;
00176     K_UCHAR     ucUser;
00177     K_USHORT    usPerms;
00178
00179     K_USHORT    usParent;
00180     K_USHORT    usChild;
00181
00182 } -- File-specific
00183 K_ULONG        ulAllocSize;
00184 K_ULONG        ulFileSize;
00185
00186 K_ULONG        ulFirstBlock;
00187 K_ULONG        ulLastBlock;
00188 } NLFS_File_Node_t;
00189
00190 //-----
00194 typedef struct
00195 {
00196     K_USHORT    usNumFiles;
00197     K_USHORT    usNumFilesFree;
00198     K_USHORT    usNextFreeNode;
00199
00200     K_ULONG     ulNumBlocks;
00201     K_ULONG     ulNumBlocksFree;
00202     K_ULONG     ulNextFreeBlock;
00203
00204     K_ULONG     ulBlockSize;
00205     K_ULONG     ulBlockOffset;
00206     K_ULONG     ulDataOffset;
00207 } NLFS_Root_Node_t;
00208
00209 //-----
00215 typedef struct
00216 {
00217     NLFS_Type_t    eBlockType;
00218
00219     union          // Depending on the block type, we use one of the following
00220     {
00221         NLFS_Root_Node_t    stRootNode;
00222         NLFS_File_Node_t    stFileNode;
00223     };
00224 } NLFS_Node_t;
00225
00226 //-----
00232 typedef struct
00233 {
00234     K_ULONG        ulNextBlock;
00235     union
00236     {
00237         K_UCHAR     ucFlags;
00238         struct
00239         {
00240             unsigned int    uAllocated;
00241             unsigned int    uCheckBit;
00242         };
00243     };
00244 } NLFS_Block_t;
00245
00246

```



```

00247 //-----
00253 typedef union
00254 {
00255     void *pvData;
00256     uint32_t u32Data;
00257     uint64_t u64Data;
00258     K_ADDR kaData;
00259 } NLFS_Host_t;
00260
00261
00262 //-----
00266 typedef struct
00267 {
00268     K_ULONG    ulAllocSize;
00269     K_ULONG    ulFileSize;
00270     K_USHORT   usPerms;
00271     K_UCHAR    ucUser;
00272     K_UCHAR    ucGroup;
00273     K_CHAR     acFileName[16];
00274 } NLFS_File_Stat_t;
00275
00276 //-----
00280 class NLFS
00281 {
00282 friend class NLFS_File;
00283 public:
00284
00311     void Format(NLFS_Host_t *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_, K_USHORT
        usDataBlockSize_);
00312
00318     void Mount(NLFS_Host_t *puHost_);
00319
00326     K_USHORT Create_File(const K_CHAR *szPath_);
00327
00334     K_USHORT Create_Dir(const K_CHAR *szPath_);
00335
00341     K_USHORT Delete_File(const K_CHAR *szPath_);
00342
00348     K_USHORT Delete_Folder(const K_CHAR *szPath_);
00349
00356     void Cleanup_Node_Links(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00357
00364     K_USHORT Find_Parent_Dir(const K_CHAR *szPath_);
00365
00371     K_USHORT Find_File(const K_CHAR *szPath_);
00372
00376     void Print(void);
00377
00382     K_ULONG GetBlockSize(void) { return m_stLocalRoot.
        ulBlockSize; }
00383
00388     K_ULONG GetNumBlocks(void) { return m_stLocalRoot.
        ulNumBlocks; }
00389
00395     K_ULONG GetNumBlocksFree(void) { return m_stLocalRoot.
        ulNumBlocksFree; }
00396
00401     K_ULONG GetNumFiles(void) { return m_stLocalRoot.
        usNumFiles; }
00402
00407     K_USHORT GetNumFilesFree(void) { return m_stLocalRoot.
        usNumFilesFree; }
00408
00409
00417     K_USHORT GetFirstChild( K_USHORT usNode_ );
00418
00424     K_USHORT GetNextPeer( K_USHORT usNode_ );
00425
00432     K_BOOL GetStat( K_USHORT usNode_, NLFS_File_Stat_t *pstStat_);
00433
00434 protected:
00435
00442     K_CHAR Find_Last_Slash(const K_CHAR *szPath_);
00443
00451     K_BOOL File_Names_Match(const K_CHAR *szPath_, NLFS_Node_t *pstNode_);
00452
00459     virtual void Read_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_) = 0;
00460
00467     virtual void Write_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_) = 0;
00468
00475     virtual void Read_Block_Header(K_ULONG ulBlock_,
        NLFS_Block_t *pstBlock_) = 0;
00476
00483     virtual void Write_Block_Header(K_ULONG ulBlock_,
        NLFS_Block_t *pstFileBlock_) = 0;
00484
00494     virtual void Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_) =

```



```

00022 #define DEBUG          0
00023
00024 #if DEBUG
00025     #include <stdio.h>
00026     #include <stdlib.h>
00027     #define DEBUG_PRINT    printf
00028 #else
00029     #define DEBUG_PRINT(...)
00030 #endif
00031
00032
00033 #endif // NLFS_CONFIG_H

```

17.127 /home/mo/mark3-source/embedded/stage/src/nlfs_file.cpp File Reference

Nice Little Filesystem - File Access Class.

```

#include "kerneltypes.h"
#include "memutil.h"
#include "nlfs_file.h"
#include "nlfs.h"
#include "nlfs_config.h"

```

17.127.1 Detailed Description

Nice Little Filesystem - File Access Class.

Definition in file [nlfs_file.cpp](#).

17.128 nlfs_file.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "kerneltypes.h"
00020 #include "memutil.h"
00021 #include "nlfs_file.h"
00022 #include "nlfs.h"
00023 #include "nlfs_config.h"
00024
00025 //-----
00026 int NLFS_File::Open(NLFS *pclFS_, const K_CHAR *szPath_, NLFS_File_Mode_t eMode_)
00027 {
00028     K_USHORT usNode;
00029     usNode = pclFS_->Find_File(szPath_);
00030
00031     if (INVALID_NODE == usNode)
00032     {
00033         DEBUG_PRINT("file does not exist in path\n");
00034         if (eMode_ & NLFS_FILE_CREATE)
00035         {
00036             DEBUG_PRINT("Attempt to create\n");
00037             usNode = pclFS_->Create_File(szPath_);
00038             if (INVALID_NODE == usNode)
00039             {
00040                 DEBUG_PRINT("unable to create node in path\n");
00041                 return -1;
00042             }
00043         }
00044     }
00045     else
00046     {

```

```

00046         return -1;
00047     }
00048 }
00049
00050 DEBUG_PRINT("Current Node: %d\n", usNode);
00051
00052 m_pclFileSystem = pclFS_;
00053 m_pclFileSystem->Read_Node(usNode, &m_stNode);
00054
00055 m_usFile = usNode;
00056
00057 if (eMode_ & NLFS_FILE_APPEND)
00058 {
00059     if (!(eMode_ & NLFS_FILE_WRITE))
00060     {
00061         DEBUG_PRINT("Open file for append in read-only mode? Why!\n");
00062         return -1;
00063     }
00064     if (-1 == Seek(m_stNode.stFileNode.ulFileSize))
00065     {
00066         DEBUG_PRINT("file open failed - error seeking to EOF for append\n");
00067         return -1;
00068     }
00069 }
00070
00071 else if (eMode_ & NLFS_FILE_TRUNCATE)
00072 {
00073     if (!(eMode_ & NLFS_FILE_WRITE))
00074     {
00075         DEBUG_PRINT("Truncate file in read-only mode? Why!\n");
00076         return -1;
00077     }
00078
00079 K_ULONG ulCurr = m_stNode.stFileNode.ulFirstBlock;
00080 K_ULONG ulPrev = ulCurr;
00081
00082 // Go through and clear all blocks allocated to the file
00083 while (INVALID_BLOCK != ulCurr)
00084 {
00085     NLFS_Block_t stBlock;
00086     pclFS->Read_Block_Header(ulCurr, &stBlock);
00087
00088     ulPrev = ulCurr;
00089     ulCurr = stBlock.ulNextBlock;
00090
00091     pclFS->Push_Free_Block(ulPrev);
00092 }
00093
00094 m_ulOffset = 0;
00095 m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00096 }
00097 else
00098 {
00099     // Open file to beginning of file, regardless of mode.
00100     m_ulOffset = 0;
00101     m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00102 }
00103
00104 m_ucFlags = eMode_;
00105
00106 DEBUG_PRINT("Current Block: %d\n", m_ulCurrentBlock);
00107 DEBUG_PRINT("file open OK\n");
00108 return 0;
00109 }
00110
00111 //-----
00112 int NLFS_File::Seek(K_ULONG ulOffset_)
00113 {
00114     NLFS_Block_t stBlock;
00115     m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00116     m_ulOffset = ulOffset_;
00117
00118     if (INVALID_NODE == m_usFile)
00119     {
00120         DEBUG_PRINT("Error - invalid file");
00121         return -1;
00122     }
00123
00124     if (INVALID_BLOCK == m_ulCurrentBlock)
00125     {
00126         DEBUG_PRINT("Invalid block\n");
00127         m_ulOffset = 0;
00128         return -1;
00129     }

```

```

00130
00131     m_pclFileSystem->Read_Block_Header(
00132         m_ulCurrentBlock, &stBlock);
00133     while (ulOffset_ >= m_pclFileSystem->GetBlockSize())
00134     {
00135         ulOffset_ -= m_pclFileSystem->GetBlockSize();
00136         m_ulCurrentBlock = stBlock.ulNextBlock;
00137         if ((ulOffset_) && (INVALID_BLOCK == m_ulCurrentBlock))
00138         {
00139             m_ulCurrentBlock = m_stNode.stFileNode.
00140             ulFirstBlock;
00141             m_ulOffset = 0;
00142             return -1;
00143         }
00144         m_pclFileSystem->Read_Block_Header(
00145             m_ulCurrentBlock, &stBlock);
00146     }
00147     m_ulOffset = ulOffset_;
00148     return 0;
00149 }
00150 //-----
00151 int NLFS_File::Read(void *pvBuf_, K_ULONG ulLen_)
00152 {
00153     K_ULONG ulBytesLeft;
00154     K_ULONG ulOffset;
00155     K_ULONG ulRead = 0;
00156     K_BOOL bBail = false;
00157
00158     K_CHAR *szCharBuf = (K_CHAR*)pvBuf_;
00159
00160     if (INVALID_NODE == m_usFile)
00161     {
00162         DEBUG_PRINT("Error - invalid file");
00163         return -1;
00164     }
00165
00166     if (!(NLFS_FILE_READ & m_ucFlags))
00167     {
00168         DEBUG_PRINT("Error - file not open for read\n");
00169         return -1;
00170     }
00171
00172     DEBUG_PRINT("Reading: %d bytes from file\n", ulLen_);
00173     while (ulLen_ && !bBail)
00174     {
00175         ulOffset = m_ulOffset & (m_pclFileSystem->
00176             GetBlockSize() - 1);
00177         ulBytesLeft = m_pclFileSystem->GetBlockSize() - ulOffset;
00178         if (ulBytesLeft > ulLen_)
00179         {
00180             ulBytesLeft = ulLen_;
00181         }
00182         if (m_ulOffset + ulBytesLeft >= m_stNode.stFileNode.
00183             ulFileSize)
00184         {
00185             ulBytesLeft = m_stNode.stFileNode.ulFileSize -
00186             m_ulOffset;
00187             bBail = true;
00188         }
00189         DEBUG_PRINT( "%d bytes left in block, %d len, %x block\n", ulBytesLeft, ulLen_,
00190             m_ulCurrentBlock);
00191         if (ulBytesLeft && ulLen_ && (INVALID_BLOCK != m_ulCurrentBlock))
00192         {
00193             m_pclFileSystem->Read_Block(
00194                 m_ulCurrentBlock, ulOffset, (void*)szCharBuf, ulBytesLeft );
00195
00196             ulRead += ulBytesLeft;
00197             ulLen_ -= ulBytesLeft;
00198             szCharBuf += ulBytesLeft;
00199             m_ulOffset += ulBytesLeft;
00200             DEBUG_PRINT( "%d bytes to go\n", ulLen_);
00201         }
00202         if (ulLen_)
00203         {
00204             DEBUG_PRINT("reading next node\n");
00205             NLFS_Block_t stBlock;
00206             m_pclFileSystem->Read_Block_Header(
00207                 m_ulCurrentBlock, &stBlock);
00208             m_ulCurrentBlock = stBlock.ulNextBlock;
00209         }
00210         if (INVALID_BLOCK == m_ulCurrentBlock)
00211         {

```

```

00208         break;
00209     }
00210
00211 }
00212 DEBUG_PRINT("Return :%d bytes read\n", ulRead);
00213 return ulRead;
00214 }
00215
00216 //-----
00217 int NLFS_File::Write(void *pvBuf_, K_ULONG ulLen_)
00218 {
00219     K_ULONG ulBytesLeft;
00220     K_ULONG ulOffset;
00221     K_ULONG ulWritten = 0;
00222     K_CHAR *szCharBuf = (K_CHAR*)pvBuf_;
00223
00224     if (INVALID_NODE == m_usFile)
00225     {
00226         DEBUG_PRINT("Error - invalid file");
00227         return -1;
00228     }
00229
00230     if (!(NLFS_FILE_WRITE & m_ucFlags))
00231     {
00232         DEBUG_PRINT("Error - file not open for write\n");
00233         return -1;
00234     }
00235
00236     DEBUG_PRINT("writing: %d bytes to file\n", ulLen_);
00237     while (ulLen_)
00238     {
00239         ulOffset = m_ulOffset & (m_pclFileSystem->
00240         GetBlockSize() - 1);
00241         ulBytesLeft = m_pclFileSystem->GetBlockSize() - ulOffset;
00242         if (ulBytesLeft > ulLen_)
00243         {
00244             ulBytesLeft = ulLen_;
00245         }
00246         if (ulBytesLeft && ulLen_ && (INVALID_BLOCK != m_ulCurrentBlock))
00247         {
00248             m_pclFileSystem->Write_Block(
00249             m_ulCurrentBlock, ulOffset, (void*)szCharBuf, ulBytesLeft );
00250             ulWritten += ulBytesLeft;
00251             ulLen_ -= ulBytesLeft;
00252             szCharBuf += ulBytesLeft;
00253             m_stNode.stFileNode.ulFileSize += ulBytesLeft;
00254             m_ulOffset += ulBytesLeft;
00255             DEBUG_PRINT( "%d bytes to go\n", ulLen_);
00256         }
00257         if (!ulLen_)
00258         {
00259             m_pclFileSystem->Write_Node(m_usFile, &
00260             m_stNode);
00261         }
00262         else
00263         {
00264             DEBUG_PRINT("appending\n");
00265             m_ulCurrentBlock = m_pclFileSystem->
00266             Append_Block_To_Node(&m_stNode);
00267         }
00268         DEBUG_PRINT("writing node to file\n");
00269         m_pclFileSystem->Write_Node(m_usFile, &
00270         m_stNode);
00271     }
00272     return ulWritten;
00273 }
00274
00275 //-----
00276 int NLFS_File::Close(void)
00277 {
00278     m_usFile = INVALID_NODE;
00279     m_ulCurrentBlock = INVALID_BLOCK;
00280     m_ulOffset = 0;
00281     m_ucFlags = 0;
00282     return 0;
00283 }

```

17.129 /home/mo/mark3-source/embedded/stage/src/nlfs_file.h File Reference

NLFS file access class.

```
#include "kerneltypes.h"
#include "nlfs.h"
#include "nlfs_config.h"
```

Classes

- class [NLFS_File](#)
The [NLFS_File](#) class.

Typedefs

- typedef K_UCHAR **NLFS_File_Mode_t**

Enumerations

- enum [NLFS_File_Mode](#) {
 [NLFS_FILE_CREATE](#) = 0x01, [NLFS_FILE_APPEND](#) = 0x02, [NLFS_FILE_TRUNCATE](#) = 0x04, [NLFS_FILE_READ](#) = 0x08,
 [NLFS_FILE_WRITE](#) = 0x10 }

17.129.1 Detailed Description

[NLFS](#) file access class.

Definition in file [nlfs_file.h](#).

17.129.2 Enumeration Type Documentation

17.129.2.1 enum NLFS_File_Mode

Enumerator

NLFS_FILE_CREATE Create the file if it does not exist.

NLFS_FILE_APPEND Open to end of file.

NLFS_FILE_TRUNCATE Truncate file size to 0-bytes.

NLFS_FILE_READ Open file for read.

NLFS_FILE_WRITE Open file for write.

Definition at line 27 of file [nlfs_file.h](#).

17.130 nlfs_file.h

```
00001  /*=====
00002
00003  _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |  / \ / \  |  / \ / \  |  / \ / \  |  / \ / \  |  / \ / \  |
00006  |_/___\_\_|_/___\_\_|_/___\_\_|_/___\_\_|_/___\_\_|_/___\_\_|
00007  |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
```

```

00019 #ifndef __NLFS_FILE_H
00020 #define __NLFS_FILE_H
00021
00022 #include "kerneltypes.h"
00023 #include "nlfs.h"
00024 #include "nlfs_config.h"
00025
00026 //-----
00027 typedef enum
00028 {
00029     NLFS_FILE_CREATE = 0x01,
00030     NLFS_FILE_APPEND = 0x02,
00031     NLFS_FILE_TRUNCATE = 0x04,
00032     NLFS_FILE_READ = 0x08,
00033     NLFS_FILE_WRITE = 0x10
00034 } NLFS_File_Mode;
00035 typedef K_UCHAR NLFS_File_Mode_t;
00036
00037 //-----
00045 class NLFS_File
00046 {
00047
00048 public:
00056     int      Open(NLFS *pclFS_, const K_CHAR *szPath_, NLFS_File_Mode_t eMode_);
00057
00064     int      Read(void *pvBuf_, K_ULONG ulLen_);
00065
00073     int      Write(void *pvBuf_, K_ULONG ulLen_);
00074
00080     int      Seek(K_ULONG ulOffset_);
00081
00086     int      Close(void);
00087
00088 private:
00089     NLFS      *m_pclFileSystem;
00090     K_ULONG    m_ulOffset;
00091     K_ULONG    m_ulCurrentBlock;
00092     K_USHORT   m_usFile;
00093     NLFS_File_Mode_t m_ucFlags;
00094     NLFS_Node_t m_stNode;
00095 };
00096
00097 #endif // __NLFS_FILE_H

```

17.131 /home/mo/mark3-source/embedded/stage/src/nlfs_ram.cpp File Reference

RAM-based Nice Little Filesystem (NLFS) driver.

```

#include "nlfs.h"
#include "nlfs_ram.h"
#include "memutil.h"
#include "nlfs_config.h"

```

17.131.1 Detailed Description

RAM-based Nice Little Filesystem (NLFS) driver.

Definition in file [nlfs_ram.cpp](#).

17.132 nlfs_ram.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.

```



```

00012 See license.txt for more information
00013 =====*/
00019 #include "nlfs.h"
00020 #include "nlfs_ram.h"
00021 #include "memutil.h"
00022 #include "nlfs_config.h"
00023
00024 //-----
00025 void NLFS_RAM::Read_Node( K_USHORT usNode_, NLFS_Node_t *pstFileNode_)
00026 {
00027     NLFS_Node_t *pstFileNode = (NLFS_Node_t*)(m_puHost->kaData
00028                                             + (usNode_ * sizeof(
00029         NLFS_Node_t)));
00029     MemUtil::CopyMemory(pstFileNode_, pstFileNode, sizeof(
00030         NLFS_Node_t));
00031 }
00032
00033 //-----
00034 void NLFS_RAM::Write_Node(K_USHORT usNode_, NLFS_Node_t *pstFileNode_)
00035 {
00036     NLFS_Node_t *pstFileNode = (NLFS_Node_t*)(m_puHost->kaData
00037                                             + (usNode_ * sizeof(
00038         NLFS_Node_t)));
00039     MemUtil::CopyMemory(pstFileNode, pstFileNode_, sizeof(
00040         NLFS_Node_t));
00041 }
00042 //-----
00043 void NLFS_RAM::Read_Block_Header(K_ULONG ulBlock_,
00044     NLFS_Block_t *pstFileBlock_)
00045 {
00046     NLFS_Block_t *pstFileBlock = (NLFS_Block_t*)(
00047         m_puHost->kaData
00048         + m_stLocalRoot.
00049         ulBlockOffset
00050         + (ulBlock_ * sizeof(
00051             NLFS_Block_t)));
00052     MemUtil::CopyMemory(pstFileBlock_, pstFileBlock, sizeof(
00053         NLFS_Block_t));
00054 }
00055 //-----
00056 void NLFS_RAM::Write_Block_Header(K_ULONG ulBlock_,
00057     NLFS_Block_t *pstFileBlock_)
00058 {
00059     NLFS_Block_t *pstFileBlock = (NLFS_Block_t*)(
00060         m_puHost->kaData
00061         + m_stLocalRoot.
00062         ulBlockOffset
00063         + (ulBlock_ * sizeof(
00064             NLFS_Block_t)));
00065     MemUtil::CopyMemory(pstFileBlock, pstFileBlock_, sizeof(
00066         NLFS_Block_t));
00067 }
00068 //-----
00069 void NLFS_RAM::Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG
00070     ulLen_)
00071 {
00072     void *pvSrc_ = (void*)( m_puHost->kaData
00073                             + m_stLocalRoot.ulDataOffset
00074                             + ulOffset_
00075                             + (ulBlock_ * m_stLocalRoot.ulBlockSize) );
00076     MemUtil::CopyMemory(pvData_, pvSrc_, (K_USHORT)ulLen_);
00077 }
00078 //-----
00079 void NLFS_RAM::Write_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG
00080     ulLen_)
00081 {
00082     void *pvDst_ = (void*)( m_puHost->kaData
00083                             + m_stLocalRoot.ulDataOffset
00084                             + ulOffset_
00085                             + (ulBlock_ * m_stLocalRoot.ulBlockSize) );
00086     MemUtil::CopyMemory(pvDst_, pvData_, (K_USHORT)ulLen_);
00087 }

```

17.133 /home/mo/mark3-source/embedded/stage/src/nlfs_ram.h File Reference

RAM-based Nice Little Filesystem (NLFS) driver.

```
#include "nlfs.h"
```

Classes

- class [NLFS_RAM](#)

The [NLFS_RAM](#) class.

17.133.1 Detailed Description

RAM-based Nice Little Filesystem (NLFS) driver.

Definition in file [nlfs_ram.h](#).

17.134 nlfs_ram.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __NLFS_RAM_H
00020 #define __NLFS_RAM_H
00021
00022 #include "nlfs.h"
00023
00031 class NLFS_RAM : public NLFS
00032 {
00033 private:
00034
00041     virtual void Read_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00042
00049     virtual void Write_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00050
00057     virtual void Read_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstBlock_);
00058
00065     virtual void Write_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstFileBlock_);
00066
00076     virtual void Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_);
00077
00088     void Write_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_);
00089
00090 };
00091
00092 #endif // NLFS_RAM_H
```

17.135 /home/mo/mark3-source/embedded/stage/src/panic_codes.h File Reference

Define and enumerate the possible causes of a kernel panic.

Macros

- **#define PANIC_ASSERT_FAILED (1)**
- **#define PANIC_LIST_UNLINK_FAILED (2)**
- **#define PANIC_STACK_SLACK_VIOLATED (3)**
- **#define PANIC_PEND_LOCK_VIOLATION (4)**
- **#define PANIC_EVENT_LOCK_VIOLATION (5)**
- **#define PANIC_MUTEX_LOCK_VIOLATION (6)**


```

00081     if (m_usIterations)
00082     {
00083         return m_ulCumulative / (K_ULONG)m_usIterations;
00084     }
00085     return 0;
00086 }
00087
00088 //-----
00089 K_ULONG ProfileTimer::GetCurrent()
00090 {
00091
00092     if (m_bActive)
00093     {
00094         K_USHORT usCurrent;
00095         K_ULONG ulEpoch;
00096         CS_ENTER();
00097         usCurrent = Profiler::Read();
00098         ulEpoch = Profiler::GetEpoch();
00099         CS_EXIT();
00100         return ComputeCurrentTicks(usCurrent, ulEpoch);
00101     }
00102     return m_ulCurrentIteration;
00103 }
00104
00105 //-----
00106 K_ULONG ProfileTimer::ComputeCurrentTicks(K_USHORT usCurrent_, K_ULONG
ulEpoch_)
00107 {
00108     K_ULONG ulTotal;
00109     K_ULONG ulOverflows;
00110
00111     ulOverflows = ulEpoch_ - m_ulInitialEpoch;
00112
00113     // More than one overflow...
00114     if (ulOverflows > 1)
00115     {
00116         ulTotal = ((K_ULONG)(ulOverflows-1) * TICKS_PER_OVERFLOW)
+ (K_ULONG)(TICKS_PER_OVERFLOW - m_usInitial) +
(K_ULONG)usCurrent_;
00117
00118     }
00119     // Only one overflow, or one overflow that has yet to be processed
00120     else if (ulOverflows || (usCurrent_ < m_usInitial))
00121     {
00122         ulTotal = (K_ULONG)(TICKS_PER_OVERFLOW - m_usInitial) +
(K_ULONG)usCurrent_;
00123
00124     }
00125     // No overflows, none pending.
00126     else
00127     {
00128         ulTotal = (K_ULONG)(usCurrent_ - m_usInitial);
00129     }
00130
00131     return ulTotal;
00132 }
00133 }
00134
00135 #endif

```

17.139 /home/mo/mark3-source/embedded/stage/src/profile.h File Reference

High-precision profiling timers.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"

```

Classes

- class [ProfileTimer](#)

Profiling timer.

17.139.1 Detailed Description

High-precision profiling timers. Enables the profiling and instrumentation of performance-critical code. Multiple timers can be used simultaneously to enable system-wide performance metrics to be computed in a lightweight manner.

Usage:

```
ProfileTimer clMyTimer;
int i;

clMyTimer.Init();

// Profile the same block of code ten times
for (i = 0; i < 10; i++)
{
    clMyTimer.Start();
    ...
    //Block of code to profile
    ...
    clMyTimer.Stop();
}

// Get the average execution time of all iterations
ulAverageTimer = clMyTimer.GetAverage();

// Get the execution time from the last iteration
ulLastTimer = clMyTimer.GetCurrent();
```

Definition in file [profile.h](#).

17.140 profile.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00053 #ifndef __PROFILE_H__
00054 #define __PROFILE_H__
00055
00056 #include "kerneltypes.h"
00057 #include "mark3cfg.h"
00058 #include "ll.h"
00059
00060 #if KERNEL_USE_PROFILER
00061
00069 class ProfileTimer
00070 {
00071
00072 public:
00073     void Init();
00074
00075     void Start();
00076
00077     void Stop();
00078
00079     K_ULONG GetAverage();
00080
00081     K_ULONG GetCurrent();
00082
00083 private:
00084
00085     K_ULONG ComputeCurrentTicks(K_USHORT usCount_, K_ULONG ulEpoch_);
00086
00087     K_ULONG m_ulCumulative;
00088     K_ULONG m_ulCurrentIteration;
00089     K_USHORT m_usInitial;
00090     K_ULONG m_ulInitialEpoch;
00091     K_USHORT m_usIterations;
00092     K_UCHAR m_bActive;
```

17.141 /home/mo/mark3-source/embedded/stage/src/quantum.cpp File Reference

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024
00025 #include "timerlist.h"
00026 #include "thread.h"
00027 #include "quantum.h"
00028 #include "kernel_debug.h"
00029 //-----
00030 #if defined __FILE_ID__
00031     #undef __FILE_ID__
00032 #endif
```

```

00033 #define __FILE_ID__      QUANTUM_CPP
00034
00035 #if KERNEL_USE_QUANTUM
00036
00037 //-----
00038 static volatile K_BOOL bAddQuantumTimer;    // Indicates that a timer add is pending
00039
00040 //-----
00041 Timer Quantum::m_clQuantumTimer;    // The global timernodelist_t object
00042 K_UCHAR Quantum::m_bActive;
00043 //-----
00044 static void QuantumCallback(Thread *pclThread_, void *pvData_)
00045 {
00046     // Validate thread pointer, check that source/destination match (it's
00047     // in its real priority list). Also check that this thread was part of
00048     // the highest-running priority level.
00049     if (pclThread_>GetPriority() >= Scheduler::GetCurrentThread()->
        GetPriority())
00050     {
00051         if (pclThread_>GetCurrent()->GetHead() != pclThread_>
            GetCurrent()->GetTail() )
00052         {
00053             bAddQuantumTimer = true;
00054             pclThread_>GetCurrent()->PivotForward();
00055         }
00056     }
00057 }
00058
00059 //-----
00060 void Quantum::SetTimer(Thread *pclThread_)
00061 {
00062     m_clQuantumTimer.SetIntervalMSeconds(pclThread_>
        GetQuantum());
00063     m_clQuantumTimer.SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00064     m_clQuantumTimer.SetData(NULL);
00065     m_clQuantumTimer.SetCallback((TimerCallback_t)QuantumCallback);
00066     m_clQuantumTimer.SetOwner(pclThread_);
00067 }
00068
00069 //-----
00070 void Quantum::AddThread(Thread *pclThread_)
00071 {
00072     if (m_bActive)
00073     {
00074         return;
00075     }
00076     // If this isn't the only thread in the list.
00077     if ( pclThread_>GetCurrent()->GetHead() !=
        pclThread_>GetCurrent()->GetTail() )
00078     {
00079         Quantum::SetTimer(pclThread_);
00080         TimerScheduler::Add(&m_clQuantumTimer);
00081         m_bActive = 1;
00082     }
00083 }
00084
00085 //-----
00086 void Quantum::RemoveThread(void)
00087 {
00088     if (!m_bActive)
00089     {
00090         return;
00091     }
00092
00093     // Cancel the current timer
00094     TimerScheduler::Remove(&m_clQuantumTimer);
00095     m_bActive = 0;
00096 }
00097
00098 //-----
00099 void Quantum::UpdateTimer(void)
00100 {
00101     // If we have to re-add the quantum timer (more than 2 threads at the
00102     // high-priority level...)
00103     if (bAddQuantumTimer)
00104     {
00105         // Trigger a thread yield - this will also re-schedule the
00106         // thread *and* reset the round-robin scheduler.
00107         Thread::Yield();
00108         bAddQuantumTimer = false;
00109     }
00110 }
00111
00112
00113 #endif //KERNEL_USE_QUANTUM

```


17.145 /home/mo/mark3-source/embedded/stage/src/scheduler.cpp File Reference

Strict-Priority + Round-Robin thread scheduler implementation.

```
#include "kerneltypes.h"
#include "ll.h"
#include "scheduler.h"
#include "thread.h"
#include "threadport.h"
#include "kernel_debug.h"
```

Macros

- `#define __FILE_ID__ SCHEDULER_CPP`

Variables

- `Thread * g_pstNext`
- `Thread * g_pstCurrent`
- `K_UCHAR g_ucFlag`
- `static const K_UCHAR aucCLZ [16] = {255,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3}`

17.145.1 Detailed Description

Strict-Priority + Round-Robin thread scheduler implementation.

Definition in file [scheduler.cpp](#).

17.146 scheduler.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "ll.h"
00024 #include "scheduler.h"
00025 #include "thread.h"
00026 #include "threadport.h"
00027 #include "kernel_debug.h"
00028 //-----
00029 #if defined __FILE_ID__
00030     #undef __FILE_ID__
00031 #endif
00032 #define __FILE_ID__ SCHEDULER_CPP
00033
00034 //-----
00035 Thread *g_pstNext;
00036 Thread *g_pstCurrent;
00037
00038 //-----
00039 K_BOOL Scheduler::m_bEnabled;
00040 K_BOOL Scheduler::m_bQueuedSchedule;
00041
00042 ThreadList Scheduler::m_clStopList;
00043 ThreadList Scheduler::m_aclPriorities [NUM_PRIORITIES];
```

```

00044 K_UCHAR Scheduler::m_ucPriFlag;
00045
00046 K_UCHAR g_ucFlag;
00047 //-----
00048 static const K_UCHAR aucCLZ[16] = {255,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3};
00049
00050 //-----
00051 void Scheduler::Init()
00052 {
00053     m_ucPriFlag = 0;
00054     for (int i = 0; i < NUM_PRIORITIES; i++)
00055     {
00056         m_aclPriorities[i].SetPriority(i);
00057         m_aclPriorities[i].SetFlagPointer(&
m_ucPriFlag);
00058     }
00059     g_ucFlag = m_ucPriFlag;
00060     m_bQueuedSchedule = false;
00061 }
00062
00063 //-----
00064 void Scheduler::Schedule()
00065 {
00066     K_UCHAR ucPri = 0;
00067
00068     // Figure out what priority level has ready tasks (8 priorities max)
00069     ucPri = aucCLZ[m_ucPriFlag >> 4 ];
00070     if (ucPri == 0xFF) { ucPri = aucCLZ[m_ucPriFlag & 0x0F]; }
00071     else { ucPri += 4; }
00072
00073     // Get the thread node at this priority.
00074     g_pstNext = (Thread*)( m_aclPriorities[ucPri].GetHead() );
00075     g_ucFlag = m_ucPriFlag;
00076
00077     KERNEL_TRACE_1( STR_SCHEDULE_1, (K_USHORT)g_pstNext->GetID() );
00078 }
00079
00080 //-----
00081 void Scheduler::Add(Thread *pclThread_)
00082 {
00083     m_aclPriorities[pclThread_->GetPriority()].Add(pclThread_);
00084     g_ucFlag = m_ucPriFlag;
00085 }
00086
00087 //-----
00088 void Scheduler::Remove(Thread *pclThread_)
00089 {
00090     m_aclPriorities[pclThread_->GetPriority()].Remove(pclThread_);
00091     g_ucFlag = m_ucPriFlag;
00092 }
00093
00094 //-----
00095 K_BOOL Scheduler::SetScheduler(K_BOOL bEnable_)
00096 {
00097     K_BOOL bRet ;
00098     CS_ENTER();
00099     bRet = m_bEnabled;
00100     m_bEnabled = bEnable_;
00101     // If there was a queued scheduler event, dequeue and trigger an
00102     // immediate Yield
00103     if (m_bEnabled && m_bQueuedSchedule)
00104     {
00105         m_bQueuedSchedule = false;
00106         Thread::Yield();
00107     }
00108     CS_EXIT();
00109     return bRet;
00110 }

```

17.147 /home/mo/mark3-source/embedded/stage/src/scheduler.h File Reference

[Thread](#) scheduler function declarations.

```

#include "kerneltypes.h"
#include "thread.h"
#include "threadport.h"

```

Classes

- class [Scheduler](#)

Priority-based round-robin [Thread](#) scheduling, using [ThreadLists](#) for housekeeping.

Macros

- `#define NUM_PRIORITIES (8)`

Variables

- [Thread](#) * `g_pstNext`
- [Thread](#) * `g_pstCurrent`

17.147.1 Detailed Description

[Thread](#) scheduler function declarations. This scheduler implements a very flexible type of scheduling, which has become the defacto industry standard when it comes to real-time operating systems. This scheduling mechanism is referred to as priority round- robin.

From the name, there are two concepts involved here:

- 1) Priority scheduling:

Threads are each assigned a priority, and the thread with the highest priority which is ready to run gets to execute.

- 2) Round-robin scheduling:

Where there are multiple ready threads at the highest-priority level, each thread in that group gets to share time, ensuring that progress is made.

The scheduler uses an array of [ThreadList](#) objects to provide the necessary housekeeping required to keep track of threads at the various priorities. As a result, the scheduler contains one [ThreadList](#) per priority, with an additional list to manage the storage of threads which are in the "stopped" state (either have been stopped, or have not been started yet).

Definition in file [scheduler.h](#).

17.148 scheduler.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00046 #ifndef __SCHEDULER_H__
00047 #define __SCHEDULER_H__
00048
00049 #include "kerneltypes.h"
00050 #include "thread.h"
00051 #include "threadport.h"
00052
00053 extern Thread *g_pstNext;
00054 extern Thread *g_pstCurrent;
00055
00056 #define NUM_PRIORITIES (8)
00057 //-----
00062 class Scheduler

```

```

00063 {
00064 public:
00070     static void Init();
00071
00079     static void Schedule();
00080
00088     static void Add(Thread *pclThread_);
00089
00098     static void Remove(Thread *pclThread_);
00099
00112     static K_BOOL SetScheduler(K_BOOL bEnable_);
00113
00119     static Thread *GetCurrentThread(){ return g_pstCurrent; }
00120
00127     static Thread *GetNextThread(){ return g_pstNext; }
00128
00137     static ThreadList *GetThreadList(K_UCHAR ucPriority_){ return &
m_aclPriorities[ucPriority_]; }
00138
00145     static ThreadList *GetStopList(){ return &m_clStopList; }
00146
00155     static K_UCHAR IsEnabled(){ return m_bEnabled; }
00156
00157     static void QueueScheduler() { m_bQueuedSchedule = true; }
00158
00159 private:
00161     static K_BOOL m_bEnabled;
00162
00164     static K_BOOL m_bQueuedSchedule;
00165
00167     static ThreadList m_clStopList;
00168
00170     static ThreadList m_aclPriorities[NUM_PRIORITIES];
00171
00173     static K_UCHAR m_ucPriFlag;
00174 };
00175 #endif
00176

```

17.149 /home/mo/mark3-source/embedded/stage/src/screen.cpp File Reference

Higher level window management framework.

```
#include "kerneltypes.h"
#include "screen.h"
#include "gui.h"
#include "memutil.h"
```

17.149.1 Detailed Description

Higher level window management framework.

Definition in file [screen.cpp](#).

17.150 screen.cpp

```
00001 /*-----  
00002  
00003      |_____|   |_____|   |_____|   |_____|  
00004      / \    / \    / \    / \    / \  
00005     /   \  /   \  /   \  /   \  /   \  
00006    /_____\/_\_____\/_\_____\/_\_____\  
00007  
00008  
00009 --[Mark3 Realtime Platform]-----  
00010  
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.  
00012 See license.txt for more information  
00013 =====*/  
00014 #include "kerneltypes.h"  
00015 #include "screen.h"  
00016 #include "gui.h"
```

```

00022 #include "memutil.h"
00023
00024 //-----
00025 void Screen::SetManager( ScreenManager *pclScreenManager_ )
00026 {
00027     m_pclScreenManager = pclScreenManager_;
00028 }
00029
00030 //-----
00031 void Screen::SetWindowAffinity( const K_CHAR *szWindowName_ )
00032 {
00033     m_pclWindow = m_pclScreenManager->FindWindowByName( szWindowName_ );
00034 }
00035
00036 //-----
00037 GuiWindow *ScreenManager::FindWindowByName( const K_CHAR *m_szName_
)
00038 {
00039     return m_pclSurface->FindWindowByName( m_szName_ );
00040 }
00041
00042 //-----
00043 Screen *ScreenManager::FindScreenByName( const K_CHAR *szName_ )
00044 {
00045     LinkListNode *pclTempNode = static_cast<LinkListNode*>(
m_clScreenList.GetHead());
00046
00047     while (pclTempNode)
00048     {
00049         if (MemUtil::CompareStrings( szName_, static_cast<Screen*>(pclTempNode)->
GetName()))
00050         {
00051             return static_cast<Screen*>(pclTempNode);
00052         }
00053         pclTempNode = pclTempNode->GetNext();
00054     }
00055     return NULL;
00056 }
00057 }
00058

```

17.151 /home/mo/mark3-source/embedded/stage/src/screen.h File Reference

Higher level window management framework.

```

#include "kerneltypes.h"
#include "gui.h"
#include "ll.h"

```

Classes

- class [Screen](#)
- class [ScreenList](#)
- class [ScreenManager](#)

17.151.1 Detailed Description

Higher level window management framework.

Definition in file [screen.h](#).

17.152 screen.h

```

00001 /*=====
00002
00003 |-----|-----|-----|-----|-----|
00004 |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00005 |  /  \  |  /  \  |  /  \  |  /  \  |  /  \  |

```

Generated on Sat Jan 25 2014 18:56:55 for Mark3 Realtime Kernel by Doxygen

```

00141
00142 private:
00143
00144     ScreenList m_clScreenList;
00145     GuiEventSurface *m_pclSurface;
00146 };
00147
00148 #endif

```

17.153 /home/mo/mark3-source/embedded/stage/src/shell_support.cpp File Reference

Support functions & data structures useful in implementing a shell.

```

#include "kerneltypes.h"
#include "memutil.h"
#include "shell_support.h"

```

17.153.1 Detailed Description

Support functions & data structures useful in implementing a shell.

Definition in file [shell_support.cpp](#).

17.154 shell_support.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00023 #include "kerneltypes.h"
00024 #include "memutil.h"
00025 #include "shell_support.h"
00026
00027 //-----
00028 K_CHAR ShellSupport::RunCommand( CommandLine_t *pstCommand_, const
ShellCommand_t *pastShellCommands_ )
00029 {
00030     K_UCHAR i = 0;
00031     K_UCHAR tmp_len;
00032     while (pastShellCommands_[i].szCommand)
00033     {
00034         tmp_len = MIN(pstCommand_>pstCommand->ucLen,
MemUtil::StringLength(pastShellCommands_[i].szCommand));
00035
00036         if (true == MemUtil::CompareMemory( (const void*)pastShellCommands_[i].
szCommand,
00037                                             (const void*) (pstCommand_>
pstCommand->pcToken),
00038                                             tmp_len ) )
00039         {
00040             pastShellCommands_[i].pfHandler( pstCommand_ );
00041             return 1;
00042         }
00043         i++;
00044     }
00045     return 0;
00046 }
00047
00048 //-----
00049 void ShellSupport::UnescapeToken( Token_t *pstToken_, K_CHAR *szDest_ )
00050 {
00051     const K_CHAR *szSrc = pstToken_>pcToken;
00052     int i;

```



```

00053     int j = 0;
00054     for (i = 0; i < pstToken_>ucLen; i++)
00055     {
00056         //-- Escape characters
00057         if ('\\' == szSrc[i])
00058         {
00059             i++;
00060             if (i >= pstToken_>ucLen)
00061             {
00062                 break;
00063             }
00064             switch (szSrc[i])
00065             {
00066                 case 't':
00067                     szDest_[j++] = '\t';
00068                     break;
00069                 case 'r':
00070                     szDest_[j++] = '\r';
00071                     break;
00072                 case 'n':
00073                     szDest_[j++] = '\n';
00074                     break;
00075                 case ' ':
00076                     szDest_[j++] = ' ';
00077                     break;
00078                 case '\\':
00079                     szDest_[j++] = '\\';
00080                     break;
00081                 case '\"':
00082                     szDest_[j++] = '\"';
00083                     break;
00084                 default:
00085                     break;
00086             }
00087         }
00088         //-- Unescaped quotes
00089         else if ('\"' == szSrc[i])
00090         {
00091             continue;
00092         }
00093         //-- Everything else
00094         else
00095         {
00096             szDest_[j++] = szSrc[i];
00097         }
00098     }
00099     //-- Null-terminate the string
00100     szDest_[j] = '\0';
00101 }
00102
00103 //-----
00104 Option_t *ShellSupport::CheckForOption(
00105     CommandLine_t *pstCommand_, const K_CHAR *szOption_ )
00106 {
00107     K_CHAR i;
00108     K_UCHAR tmp_len;
00109     for (i = 0; i < pstCommand_>ucNumOptions; i++)
00110     {
00111         tmp_len = MIN(MemUtil::StringLength(szOption_), pstCommand_>
00112             astOptions[i].pstStart->ucLen);
00113         if (true == MemUtil::CompareMemory( (const void*)szOption_,
00114             (const void*)(pstCommand_>astOptions[i].
00115             pstStart->pcToken),
00116             tmp_len ) )
00117         {
00118             return &(pstCommand_>astOptions[i]);
00119         }
00120     }
00121     return 0;
00122 }
00123 //-----
00124 K_CHAR ShellSupport::TokensToCommandLine(
00125     Token_t *pstTokens_, K_UCHAR ucTokens_, CommandLine_t *pstCommand_)
00126 {
00127     K_CHAR count = 0;
00128     K_CHAR token = 0;
00129     K_CHAR option = 0;
00130     pstCommand_>ucNumOptions = 0;
00131     if (!ucTokens_)
00132     {
00133         return -1;
00134     }
00135     // Command is a single token...

```

```

00136     pstCommand_>pstCommand = &pastTokens_[0];
00137
00138     // Parse out options
00139     token = 1;
00140     while (token < ucTokens_ && option < 12)
00141     {
00142         pstCommand_>astOptions[option].pstStart = &pastTokens_[token];
00143         count = 1;
00144         token++;
00145         while (token < ucTokens_ && pastTokens_[token].pcToken[0] != '-')
00146         {
00147             token++;
00148             count++;
00149         }
00150         pstCommand_>astOptions[option].ucCount = count;
00151         option++;
00152     }
00153
00154     pstCommand_>ucNumOptions = option;
00155     pstCommand_>ucTokenCount = ucTokens_;
00156     pstCommand_>pastTokenList = pastTokens_;
00157     return option;
00158 }

```

17.155 /home/mo/mark3-source/embedded/stage/src/shell_support.h File Reference

Support functions & data structures useful in implementing a shell.

```

#include "kerneltypes.h"
#include "memutil.h"

```

Classes

- struct [Option_t](#)
Structure used to represent a command-line option with its arguments.
- struct [CommandLine_t](#)
Structure containing multiple representations for command-line data.
- struct [ShellCommand_t](#)
Data structure defining a lookup table correlating a command name to its handler function.
- class [ShellSupport](#)
The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

Macros

- #define [MIN](#)(x, y) ((x) < (y)) ? (x) : (y)
Utility macro used to return the lesser of two values/objects.
- #define [MAX](#)(x, y) ((x) > (y)) ? (x) : (y)
Utility macro used to return the greater of two values/objects.

Typedefs

- typedef K_CHAR(* [fp_internal_command](#))([CommandLine_t](#) *pstCommandLine)
Function pointer type used to represent shell commands, as implemented by users of this infrastructure.

17.155.1 Detailed Description

Support functions & data structures useful in implementing a shell.

Definition in file [shell_support.h](#).

17.155.2 Typedef Documentation

17.155.2.1 typedef K_CHAR(*fp_internal_command)(CommandLine_t *pstCommandLine_)

Function pointer type used to represent shell commands, as implemented by users of this infrastructure.

Commands return a signed 8-bit result, and take a command-line argument structure as the first and only argument.

Definition at line 110 of file [shell_support.h](#).

17.156 shell_support.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00023 #ifndef __SHELL_SUPPORT_H__
00024 #define __SHELL_SUPPORT_H__
00025
00026 //-----
00027 #include "kerneltypes.h"
00028 #include "memutil.h"
00029
00030 //-----
00031 #ifndef MIN
00032
00033     #define MIN(x,y)          ( ( (x) < (y) ) ? (x) : (y) )
00034 #endif
00037 #ifndef MAX
00038
00039     #define MAX(x,y)          ( ( (x) > (y) ) ? (x) : (y) )
00040 #endif
00041
00042 //-----
00083 typedef struct
00084 {
00085     Token_t *pstStart;
00086     K_UCHAR ucCount;
00087 } Option_t;
00088
00089 //-----
00093 typedef struct
00094 {
00095     Token_t *pastTokenList;
00096     K_UCHAR ucTokenCount;
00097
00098     Token_t *pstCommand;
00099
00100     Option_t astOptions[12];
00101     K_UCHAR ucNumOptions;
00102 } CommandLine_t;
00103
00104 //-----
00110 typedef K_CHAR (*fp_internal_command)( CommandLine_t *pstCommandLine_ );
00111
00112 //-----
00117 typedef struct
00118 {
00119     const K_CHAR *szCommand;
00120     fp_internal_command pfHandler;
00121 } ShellCommand_t;
00122
00123 //-----
00129 class ShellSupport
00130 {
00131 public:
00132
00133     //-----
00142     static K_CHAR RunCommand( CommandLine_t *pstCommand_, const
ShellCommand_t *pastShellCommands_ );
00143
00144     //-----

```



```

00020 #include "slip.h"
00021 #include "driver.h"
00022
00023 //-----
00024 #define FRAMING_BYTE (192)
00025 #define FRAMING_ENC_BYTE (219)
00026 #define FRAMING_SUB_BYTE (220)
00027 #define FRAMING_SUB_ENC_BYTE (221)
00028
00029 //-----
00030 #define ACchar (69)
00031 #define NACchar (96)
00032
00033 //-----
00034 K_USHORT Slip::EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ )
00035 {
00036     K_USHORT usLen = 1;
00037     switch (ucChar_)
00038     {
00039         case FRAMING_BYTE:
00040             aucBuf_[0] = FRAMING_ENC_BYTE;
00041             aucBuf_[1] = FRAMING_SUB_BYTE;
00042             usLen = 2;
00043             break;
00044         case FRAMING_ENC_BYTE:
00045             aucBuf_[0] = FRAMING_ENC_BYTE;
00046             aucBuf_[1] = FRAMING_SUB_ENC_BYTE;
00047             usLen = 2;
00048             break;
00049         default:
00050             aucBuf_[0] = ucChar_;
00051     }
00052     return usLen;
00053 }
00054
00055 //-----
00056 K_USHORT Slip::DecodeByte( K_UCHAR *ucChar_, const K_UCHAR *aucBuf_ )
00057 {
00058     K_USHORT usLen = 1;
00059
00060     if (aucBuf_[0] == FRAMING_ENC_BYTE)
00061     {
00062         if (aucBuf_[1] == FRAMING_SUB_BYTE)
00063         {
00064             *ucChar_ = FRAMING_BYTE;
00065             usLen = 2;
00066         }
00067         else if (aucBuf_[1] == FRAMING_SUB_ENC_BYTE)
00068         {
00069             *ucChar_ = FRAMING_ENC_BYTE;
00070             usLen = 2;
00071         }
00072         else
00073         {
00074             *ucChar_ = 0;
00075             usLen = 0;
00076         }
00077     }
00078     else if (aucBuf_[0] == FRAMING_BYTE)
00079     {
00080         usLen = 0;
00081         *ucChar_ = 0;
00082     }
00083     else
00084     {
00085         *ucChar_ = aucBuf_[0];
00086     }
00087     return usLen;
00088 }
00089
00090 //-----
00091 void Slip::WriteByte( K_UCHAR ucData_ )
00092 {
00093     K_USHORT usSize = 0;
00094     K_USHORT usIdx = 0;
00095     K_UCHAR aucBuf[2];
00096     usSize = EncodeByte(ucData_, aucBuf);
00097     while (usIdx < usSize)
00098     {
00099         usIdx += m_pclDriver->Write(usSize, &aucBuf[usIdx]);
00100     }
00101 }
00102
00103 //-----
00104 K_USHORT Slip::ReadData(K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_)
00105 {
00106     K_USHORT usReadCount;

```

```

00107     K_UCHAR ucTempCount;
00108     K_USHORT usValid = 0;
00109     K_USHORT usCRC;
00110     K_USHORT usCRC_Calc = 0;
00111     K_USHORT usLen;
00112     K_UCHAR *pucSrc = (K_UCHAR*)aucBuf_;
00113     K_UCHAR *pucDst = (K_UCHAR*)aucBuf_;
00114
00115     usReadCount = m_pclDriver->Read(usLen_, (K_UCHAR*)aucBuf_);
00116
00117     while (usReadCount)
00118     {
00119         K_UCHAR ucRead;
00120         ucTempCount = DecodeByte(&ucRead, pucSrc);
00121
00122         *pucDst = ucRead;
00123
00124         // Encountered a FRAMING_BYTE - end of message
00125         if (!ucTempCount)
00126         {
00127             break;
00128         }
00129
00130         // Add to the CRC
00131         usCRC_Calc += ucRead;
00132
00133         // Adjust iterators, source, and destination pointers.
00134         usReadCount -= ucTempCount;
00135         pucSrc += ucTempCount;
00136         pucDst++;
00137         usValid++;
00138     }
00139
00140     // Ensure we have enough data to try a match.
00141     if (usValid < 5) {
00142         return 0;
00143     }
00144
00145     usCRC_Calc -= aucBuf_[usValid-2];
00146     usCRC_Calc -= aucBuf_[usValid-1];
00147
00148     usLen = ((K_USHORT)aucBuf_[1]) << 8;
00149     usLen += ((K_USHORT)aucBuf_[2]);
00150     usCRC = ((K_USHORT)aucBuf_[usValid-2]) << 8;
00151     usCRC += ((K_USHORT)aucBuf_[usValid-1]);
00152
00153     if (usCRC != usCRC_Calc)
00154     {
00155         return 0;
00156     }
00157
00158     *pucChannel_ = aucBuf_[0];
00159
00160     return usLen;
00161 }
00162
00163 //-----
00164 void Slip::WriteData(K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_)
00165 {
00166     K_UCHAR aucTmp[2];
00167     K_USHORT usCRC = 0;
00168
00169     // Lightweight protocol built on-top of SLIP.
00170     // 1) Channel ID (8-bit)
00171     // 2) Data Size (16-bit)
00172     // 3) Data blob
00173     // 4) CRC16 (16-bit)
00174     aucTmp[0] = FRAMING_BYTE;
00175     while( !m_pclDriver->Write(1, aucTmp) ) {}
00176
00177     if (!usLen_) // Read to end-of-line (\0)
00178     {
00179         K_UCHAR *pucBuf = (K_UCHAR*)aucBuf_;
00180         while (*pucBuf != '\0')
00181         {
00182             usLen_++;
00183             pucBuf++;
00184         }
00185     }
00186
00187     WriteByte(ucChannel_);
00188     usCRC = ucChannel_;
00189
00190     WriteByte((K_UCHAR)(usLen_ >> 8));
00191     usCRC += (usLen_ >> 8);
00192
00193     WriteByte((K_UCHAR)(usLen_ & 0x00FF));

```

```

00194     usCRC += (usLen_ & 0x00FF);
00195
00196     while (usLen_-->0)
00197     {
00198         WriteByte(*aucBuf_);
00199         usCRC += (K_USHORT)*aucBuf_;
00200         aucBuf_++;
00201     }
00202
00203     WriteByte((K_UCHAR)(usCRC >> 8));
00204     WriteByte((K_UCHAR)(usCRC & 0x00FF));
00205
00206     aucTmp[0] = FRAMING_BYTE;
00207     while( !m_pclDriver->Write(1, aucTmp) ) {}
00208 }
00209
00210 //-----
00211 void Slip::SendAck()
00212 {
00213     WriteByte(ACchar);
00214 }
00215
00216 //-----
00217 void Slip::SendNack()
00218 {
00219     WriteByte(NACchar);
00220 }
00221
00222 //-----
00223 void Slip::WriteVector(K_UCHAR ucChannel_, SlipDataVector *astData_,
00224                       K_USHORT usLen_)
00225 {
00226     K_UCHAR aucTmp[2];
00227     K_USHORT usCRC = 0;
00228     K_UCHAR i, j;
00229     K_USHORT usTotalLen = 0;
00230
00231     // Calculate the total length of all message fragments
00232     for (i = 0; i < usLen_; i++)
00233     {
00234         usTotalLen += astData_[i].ucSize;
00235     }
00236
00237     // Send a FRAMING_BYTE to start framing a message
00238     aucTmp[0] = FRAMING_BYTE;
00239     while( !m_pclDriver->Write(1, aucTmp) ) {}
00240
00241     // Write a the channel
00242     WriteByte(ucChannel_);
00243     usCRC = ucChannel_;
00244
00245     // Write the length
00246     WriteByte((K_UCHAR)(usTotalLen >> 8));
00247     usCRC += (usTotalLen >> 8);
00248
00249     WriteByte((K_UCHAR)(usTotalLen & 0x00FF));
00250     usCRC += (usTotalLen & 0x00FF);
00251
00252     // Write the message fragments
00253     for (i = 0; i < usLen_; i++)
00254     {
00255         K_UCHAR *aucBuf = astData_[i].pucData;
00256         for (j = 0; j < astData_[i].ucSize; j++)
00257         {
00258             WriteByte(*aucBuf);
00259             usCRC += (K_USHORT)*aucBuf;
00260             aucBuf++;
00261         }
00262     }
00263
00264     // Write the CRC
00265     WriteByte((K_UCHAR)(usCRC >> 8));
00266     WriteByte((K_UCHAR)(usCRC & 0x00FF));
00267
00268     // Write the end-of-message
00269     aucTmp[0] = FRAMING_BYTE;
00270     while( !m_pclDriver->Write(1, aucTmp) ) {}
00271 }

```

17.159 /home/mo/mark3-source/embedded/stage/src/slip.h File Reference

Serial Line IP framing code.

```
#include "kerneltypes.h"
#include "driver.h"
```

Classes

- struct [SlipDataVector](#)
Data structure used for vector-based SLIP data transmission.
- class [Slip](#)
Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

Enumerations

- enum [SlipChannel](#) {
[SLIP_CHANNEL_TERMINAL](#) = 0, [SLIP_CHANNEL_UNISCOPE](#), [SLIP_CHANNEL_NVM](#), [SLIP_CHANNEL_RESET](#),
[SLIP_CHANNEL_GRAPHICS](#), [SLIP_CHANNEL_HID](#), [SLIP_CHANNEL_COUNT](#) }

17.159.1 Detailed Description

Serial Line IP framing code. Also includes code to frame data in FunkenSlip format for use with [SlipTerm](#) on a host PC.

FunkenSlip uses SLIP-framed messages with a pre-defined packet format as follows:

[Channel][Size][Data Buffer][CRC8]

Channel is 1 byte, indicating the type of data carried in the message

Size is 2 bytes, indicating the length of the binary blob that follows

Data Buffer is n bytes, and contains the raw packet data.

CRC16 is 2 byte, Providing an error detection mechanism

Definition in file [slip.h](#).

17.159.2 Enumeration Type Documentation

17.159.2.1 enum SlipChannel

Enumerator

[SLIP_CHANNEL_TERMINAL](#) ASCII text mode terminal.

[SLIP_CHANNEL_UNISCOPE](#) Uniscope VM command channel.

[SLIP_CHANNEL_NVM](#) Non-volatile memory configuration.

[SLIP_CHANNEL_RESET](#) Channel used to reset the device...

[SLIP_CHANNEL_GRAPHICS](#) Encoded drawing commands.

[SLIP_CHANNEL_HID](#) HID commands.

Definition at line 41 of file [slip.h](#).

17.160 slip.h

```
00001 /*=====
00002  _____
```



```

00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00034 #include "kerneltypes.h"
00035 #include "driver.h"
00036
00037 #ifndef __SLIP_H__
00038 #define __SLIP_H__
00039
00040 //-----
00041 typedef enum
00042 {
00043     SLIP_CHANNEL_TERMINAL = 0,
00044     SLIP_CHANNEL_UNISCOPE,
00045     SLIP_CHANNEL_NVM,
00046     SLIP_CHANNEL_RESET,
00047     SLIP_CHANNEL_GRAPHICS,
00048     SLIP_CHANNEL_HID,
00049 } SLIP_CHANNEL_COUNT;
00050 } SlipChannel;
00051
00052 //-----
00059 typedef struct
00060 {
00061     K_UCHAR ucSize;
00062     K_UCHAR *pucData;
00063 }SlipDataVector;
00064
00065 //-----
00070 class Slip
00071 {
00072 public:
00073     void SetDriver( Driver *pclDriver_ ){ m_pclDriver = pclDriver_; }
00074
00075     Driver *GetDriver() { return m_pclDriver; }
00076
00077     static K_USHORT EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ );
00078
00079     static K_USHORT DecodeByte( K_UCHAR *ucChar_, const K_UCHAR *aucBuf_ );
00080
00081     void WriteData( K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_ );
00082
00083     K_USHORT ReadData( K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_ );
00084
00085     void WriteVector( K_UCHAR ucChannel_, SlipDataVector *astData_, K_USHORT
usLen_ );
00086
00087     void SendAck();
00088
00089     void SendNack();
00090 private:
00091     void WriteByte(K_UCHAR ucData_);
00092     Driver *m_pclDriver;
00093 };
00094 #endif

```

17.161 /home/mo/mark3-source/embedded/stage/src/slip_mux.cpp File Reference

FunkenSlip Channel Multiplexer.

```

#include "kerneltypes.h"
#include "driver.h"
#include "drvUART.h"
#include "slip.h"
#include "slip_mux.h"
#include "message.h"

```



```

00063
00064     m_clSlip.SetDriver(m_pclDriver);
00065     m_clSlipSem.Init(0, 1);
00066
00067     m_pclDriver->Control(CMD_SET_BUFFERS, (void*)aucRx_, usRxSize_, (void*)aucTx_, usTxSize_);
00068     m_pclDriver->Control(CMD_SET_RX_CALLBACK, (void*)SlipMux_Callback, 0, 0, 0);
00069     {
00070         K_UCHAR ucEscape = 192;
00071         m_pclDriver->Control(CMD_SET_RX_ESCAPE, (void*)&ucEscape, 1, 0, NULL);
00072     }
00073 }
00074
00075 //-----
00076 void SlipMux::InstallHandler( K_UCHAR ucChannel_, Slip_Channel pfHandler_ )
00077 {
00078     if (pfHandler_)
00079     {
00080         m_apfChannelHandlers[ucChannel_] = pfHandler_;
00081     }
00082 }
00083
00084 //-----
00085 void SlipMux::MessageReceive(void)
00086 {
00087     K_USHORT usLen;
00088     K_UCHAR ucChannel;
00089
00090     usLen = m_clSlip.ReadData( &ucChannel, (K_CHAR*)m_aucData, SLIP_BUFFER_SIZE );
00091     if (usLen && (m_apfChannelHandlers[ucChannel] != NULL))
00092     {
00093         m_apfChannelHandlers[ucChannel]( m_pclDriver, ucChannel, &(m_aucData[3]), usLen);
00094     }
00095
00096     // Re-enable the driver once we're done.
00097     m_pclDriver->Control( CMD_SET_RX_ENABLE, 0, 0, 0, 0 );
00098 }
00099

```

17.163 /home/mo/mark3-source/embedded/stage/src/slippy_mux.h File Reference

FunkenSlip Channel Multiplexer.

```

#include "kerneltypes.h"
#include "driver.h"
#include "ksemaphore.h"
#include "message.h"
#include "slip.h"

```

Classes

- class [SlipMux](#)

Static-class which implements a multiplexed stream of SLIP data over a single interface.

Macros

- #define **SLIP_BUFFER_SIZE** (32)
- #define **SLIP_RX_MESSAGE_ID** (0xD00D)

Typedefs

- typedef void(* **Slip_Channel**)(Driver *pclDriver_, K_UCHAR ucChannel_, K_UCHAR *pucData_, K_USHORT usLen_)

17.163.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel

Definition in file [slip_mux.h](#).

17.164 slip_mux.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "driver.h"
00023 #include "ksemaphore.h"
00024 #include "message.h"
00025 #include "slip.h"
00026
00027 #ifndef __SLIP_MUX_H__
00028 #define __SLIP_MUX_H__
00029
00030 //-----
00031 #define SLIP_BUFFER_SIZE      (32)
00032
00033 #define SLIP_RX_MESSAGE_ID    (0xD00D)
00034
00035 //-----
00036 typedef void (*Slip_Channel)( Driver *pclDriver_, K_UCHAR ucChannel_, K_UCHAR *pucData_, K_USHORT
usLen_ );
00037
00038 //-----
00043 class SlipMux
00044 {
00045 public:
00065     static void Init(const K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT
usTxSize_, K_UCHAR *aucTx_);
00066
00075     static void InstallHandler( K_UCHAR ucChannel_, Slip_Channel pfHandler_ );
00076
00084     static void MessageReceive();
00085
00091     static Driver *GetDriver(){ return m_pclDriver; }
00092
00099     static MessageQueue *GetQueue(){ return m_pclMessageQueue; }
00100
00108     static void SetQueue( MessageQueue *pclMessageQueue_ )
00109     { m_pclMessageQueue = pclMessageQueue_; }
00110
00117     static Slip *GetSlip(){ return &m_clSlip; }
00118
00119 private:
00120     static MessageQueue *m_pclMessageQueue;
00121     static Driver *m_pclDriver;
00122     static Slip_Channel m_apfChannelHandlers[SLIP_CHANNEL_COUNT];
00123     static K_UCHAR m_aucData[SLIP_BUFFER_SIZE];
00124     static Semaphore m_clSlipSem;
00125     static Slip m_clSlip;
00126 };
00127
00128 #endif

```

17.165 /home/mo/mark3-source/embedded/stage/src/slipterm.cpp File Reference

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

```
#include "kerneltypes.h"
#include "slip.h"
#include "slipterm.h"
```

17.165.1 Detailed Description

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

Definition in file [slipterm.cpp](#).

17.166 slipterm.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "slip.h"
00023 #include "slipterm.h"
00024
00025 //-----
00026 void SlipTerm::Init()
00027 {
00028     m_clSlip.SetDriver( DriverList::FindByPath("/dev/tty" ) );
00029
00029     m_ucVerbosity = SEVERITY_DEBUG;
00030 }
00031
00032 //-----
00033 K_USHORT SlipTerm::StrLen( const char *szLine_ )
00034 {
00035     K_USHORT i=0;
00036     while (szLine_[i] != 0 )
00037     {
00038         i++;
00039     }
00040     return i;
00041 }
00042
00043 //-----
00044 void SlipTerm::PrintLn( const char *szLine_ )
00045 {
00046     SlipDataVector astData[2];
00047     astData[0].pucData = (K_UCHAR*)szLine_;
00048     astData[0].ucSize = StrLen(szLine_);
00049     astData[1].pucData = (K_UCHAR*)"r\n";
00050     astData[1].ucSize = 2;
00051
00052     m_clSlip.WriteVector(SLIP_CHANNEL_TERMINAL, astData, 2);
00053 }
00054
00055 //-----
00056 void SlipTerm::PrintLn( K_UCHAR ucSeverity_, const char *szLine_ )
00057 {
00058     if (ucSeverity_ <= m_ucVerbosity)
00059     {
00060         PrintLn( szLine_ );
00061     }
00062 }
```

17.167 /home/mo/mark3-source/embedded/stage/src/slipterm.h File Reference

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

```
#include "kerneltypes.h"
#include "driver.h"
#include "slip.h"
```

Classes

- class [SlipTerm](#)

Class implementing a simple debug terminal interface.

Macros

- #define **SEVERITY_DEBUG** 4
- #define **SEVERITY_INFO** 3
- #define **SEVERITY_WARN** 2
- #define **SEVERITY_CRITICAL** 1
- #define **SEVERITY_CATASTROPHIC** 0
- #define **__SLIPTERM_H__**

17.167.1 Detailed Description

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

Definition in file [slipterm.h](#).

17.168 slipterm.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "driver.h"
00023 #include "slip.h"
00024
00025 //-----
00026 #define SEVERITY_DEBUG          4
00027 #define SEVERITY_INFO          3
00028 #define SEVERITY_WARN          2
00029 #define SEVERITY_CRITICAL      1
00030 #define SEVERITY_CATASTROPHIC 0
00031
00032 //-----
00033 #ifndef __SLIPTERM_H__
00034 #define __SLIPTERM_H__
00035
00040 class SlipTerm
00041 {
00042 public:
00050     void Init();
00051
00060     void PrintLn( const char *szLine_ );
00061
00072     void PrintLn( K_UCHAR ucSeverity_, const char *szLine_ );
00073
00081     void SetVerbosity( K_UCHAR ucLevel_ ) { m_ucVerbosity = ucLevel_; }
00082 private:
00090     K_USHORT StrLen( const char *szString_ );
00091
```

```

00092     K_UCHAR m_ucVerbosity;
00093
00094     Slip m_clSlip;
00095 };
00096
00097
00098 #endif

```

17.169 /home/mo/mark3-source/embedded/stage/src/system_heap.cpp File Reference

Global system-heap implementation.

```

#include "kerneltypes.h"
#include "system_heap_config.h"
#include "system_heap.h"

```

17.169.1 Detailed Description

Global system-heap implementation. Provides a system-wide malloc/free paradigm allocation scheme.

Definition in file [system_heap.cpp](#).

17.170 system_heap.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "system_heap_config.h"
00023 #include "system_heap.h"
00024
00025 #if USE_SYSTEM_HEAP
00026
00027 //-----
00028 K_UCHAR SystemHeap::m_pucRawHeap[ HEAP_RAW_SIZE ];
00029 HeapConfig SystemHeap::m_pclSystemHeapConfig[
00030     HEAP_NUM_SIZES + 1];
00031 FixedHeap SystemHeap::m_clSystemHeap;
00032 bool SystemHeap::m_bInit;
00033
00034 //-----
00034 void SystemHeap::Init(void)
00035 {
00036     #if HEAP_NUM_SIZES > 0
00037         m_pclSystemHeapConfig[0].m_usBlockSize =
00038             HEAP_BLOCK_SIZE_1;
00039         m_pclSystemHeapConfig[0].m_usBlockCount =
00040             HEAP_BLOCK_COUNT_1;
00041     #endif
00042     #if HEAP_NUM_SIZES > 1
00043         m_pclSystemHeapConfig[1].m_usBlockSize = HEAP_BLOCK_SIZE_2;
00044         m_pclSystemHeapConfig[1].m_usBlockCount = HEAP_BLOCK_COUNT_2;
00045     #endif
00046     #if HEAP_NUM_SIZES > 2
00047         m_pclSystemHeapConfig[2].m_usBlockSize = HEAP_BLOCK_SIZE_3;
00048         m_pclSystemHeapConfig[2].m_usBlockCount = HEAP_BLOCK_COUNT_3;
00049     #endif
00050     #if HEAP_NUM_SIZES > 3
00051         m_pclSystemHeapConfig[3].m_usBlockSize = HEAP_BLOCK_SIZE_4;
00052         m_pclSystemHeapConfig[3].m_usBlockCount = HEAP_BLOCK_COUNT_4;
00053     #endif
00054     #if HEAP_NUM_SIZES > 4
00055         m_pclSystemHeapConfig[4].m_usBlockSize = HEAP_BLOCK_SIZE_5;

```

```

00054     m_pclSystemHeapConfig[4].m_usBlockCount = HEAP_BLOCK_COUNT_5;
00055 #endif
00056 #if HEAP_NUM_SIZES > 5
00057     m_pclSystemHeapConfig[5].m_usBlockSize = HEAP_BLOCK_SIZE_6;
00058     m_pclSystemHeapConfig[5].m_usBlockCount = HEAP_BLOCK_COUNT_6;
00059 #endif
00060 #if HEAP_NUM_SIZES > 6
00061     m_pclSystemHeapConfig[6].m_usBlockSize = HEAP_BLOCK_SIZE_7;
00062     m_pclSystemHeapConfig[6].m_usBlockCount = HEAP_BLOCK_COUNT_7;
00063 #endif
00064 #if HEAP_NUM_SIZES > 7
00065     m_pclSystemHeapConfig[7].m_usBlockSize = HEAP_BLOCK_SIZE_8;
00066     m_pclSystemHeapConfig[7].m_usBlockCount = HEAP_BLOCK_COUNT_8;
00067 #endif
00068 #if HEAP_NUM_SIZES > 8
00069     m_pclSystemHeapConfig[8].m_usBlockSize = HEAP_BLOCK_SIZE_9;
00070     m_pclSystemHeapConfig[8].m_usBlockCount = HEAP_BLOCK_COUNT_9;
00071 #endif
00072 #if HEAP_NUM_SIZES > 9
00073     m_pclSystemHeapConfig[9].m_usBlockSize = HEAP_BLOCK_SIZE_10;
00074     m_pclSystemHeapConfig[9].m_usBlockCount = HEAP_BLOCK_COUNT_10;
00075 #endif
00076 #if HEAP_NUM_SIZES > 10
00077     m_pclSystemHeapConfig[10].m_usBlockSize = HEAP_BLOCK_SIZE_11;
00078     m_pclSystemHeapConfig[10].m_usBlockCount = HEAP_BLOCK_COUNT_11;
00079 #endif
00080 #if HEAP_NUM_SIZES > 11
00081     m_pclSystemHeapConfig[11].m_usBlockSize = HEAP_BLOCK_SIZE_12;
00082     m_pclSystemHeapConfig[11].m_usBlockCount = HEAP_BLOCK_COUNT_12;
00083 #endif
00084 #if HEAP_NUM_SIZES > 12
00085     m_pclSystemHeapConfig[12].m_usBlockSize = HEAP_BLOCK_SIZE_13;
00086     m_pclSystemHeapConfig[12].m_usBlockCount = HEAP_BLOCK_COUNT_13;
00087 #endif
00088 #if HEAP_NUM_SIZES > 13
00089     m_pclSystemHeapConfig[13].m_usBlockSize = HEAP_BLOCK_SIZE_14;
00090     m_pclSystemHeapConfig[13].m_usBlockCount = HEAP_BLOCK_COUNT_14;
00091 #endif
00092 #if HEAP_NUM_SIZES > 14
00093     m_pclSystemHeapConfig[14].m_usBlockSize = HEAP_BLOCK_SIZE_15;
00094     m_pclSystemHeapConfig[14].m_usBlockCount = HEAP_BLOCK_COUNT_15;
00095 #endif
00096 #if HEAP_NUM_SIZES > 15
00097     m_pclSystemHeapConfig[15].m_usBlockSize = HEAP_BLOCK_SIZE_16;
00098     m_pclSystemHeapConfig[15].m_usBlockCount = HEAP_BLOCK_COUNT_16;
00099 #endif
00100 #if HEAP_NUM_SIZES > 16
00101     m_pclSystemHeapConfig[16].m_usBlockSize = HEAP_BLOCK_SIZE_17;
00102     m_pclSystemHeapConfig[16].m_usBlockCount = HEAP_BLOCK_COUNT_17;
00103 #endif
00104 #if HEAP_NUM_SIZES > 17
00105     m_pclSystemHeapConfig[17].m_usBlockSize = HEAP_BLOCK_SIZE_18;
00106     m_pclSystemHeapConfig[17].m_usBlockCount = HEAP_BLOCK_COUNT_18;
00107 #endif
00108 #if HEAP_NUM_SIZES > 18
00109     m_pclSystemHeapConfig[18].m_usBlockSize = HEAP_BLOCK_SIZE_19;
00110     m_pclSystemHeapConfig[18].m_usBlockCount = HEAP_BLOCK_COUNT_19;
00111 #endif
00112 #if HEAP_NUM_SIZES > 19
00113     m_pclSystemHeapConfig[19].m_usBlockSize = HEAP_BLOCK_SIZE_20;
00114     m_pclSystemHeapConfig[19].m_usBlockCount = HEAP_BLOCK_COUNT_20;
00115 #endif
00116 #if HEAP_NUM_SIZES > 20
00117     m_pclSystemHeapConfig[20].m_usBlockSize = HEAP_BLOCK_SIZE_21;
00118     m_pclSystemHeapConfig[20].m_usBlockCount = HEAP_BLOCK_COUNT_21;
00119 #endif
00120
00121     m_pclSystemHeapConfig[HEAP_NUM_SIZES].
00122     m_usBlockSize = 0;
00122     m_pclSystemHeapConfig[HEAP_NUM_SIZES].
00123     m_usBlockCount = 0;
00123
00124     m_clSystemHeap.Create((void*)m_pucRawHeap,
00125     m_pclSystemHeapConfig);
00125
00126     m_bInit = true;
00127 }
00128
00129 //-----
00130 void *SystemHeap::Alloc(K_USHORT usSize_)
00131 {
00132     if (!m_bInit)
00133     {
00134         return NULL;
00135     }
00136     return m_clSystemHeap.Alloc(usSize_);
00137 }

```



```

00138
00139 //-----
00140 void SystemHeap::Free(void* pvBlock_)
00141 {
00142     if (!m_bInit)
00143     {
00144         return;
00145     }
00146     m_clSystemHeap.Free(pvBlock_);
00147 }
00148
00149 #endif // USE_SYSTEM_HEAP

```

17.171 /home/mo/mark3-source/embedded/stage/src/system_heap.h File Reference

Global system-heap implmentation.

```

#include "system_heap_config.h"
#include "fixed_heap.h"

```

Classes

- class [SystemHeap](#)

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

Macros

- #define [HEAP_RAW_SIZE_1](#) ((HEAP_BLOCK_SIZE_1 + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_1](#))

Really ugly computations used to auto-size the heap footprint based on the user-configuration data.

- #define [HEAP_RAW_SIZE_2](#) ((HEAP_BLOCK_SIZE_2 + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_2](#))
- #define [HEAP_RAW_SIZE_3](#) ((HEAP_BLOCK_SIZE_3 + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_3](#))
- #define [HEAP_RAW_SIZE_4](#) 0
- #define [HEAP_RAW_SIZE_5](#) 0
- #define [HEAP_RAW_SIZE_6](#) 0
- #define [HEAP_RAW_SIZE_7](#) 0
- #define [HEAP_RAW_SIZE_8](#) 0
- #define [HEAP_RAW_SIZE_9](#) 0
- #define [HEAP_RAW_SIZE_10](#) 0
- #define [HEAP_RAW_SIZE_11](#) 0
- #define [HEAP_RAW_SIZE_12](#) 0
- #define [HEAP_RAW_SIZE_13](#) 0
- #define [HEAP_RAW_SIZE_14](#) 0
- #define [HEAP_RAW_SIZE_15](#) 0
- #define [HEAP_RAW_SIZE_16](#) 0
- #define [HEAP_RAW_SIZE_17](#) 0
- #define [HEAP_RAW_SIZE_18](#) 0
- #define [HEAP_RAW_SIZE_19](#) 0
- #define [HEAP_RAW_SIZE_20](#) 0
- #define [HEAP_RAW_SIZE_21](#) 0
- #define [HEAP_RAW_SIZE](#)


```

        HEAP_BLOCK_COUNT_1 )
00036 #else
00037     #define HEAP_RAW_SIZE_1 0
00038 #endif
00039
00040 #if HEAP_NUM_SIZES > 1
00041     #define HEAP_RAW_SIZE_2 ((HEAP_BLOCK_SIZE_2 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_2 )
00042 #else
00043     #define HEAP_RAW_SIZE_2 0
00044 #endif
00045
00046 #if HEAP_NUM_SIZES > 2
00047     #define HEAP_RAW_SIZE_3 ((HEAP_BLOCK_SIZE_3 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_3 )
00048 #else
00049     #define HEAP_RAW_SIZE_3 0
00050 #endif
00051
00052 #if HEAP_NUM_SIZES > 3
00053     #define HEAP_RAW_SIZE_4 ((HEAP_BLOCK_SIZE_4 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_4 )
00054 #else
00055     #define HEAP_RAW_SIZE_4 0
00056 #endif
00057
00058 #if HEAP_NUM_SIZES > 4
00059     #define HEAP_RAW_SIZE_5 ((HEAP_BLOCK_SIZE_5 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_5 )
00060 #else
00061     #define HEAP_RAW_SIZE_5 0
00062 #endif
00063
00064 #if HEAP_NUM_SIZES > 5
00065     #define HEAP_RAW_SIZE_6 ((HEAP_BLOCK_SIZE_6 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_6 )
00066 #else
00067     #define HEAP_RAW_SIZE_6 0
00068 #endif
00069
00070 #if HEAP_NUM_SIZES > 6
00071     #define HEAP_RAW_SIZE_7 ((HEAP_BLOCK_SIZE_7 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_7 )
00072 #else
00073     #define HEAP_RAW_SIZE_7 0
00074 #endif
00075
00076 #if HEAP_NUM_SIZES > 7
00077     #define HEAP_RAW_SIZE_8 ((HEAP_BLOCK_SIZE_8 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_8 )
00078 #else
00079     #define HEAP_RAW_SIZE_8 0
00080 #endif
00081
00082 #if HEAP_NUM_SIZES > 8
00083     #define HEAP_RAW_SIZE_9 ((HEAP_BLOCK_SIZE_9 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_9 )
00084 #else
00085     #define HEAP_RAW_SIZE_9 0
00086 #endif
00087
00088 #if HEAP_NUM_SIZES > 9
00089     #define HEAP_RAW_SIZE_10 ((HEAP_BLOCK_SIZE_10 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_10 )
00090 #else
00091     #define HEAP_RAW_SIZE_10 0
00092 #endif
00093
00094 #if HEAP_NUM_SIZES > 10
00095     #define HEAP_RAW_SIZE_11 ((HEAP_BLOCK_SIZE_11 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_11 )
00096 #else
00097     #define HEAP_RAW_SIZE_11 0
00098 #endif
00099
00100 #if HEAP_NUM_SIZES > 11
00101     #define HEAP_RAW_SIZE_12 ((HEAP_BLOCK_SIZE_12 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_12 )
00102 #else
00103     #define HEAP_RAW_SIZE_12 0
00104 #endif
00105
00106 #if HEAP_NUM_SIZES > 12
00107     #define HEAP_RAW_SIZE_13 ((HEAP_BLOCK_SIZE_13 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_13 )
00108 #else
00109     #define HEAP_RAW_SIZE_13 0

```

```

00110 #endif
00111
00112 #if HEAP_NUM_SIZES > 13
00113     #define HEAP_RAW_SIZE_14 ((HEAP_BLOCK_SIZE_14 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_14 )
00114 #else
00115     #define HEAP_RAW_SIZE_14 0
00116 #endif
00117
00118 #if HEAP_NUM_SIZES > 14
00119     #define HEAP_RAW_SIZE_15 ((HEAP_BLOCK_SIZE_15 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_15 )
00120 #else
00121     #define HEAP_RAW_SIZE_15 0
00122 #endif
00123
00124 #if HEAP_NUM_SIZES > 15
00125     #define HEAP_RAW_SIZE_16 ((HEAP_BLOCK_SIZE_16 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_16 )
00126 #else
00127     #define HEAP_RAW_SIZE_16 0
00128 #endif
00129
00130 #if HEAP_NUM_SIZES > 16
00131     #define HEAP_RAW_SIZE_17 ((HEAP_BLOCK_SIZE_17 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_17 )
00132 #else
00133     #define HEAP_RAW_SIZE_17 0
00134 #endif
00135
00136 #if HEAP_NUM_SIZES > 17
00137     #define HEAP_RAW_SIZE_18 ((HEAP_BLOCK_SIZE_18 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_18 )
00138 #else
00139     #define HEAP_RAW_SIZE_18 0
00140 #endif
00141
00142 #if HEAP_NUM_SIZES > 18
00143     #define HEAP_RAW_SIZE_19 ((HEAP_BLOCK_SIZE_19 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_19 )
00144 #else
00145     #define HEAP_RAW_SIZE_19 0
00146 #endif
00147
00148 #if HEAP_NUM_SIZES > 19
00149     #define HEAP_RAW_SIZE_20 ((HEAP_BLOCK_SIZE_20 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_20 )
00150 #else
00151     #define HEAP_RAW_SIZE_20 0
00152 #endif
00153
00154 #if HEAP_NUM_SIZES > 20
00155     #define HEAP_RAW_SIZE_21 ((HEAP_BLOCK_SIZE_21 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_21 )
00156 #else
00157     #define HEAP_RAW_SIZE_21 0
00158 #endif
00159
00160 //-----
00161 #define HEAP_RAW_SIZE \
00162     HEAP_RAW_SIZE_1 + \
00163     HEAP_RAW_SIZE_2 + \
00164     HEAP_RAW_SIZE_3 + \
00165     HEAP_RAW_SIZE_4 + \
00166     HEAP_RAW_SIZE_5 + \
00167     HEAP_RAW_SIZE_6 + \
00168     HEAP_RAW_SIZE_7 + \
00169     HEAP_RAW_SIZE_8 + \
00170     HEAP_RAW_SIZE_9 + \
00171     HEAP_RAW_SIZE_10 + \
00172     HEAP_RAW_SIZE_11 + \
00173     HEAP_RAW_SIZE_12 + \
00174     HEAP_RAW_SIZE_13 + \
00175     HEAP_RAW_SIZE_14 + \
00176     HEAP_RAW_SIZE_15 + \
00177     HEAP_RAW_SIZE_16 + \
00178     HEAP_RAW_SIZE_17 + \
00179     HEAP_RAW_SIZE_18 + \
00180     HEAP_RAW_SIZE_19 + \
00181     HEAP_RAW_SIZE_20 + \
00182     HEAP_RAW_SIZE_21
00183
00184 //-----
00189 class SystemHeap
00190 {
00191 public:
00195     static void Init(void);

```

```

00196
00203     static void* Alloc(K_USHORT usSize_);
00204
00209     static void Free(void *pvData_);
00210
00211 private:
00212     static K_UCHAR m_pucRawHeap[ HEAP_RAW_SIZE ];
00213     static HeapConfig m_pclSystemHeapConfig[
00214         HEAP_NUM_SIZES + 1 ];
00214     static FixedHeap m_clSystemHeap;
00215     static bool m_bInit;
00216 };
00217
00218 #endif // USE_SYSTEM_HEAP
00219
00220 #endif // __SYSTEM_HEAP_H__

```

17.173 /home/mo/mark3-source/embedded/stage/src/system_heap_config.h File Reference

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

```
#include "kerneltypes.h"
```

Macros

- `#define USE_SYSTEM_HEAP (1)`
Set this to "1" if you want the system heap to be built as part of this library.
- `#define HEAP_NUM_SIZES (3)`
Define the number of heap block sizes that we want to have attached to our system heap.
- `#define HEAP_BLOCK_SIZE_1 ((K_USHORT) 8)`
Define the block sizes for each of the fixed-size blocks that will be managed by our heaps.
- `#define HEAP_BLOCK_SIZE_2 ((K_USHORT) 16)`
- `#define HEAP_BLOCK_SIZE_3 ((K_USHORT) 24)`
- `#define HEAP_BLOCK_SIZE_4 ((K_USHORT) 32)`
- `#define HEAP_BLOCK_SIZE_5 ((K_USHORT) 48)`
- `#define HEAP_BLOCK_SIZE_6 ((K_USHORT) 64)`
- `#define HEAP_BLOCK_SIZE_7 ((K_USHORT) 96)`
- `#define HEAP_BLOCK_SIZE_8 ((K_USHORT) 128)`
- `#define HEAP_BLOCK_SIZE_9 ((K_USHORT) 192)`
- `#define HEAP_BLOCK_SIZE_10 ((K_USHORT) 256)`
- `#define HEAP_BLOCK_COUNT_1 ((K_USHORT) 4)`
Define the number of blocks in each bin, tailored for a particular application.
- `#define HEAP_BLOCK_COUNT_2 ((K_USHORT) 4)`
- `#define HEAP_BLOCK_COUNT_3 ((K_USHORT) 2)`
- `#define HEAP_BLOCK_COUNT_4 ((K_USHORT) 2)`
- `#define HEAP_BLOCK_COUNT_5 ((K_USHORT) 2)`
- `#define HEAP_BLOCK_COUNT_6 ((K_USHORT) 2)`
- `#define HEAP_BLOCK_COUNT_7 ((K_USHORT) 1)`
- `#define HEAP_BLOCK_COUNT_8 ((K_USHORT) 1)`
- `#define HEAP_BLOCK_COUNT_9 ((K_USHORT) 1)`
- `#define HEAP_BLOCK_COUNT_10 ((K_USHORT) 1)`

17.173.1 Detailed Description

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

Definition in file [system_heap_config.h](#).

17.173.2 Macro Definition Documentation

17.173.2.1 #define HEAP_BLOCK_SIZE_1 ((K_USHORT) 8)

Define the block sizes for each of the fixed-size blocks that will be managed by our heaps.

Must be defined in incrementing order.

Definition at line 44 of file [system_heap_config.h](#).

17.174 system_heap_config.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #ifndef __SYSTEM_HEAP_CONFIG_H__
00021 #define __SYSTEM_HEAP_CONFIG_H__
00022
00023 #include "kerneltypes.h"
00024
00025 //-----
00030 #define USE_SYSTEM_HEAP          (1)
00031
00032 //-----
00037 #define HEAP_NUM_SIZES          (3)
00038
00039 //-----
00044 #define HEAP_BLOCK_SIZE_1      ((K_USHORT) 8)
00045 #define HEAP_BLOCK_SIZE_2      ((K_USHORT) 16)
00046 #define HEAP_BLOCK_SIZE_3      ((K_USHORT) 24)
00047 #define HEAP_BLOCK_SIZE_4      ((K_USHORT) 32)
00048 #define HEAP_BLOCK_SIZE_5      ((K_USHORT) 48)
00049 #define HEAP_BLOCK_SIZE_6      ((K_USHORT) 64)
00050 #define HEAP_BLOCK_SIZE_7      ((K_USHORT) 96)
00051 #define HEAP_BLOCK_SIZE_8      ((K_USHORT) 128)
00052 #define HEAP_BLOCK_SIZE_9      ((K_USHORT) 192)
00053 #define HEAP_BLOCK_SIZE_10     ((K_USHORT) 256)
00054
00055 //-----
00060 #define HEAP_BLOCK_COUNT_1      ((K_USHORT) 4)
00061 #define HEAP_BLOCK_COUNT_2      ((K_USHORT) 4)
00062 #define HEAP_BLOCK_COUNT_3      ((K_USHORT) 2)
00063 #define HEAP_BLOCK_COUNT_4      ((K_USHORT) 2)
00064 #define HEAP_BLOCK_COUNT_5      ((K_USHORT) 2)
00065 #define HEAP_BLOCK_COUNT_6      ((K_USHORT) 2)
00066 #define HEAP_BLOCK_COUNT_7      ((K_USHORT) 1)
00067 #define HEAP_BLOCK_COUNT_8      ((K_USHORT) 1)
00068 #define HEAP_BLOCK_COUNT_9      ((K_USHORT) 1)
00069 #define HEAP_BLOCK_COUNT_10     ((K_USHORT) 1)
00070
00071 #endif
00072

```

17.175 /home/mo/mark3-source/embedded/stage/src/thread.cpp File Reference

Platform-Independent thread class Definition.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "scheduler.h"
#include "kernelswi.h"
#include "timerlist.h"
#include "ksemaphore.h"
#include "quantum.h"
#include "kernel.h"
#include "kernel_debug.h"
```

Macros

- `#define __FILE_ID__ THREAD_CPP`

Functions

- static void [ThreadSleepCallback](#) ([Thread](#) *pclOwner_, void *pvData_)

This callback is used to wake up a thread once the interval has expired.

17.175.1 Detailed Description

Platform-Independent thread class Definition.

Definition in file [thread.cpp](#).

17.176 thread.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024
00025 #include "thread.h"
00026 #include "scheduler.h"
00027 #include "kernelswi.h"
00028 #include "timerlist.h"
00029 #include "ksemaphore.h"
00030 #include "quantum.h"
00031 #include "kernel.h"
00032 #include "kernel_debug.h"
00033
00034 //-----
00035 #if defined __FILE_ID__
00036     #undef __FILE_ID__
00037 #endif
00038 #define __FILE_ID__      THREAD_CPP
00039
00040 //-----
00041 void Thread::Init( K_WORD *pwStack_,
00042                  K_USHORT usStackSize_,
00043                  K_UCHAR ucPriority_,
00044                  ThreadEntry_t pfEntryPoint_,
00045                  void *pvArg_ )
00046 {
```

```

00047     static K_UCHAR ucThreadID = 0;
00048
00049     KERNEL_ASSERT( pwStack_ );
00050     KERNEL_ASSERT( pfEntryPoint_ );
00051
00052     m_ucThreadID = ucThreadID++;
00053
00054     KERNEL_TRACE_1( STR_STACK_SIZE_1, usStackSize_ );
00055     KERNEL_TRACE_1( STR_PRIORITY_1, (K_UCHAR)ucPriority_ );
00056     KERNEL_TRACE_1( STR_THREAD_ID_1, (K_USHORT)m_ucThreadID );
00057     KERNEL_TRACE_1( STR_ENTRYPOINT_1, (K_USHORT)pfEntryPoint_ );
00058
00059     // Initialize the thread parameters to their initial values.
00060     m_pwStack = pwStack_;
00061     m_pwStackTop = TOP_OF_STACK(pwStack_, usStackSize_);
00062
00063     m_usStackSize = usStackSize_;
00064
00065     #if KERNEL_USE_QUANTUM
00066         m_usQuantum = 4;
00067     #endif
00068
00069     m_ucPriority = ucPriority_;
00070     m_ucCurPriority = m_ucPriority;
00071     m_pfEntryPoint = pfEntryPoint_;
00072     m_pvArg = pvArg_;
00073
00074     #if KERNEL_USE_THREADNAME
00075         m_szName = NULL;
00076     #endif
00077
00078     // Call CPU-specific stack initialization
00079     ThreadPort::InitStack(this);
00080
00081     // Add to the global "stop" list.
00082     CS_ENTER();
00083     m_pclOwner = Scheduler::GetThreadList(
m_ucPriority);
00084     m_pclCurrent = Scheduler::GetStopList();
00085     m_pclCurrent->Add(this);
00086     CS_EXIT();
00087 }
00088
00089 //-----
00090 void Thread::Start(void)
00091 {
00092     // Remove the thread from the scheduler's "stopped" list, and add it
00093     // to the scheduler's ready list at the proper priority.
00094     KERNEL_TRACE_1( STR_THREAD_START_1, (K_USHORT)m_ucThreadID );
00095
00096     CS_ENTER();
00097     Scheduler::GetStopList()->Remove(this);
00098     Scheduler::Add(this);
00099     m_pclOwner = Scheduler::GetThreadList(
m_ucPriority);
00100     m_pclCurrent = m_pclOwner;
00101
00102     if (Kernel::IsStarted())
00103     {
00104         if (m_ucPriority >= Scheduler::GetCurrentThread()->
GetCurPriority())
00105         {
00106             #if KERNEL_USE_QUANTUM
00107                 // Deal with the thread Quantum
00108                 Quantum::RemoveThread();
00109                 Quantum::AddThread(this);
00110             #endif
00111         }
00112         if (m_ucPriority > Scheduler::GetCurrentThread()->
GetPriority())
00113         {
00114             Thread::Yield();
00115         }
00116     }
00117     CS_EXIT();
00118 }
00119
00120 //-----
00121 void Thread::Stop()
00122 {
00123     K_UCHAR bReschedule = 0;
00124
00125     CS_ENTER();
00126
00127     // If a thread is attempting to stop itself, ensure we call the scheduler
00128     if (this == Scheduler::GetCurrentThread())
00129     {

```



```

00130         bReschedule = true;
00131     }
00132
00133     // Add this thread to the stop-list (removing it from active scheduling)
00134     Scheduler::Remove(this);
00135     m_pclOwner = Scheduler::GetStopList();
00136     m_pclCurrent = m_pclOwner;
00137     m_pclOwner->Add(this);
00138
00139     CS_EXIT();
00140
00141     if (bReschedule)
00142     {
00143         Thread::Yield();
00144     }
00145 }
00146
00147 #if KERNEL_USE_DYNAMIC_THREADS
00148 //-----
00149 void Thread::Exit()
00150 {
00151     K_UCHAR bReschedule = 0;
00152
00153     KERNEL_TRACE_1( STR_THREAD_EXIT_1, m_ucThreadID );
00154
00155     CS_ENTER();
00156
00157     // If this thread is the actively-running thread, make sure we run the
00158     // scheduler again.
00159     if (this == Scheduler::GetCurrentThread())
00160     {
00161         bReschedule = 1;
00162     }
00163
00164     // Remove the thread from scheduling
00165     m_pclCurrent->Remove(this);
00166
00167 #if KERNEL_USE_TIMERS
00168     // Just to be safe - attempt to remove the thread's timer
00169     // from the timer-scheduler (does no harm if it isn't
00170     // in the timer-list)
00171     TimerScheduler::Remove(&m_clTimer);
00172 #endif
00173
00174     CS_EXIT();
00175
00176     if (bReschedule)
00177     {
00178         // Choose a new "next" thread if we must
00179         Thread::Yield();
00180     }
00181 }
00182 #endif
00183
00184 #if KERNEL_USE_SLEEP
00185 //-----
00186 static void ThreadSleepCallback( Thread *pclOwner_, void *pvData_ )
00187 {
00188     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_);
00189     // Post the semaphore, which will wake the sleeping thread.
00190     pclSemaphore->Post();
00191 }
00192
00193 //-----
00194 void Thread::Sleep(K_ULONG ulTimeMs_)
00195 {
00196     Semaphore clSemaphore;
00197     Timer *pclTimer = g_pstCurrent->GetTimer();
00198
00199     // Create a semaphore that this thread will block on
00200     clSemaphore.Init(0, 1);
00201
00202     // Create a one-shot timer that will call a callback that posts the
00203     // semaphore, waking our thread.
00204     pclTimer->Init();
00205     pclTimer->SetIntervalMSeconds(ulTimeMs_);
00206     pclTimer->SetCallback(ThreadSleepCallback);
00207     pclTimer->SetData((void*)&clSemaphore);
00208     pclTimer->SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00209
00210     // Add the new timer to the timer scheduler, and block the thread
00211     TimerScheduler::Add(pclTimer);
00212     clSemaphore.Pend();
00213 }
00214
00215 //-----
00216 void Thread::USleep(K_ULONG ulTimeUs_)

```

```

00218 {
00219     Semaphore clSemaphore;
00220     Timer *pclTimer = g_pstCurrent->GetTimer();
00221
00222     // Create a semaphore that this thread will block on
00223     clSemaphore.Init(0, 1);
00224
00225     // Create a one-shot timer that will call a callback that posts the
00226     // semaphore, waking our thread.
00227     pclTimer->Init();
00228     pclTimer->SetIntervalUseconds(ulTimeUs_);
00229     pclTimer->SetCallback(ThreadSleepCallback);
00230     pclTimer->SetData((void*)&clSemaphore);
00231     pclTimer->SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00232
00233     // Add the new timer to the timer scheduler, and block the thread
00234     TimerScheduler::Add(pclTimer);
00235     clSemaphore.Pend();
00236 }
00237 #endif // KERNEL_USE_SLEEP
00238
00239 //-----
00240 K_USHORT Thread::GetStackSlack()
00241 {
00242     K_USHORT usCount = 0;
00243
00244     CS_ENTER();
00245
00246     for (usCount = 0; usCount < m_usStackSize; usCount++)
00247     {
00248         if (m_pwStack[usCount] != 0xFF)
00249         {
00250             break;
00251         }
00252     }
00253
00254     CS_EXIT();
00255
00256     return usCount;
00257 }
00258
00259 //-----
00260 void Thread::Yield()
00261 {
00262     CS_ENTER();
00263
00264     // Run the scheduler
00265     if (Scheduler::IsEnabled())
00266     {
00267         Scheduler::Schedule();
00268
00269         // Only switch contexts if the new task is different than the old task
00270         if (Scheduler::GetCurrentThread() !=
00271             Scheduler::GetNextThread())
00272         {
00273             #if KERNEL_USE_QUANTUM
00274                 // new thread scheduled. Stop current quantum timer (if it exists),
00275                 // and restart it for the new thread (if required).
00276                 Quantum::RemoveThread();
00277                 Quantum::AddThread(g_pstNext);
00278             #endif
00279             Thread::ContextSwitchSWI();
00280         }
00281     }
00282     else
00283     {
00284         Scheduler::QueueScheduler();
00285     }
00286
00287     CS_EXIT();
00288 }
00289
00290 //-----
00291 void Thread::SetPriorityBase(K_UCHAR ucPriority_)
00292 {
00293     GetCurrent()->Remove(this);
00294
00295     SetCurrent(Scheduler::GetThreadList(
00296         m_ucPriority));
00297
00298     GetCurrent()->Add(this);
00299 }
00300 //-----
00301 void Thread::SetPriority(K_UCHAR ucPriority_)
00302 {
00303     K_UCHAR bSchedule = 0;

```

```

00304     CS_ENTER();
00305     // If this is the currently running thread, it's a good idea to reschedule
00306     // Or, if the new priority is a higher priority than the current thread's.
00307     if ((g_pstCurrent == this) || (ucPriority_ > g_pstCurrent->GetPriority()))
00308     {
00309         bSchedule = 1;
00310     }
00311     Scheduler::Remove(this);
00312     CS_EXIT();
00313
00314     m_ucCurPriority = ucPriority_;
00315     m_ucPriority = ucPriority_;
00316
00317     CS_ENTER();
00318     Scheduler::Add(this);
00319     CS_EXIT();
00320
00321     if (bSchedule)
00322     {
00323         if (Scheduler::IsEnabled())
00324         {
00325             CS_ENTER();
00326             Scheduler::Schedule();
00327             #if KERNEL_USE_QUANTUM
00328                 // new thread scheduled. Stop current quantum timer (if it exists),
00329                 // and restart it for the new thread (if required).
00330                 Quantum::RemoveThread();
00331                 Quantum::AddThread(g_pstNext);
00332             #endif
00333             CS_EXIT();
00334             Thread::ContextSwitchSWI();
00335         }
00336         else
00337         {
00338             Scheduler::QueueScheduler();
00339         }
00340     }
00341 }
00342
00343 //-----
00344 void Thread::InheritPriority(K_UCHAR ucPriority_)
00345 {
00346     SetOwner(Scheduler::GetThreadList(ucPriority_));
00347     m_ucCurPriority = ucPriority_;
00348 }
00349
00350 //-----
00351 void Thread::ContextSwitchSWI()
00352 {
00353     // Call the context switch interrupt if the scheduler is enabled.
00354     if (Scheduler::IsEnabled() == 1)
00355     {
00356         KERNEL_TRACE_1( STR_CONTEXT_SWITCH_1, (K_USHORT)g_pstNext->GetID() );
00357         KernelSWI::Trigger();
00358     }
00359 }
00360
00361 //-----
00362 Timer *Thread::GetTimer() { return &
    m_clTimer; }
00363 //-----
00364 void Thread::SetExpired( K_BOOL bExpired_ ) { m_bExpired = bExpired; }
00365 //-----
00366 K_BOOL Thread::GetExpired() { return m_bExpired; }

```

17.177 /home/mo/mark3-source/embedded/stage/src/thread.h File Reference

Platform independent thread class declarations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "threadlist.h"
#include "scheduler.h"
#include "threadport.h"
#include "quantum.h"

```

Classes

- class [Thread](#)

Object providing fundamental multitasking support in the kernel.

Typedefs

- typedef void(* [ThreadEntry_t](#))(void *pvArg_)

Function pointer type used for thread entrypoint functions.

17.177.1 Detailed Description

Platform independent thread class declarations. Threads are an atomic unit of execution, and each instance of the thread class represents an instance of a program running on the processor. The [Thread](#) is the fundamental user-facing object in the kernel - it is what makes multiprocessing possible from application code.

In Mark3, threads each have their own context - consisting of a stack, and all of the registers required to multiplex a processor between multiple threads.

The [Thread](#) class inherits directly from the [LinkedListNode](#) class to facilitate efficient thread management using Double, or Double-Circular linked lists.

Definition in file [thread.h](#).

17.178 thread.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00035 #ifndef __THREAD_H__
00036 #define __THREAD_H__
00037
00038 #include "kerneltypes.h"
00039 #include "mark3cfg.h"
00040
00041 #include "ll.h"
00042 #include "threadlist.h"
00043 #include "scheduler.h"
00044 #include "threadport.h"
00045 #include "quantum.h"
00046
00047 //-----
00051 typedef void (*ThreadEntry_t)(void *pvArg_);
00052 class Timer;
00053 //-----
00057 class Thread : public LinkedListNode
00058 {
00059 public:
00079     void Init(K_WORD *paucStack_,
00080              K_USHORT usStackSize_,
00081              K_UCHAR ucPriority_,
00082              ThreadEntry_t pfEntryPoint_,
00083              void *pvArg_ );
00084
00092     void Start();
00093
00094
00101     void Stop();
00102
00103 #if KERNEL_USE_THREADNAME
00104

```

```

00113     void SetName(const K_CHAR *szName_) { m_szName = szName_; }
00114
00121     const K_CHAR* GetName() { return m_szName; }
00122 #endif
00123
00132     ThreadList *GetOwner(void) { return m_pclOwner; }
00133
00141     ThreadList *GetCurrent(void) { return m_pclCurrent; }
00142
00151     K_UCHAR GetPriority(void) { return m_ucPriority; }
00152
00160     K_UCHAR GetCurPriority(void) { return m_ucCurPriority; }
00161
00162 #if KERNEL_USE_QUANTUM
00163
00170     void SetQuantum( K_USHORT usQuantum_ ) { m_usQuantum = usQuantum_; }
00171
00179     K_USHORT GetQuantum(void) { return m_usQuantum; }
00180 #endif
00181
00189     void SetCurrent( ThreadList *pclNewList_ ) {
m_pclCurrent = pclNewList_; }
00190
00198     void SetOwner( ThreadList *pclNewList_ ) { m_pclOwner = pclNewList_; }
00199
00200
00213     void SetPriority(K_UCHAR ucPriority_);
00214
00224     void InheritPriority(K_UCHAR ucPriority_);
00225
00226 #if KERNEL_USE_DYNAMIC_THREADS
00227
00238     void Exit();
00239 #endif
00240
00241 #if KERNEL_USE_SLEEP
00242
00250     static void Sleep(K_ULONG ulTimeMs_);
00251
00260     static void USleep(K_ULONG ulTimeUs_);
00261 #endif
00262
00270     static void Yield(void);
00271
00279     void SetID( K_UCHAR ucID_ ) { m_ucThreadID = ucID_; }
00280
00288     K_UCHAR GetID() { return m_ucThreadID; }
00289
00290
00303     K_USHORT GetStackSlack();
00304
00305 #if KERNEL_USE_EVENTFLAG
00306
00313     K_USHORT GetEventFlagMask() { return m_usFlagMask; }
00314
00319     void SetEventFlagMask(K_USHORT usMask_) { m_usFlagMask = usMask_; }
00320
00326     void SetEventFlagMode(EventFlagOperation_t eMode_ ) {
m_eFlagMode = eMode_; }
00327
00332     EventFlagOperation_t GetEventFlagMode() { return m_eFlagMode; }
00333 #endif
00334
00335 #if KERNEL_USE_TIMERS
00336
00339     Timer *GetTimer();
00340     void SetExpired( K_BOOL bExpired_ );
00341     K_BOOL GetExpired();
00342 #endif
00343
00344     friend class ThreadPort;
00345
00346 private:
00354     static void ContextSwitchSWI(void);
00355
00360     void SetPriorityBase(K_UCHAR ucPriority_);
00361
00363     K_WORD *m_pwStackTop;
00364
00366     K_WORD *m_pwStack;
00367
00369     K_USHORT m_usStackSize;
00370
00371 #if KERNEL_USE_QUANTUM
00372     K_USHORT m_usQuantum;
00374 #endif
00375

```



```

00023 #include "ll.h"
00024 #include "threadlist.h"
00025 #include "thread.h"
00026 #include "kernel_debug.h"
00027 //-----
00028 #if defined __FILE_ID__
00029     #undef __FILE_ID__
00030 #endif
00031 #define __FILE_ID__      THREADLIST_CPP
00032
00033 //-----
00034 void ThreadList::SetPriority(K_UCHAR ucPriority_)
00035 {
00036     m_ucPriority = ucPriority_;
00037 }
00038
00039 //-----
00040 void ThreadList::SetFlagPointer( K_UCHAR *pucFlag_)
00041 {
00042     m_pucFlag = pucFlag_;
00043 }
00044
00045 //-----
00046 void ThreadList::Add(LinkListNode *node_) {
00047     CircularLinkList::Add(node_);
00048
00049     // If the head of the list isn't empty,
00050     if (m_pstHead != NULL)
00051     {
00052         // We've specified a bitmap for this threadlist
00053         if (m_pucFlag)
00054         {
00055             // Set the flag for this priority level
00056             *m_pucFlag |= (1 << m_ucPriority);
00057         }
00058     }
00059 }
00060
00061 //-----
00062 void ThreadList::Add(LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_)
00063 {
00064     // Set the threadlist's priority level, flag pointer, and then add the
00065     // thread to the threadlist
00066     SetPriority(ucPriority_);
00067     SetFlagPointer(pucFlag_);
00068     Add(node_);
00069 }
00070 //-----
00071 void ThreadList::Remove(LinkListNode *node_) {
00072     // Remove the thread from the list
00073     CircularLinkList::Remove(node_);
00074
00075     // If the list is empty...
00076     if (!m_pstHead)
00077     {
00078         // Clear the bit in the bitmap at this priority level
00079         if (m_pucFlag)
00080         {
00081             *m_pucFlag &= ~(1 << m_ucPriority);
00082         }
00083     }
00084 }
00085
00086 //-----
00087 Thread *ThreadList::HighestWaiter()
00088 {
00089     Thread *pclTemp = static_cast<Thread*>(GetHead());
00090     Thread *pclChosen = pclTemp;
00091
00092     K_UCHAR ucMaxPri = 0;
00093
00094     // Go through the list, return the highest-priority thread in this list.
00095     while(1)
00096     {
00097         // Compare against current max-priority thread
00098         if (pclTemp->GetPriority() >= ucMaxPri)
00099         {
00100             ucMaxPri = pclTemp->GetPriority();
00101             pclChosen = pclTemp;
00102         }
00103
00104         // Break out if this is the last thread in the list
00105         if (pclTemp == static_cast<Thread*>(GetTail()))
00106         {
00107             break;
00108         }
00109     }

```

```

00109
00110     pclTemp = static_cast<Thread*>(pclTemp->GetNext());
00111 }
00112 return pclChosen;
00113 }

```

17.181 /home/mo/mark3-source/embedded/stage/src/threadlist.h File Reference

[Thread](#) linked-list declarations.

```

#include "kerneltypes.h"
#include "ll.h"

```

Classes

- class [ThreadList](#)

This class is used for building thread-management facilities, such as schedulers, and blocking objects.

17.181.1 Detailed Description

[Thread](#) linked-list declarations.

Definition in file [threadlist.h](#).

17.182 threadlist.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #ifndef __THREADLIST_H__
00023 #define __THREADLIST_H__
00024
00025 #include "kerneltypes.h"
00026 #include "ll.h"
00027
00028 class Thread;
00029
00034 class ThreadList : public CircularLinkedList
00035 {
00036 public:
00040     ThreadList() { m_ucPriority = 0; m_pucFlag = NULL; }
00041
00049     void SetPriority(K_UCHAR ucPriority_);
00050
00059     void SetFlagPointer(K_UCHAR *pucFlag_);
00060
00068     void Add(LinkListNode *node_);
00069
00083     void Add(LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_);
00084
00092     void Remove(LinkListNode *node_);
00093
00101     Thread *HighestWaiter();
00102 private:
00103
00105     K_UCHAR m_ucPriority;
00106
00108     K_UCHAR *m_pucFlag;
00109 };

```


17.183 /home/mo/mark3-source/embedded/stage/src/threadport.cpp File Reference

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "threadport.h"
#include "kernelswi.h"
#include "kerneltimer.h"
#include "timerlist.h"
#include "quantum.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

- static void **Thread_Switch** (void)
- **ISR** (INT0_vect) __attribute__((signal))
SWI using INT0 - used to trigger a context switch.
- **ISR** (TIMER1_COMPA_vect)
Timer interrupt ISR - causes a tick, which may cause a context switch.

- Thread * g_pstCurrentThread
- naked

Definition in file [threadport.cpp](#).

[illegible]

```

00028 #include "timerlist.h"
00029 #include "quantum.h"
00030 #include <avr/io.h>
00031 #include <avr/interrupt.h>
00032
00033 //-----
00034 Thread *g_pstCurrentThread;
00035
00036 //-----
00037 void ThreadPort::InitStack(Thread *pclThread_)
00038 {
00039     // Initialize the stack for a Thread
00040     K_USHORT usAddr;
00041     K_UCHAR *pucStack;
00042     K_USHORT i;
00043
00044     // Get the address of the thread's entry function
00045     usAddr = (K_USHORT) (pclThread_>m_pfEntryPoint);
00046
00047     // Start by finding the bottom of the stack
00048     pucStack = (K_UCHAR*)pclThread_>m_pwStackTop;
00049
00050     // clear the stack, and initialize it to a known-default value (easier
00051     // to debug when things go sour with stack corruption or overflow)
00052     for (i = 0; i < pclThread_>m_usStackSize; i++)
00053     {
00054         pclThread_>m_pwStack[i] = 0xFF;
00055     }
00056
00057     // Our context starts with the entry function
00058     PUSH_TO_STACK(pucStack, (K_UCHAR) (usAddr & 0x00FF));
00059     PUSH_TO_STACK(pucStack, (K_UCHAR) ((usAddr >> 8) & 0x00FF));
00060
00061     // R0
00062     PUSH_TO_STACK(pucStack, 0x00); // R0
00063
00064     // Push status register and R1 (which is used as a constant zero)
00065     PUSH_TO_STACK(pucStack, 0x80); // SR
00066     PUSH_TO_STACK(pucStack, 0x00); // R1
00067
00068     // Push other registers
00069     for (i = 2; i <= 23; i++) //R2-R23
00070     {
00071         PUSH_TO_STACK(pucStack, i);
00072     }
00073
00074     // Assume that the argument is the only stack variable
00075     PUSH_TO_STACK(pucStack, (K_UCHAR) (((K_USHORT) (pclThread_>
m_pvArg)) & 0x00FF)); //R24
00076     PUSH_TO_STACK(pucStack, (K_UCHAR) (((K_USHORT) (pclThread_>
m_pvArg))>>8) & 0x00FF)); //R25
00077
00078     // Push the rest of the registers in the context
00079     for (i = 26; i <= 31; i++)
00080     {
00081         PUSH_TO_STACK(pucStack, i);
00082     }
00083
00084     // Set the top o' the stack.
00085     pclThread_>m_pwStackTop = (K_UCHAR*)pucStack;
00086
00087     // That's it! the thread is ready to run now.
00088 }
00089
00090 //-----
00091 static void Thread_Switch(void)
00092 {
00093     g_pstCurrent = g_pstNext;
00094 }
00095
00096
00097 //-----
00098 void ThreadPort::StartThreads()
00099 {
00100     KernelSWI::Config(); // configure the task switch SWI
00101     KernelTimer::Config(); // configure the kernel timer
00102
00103     Scheduler::SetScheduler(1); // enable the scheduler
00104     Scheduler::Schedule(); // run the scheduler - determine the first
thread to run
00105
00106     Thread_Switch(); // Set the next scheduled thread to the current thread
00107
00108     KernelTimer::Start(); // enable the kernel timer
00109     KernelSWI::Start(); // enable the task switch SWI
00110
00111     // Restore the context...

```

```

00112     Thread_RestoreContext();           // restore the context of the first running thread
00113     ASM("reti");                       // return from interrupt - will return to the first scheduled thread
00114 }
00115
00116 //-----
00121 //-----
00122 ISR(INT0_vect) __attribute__ ( ( signal, naked ) );
00123 ISR(INT0_vect)
00124 {
00125     Thread_SaveContext();              // Push the context (registers) of the current task
00126     Thread_Switch();                  // Switch to the next task
00127     Thread_RestoreContext();           // Pop the context (registers) of the next task
00128     ASM("reti");                      // Return to the next task
00129 }
00130
00131 //-----
00136 //-----
00137 ISR(TIMER1_COMPA_vect)
00138 {
00139     #if KERNEL_USE_TIMERS
00140         TimerScheduler::Process();
00141     #endif
00142     #if KERNEL_USE_QUANTUM
00143         Quantum::UpdateTimer();
00144     #endif
00145 }

```

17.185 /home/mo/mark3-source/embedded/stage/src/threadport.h File Reference

ATMega328p Multithreading support.

```

#include "kerneltypes.h"
#include "thread.h"
#include <avr/io.h>
#include <avr/interrupt.h>

```

Classes

- class [ThreadPort](#)
Class defining the architecture specific functions required by the kernel.

Macros

- #define [ASM](#)(x) asm volatile(x);
ASM Macro - simplify the use of ASM directive in C.
- #define [SR_](#) 0x3F
Status register define - map to 0x003F.
- #define [SPH_](#) 0x3E
Stack pointer define.
- #define [SPL_](#) 0x3D
- #define [TOP_OF_STACK](#)(x, y) (K_UCHAR*) (((K_USHORT)x) + (y-1))
Macro to find the top of a stack given its size and top address.
- #define [PUSH_TO_STACK](#)(x, y) *x = y; x--;
Push a value y to the stack pointer x and decrement the stack pointer.
- #define [Thread_SaveContext](#)()
Save the context of the [Thread](#).
- #define [Thread_RestoreContext](#)()
Restore the context of the [Thread](#).
- #define [CS_ENTER](#)()
These macros must be used in pairs !


```

00030 //-----
00032 #define ASM(x)      asm volatile(x);
00033 #define SR_        0x3F
00035 #define SPH_        0x3E
00037 #define SPL_        0x3D
00038
00039
00040 //-----
00042 #define TOP_OF_STACK(x, y)      ((K_UCHAR*) ( ((K_USHORT)x) + (y-1) ))
00043 #define PUSH_TO_STACK(x, y)    *x = y; x--;
00045
00046 //-----
00048 #define Thread_SaveContext() \
00049 ASM("push r0"); \
00050 ASM("in r0, __SREG__"); \
00051 ASM("cli"); \
00052 ASM("push r0"); \
00053 ASM("push r1"); \
00054 ASM("clr r1"); \
00055 ASM("push r2"); \
00056 ASM("push r3"); \
00057 ASM("push r4"); \
00058 ASM("push r5"); \
00059 ASM("push r6"); \
00060 ASM("push r7"); \
00061 ASM("push r8"); \
00062 ASM("push r9"); \
00063 ASM("push r10"); \
00064 ASM("push r11"); \
00065 ASM("push r12"); \
00066 ASM("push r13"); \
00067 ASM("push r14"); \
00068 ASM("push r15"); \
00069 ASM("push r16"); \
00070 ASM("push r17"); \
00071 ASM("push r18"); \
00072 ASM("push r19"); \
00073 ASM("push r20"); \
00074 ASM("push r21"); \
00075 ASM("push r22"); \
00076 ASM("push r23"); \
00077 ASM("push r24"); \
00078 ASM("push r25"); \
00079 ASM("push r26"); \
00080 ASM("push r27"); \
00081 ASM("push r28"); \
00082 ASM("push r29"); \
00083 ASM("push r30"); \
00084 ASM("push r31"); \
00085 ASM("lds r26, g_pstCurrent"); \
00086 ASM("lds r27, g_pstCurrent + 1"); \
00087 ASM("adiw r26, 4"); \
00088 ASM("in r0, 0x3D"); \
00089 ASM("st x+, r0"); \
00090 ASM("in r0, 0x3E"); \
00091 ASM("st x+, r0");
00092
00093 //-----
00095 #define Thread_RestoreContext() \
00096 ASM("lds r26, g_pstCurrent"); \
00097 ASM("lds r27, g_pstCurrent + 1"); \
00098 ASM("adiw r26, 4"); \
00099 ASM("ld r28, x+"); \
00100 ASM("out 0x3D, r28"); \
00101 ASM("ld r29, x+"); \
00102 ASM("out 0x3E, r29"); \
00103 ASM("pop r31"); \
00104 ASM("pop r30"); \
00105 ASM("pop r29"); \
00106 ASM("pop r28"); \
00107 ASM("pop r27"); \
00108 ASM("pop r26"); \
00109 ASM("pop r25"); \
00110 ASM("pop r24"); \
00111 ASM("pop r23"); \
00112 ASM("pop r22"); \
00113 ASM("pop r21"); \
00114 ASM("pop r20"); \
00115 ASM("pop r19"); \
00116 ASM("pop r18"); \
00117 ASM("pop r17"); \
00118 ASM("pop r16"); \
00119 ASM("pop r15"); \
00120 ASM("pop r14"); \
00121 ASM("pop r13"); \
00122 ASM("pop r12"); \
00123 ASM("pop r11"); \

```

```

00124 ASM("pop r10"); \
00125 ASM("pop r9"); \
00126 ASM("pop r8"); \
00127 ASM("pop r7"); \
00128 ASM("pop r6"); \
00129 ASM("pop r5"); \
00130 ASM("pop r4"); \
00131 ASM("pop r3"); \
00132 ASM("pop r2"); \
00133 ASM("pop r1"); \
00134 ASM("pop r0"); \
00135 ASM("out __SREG__, r0"); \
00136 ASM("pop r0");
00137
00138 //-----
00140 //-----
00142 #define CS_ENTER() \
00143 { \
00144 volatile K_UCHAR x; \
00145 x = _SFR_IO8(SR_); \
00146 ASM("cli");
00147 //-----
00149 #define CS_EXIT() \
00150 _SFR_IO8(SR_) = x;\
00151 }
00152
00153 //-----
00155 #define ENABLE_INTS()      ASM("sei");
00156 #define DISABLE_INTS()    ASM("cli");
00157
00158 //-----
00159 class Thread;
00167 class ThreadPort
00168 {
00169 public:
00175     static void StartThreads();
00176     friend class Thread;
00177 private:
00178
00186     static void InitStack(Thread *pstThread_);
00187 };
00188
00189 #endif //__ThreadPORT_H_

```

17.187 /home/mo/mark3-source/embedded/stage/src/timerlist.cpp File Reference

[Timer](#) data structure + scheduler implementations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "timerlist.h"
#include "kerneltimer.h"
#include "threadport.h"
#include "kernel_debug.h"

```

Macros

- `#define __FILE_ID__ TIMERLIST_CPP`

17.187.1 Detailed Description

[Timer](#) data structure + scheduler implementations.

Definition in file [timerlist.cpp](#).

17.188 timerlist.cpp

```

00001 /*=====

```

```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024
00025 #include "timerlist.h"
00026 #include "kerneltimer.h"
00027 #include "threadport.h"
00028 #include "kernel_debug.h"
00029 //-----
00030 #if defined __FILE_ID__
00031 #undef __FILE_ID__
00032 #endif
00033 #define __FILE_ID__ TIMERLIST_CPP
00034
00035 #if KERNEL_USE_TIMERS
00036
00037 //-----
00038 TimerList TimerScheduler::m_clTimerList;
00039
00040 //-----
00041 void TimerList::Init(void)
00042 {
00043     m_bTimerActive = 0;
00044     m_ulNextWakeup = 0;
00045 }
00046
00047 //-----
00048 void TimerList::Add(Timer *pclListNode_)
00049 {
00050     #if KERNEL_TIMERS_TICKLESS
00051         K_UCHAR bStart = 0;
00052     #endif
00053
00054     K_LONG lDelta;
00055     CS_ENTER();
00056
00057     #if KERNEL_TIMERS_TICKLESS
00058         if (GetHead() == NULL)
00059         {
00060             bStart = 1;
00061         }
00062     #endif
00063
00064     pclListNode_>ClearNode();
00065     DoubleLinkedList::Add(pclListNode_);
00066
00067     // Set the initial timer value
00068     pclListNode_>m_ulTimeLeft = pclListNode_>m_ulInterval;
00069
00070     #if KERNEL_TIMERS_TICKLESS
00071         if (!bStart)
00072         {
00073             // If the new interval is less than the amount of time remaining...
00074             lDelta = KernelTimer::TimeToExpiry() - pclListNode_>
m_ulInterval;
00075
00076             if (lDelta > 0)
00077             {
00078                 // Set the new expiry time on the timer.
00079                 m_ulNextWakeup = KernelTimer::SubtractExpiry((K_ULONG)
lDelta);
00080             }
00081         }
00082         else
00083         {
00084             m_ulNextWakeup = pclListNode_>m_ulInterval;
00085             KernelTimer::SetExpiry(m_ulNextWakeup);
00086             KernelTimer::Start();
00087         }
00088     #endif
00089
00090     // Set the timer as active.
00091     pclListNode_>m_ucFlags |= TIMERLIST_FLAG_ACTIVE;
00092     CS_EXIT();
00093 }
00094

```

```

00095 //-----
00096 void TimerList::Remove(Timer *pclLinkListNode_)
00097 {
00098     CS_ENTER();
00099
00100     DoubleLinkedList::Remove(pclLinkListNode_);
00101
00102     #if KERNEL_TIMERS_TICKLESS
00103     if (this->GetHead() == NULL)
00104     {
00105         KernelTimer::Stop();
00106     }
00107     #endif
00108
00109     CS_EXIT();
00110 }
00111
00112 //-----
00113 void TimerList::Process(void)
00114 {
00115     #if KERNEL_TIMERS_TICKLESS
00116     K_ULONG ulNewExpiry;
00117     K_ULONG ulOvertime;
00118     K_UCHAR bContinue;
00119     #endif
00120
00121     Timer *pclNode;
00122     Timer *pclPrev;
00123
00124     #if KERNEL_TIMERS_TICKLESS
00125     // Clear the timer and its expiry time - keep it running though
00126     KernelTimer::ClearExpiry();
00127     do
00128     {
00129     #endif
00130         pclNode = static_cast<Timer*>(GetHead());
00131         pclPrev = NULL;
00132
00133         #if KERNEL_TIMERS_TICKLESS
00134         bContinue = 0;
00135         ulNewExpiry = MAX_TIMER_TICKS;
00136         #endif
00137
00138         // Subtract the elapsed time interval from each active timer.
00139         while (pclNode)
00140         {
00141             // Active timers only...
00142             if (pclNode->m_ucFlags & TIMERLIST_FLAG_ACTIVE)
00143             {
00144                 // Did the timer expire?
00145                 #if KERNEL_TIMERS_TICKLESS
00146                 if (pclNode->m_ulTimeLeft <= m_ulNextWakeup)
00147                 #else
00148                 pclNode->m_ulTimeLeft--;
00149                 if (0 == pclNode->m_ulTimeLeft)
00150                 #endif
00151                 {
00152                     // Yes - set the "callback" flag - we'll execute the callbacks later
00153                     pclNode->m_ucFlags |= TIMERLIST_FLAG_CALLBACK;
00154
00155                     if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
00156                     {
00157                         // If this was a one-shot timer, deactivate the timer.
00158                         pclNode->m_ucFlags |= TIMERLIST_FLAG_EXPIRED;
00159                         pclNode->m_ucFlags &= ~TIMERLIST_FLAG_ACTIVE;
00160                     }
00161                     else
00162                     {
00163                         // Reset the interval timer.
00164                         // I think we're good though...
00165                         pclNode->m_ulTimeLeft = pclNode->
00166                             m_ulInterval;
00167
00168                         #if KERNEL_TIMERS_TICKLESS
00169                         // If the time remaining (plus the length of the tolerance interval)
00170                         // is less than the next expiry interval, set the next expiry interval.
00171                         if ((pclNode->m_ulTimeLeft + pclNode->
00172                             m_ulTimerTolerance) < ulNewExpiry)
00173                         {
00174                             ulNewExpiry = pclNode->m_ulTimeLeft + pclNode->
00175                                 m_ulTimerTolerance;
00176                         }
00177                     }
00178                 }
00179             }
00180         }
00181     }
00182     #if KERNEL_TIMERS_TICKLESS
00183     #endif

```



```

00179         else
00180         {
00181             // Not expiring, but determine how K_LONG to run the next timer interval for.
00182             pclNode->m_ulTimeLeft -= m_ulNextWakeup;
00183             if (pclNode->m_ulTimeLeft < ulNewExpiry)
00184             {
00185                 ulNewExpiry = pclNode->m_ulTimeLeft;
00186             }
00187         }
00188     #endif
00189     }
00190     pclNode = static_cast<Timer*>(pclNode->GetNext());
00191 }
00192
00193 // Process the expired timers callbacks.
00194 pclNode = static_cast<Timer*>(GetHead());
00195 while (pclNode)
00196 {
00197     pclPrev = NULL;
00198
00199     // If the timer expired, run the callbacks now.
00200     if (pclNode->m_ucFlags & TIMERLIST_FLAG_CALLBACK)
00201     {
00202         // Run the callback. these callbacks must be very fast...
00203         pclNode->m_pfCallback( pclNode->m_pclOwner, pclNode->
00204             m_pvData );
00205         pclNode->m_ucFlags &= ~TIMERLIST_FLAG_CALLBACK;
00206
00207         // If this was a one-shot timer, let's remove it.
00208         if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
00209         {
00210             pclPrev = pclNode;
00211         }
00212         pclNode = static_cast<Timer*>(pclNode->GetNext());
00213
00214         // Remove one-shot-timers
00215         if (pclPrev)
00216         {
00217             Remove(pclPrev);
00218         }
00219     }
00220
00221     #if KERNEL_TIMERS_TICKLESS
00222     // Check to see how much time has elapsed since the time we
00223     // acknowledged the interrupt...
00224     ulOvertime = KernelTimer::GetOvertime();
00225
00226     if( ulOvertime >= ulNewExpiry ) {
00227         m_ulNextWakeup = ulOvertime;
00228         bContinue = 1;
00229     }
00230
00231     // If it's taken longer to go through this loop than would take us to
00232     // the next expiry, re-run the timing loop
00233     } while (bContinue);
00234
00235     // This timer elapsed, but there's nothing more to do...
00236     // Turn the timer off.
00237     if (ulNewExpiry >= MAX_TIMER_TICKS)
00238     {
00239         KernelTimer::Stop();
00240     }
00241     else
00242     {
00243         // Update the timer with the new "Next Wakeup" value, plus whatever
00244         // overtime has accumulated since the last time we called this handler
00245         m_ulNextWakeup = KernelTimer::SetExpiry(ulNewExpiry +
00246             ulOvertime);
00247     }
00248     #endif
00249 }
00250
00251 //-----
00252 void Timer::Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *
00253     pvData_ )
00254 {
00255     SetIntervalMSeconds(ulIntervalMs_);
00256     m_pfCallback = pfCallback_;
00257     m_pvData = pvData_;
00258     if (!bRepeat_)
00259     {
00260         m_ucFlags = TIMERLIST_FLAG_ONE_SHOT;
00261     }
00262     else
00263     {

```

```

00263         m_ucFlags = 0;
00264     }
00265     m_pclOwner = Scheduler::GetCurrentThread();
00266     TimerScheduler::Add(this);
00267 }
00268
00269 //-----
00270 void Timer::Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_,
TimerCallback_t pfCallback_, void *pvData_ )
00271 {
00272     m_ulTimerTolerance = MSECONDS_TO_TICKS(ulToleranceMs_);
00273     Start(bRepeat_, ulIntervalMs_, pfCallback_, pvData_);
00274 }
00275
00276 //-----
00277 void Timer::Stop()
00278 {
00279     TimerScheduler::Remove(this);
00280 }
00281
00282 //-----
00283 void Timer::SetIntervalTicks( K_ULONG ulTicks_ )
00284 {
00285     m_ulInterval = ulTicks_;
00286 }
00287
00288 //-----
00289 //-----
00290 void Timer::SetIntervalSeconds( K_ULONG ulSeconds_ )
00291 {
00292     m_ulInterval = SECONDS_TO_TICKS(ulSeconds_);
00293 }
00294
00295 //-----
00296 void Timer::SetIntervalMSeconds( K_ULONG ulMSeconds_ )
00297 {
00298     m_ulInterval = MSECONDS_TO_TICKS(ulMSeconds_);
00299 }
00300
00301 //-----
00302 void Timer::SetIntervalUSeconds( K_ULONG ulUSeconds_ )
00303 {
00304     m_ulInterval = USECONDS_TO_TICKS(ulUSeconds_);
00305 }
00306
00307 //-----
00308 void Timer::SetTolerance(K_ULONG ulTicks_)
00309 {
00310     m_ulTimerTolerance = ulTicks_;
00311 }
00312
00313
00314
00315 #endif //KERNEL_USE_TIMERS

```

17.189 /home/mo/mark3-source/embedded/stage/src/timerlist.h File Reference

[Timer](#) list and timer-scheduling declarations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"

```

Classes

- class [Timer](#)
 - [Timer](#) - an event-driven execution context based on a specified time interval.*
- class [TimerList](#)
 - [TimerList](#) class - a doubly-linked-list of timer objects.*
- class [TimerScheduler](#)
 - "Static" Class used to interface a global [TimerList](#) with the rest of the kernel.*

Macros

- #define **TIMERLIST_FLAG_ONE_SHOT** (0x01)
Timer is one-shot.
- #define **TIMERLIST_FLAG_ACTIVE** (0x02)
Timer is currently active.
- #define **TIMERLIST_FLAG_CALLBACK** (0x04)
Timer is pending a callback.
- #define **TIMERLIST_FLAG_EXPIRED** (0x08)
Timer is actually expired.
- #define **MAX_TIMER_TICKS** (0x7FFFFFFF)
Maximum value to set.
- #define **SECONDS_TO_TICKS**(x) (((K_ULONG)x) * TIMER_FREQ))
- #define **MSECONDS_TO_TICKS**(x) (((((K_ULONG)x) * (TIMER_FREQ/100)) + 5) / 10))
- #define **USECONDS_TO_TICKS**(x) (((((K_ULONG)x) * TIMER_FREQ) + 50000) / 1000000))
- #define **MIN_TICKS** (3)
The minimum tick value to set.

Typedefs

- typedef void(* **TimerCallback_t**)(**Thread** *pclOwner_, void *pvData_)

17.189.1 Detailed Description

Timer list and timer-scheduling declarations. These classes implements a linked list of timer objects attached to the global kernel timer. Unlike other kernels which use a fully-synchronous "tick-based" timing mechanism, where the OS timing facilities are based on a fixed-frequency timer (which causes regular timer interrupts), Mark3 uses a "tickless" timer implementation, which only triggers interrupts when absolutely required. This is much more efficient in most cases - timer interrupts occur less frequently, allowing the kernel to stay in sleep much longer than it would otherwise.

Definition in file [timerlist.h](#).

17.189.2 Macro Definition Documentation

17.189.2.1 #define TIMERLIST_FLAG_EXPIRED (0x08)

Timer is actually expired.

Definition at line 45 of file timerlist.h.

17.190 timerlist.h

```
00001 /*=====
00002
00003 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
00004 | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / |
00005 |  V  |  V  |  V  |  V  |  V  |  V  |  V  |  V  |  V  |  V  |
00006 |_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|_/ \_|
00007 |_____||_____||_____||_____||_____||_____||_____||
00008
00009 --[Mark3 Realtime Platform]-----*/
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00030 #ifndef __TIMERLIST_H__
00031 #define    TIMERLIST H
```

```

00032
00033 #include "kerneltypes.h"
00034 #include "mark3cfg.h"
00035
00036 #include "ll.h"
00037
00038 #if KERNEL_USE_TIMERS
00039 class Thread;
00040
00041 //-----
00042 #define TIMERLIST_FLAG_ONE_SHOT      (0x01)
00043 #define TIMERLIST_FLAG_ACTIVE       (0x02)
00044 #define TIMERLIST_FLAG_CALLBACK     (0x04)
00045 #define TIMERLIST_FLAG_EXPIRED      (0x08)
00046
00047 //-----
00048 #if KERNEL_TIMERS_TICKLESS
00049
00050 //-----
00051 #define MAX_TIMER_TICKS              (0x7FFFFFFF)
00052
00053 //-----
00054 /*
00055     Ugly macros to support a wide resolution of delays.
00056     Given a 16-bit timer @ 16MHz & 256 cycle prescaler, this gives us...
00057     Max time, SECONDS_TO_TICKS:  68719s
00058     Max time, MSECONDS_TO_TICKS: 6871.9s
00059     Max time, USECONDS_TO_TICKS: 6.8719s
00060     With a 16us tick resolution.
00061 */
00062 //-----
00063 #define SECONDS_TO_TICKS(x)          (((K_ULONG)x) * TIMER_FREQ)
00064 #define MSECONDS_TO_TICKS(x)        (((((K_ULONG)x) * (TIMER_FREQ/100)) + 5) / 10))
00065 #define USECONDS_TO_TICKS(x)        (((((K_ULONG)x) * TIMER_FREQ) + 50000) / 1000000))
00066
00067 //-----
00068 #define MIN_TICKS                    (3)
00069 //-----
00070
00071 #else
00072 //-----
00073 // Tick-based timers, assuming 1khz tick rate
00074 #define MAX_TIMER_TICKS              (0x7FFFFFFF)
00075
00076 //-----
00077 #define SECONDS_TO_TICKS(x)          ((K_ULONG)(x) * 1000)
00078 #define MSECONDS_TO_TICKS(x)        ((K_ULONG)(x))
00079 #define USECONDS_TO_TICKS(x)        (((K_ULONG)(x) + 999) / 1000)
00080
00081 //-----
00082 #define MIN_TICKS                    (1)
00083 //-----
00084
00085 #endif // KERNEL_TIMERS_TICKLESS
00086
00087 typedef void (*TimerCallback_t)(Thread *pclOwner_, void *pvData_);
00088
00089 //-----
00090 class TimerList;
00091 class TimerScheduler;
00092 class Quantum;
00093 class Timer : public LinkListNode
00094 {
00095 public:
00104     Timer() { Init(); }
00105
00109     void Init() { m_ulInterval = 0; m_ulTimerTolerance = 0;
m_ulTimeLeft = 0; m_ucFlags = 0; }
00110
00120     void Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *pvData_ );
00121
00133     void Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_, TimerCallback_t
pfCallback_, void *pvData_ );
00134
00139     void Stop();
00140
00150     void SetFlags (K_UCHAR ucFlags_) { m_ucFlags = ucFlags_; }
00151
00159     void SetCallback( TimerCallback_t pfCallback_){ m_pfCallback = pfCallback_; }
00160
00168     void SetData( void *pvData_ ){ m_pvData = pvData_; }
00169
00178     void SetOwner( Thread *pclOwner_){ m_pclOwner = pclOwner_; }
00179
00187     void SetIntervalTicks(K_ULONG ulTicks_);
00188
00196     void SetIntervalSeconds(K_ULONG ulSeconds_);

```

```

00197
00198
00199     K_ULONG GetInterval()    { return m_ulInterval; }
00200
00201     void SetIntervalMSeconds(K_ULONG ulMSeconds_);
00202
00203     void SetIntervalUSeconds(K_ULONG ulUSeconds_);
00204
00205     void SetTolerance(K_ULONG ulTicks_);
00206
00207 private:
00208
00209     friend class TimerList;
00210
00211     K_UCHAR m_ucFlags;
00212
00213     TimerCallback_t m_pfnCallback;
00214
00215     K_ULONG m_ulInterval;
00216
00217     K_ULONG m_ulTimeLeft;
00218
00219     K_ULONG m_ulTimerTolerance;
00220
00221     Thread *m_pclOwner;
00222
00223     void *m_pvData;
00224 };
00225
00226 //-----
00227 class TimerList : public DoubleLinkedList
00228 {
00229 public:
00230     void Init();
00231
00232     void Add(Timer *pclListNode_);
00233
00234     void Remove(Timer *pclListNode_);
00235
00236     void Process();
00237 private:
00238     K_ULONG m_ulNextWakeUp;
00239
00240     K_UCHAR m_bTimerActive;
00241 };
00242
00243 //-----
00244 class TimerScheduler
00245 {
00246 public:
00247     static void Init() { m_clTimerList.Init(); }
00248
00249     static void Add(Timer *pclListNode_)
00250     { m_clTimerList.Add(pclListNode_); }
00251
00252     static void Remove(Timer *pclListNode_)
00253     { m_clTimerList.Remove(pclListNode_); }
00254
00255     static void Process() { m_clTimerList.Process(); }
00256 private:
00257     static TimerList m_clTimerList;
00258 };
00259
00260 #endif // KERNEL_USE_TIMERS
00261
00262 #endif

```

17.191 /home/mo/mark3-source/embedded/stage/src/tracebuffer.cpp File Reference

Kernel trace buffer class definition.

```

#include "kerneltypes.h"
#include "tracebuffer.h"
#include "mark3cfg.h"
#include "writebuf16.h"
#include "kernel_debug.h"

```

17.191.1 Detailed Description

[Kernel](#) trace buffer class definition.

Definition in file [tracebuffer.cpp](#).

17.192 tracebuffer.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "kerneltypes.h"
00020 #include "tracebuffer.h"
00021 #include "mark3cfg.h"
00022 #include "writebuf16.h"
00023 #include "kernel_debug.h"
00024
00025 #if KERNEL_USE_DEBUG
00026
00027 //-----
00028 WriteBuffer16 TraceBuffer::m_clBuffer;
00029 volatile K_USHORT TraceBuffer::m_usIndex;
00030 K_USHORT TraceBuffer::m_ausBuffer[ (TRACE_BUFFER_SIZE/sizeof(K_USHORT)) ];
00031
00032 //-----
00033 void TraceBuffer::Init()
00034 {
00035     m_clBuffer.SetBuffers(m_ausBuffer, TRACE_BUFFER_SIZE/sizeof(K_USHORT));
00036     m_usIndex = 0;
00037 }
00038
00039 //-----
00040 K_USHORT TraceBuffer::Increment()
00041 {
00042     return m_usIndex++;
00043 }
00044
00045 //-----
00046 void TraceBuffer::Write( K_USHORT *pusData_, K_USHORT usSize_ )
00047 {
00048     // Pipe the data directly to the circular buffer
00049     m_clBuffer.WriteData(pusData_, usSize_);
00050 }
00051
00052 #endif
00053

```

17.193 /home/mo/mark3-source/embedded/stage/src/tracebuffer.h File Reference

[Kernel](#) trace buffer class declaration.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "writebuf16.h"

```

17.193.1 Detailed Description

[Kernel](#) trace buffer class declaration. Global kernel trace-buffer. Used to instrument the kernel with lightweight encoded print statements. If something goes wrong, the tracebuffer can be examined for debugging purposes. Also,


```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "transaction.h"
00022
00023 //-----
00024 DoubleLinkedList TransactionQueue::m_clGlobalQueue;
00025 Transaction TransactionQueue::m_aclTransactions[
TRANSACTION_QUEUE_SIZE];
00026
00027 //-----
00028 void TransactionQueue::GlobalQueueInit()
00029 {
00030     for (K_UCHAR i = 0; i < TRANSACTION_QUEUE_SIZE; i++)
00031     {
00032         m_clGlobalQueue.Add(&m_aclTransactions[i]);
00033     }
00034 }
00035
00036 //-----
00037 void TransactionQueue::Enqueue( K_USHORT usData_, void *pvData_)
00038 {
00039     // Note - We do not do this from a critical section, as we assume
00040     // that anything calling Enqueue() is already running in a critical
00041     // section.
00042
00043     Transaction *pclTrx;
00044
00045     pclTrx = static_cast<Transaction*>(m_clGlobalQueue.
GetHead());
00046
00047     KERNEL_ASSERT(pclTrx);
00048
00049     m_clGlobalQueue.Remove(pclTrx);
00050
00051     pclTrx->Set( usData_, pvData_ );
00052     Add(pclTrx);
00053 }
00054
00055 //-----
00056 Transaction *TransactionQueue::Dequeue()
00057 {
00058     Transaction *pclTrx;
00059
00060     CS_ENTER();
00061     pclTrx = static_cast<Transaction*>(GetHead());
00062
00063     KERNEL_ASSERT(pclTrx);
00064
00065     Remove(pclTrx);
00066     CS_EXIT();
00067
00068     return pclTrx;
00069 }
00070
00071 //-----
00072 void TransactionQueue::Finish( Transaction *pclTransaction_ )
00073 {
00074     CS_ENTER();
00075     m_clGlobalQueue.Add(pclTransaction_);
00076     CS_EXIT();
00077 }
00078

```

17.197 /home/mo/mark3-source/embedded/stage/src/transaction.h File Reference

[Transaction](#) Queue Implementation.


```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "threadport.h"
#include "ll.h"
#include "kernel_debug.h"
```

Classes

- class [Transaction](#)
The *Transaction* class.
- class [TransactionQueue](#)
The *TransactionQueue* class.

17.197.1 Detailed Description

[Transaction](#) Queue Implementation.

Definition in file [transaction.h](#).

17.198 transaction.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #ifndef __TRANSACTION_H__
00022 #define __TRANSACTION_H__
00023
00024 #include "kerneltypes.h"
00025 #include "mark3cfg.h"
00026 #include "threadport.h"
00027 #include "ll.h"
00028 #include "kernel_debug.h"
00029
00030 //-----
00051 class Transaction : public LinkedListNode
00052 {
00053 public:
00062 void Set( K_USHORT usCode_, void *pvData_ )
00063 {
00064     m_usCode = usCode_;
00065     m_pvData = pvData_;
00066 }
00067
00075 K_USHORT GetCode()
00076 {
00077     return m_usCode;
00078 }
00079
00087 void *GetData()
00088 {
00089     return m_pvData;
00090 }
00091
00092 private:
00093     K_USHORT    m_usCode;
00094     void        *m_pvData;
00095 };
00096
00097 //-----
00138 class TransactionQueue : public DoubleLinkedList
```

```

00139 {
00140 public:
00141     static void GlobalQueueInit();
00142
00143     void Enqueue( K_USHORT usData_, void *pvData_);
00144
00145     Transaction *Dequeue();
00146
00147     void Finish( Transaction *pclTransaction_ );
00148
00149 private:
00150
00151     static DoubleLinkedList m_clGlobalQueue;
00152     static Transaction m_aclTransactions[
00153         TRANSACTION_QUEUE_SIZE];
00154 };
00155 #endif

```

17.199 /home/mo/mark3-source/embedded/stage/src/unit_test.cpp File Reference

Unit test class definition.

```

#include "kerneltypes.h"
#include "unit_test.h"

```

17.199.1 Detailed Description

Unit test class definition.

Definition in file [unit_test.cpp](#).

17.200 unit_test.cpp

```

00001 /*=====
00002
00003  _____  _____  _____  _____  _____
00004  |   /   \  |   /   \  |   /   \  |   /   \  |   /   \
00005  |  /     \  |  /     \  |  /     \  |  /     \  |  /     \
00006  | /       \ | /       \ | /       \ | /       \ | /       \
00007  |/        \|/        \|/        \|/        \|/        \|
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00014
00015 #include "kerneltypes.h"
00016 #include "unit_test.h"
00017
00018 //-----
00019
00020 UnitTest::UnitTest()
00021 {
00022     m_bIsActive = false;
00023     m_usIterations = 0;
00024     m_usPassed = 0;
00025     m_bComplete = false;
00026 }
00027
00028 //-----
00029
00030 void UnitTest::Pass()
00031 {
00032     if (m_bComplete)
00033     {
00034         return;
00035     }
00036
00037     if (m_bIsActive)
00038     {
00039         m_bIsActive = false;
00040         m_usIterations++;
00041         m_usPassed++;
00042         m_bStatus = true;
00043     }
00044 }

```

```
00045     }
00046 }
00047
00048 //-----
00049 void UnitTest::Fail()
00050 {
00051     if (m_bComplete)
00052     {
00053         return;
00054     }
00055
00056     if (m_bIsActive)
00057     {
00058         m_bIsActive = false;
00059         m_usIterations++;
00060         m_bStatus = false;
00061     }
00062 }
```

Unit test class declarations.

```
#include "kerneltypes.h"
```

Classes

- class `UnitTest`
Class used to implement a simple unit-testing framework.

Unit test class declarations.

Definition in file [unit_test.h](#).

```

00001 /*-----*/
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00014 #ifndef __UNIT_TEST_H__
00015 #define __UNIT_TEST_H__
00016
00017
00018
00019 #include "kerneltypes.h"
00020
00021
00022 //-----
00023
00024
00025
00026
00027
00028 class UnitTest
00029 {
00030 public:
00031     UnitTest();
00032
00033
00034 void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }
00035
00036 void Start() { m_bIsActive = 1; }
00037
00038 void Pass();
00039
00040
00041
00042
00043
00044
00045
00046
00047
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
0
```

```

00064     void Fail();
00065
00066     void ExpectTrue( bool bExpression_ )
00067     { bExpression_ ? Pass() : Fail(); }
00068
00069     void ExpectFalse( bool bExpression_ )
00070     { !bExpression_ ? Pass() : Fail(); }
00071
00072     void ExpectEquals( bool bVal_, bool bExpression_ )
00073     { (bVal_ == bExpression_) ? Pass() : Fail(); }
00074
00075     void ExpectEquals( K_UCHAR ucVal_, K_UCHAR ucExpression_ )
00076     { (ucVal_ == ucExpression_) ? Pass() : Fail(); }
00077
00078     void ExpectEquals( K_USHORT usVal_, K_USHORT usExpression_ )
00079     { (usVal_ == usExpression_) ? Pass() : Fail(); }
00080
00081     void ExpectEquals( K_ULONG ulVal_, K_ULONG ulExpression_ )
00082     { (ulVal_ == ulExpression_) ? Pass() : Fail(); }
00083
00084     void ExpectEquals( K_CHAR cVal_, K_CHAR cExpression_ )
00085     { (cVal_ == cExpression_) ? Pass() : Fail(); }
00086
00087     void ExpectEquals( K_SHORT sVal_, K_SHORT sExpression_ )
00088     { (sVal_ == sExpression_) ? Pass() : Fail(); }
00089
00090     void ExpectEquals( K_LONG lVal_, K_LONG lExpression_ )
00091     { (lVal_ == lExpression_) ? Pass() : Fail(); }
00092
00093     void ExpectEquals( void* pvVal_, void* pvExpression_ )
00094     { (pvVal_ == pvExpression_) ? Pass() : Fail(); }
00095
00096
00097     void ExpectFailTrue( bool bExpression_ )
00098     { bExpression_ ? Fail() : Pass(); }
00099
00100     void ExpectFailFalse( bool bExpression_ )
00101     { !bExpression_ ? Fail() : Pass(); }
00102
00103     void ExpectFailEquals( bool bVal_, bool bExpression_ )
00104     { (bVal_ == bExpression_) ? Fail() : Pass(); }
00105
00106     void ExpectFailEquals( K_UCHAR ucVal_, K_UCHAR ucExpression_ )
00107     { (ucVal_ == ucExpression_) ? Fail() : Pass(); }
00108
00109     void ExpectFailEquals( K_USHORT usVal_, K_USHORT usExpression_ )
00110     { (usVal_ == usExpression_) ? Fail() : Pass(); }
00111
00112     void ExpectFailEquals( K_ULONG ulVal_, K_ULONG ulExpression_ )
00113     { (ulVal_ == ulExpression_) ? Fail() : Pass(); }
00114
00115     void ExpectFailEquals( K_CHAR cVal_, K_CHAR cExpression_ )
00116     { (cVal_ == cExpression_) ? Fail() : Pass(); }
00117
00118     void ExpectFailEquals( K_SHORT sVal_, K_SHORT sExpression_ )
00119     { (sVal_ == sExpression_) ? Fail() : Pass(); }
00120
00121     void ExpectFailEquals( K_LONG lVal_, K_LONG lExpression_ )
00122     { (lVal_ == lExpression_) ? Fail() : Pass(); }
00123
00124     void ExpectFailEquals( void* pvVal_, void* pvExpression_ )
00125     { (pvVal_ == pvExpression_) ? Fail() : Pass(); }
00126
00127     void ExpectGreaterThan( K_LONG lVal_, K_LONG lExpression_ )
00128     { (lVal_ > lExpression_) ? Pass() : Fail(); }
00129
00130     void ExpectLessThan( K_LONG lVal_, K_LONG lExpression_ )
00131     { (lVal_ < lExpression_) ? Pass() : Fail(); }
00132
00133     void ExpectGreaterThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00134     { (lVal_ >= lExpression_) ? Pass() : Fail(); }
00135
00136     void ExpectLessThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00137     { (lVal_ <= lExpression_) ? Pass() : Fail(); }
00138
00139     void ExpectFailGreaterThan( K_LONG lVal_, K_LONG lExpression_ )
00140     { (lVal_ > lExpression_) ? Fail() : Pass(); }
00141
00142     void ExpectFailLessThan( K_LONG lVal_, K_LONG lExpression_ )
00143     { (lVal_ < lExpression_) ? Fail() : Pass(); }
00144
00145     void ExpectFailGreaterThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00146     { (lVal_ >= lExpression_) ? Fail() : Pass(); }
00147
00148     void ExpectFailLessThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00149     { (lVal_ <= lExpression_) ? Fail() : Pass(); }
00150

```

```

00157     void Complete() { m_bComplete = 1; }
00158
00166     const K_CHAR *GetName() { return m_szName; }
00167
00175     K_BOOL GetResult() { return m_bStatus; }
00176
00184     K_USHORT GetPassed() { return m_usPassed; }
00185
00193     K_USHORT GetFailed() { return m_usIterations -
m_usPassed; }
00194
00202     K_USHORT GetTotal() { return m_usIterations; }
00203
00204 private:
00205     const K_CHAR *m_szName;
00206     K_BOOL m_bIsActive;
00207     K_UCHAR m_bComplete;
00208     K_BOOL m_bStatus;
00209     K_USHORT m_usIterations;
00210     K_USHORT m_usPassed;
00211 };
00212
00213 #endif

```

17.203 /home/mo/mark3-source/embedded/stage/src/writebuf16.cpp File Reference

16 bit circular buffer implementation with callbacks.

```

#include "kerneltypes.h"
#include "writebuf16.h"
#include "kernel_debug.h"
#include "threadport.h"

```

17.203.1 Detailed Description

16 bit circular buffer implementation with callbacks.

Definition in file [writebuf16.cpp](#).

17.204 writebuf16.cpp

```

00001 /*=====
00002
00003 |-----|-----|-----|-----|-----|-----|-----|-----|
00004 | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / |
00005 | / \ | / \ | / \ | / \ | / \ | / \ | / \ | / \ | / \ |
00006 |-----|-----|-----|-----|-----|-----|-----|-----|
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "kerneltypes.h"
00021 #include "writebuf16.h"
00022 #include "kernel_debug.h"
00023 #include "threadport.h"
00024 //-----
00025 void WriteBuffer16::WriteData( K_USHORT *pusBuf_, K_USHORT usLen_ )
00026 {
00027     K_USHORT *apusBuf[1];
00028     K_USHORT ausLen[1];
00029
00030     apusBuf[0] = pusBuf_;
00031     ausLen[0] = usLen_;
00032
00033     WriteVector( apusBuf, ausLen, 1 );
00034 }
00035
00036 //-----
00037 void WriteBuffer16::WriteVector( K_USHORT **ppusBuf_, K_USHORT *pusLen_, K_UCHAR

```

```

        ucCount_ )
00038 {
00039     K_USHORT usTempHead;
00040     K_UCHAR i;
00041     K_UCHAR j;
00042     K_USHORT usTotalLen = 0;
00043     bool bCallback = false;
00044     bool bRollover = false;
00045     // Update the head pointer synchronously, using a small
00046     // critical section in order to provide thread safety without
00047     // compromising on responsiveness by adding lots of extra
00048     // interrupt latency.
00049
00050     CS_ENTER();
00051
00052     usTempHead = m_usHead;
00053     {
00054         for (i = 0; i < ucCount_; i++)
00055         {
00056             usTotalLen += pusLen_[i];
00057         }
00058         m_usHead = (usTempHead + usTotalLen) % m_usSize;
00059     }
00060     CS_EXIT();
00061
00062     // Call the callback if we cross the 50% mark or rollover
00063     if (m_usHead < usTempHead)
00064     {
00065         if (m_pfCallback)
00066         {
00067             bCallback = true;
00068             bRollover = true;
00069         }
00070     }
00071     else if ((usTempHead < (m_usSize >> 1)) && (m_usHead >= (
m_usSize >> 1)))
00072     {
00073         // Only trigger the callback if it's non-null
00074         if (m_pfCallback)
00075         {
00076             bCallback = true;
00077         }
00078     }
00079
00080     // Are we going to roll-over?
00081     for (j = 0; j < ucCount_; j++)
00082     {
00083         K_USHORT usSegmentLength = pusLen_[j];
00084         if (usSegmentLength + usTempHead >= m_usSize)
00085         {
00086             // We need to two-part this... First part: before the rollover
00087             K_USHORT usTempLen;
00088             K_USHORT *pusTmp = &m_pusData[ usTempHead ];
00089             K_USHORT *pusSrc = ppusBuf_[j];
00090             usTempLen = m_usSize - usTempHead;
00091             for (i = 0; i < usTempLen; i++)
00092             {
00093                 *pusTmp++ = *pusSrc++;
00094             }
00095
00096             // Second part: after the rollover
00097             usTempLen = usSegmentLength - usTempLen;
00098             pusTmp = m_pusData;
00099             for (i = 0; i < usTempLen; i++)
00100             {
00101                 *pusTmp++ = *pusSrc++;
00102             }
00103         }
00104         else
00105         {
00106             // No rollover - do the copy all at once.
00107             K_USHORT *pusSrc = ppusBuf_[j];
00108             K_USHORT *pusTmp = &m_pusData[ usTempHead ];
00109             for (K_USHORT i = 0; i < usSegmentLength; i++)
00110             {
00111                 *pusTmp++ = *pusSrc++;
00112             }
00113         }
00114     }
00115
00116     // Call the callback if necessary
00117     if (bCallback)
00118     {
00119         if (bRollover)
00120         {
00121             // Rollover - process the back-half of the buffer

```

```

00123         m_pfCallback( &m_pusData[ m_usSize >> 1],
00124             m_usSize >> 1 );
00125     }
00126     else
00127     {
00128         // 50% point - process the front-half of the buffer
00129         m_pfCallback( m_pusData, m_usSize >> 1);
00130     }
00131 }

```

17.205 /home/mo/mark3-source/embedded/stage/src/writebuf16.h File Reference

Thread-safe circular buffer implementation with 16-bit elements.

```
#include "kerneltypes.h"
```

Classes

- class [WriteBuffer16](#)

This class is used to provide a general-purpose, fully thread-safe circular buffer implementation which can be used for creating tracebuffers, data logging queues, transaction queues, etc.

Typedefs

- typedef void(* [WriteBufferCallback](#))(K_USHORT *pusData_, K_USHORT usSize_)

Function pointer type used to define a callback handler for when the circular buffer reaches 50% capacity.

17.205.1 Detailed Description

Thread-safe circular buffer implementation with 16-bit elements.

Definition in file [writebuf16.h](#).

17.206 writebuf16.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #ifndef __WRITEBUF16_H__
00021 #define __WRITEBUF16_H__
00022
00023 #include "kerneltypes.h"
00024
00029 typedef void (*WriteBufferCallback)( K_USHORT *pusData_, K_USHORT usSize_ );
00030
00037 class WriteBuffer16
00038 {
00039 public:
00050     void SetBuffers( K_USHORT *pusData_, K_USHORT usSize_ )
00051     {
00052         m_pusData = pusData_;
00053         m_usSize = usSize_;
00054         m_usHead = 0;
00055         m_usTail = 0;

```

```
00056     }
00057
00069     void SetCallback( WriteBufferCallback pfCallback_ )
00070     { m_pfCallback = pfCallback_; }
00071
00080     void WriteData( K_USHORT *pusBuf_, K_USHORT usLen_ );
00081
00091     void WriteVector( K_USHORT **ppusBuf_, K_USHORT *pusLen_, K_UCHAR ucCount_);
00092
00093 private:
00094     K_USHORT *m_pusData;
00095
00096     volatile K_USHORT m_usSize;
00097     volatile K_USHORT m_usHead;
00098     volatile K_USHORT m_usTail;
00099
00100     WriteBufferCallback m_pfCallback;
00101 };
00102
00103 #endif
```


Index

/home/mo/mark3-source/embedded/stage/src/atomic.-
cpp, [243](#)

/home/mo/mark3-source/embedded/stage/src/atomic.h,
[245](#)

/home/mo/mark3-source/embedded/stage/src/blocking.-
cpp, [246](#)

/home/mo/mark3-source/embedded/stage/src/blocking.-
h, [247](#), [248](#)

/home/mo/mark3-source/embedded/stage/src/control_
button.cpp, [249](#)

/home/mo/mark3-source/embedded/stage/src/control_
button.h, [252](#)

/home/mo/mark3-source/embedded/stage/src/control_
checkbox.cpp, [253](#), [254](#)

/home/mo/mark3-source/embedded/stage/src/control_
checkbox.h, [256](#), [257](#)

/home/mo/mark3-source/embedded/stage/src/control_
gamepanel.cpp, [257](#), [258](#)

/home/mo/mark3-source/embedded/stage/src/control_
gamepanel.h, [258](#), [259](#)

/home/mo/mark3-source/embedded/stage/src/control_
groupbox.cpp, [259](#), [260](#)

/home/mo/mark3-source/embedded/stage/src/control_
groupbox.h, [261](#)

/home/mo/mark3-source/embedded/stage/src/control_
label.h, [262](#)

/home/mo/mark3-source/embedded/stage/src/control_
notification.cpp, [263](#)

/home/mo/mark3-source/embedded/stage/src/control_
notification.h, [265](#)

/home/mo/mark3-source/embedded/stage/src/control_
panel.cpp, [266](#)

/home/mo/mark3-source/embedded/stage/src/control_
panel.h, [267](#)

/home/mo/mark3-source/embedded/stage/src/control_
progress.cpp, [268](#)

/home/mo/mark3-source/embedded/stage/src/control_
progress.h, [269](#), [270](#)

/home/mo/mark3-source/embedded/stage/src/control_
slickbutton.h, [270](#), [271](#)

/home/mo/mark3-source/embedded/stage/src/control_
slickprogress.cpp, [271](#), [272](#)

/home/mo/mark3-source/embedded/stage/src/control_
slickprogress.h, [273](#)

/home/mo/mark3-source/embedded/stage/src/dcpu.-
cpp, [274](#), [275](#)

/home/mo/mark3-source/embedded/stage/src/dcpu.h,
[285](#), [289](#)

/home/mo/mark3-source/embedded/stage/src/debug_
tokens.h, [293](#), [295](#)

/home/mo/mark3-source/embedded/stage/src/draw.h,
[296](#), [297](#)

/home/mo/mark3-source/embedded/stage/src/driver.-
cpp, [299](#), [300](#)

/home/mo/mark3-source/embedded/stage/src/driver.h,
[301](#), [302](#)

/home/mo/mark3-source/embedded/stage/src/eventflag.-
cpp, [303](#), [304](#)

/home/mo/mark3-source/embedded/stage/src/eventflag.-
h, [308](#), [309](#)

/home/mo/mark3-source/embedded/stage/src/fixed_
heap.cpp, [310](#)

/home/mo/mark3-source/embedded/stage/src/fixed_
heap.h, [312](#)

/home/mo/mark3-source/embedded/stage/src/font.h,
[313](#), [314](#)

/home/mo/mark3-source/embedded/stage/src/graphics.-
cpp, [314](#), [315](#)

/home/mo/mark3-source/embedded/stage/src/graphics.-
h, [327](#), [328](#)

/home/mo/mark3-source/embedded/stage/src/gui.cpp,
[329](#)

/home/mo/mark3-source/embedded/stage/src/gui.h,
[337](#), [339](#)

/home/mo/mark3-source/embedded/stage/src/kernel.-
cpp, [344](#), [345](#)

/home/mo/mark3-source/embedded/stage/src/kernel.h,
[346](#)

/home/mo/mark3-source/embedded/stage/src/kernel_
debug.h, [347](#)

/home/mo/mark3-source/embedded/stage/src/kernelswi.-
cpp, [348](#), [349](#)

/home/mo/mark3-source/embedded/stage/src/kernelswi.-
h, [350](#)

/home/mo/mark3-source/embedded/stage/src/kerneltimer.-
cpp, [351](#)

/home/mo/mark3-source/embedded/stage/src/kerneltimer.-
h, [353](#), [354](#)

/home/mo/mark3-source/embedded/stage/src/kerneltypes.-
h, [354](#), [355](#)

/home/mo/mark3-source/embedded/stage/src/keycodes.-
h, [356](#)

/home/mo/mark3-source/embedded/stage/src/kprofile.-
cpp, [358](#)

/home/mo/mark3-source/embedded/stage/src/kprofile.-
h, [359](#), [360](#)

/home/mo/mark3-source/embedded/stage/src/ksemaphore.-
cpp, [361](#)

- /home/mo/mark3-source/embedded/stage/src/ksemaphore.h, 365
- /home/mo/mark3-source/embedded/stage/src/ll.cpp, 366
- /home/mo/mark3-source/embedded/stage/src/ll.h, 368, 369
- /home/mo/mark3-source/embedded/stage/src/manual.h, 370, 371
- /home/mo/mark3-source/embedded/stage/src/mark3cfg.h, 371, 374
- /home/mo/mark3-source/embedded/stage/src/memutil.cpp, 375
- /home/mo/mark3-source/embedded/stage/src/memutil.h, 381
- /home/mo/mark3-source/embedded/stage/src/message.cpp, 382
- /home/mo/mark3-source/embedded/stage/src/message.h, 384, 385
- /home/mo/mark3-source/embedded/stage/src/mutex.cpp, 387
- /home/mo/mark3-source/embedded/stage/src/mutex.h, 391, 392
- /home/mo/mark3-source/embedded/stage/src/nlfs.cpp, 393
- /home/mo/mark3-source/embedded/stage/src/nlfs.h, 405, 407
- /home/mo/mark3-source/embedded/stage/src/nlfs_config.h, 410
- /home/mo/mark3-source/embedded/stage/src/nlfs_file.cpp, 411
- /home/mo/mark3-source/embedded/stage/src/nlfs_file.h, 414, 415
- /home/mo/mark3-source/embedded/stage/src/nlfs_ram.cpp, 416
- /home/mo/mark3-source/embedded/stage/src/nlfs_ram.h, 418
- /home/mo/mark3-source/embedded/stage/src/panic_codes.h, 418, 419
- /home/mo/mark3-source/embedded/stage/src/profile.cpp, 419, 420
- /home/mo/mark3-source/embedded/stage/src/profile.h, 421, 422
- /home/mo/mark3-source/embedded/stage/src/quantum.cpp, 423
- /home/mo/mark3-source/embedded/stage/src/quantum.h, 425
- /home/mo/mark3-source/embedded/stage/src/scheduler.cpp, 426
- /home/mo/mark3-source/embedded/stage/src/scheduler.h, 427, 428
- /home/mo/mark3-source/embedded/stage/src/screen.cpp, 429
- /home/mo/mark3-source/embedded/stage/src/screen.h, 430
- /home/mo/mark3-source/embedded/stage/src/shell_support.cpp, 432
- /home/mo/mark3-source/embedded/stage/src/shell_support.h, 434, 435
- /home/mo/mark3-source/embedded/stage/src/slip.cpp, 436
- /home/mo/mark3-source/embedded/stage/src/slip.h, 439, 440
- /home/mo/mark3-source/embedded/stage/src/slip_mux.cpp, 441, 442
- /home/mo/mark3-source/embedded/stage/src/slip_mux.h, 443, 444
- /home/mo/mark3-source/embedded/stage/src/slipterm.cpp, 444, 445
- /home/mo/mark3-source/embedded/stage/src/slipterm.h, 445, 446
- /home/mo/mark3-source/embedded/stage/src/system_heap.cpp, 447
- /home/mo/mark3-source/embedded/stage/src/system_heap.h, 449, 450
- /home/mo/mark3-source/embedded/stage/src/system_heap_config.h, 453, 454
- /home/mo/mark3-source/embedded/stage/src/thread.cpp, 454, 455
- /home/mo/mark3-source/embedded/stage/src/thread.h, 459, 460
- /home/mo/mark3-source/embedded/stage/src/threadlist.cpp, 462
- /home/mo/mark3-source/embedded/stage/src/threadlist.h, 464
- /home/mo/mark3-source/embedded/stage/src/threadport.cpp, 465
- /home/mo/mark3-source/embedded/stage/src/threadport.h, 467, 468
- /home/mo/mark3-source/embedded/stage/src/timerlist.cpp, 470
- /home/mo/mark3-source/embedded/stage/src/timerlist.h, 474, 475
- /home/mo/mark3-source/embedded/stage/src/tracebuffer.cpp, 477, 478
- /home/mo/mark3-source/embedded/stage/src/tracebuffer.h, 478, 479
- /home/mo/mark3-source/embedded/stage/src/transaction.cpp, 479
- /home/mo/mark3-source/embedded/stage/src/transaction.h, 480, 481
- /home/mo/mark3-source/embedded/stage/src/unit_test.cpp, 482
- /home/mo/mark3-source/embedded/stage/src/unit_test.h, 483
- /home/mo/mark3-source/embedded/stage/src/writebuf16.cpp, 485
- /home/mo/mark3-source/embedded/stage/src/writebuf16.h, 487
- Activate
 - ButtonControl, 71
 - CheckBoxControl, 73
 - GamePanelControl, 103
 - GroupBoxControl, 111
 - GuiControl, 114
 - LabelControl, 137
 - NotificationControl, 172

- PanelControl, 175
- ProgressControl, 179
- Screen, 186
- SlickButtonControl, 195
- SlickGroupBoxControl, 197
- SlickProgressControl, 199
- StubControl, 208
- Add
 - Atomic, 63
 - CircularLinkList, 75
 - DoubleLinkList, 86
 - DriverList, 97
 - LinkList, 138
 - Scheduler, 182
 - ThreadList, 219
 - TimerList, 228
 - TimerScheduler, 230
- AddControl
 - GuiWindow, 124
- AddPlugin
 - DCPU, 78
- AddThread
 - Quantum, 181
- AddWindow
 - GuiEventSurface, 120
- AdvanceByte
 - BitStreamer, 65
- Alloc
 - BlockHeap, 67
 - FixedHeap, 101
 - SystemHeap, 209
- Append_Block_To_Node
 - NLFS, 154
- Atomic, 63
 - Add, 63
 - Set, 64
 - Sub, 64
 - TestAndSet, 64
- aucBox
 - control_checkbox.cpp, 253
- aucCheck
 - control_checkbox.cpp, 254
- BitStreamer, 65
 - AdvanceByte, 65
 - Init, 65
 - ReadBits, 66
- Bitmap
 - GraphicsDriver, 107
- Block
 - BlockingObject, 68
- BlockHeap, 66
 - Alloc, 67
 - Create, 67
 - Free, 67
 - IsFree, 67
- BlockingObject, 68
 - Block, 68
 - LockAndQueue, 69
- UnBlock, 69
- UnLock, 69
- ButtonControl, 70
 - Activate, 71
 - Draw, 72
 - Init, 72
 - ProcessEvent, 72
- CS_ENTER
 - threadport.h, 468
- CS_EXIT
 - threadport.h, 468
- CheckBoxControl, 72
 - Activate, 73
 - Draw, 73
 - Init, 73
 - ProcessEvent, 74
- CheckForOption
 - ShellSupport, 193
- Checksum16
 - MemUtil, 142
- Checksum8
 - MemUtil, 142
- Circle
 - GraphicsDriver, 107
- CircularLinkList, 74
 - Add, 75
 - Remove, 75
- Claim
 - Mutex, 150
- ClaimTransaction
 - Mutex, 150
- Cleanup_Node_Links
 - NLFS, 154
- Clear
 - EventFlag, 98
- ClearState
 - GuiControl, 114
- ClearTransaction
 - EventFlag, 99
- Close
 - DevNull, 84
 - Driver, 94
 - NLFS_File, 164
- CommandLine_t, 75
- CompareMemory
 - MemUtil, 143
- CompareStrings
 - MemUtil, 143
- Complete
 - UnitTest, 237
- ComputeCurrentTicks
 - ProfileTimer, 177
- ContextSwitchSWI
 - Thread, 213
- Control
 - DevNull, 84
 - Driver, 94
- control_checkbox.cpp

- aucBox, [253](#)
 - aucCheck, [254](#)
- CopyEvent
 - GuiEventSurface, [121](#)
- CopyMemory
 - MemUtil, [143](#)
- CopyString
 - MemUtil, [143](#)
- Create
 - BlockHeap, [67](#)
 - FixedHeap, [102](#)
- Create_Dir
 - NLFS, [154](#)
- Create_File
 - NLFS, [154](#)
- Create_File_i
 - NLFS, [156](#)
- CycleFocus
 - GuiWindow, [124](#)
- DCPU, [76](#)
 - AddPlugin, [78](#)
 - GetOperand, [78](#)
 - GetRegisters, [78](#)
 - HWN, [78](#)
 - IAQ, [78](#)
 - Init, [78](#)
 - m_clPluginList, [80](#)
 - RFI, [80](#)
 - SendInterrupt, [80](#)
- DCPU_OpBasic
 - dcpu.h, [287](#)
- DCPU_OpExtended
 - dcpu.h, [288](#)
- DCPU_Registers, [80](#)
- DCPUPlugin, [81](#)
 - Enumerate, [82](#)
 - GetDeviceNumber, [82](#)
 - Init, [82](#)
 - Interrupt, [83](#)
- DI
 - KernelSWI, [133](#)
- dcpu.h
 - OP_18, [288](#)
 - OP_19, [288](#)
 - OP_1C, [288](#)
 - OP_1D, [288](#)
 - OP_ADD, [287](#)
 - OP_ADX, [288](#)
 - OP_AND, [287](#)
 - OP_ASR, [288](#)
 - OP_BOR, [287](#)
 - OP_DIV, [287](#)
 - OP_DVI, [287](#)
 - OP_EX_13, [289](#)
 - OP_EX_14, [289](#)
 - OP_EX_15, [289](#)
 - OP_EX_16, [289](#)
 - OP_EX_17, [289](#)
 - OP_EX_18, [289](#)
 - OP_EX_19, [289](#)
 - OP_EX_1A, [289](#)
 - OP_EX_1B, [289](#)
 - OP_EX_1C, [289](#)
 - OP_EX_1D, [289](#)
 - OP_EX_1E, [289](#)
 - OP_EX_1F, [289](#)
 - OP_EX_2, [288](#)
 - OP_EX_3, [288](#)
 - OP_EX_4, [288](#)
 - OP_EX_5, [288](#)
 - OP_EX_6, [288](#)
 - OP_EX_7, [288](#)
 - OP_EX_D, [288](#)
 - OP_EX_E, [288](#)
 - OP_EX_F, [288](#)
 - OP_EX_HWI, [289](#)
 - OP_EX_HWN, [288](#)
 - OP_EX_HWQ, [288](#)
 - OP_EX_IAG, [288](#)
 - OP_EX_IAQ, [288](#)
 - OP_EX_IAS, [288](#)
 - OP_EX_INT, [288](#)
 - OP_EX_JSR, [288](#)
 - OP_EX_RFI, [288](#)
 - OP_IFA, [288](#)
 - OP_IFB, [288](#)
 - OP_IFC, [288](#)
 - OP_IFE, [288](#)
 - OP_IFG, [288](#)
 - OP_IFL, [288](#)
 - OP_IFN, [288](#)
 - OP_IFU, [288](#)
 - OP_MDI, [287](#)
 - OP_MLI, [287](#)
 - OP_MOD, [287](#)
 - OP_MUL, [287](#)
 - OP_NON_BASIC, [287](#)
 - OP_SBX, [288](#)
 - OP_SET, [287](#)
 - OP_SHL, [288](#)
 - OP_SHR, [288](#)
 - OP_STD, [288](#)
 - OP_STI, [288](#)
 - OP_SUB, [287](#)
 - OP_XOR, [288](#)
- dcpu.h
 - DCPU_OpBasic, [287](#)
 - DCPU_OpExtended, [288](#)
- Deactivate
 - Screen, [186](#)
- DecimalToHex
 - MemUtil, [144](#)
- DecimalToString
 - MemUtil, [144](#)
- DecodeByte
 - Slip, [201](#)

- Delete_File
 - NLFS, [156](#)
- Delete_Folder
 - NLFS, [156](#)
- Dequeue
 - TransactionQueue, [234](#)
- DevNull, [83](#)
 - Close, [84](#)
 - Control, [84](#)
 - Open, [84](#)
 - Read, [84](#)
 - Write, [85](#)
- DoubleLinkedList, [85](#)
 - Add, [86](#)
 - Remove, [86](#)
- Draw
 - ButtonControl, [72](#)
 - CheckBoxControl, [73](#)
 - GamePanelControl, [104](#)
 - GroupBoxControl, [111](#)
 - GuiControl, [114](#)
 - LabelControl, [137](#)
 - NotificationControl, [172](#)
 - PanelControl, [175](#)
 - ProgressControl, [179](#)
 - SlickButtonControl, [195](#)
 - SlickGroupBoxControl, [198](#)
 - SlickProgressControl, [199](#)
 - StubControl, [208](#)
- DrawBitmap_t, [86](#)
- DrawCircle_t, [87](#)
- DrawEllipse_t, [88](#)
- DrawLine_t, [88](#)
- DrawMove_t, [89](#)
- DrawPixel
 - GraphicsDriver, [107](#)
- DrawPoint_t, [89](#)
- DrawPoly_t, [90](#)
- DrawRectangle_t, [90](#)
- DrawStamp_t, [91](#)
- DrawText_t, [92](#)
- DrawVector_t, [92](#)
- DrawWindow_t, [93](#)
- Driver, [93](#)
 - Close, [94](#)
 - Control, [94](#)
 - GetPath, [95](#)
 - Open, [95](#)
 - Read, [95](#)
 - SetName, [95](#)
 - Write, [96](#)
- DriverList, [96](#)
 - Add, [97](#)
 - FindByPath, [97](#)
 - Init, [97](#)
 - Remove, [97](#)
- EVENT_TYPE_COUNT
 - gui.h, [339](#)
- EVENT_TYPE_JOYSTICK
 - gui.h, [339](#)
- EVENT_TYPE_KEYBOARD
 - gui.h, [339](#)
- EVENT_TYPE_MOUSE
 - gui.h, [339](#)
- EVENT_TYPE_TIMER
 - gui.h, [339](#)
- EVENT_TYPE_TOUCH
 - gui.h, [339](#)
- Ellipse
 - GraphicsDriver, [108](#)
- EncodeByte
 - Slip, [201](#)
- Enqueue
 - TransactionQueue, [234](#)
- Enumerate
 - DCPUPlugin, [82](#)
- EventFlag, [97](#)
 - Clear, [98](#)
 - ClearTransaction, [99](#)
 - GetMask, [99](#)
 - ProcessQueue, [99](#)
 - Set, [99](#)
 - SetTransaction, [99](#)
 - TimeoutTransaction, [100](#)
 - Wait, [100](#)
 - WaitTransaction, [100](#)
- Exit
 - Thread, [213](#)
- File_Names_Match
 - NLFS, [156](#)
- Find_File
 - NLFS, [157](#)
- Find_Last_Slash
 - NLFS, [157](#)
- Find_Parent_Dir
 - NLFS, [157](#)
- FindByPath
 - DriverList, [97](#)
- Finish
 - TransactionQueue, [235](#)
- FixedHeap, [101](#)
 - Alloc, [101](#)
 - Create, [102](#)
 - Free, [102](#)
- Font_t, [102](#)
- Format
 - NLFS, [157](#)
- fp_internal_command
 - shell_support.h, [435](#)
- Free
 - BlockHeap, [67](#)
 - FixedHeap, [102](#)
 - SystemHeap, [209](#)
- GUI_EVENT_CANCEL
 - gui.h, [339](#)

- GUI_EVENT_CONSUMED
 - gui.h, 339
- GUI_EVENT_OK
 - gui.h, 339
- GUI_EVENT_RETRY
 - gui.h, 339
- GamePanelControl, 103
 - Activate, 103
 - Draw, 104
 - Init, 104
 - ProcessEvent, 104
- GetAverage
 - ProfileTimer, 178
- GetBlockSize
 - NLFS, 158
- GetCode
 - Message, 146
 - Transaction, 232
- GetControlIndex
 - GuiControl, 114
- GetControlOffset
 - GuiControl, 115
- GetCount
 - MessageQueue, 147
 - Semaphore, 189
- GetCurPriority
 - Thread, 213
- GetCurrent
 - ProfileTimer, 178
 - Thread, 213
- GetCurrentThread
 - Scheduler, 183
- GetData
 - Message, 146
 - Transaction, 232
- GetDeviceNumber
 - DCPUPlugin, 82
- GetDriver
 - GuiWindow, 124
 - Slip, 201
 - SlipMux, 204
- GetEventFlagMask
 - Thread, 213
- GetEventFlagMode
 - Thread, 213
- GetFailed
 - UnitTest, 237
- GetFirstChild
 - NLFS, 158
- GetHead
 - LinkList, 139
- GetHeight
 - GuiControl, 115
 - GuiWindow, 124
- GetID
 - Thread, 214
- GetLeft
 - GuiControl, 115
- GuiWindow, 124
- GetMask
 - EventFlag, 99
- GetMaxZOrder
 - GuiWindow, 125
- GetName
 - Thread, 214
 - UnitTest, 237
- GetNext
 - LinkListNode, 141
- GetNextPeer
 - NLFS, 158
- GetNextThread
 - Scheduler, 183
- GetNumBlocks
 - NLFS, 158
- GetNumBlocksFree
 - NLFS, 159
- GetNumFiles
 - NLFS, 159
- GetNumFilesFree
 - NLFS, 159
- GetOperand
 - DCPU, 78
- GetOvertime
 - KernelTimer, 134
- GetOwner
 - Thread, 214
- GetParentControl
 - GuiControl, 115
- GetParentWindow
 - GuiControl, 115
- GetPassed
 - UnitTest, 237
- GetPath
 - Driver, 95
- GetPrev
 - LinkListNode, 141
- GetPriority
 - Thread, 214
- GetQuantum
 - Thread, 214
- GetQueue
 - SlipMux, 204
- GetRegisters
 - DCPU, 78
- GetResult
 - UnitTest, 237
- GetSlip
 - SlipMux, 204
- GetStackSlack
 - Thread, 214
- GetStat
 - NLFS, 159
- GetStopList
 - Scheduler, 183
- GetTail
 - LinkList, 139

- GetThreadList
 - Scheduler, 183
- GetTop
 - GuiControl, 116
 - GuiWindow, 125
- GetTotal
 - UnitTest, 237
- GetWidth
 - GuiControl, 116
 - GuiWindow, 125
- GetZOrder
 - GuiControl, 116
- GlobalMessagePool, 104
 - Pop, 105
 - Push, 105
- GlobalQueueInit
 - TransactionQueue, 235
- Glyph_t, 105
- GraphicsDriver, 106
 - Bitmap, 107
 - Circle, 107
 - DrawPixel, 107
 - Ellipse, 108
 - Line, 108
 - Move, 108
 - Point, 108
 - ReadPixel, 108
 - Rectangle, 108
 - SetWindow, 109
 - Stamp, 109
 - Text, 109
 - TextFX, 109
 - TriangleFill, 109
 - TriangleWire, 109
- GroupBoxControl, 110
 - Activate, 111
 - Draw, 111
 - Init, 111
 - ProcessEvent, 111
- gui.h
 - EVENT_TYPE_COUNT, 339
 - EVENT_TYPE_JOYSTICK, 339
 - EVENT_TYPE_KEYBOARD, 339
 - EVENT_TYPE_MOUSE, 339
 - EVENT_TYPE_TIMER, 339
 - EVENT_TYPE_TOUCH, 339
 - GUI_EVENT_CANCEL, 339
 - GUI_EVENT_CONSUMED, 339
 - GUI_EVENT_OK, 339
 - GUI_EVENT_RETRY, 339
- gui.h
 - GuiEventType_t, 339
 - GuiReturn_t, 339
- GuiControl, 111
 - Activate, 114
 - ClearStale, 114
 - Draw, 114
 - GetControlIndex, 114
 - GetControlOffset, 115
 - GetHeight, 115
 - GetLeft, 115
 - GetParentControl, 115
 - GetParentWindow, 115
 - GetTop, 116
 - GetWidth, 116
 - GetZOrder, 116
 - Init, 116
 - IsInFocus, 116
 - IsStale, 116
 - m_ucControlIndex, 118
 - m_ucZOrder, 118
 - ProcessEvent, 117
 - SetControlIndex, 117
 - SetHeight, 117
 - SetLeft, 117
 - SetParentControl, 117
 - SetParentWindow, 118
 - SetTop, 118
 - SetWidth, 118
 - SetZOrder, 118
- GuiEvent_t, 119
- GuiEventSurface, 119
 - AddWindow, 120
 - CopyEvent, 121
 - Init, 121
 - InvalidateRegion, 121
 - ProcessEvent, 121
 - RemoveWindow, 121
 - SendEvent, 121
- GuiEventType_t
 - gui.h, 339
- GuiReturn_t
 - gui.h, 339
- GuiWindow, 122
 - AddControl, 124
 - CycleFocus, 124
 - GetDriver, 124
 - GetHeight, 124
 - GetLeft, 124
 - GetMaxZOrder, 125
 - GetTop, 125
 - GetWidth, 125
 - Init, 125
 - InvalidateRegion, 125
 - IsInFocus, 125
 - m_pclDriver, 128
 - ProcessEvent, 127
 - Redraw, 127
 - RemoveControl, 127
 - SetDriver, 127
 - SetFocus, 127
 - SetHeight, 128
 - SetLeft, 128
 - SetTop, 128
 - SetWidth, 128
- HEAP_BLOCK_SIZE_1

- system_heap_config.h, [454](#)
- HEAP_RAW_SIZE
 - system_heap.h, [450](#)
- HEAP_RAW_SIZE_1
 - system_heap.h, [450](#)
- HWN
 - DCPU, [78](#)
- HeapConfig, [129](#)
- HighestWaiter
 - ThreadList, [219](#)
- IAQ
 - DCPU, [78](#)
- InheritPriority
 - Thread, [215](#)
- Init
 - BitStreamer, [65](#)
 - ButtonControl, [72](#)
 - CheckBoxControl, [73](#)
 - DCPU, [78](#)
 - DCPUPlugin, [82](#)
 - DriverList, [97](#)
 - GamePanelControl, [104](#)
 - GroupBoxControl, [111](#)
 - GuiControl, [116](#)
 - GuiEventSurface, [121](#)
 - GuiWindow, [125](#)
 - Kernel, [131](#)
 - LabelControl, [137](#)
 - NotificationControl, [172](#)
 - PanelControl, [175](#)
 - Profiler, [176](#)
 - ProfileTimer, [178](#)
 - ProgressControl, [180](#)
 - Semaphore, [189](#)
 - SlickButtonControl, [195](#)
 - SlickGroupBoxControl, [198](#)
 - SlickProgressControl, [199](#)
 - SlipMux, [204](#)
 - SlipTerm, [206](#)
 - StubControl, [208](#)
 - Thread, [215](#)
 - TimerList, [228](#)
 - TimerScheduler, [230](#)
- InitStack
 - ThreadPort, [221](#)
- InstallHandler
 - SlipMux, [204](#)
- Interrupt
 - DCPUPlugin, [83](#)
- InvalidateRegion
 - GuiEventSurface, [121](#)
 - GuiWindow, [125](#)
- IsEnabled
 - Scheduler, [183](#)
- IsFree
 - BlockHeap, [67](#)
- IsInFocus
 - GuiControl, [116](#)
- GuiWindow, [125](#)
- IsPanic
 - Kernel, [131](#)
- IsStale
 - GuiControl, [116](#)
- IsStarted
 - Kernel, [131](#)
- JoystickEvent_t, [129](#)
- KERNEL_USE_ATOMIC
 - mark3cfg.h, [372](#)
- KERNEL_USE_MESSAGE
 - mark3cfg.h, [373](#)
- KERNEL_USE_PROFILER
 - mark3cfg.h, [373](#)
- KERNEL_USE_QUANTUM
 - mark3cfg.h, [373](#)
- KERNEL_USE_TIMERS
 - mark3cfg.h, [373](#)
- Kernel, [130](#)
 - Init, [131](#)
 - IsPanic, [131](#)
 - IsStarted, [131](#)
 - Panic, [131](#)
 - SetPanic, [132](#)
 - Start, [132](#)
- KernelSWI, [132](#)
 - DI, [133](#)
 - RI, [133](#)
- KernelTimer, [133](#)
 - GetOvertime, [134](#)
 - RI, [134](#)
 - Read, [134](#)
 - SetExpiry, [134](#)
 - SubtractExpiry, [135](#)
 - TimeToExpiry, [135](#)
- KeyEvent_t, [135](#)
- LabelControl, [136](#)
 - Activate, [137](#)
 - Draw, [137](#)
 - Init, [137](#)
 - ProcessEvent, [137](#)
- Line
 - GraphicsDriver, [108](#)
- LinkedList, [138](#)
 - Add, [138](#)
 - GetHead, [139](#)
 - GetTail, [139](#)
 - Remove, [139](#)
- LinkedListNode, [139](#)
 - GetNext, [141](#)
 - GetPrev, [141](#)
- LoadTile
 - Tile_8x8, [222](#)
- LockAndQueue
 - BlockingObject, [69](#)

- m_clPluginList
 - DCPU, [80](#)
- m_pclDriver
 - GuiWindow, [128](#)
- m_ucControlIndex
 - GuiControl, [118](#)
- m_ucVerbosity
 - SlipTerm, [207](#)
- m_ucZOrder
 - GuiControl, [118](#)
- mark3cfg.h
 - KERNEL_USE_ATOMIC, [372](#)
 - KERNEL_USE_MESSAGE, [373](#)
 - KERNEL_USE_PROFILER, [373](#)
 - KERNEL_USE_QUANTUM, [373](#)
 - KERNEL_USE_TIMERS, [373](#)
 - SAFE_UNLINK, [374](#)
- MemUtil, [141](#)
 - Checksum16, [142](#)
 - Checksum8, [142](#)
 - CompareMemory, [143](#)
 - CompareStrings, [143](#)
 - CopyMemory, [143](#)
 - CopyString, [143](#)
 - DecimalToHex, [144](#)
 - DecimalToString, [144](#)
 - SetMemory, [144](#)
 - StringLength, [144](#)
 - StringSearch, [144](#)
 - Tokenize, [145](#)
- Message, [145](#)
 - GetCode, [146](#)
 - GetData, [146](#)
 - SetCode, [146](#)
 - SetData, [146](#)
- MessageQueue, [147](#)
 - GetCount, [147](#)
 - Receive, [147](#), [148](#)
 - Send, [148](#)
- MessageReceive
 - SlipMux, [205](#)
- Mount
 - NLFS, [159](#)
- MouseEvent_t, [148](#)
- Move
 - GraphicsDriver, [108](#)
- Mutex, [149](#)
 - Claim, [150](#)
 - ClaimTransaction, [150](#)
 - Release, [151](#)
 - ReleaseTransaction, [151](#)
 - Timeout, [151](#)
 - TimeoutTransaction, [151](#)
- NLFS_FILE_APPEND
 - nlfs_file.h, [415](#)
- NLFS_FILE_CREATE
 - nlfs_file.h, [415](#)
- NLFS_FILE_READ
 - nlfs_file.h, [415](#)
- NLFS_FILE_TRUNCATE
 - nlfs_file.h, [415](#)
- NLFS_FILE_WRITE
 - nlfs_file.h, [415](#)
- NLFS_NODE_DIR
 - nlfs.h, [407](#)
- NLFS_NODE_FILE
 - nlfs.h, [407](#)
- NLFS_NODE_FREE
 - nlfs.h, [407](#)
- NLFS_NODE_ROOT
 - nlfs.h, [407](#)
- NLFS, [152](#)
 - Append_Block_To_Node, [154](#)
 - Cleanup_Node_Links, [154](#)
 - Create_Dir, [154](#)
 - Create_File, [154](#)
 - Create_File_i, [156](#)
 - Delete_File, [156](#)
 - Delete_Folder, [156](#)
 - File_Names_Match, [156](#)
 - Find_File, [157](#)
 - Find_Last_Slash, [157](#)
 - Find_Parent_Dir, [157](#)
 - Format, [157](#)
 - GetBlockSize, [158](#)
 - GetFirstChild, [158](#)
 - GetNextPeer, [158](#)
 - GetNumBlocks, [158](#)
 - GetNumBlocksFree, [159](#)
 - GetNumFiles, [159](#)
 - GetNumFilesFree, [159](#)
 - GetStat, [159](#)
 - Mount, [159](#)
 - Pop_Free_Block, [160](#)
 - Pop_Free_Node, [160](#)
 - Print_Dir_Details, [160](#)
 - Print_File_Details, [160](#)
 - Print_Free_Details, [160](#)
 - Print_Node_Details, [160](#)
 - Push_Free_Block, [161](#)
 - Push_Free_Node, [161](#)
 - Read_Block, [161](#)
 - Read_Block_Header, [161](#)
 - Read_Node, [161](#)
 - RootSync, [162](#)
 - Set_Node_Name, [162](#)
 - Write_Block, [162](#)
 - Write_Block_Header, [162](#)
 - Write_Node, [162](#)
- NLFS_Block_t, [163](#)
- NLFS_File, [163](#)
 - Close, [164](#)
 - Open, [164](#)
 - Read, [165](#)
 - Seek, [165](#)
 - Write, [165](#)

NLFS_File_Mode
 nlfs_file.h, 415
 NLFS_File_Node_t, 166
 NLFS_File_Stat_t, 166
 NLFS_Host_t, 167
 NLFS_Node_t, 167
 NLFS_RAM, 168
 Read_Block, 169
 Read_Block_Header, 169
 Read_Node, 169
 Write_Block, 170
 Write_Block_Header, 170
 Write_Node, 170
 NLFS_Root_Node_t, 171
 NLFS_Type_t
 nlfs.h, 407
 nlfs.h
 NLFS_NODE_DIR, 407
 NLFS_NODE_FILE, 407
 NLFS_NODE_FREE, 407
 NLFS_NODE_ROOT, 407
 nlfs.h
 NLFS_Type_t, 407
 nlfs_file.h
 NLFS_FILE_APPEND, 415
 NLFS_FILE_CREATE, 415
 NLFS_FILE_READ, 415
 NLFS_FILE_TRUNCATE, 415
 NLFS_FILE_WRITE, 415
 nlfs_file.h
 NLFS_File_Mode, 415
 NotificationControl, 171
 Activate, 172
 Draw, 172
 Init, 172
 ProcessEvent, 173

 OP_18
 dcpu.h, 288
 OP_19
 dcpu.h, 288
 OP_1C
 dcpu.h, 288
 OP_1D
 dcpu.h, 288
 OP_ADD
 dcpu.h, 287
 OP_ADX
 dcpu.h, 288
 OP_AND
 dcpu.h, 287
 OP_ASR
 dcpu.h, 288
 OP_BOR
 dcpu.h, 287
 OP_DIV
 dcpu.h, 287
 OP_DVI
 dcpu.h, 287

 OP_EX_13
 dcpu.h, 289
 OP_EX_14
 dcpu.h, 289
 OP_EX_15
 dcpu.h, 289
 OP_EX_16
 dcpu.h, 289
 OP_EX_17
 dcpu.h, 289
 OP_EX_18
 dcpu.h, 289
 OP_EX_19
 dcpu.h, 289
 OP_EX_1A
 dcpu.h, 289
 OP_EX_1B
 dcpu.h, 289
 OP_EX_1C
 dcpu.h, 289
 OP_EX_1D
 dcpu.h, 289
 OP_EX_1E
 dcpu.h, 289
 OP_EX_1F
 dcpu.h, 289
 OP_EX_2
 dcpu.h, 288
 OP_EX_3
 dcpu.h, 288
 OP_EX_4
 dcpu.h, 288
 OP_EX_5
 dcpu.h, 288
 OP_EX_6
 dcpu.h, 288
 OP_EX_7
 dcpu.h, 288
 OP_EX_D
 dcpu.h, 288
 OP_EX_E
 dcpu.h, 288
 OP_EX_F
 dcpu.h, 288
 OP_EX_HWI
 dcpu.h, 289
 OP_EX_HWN
 dcpu.h, 288
 OP_EX_HWQ
 dcpu.h, 288
 OP_EX_IAG
 dcpu.h, 288
 OP_EX_IAQ
 dcpu.h, 288
 OP_EX_IAS
 dcpu.h, 288
 OP_EX_INT
 dcpu.h, 288

- OP_EX_JSR
 - dcpu.h, [288](#)
- OP_EX_RFI
 - dcpu.h, [288](#)
- OP_IFA
 - dcpu.h, [288](#)
- OP_IFB
 - dcpu.h, [288](#)
- OP_IFC
 - dcpu.h, [288](#)
- OP_IFE
 - dcpu.h, [288](#)
- OP_IFG
 - dcpu.h, [288](#)
- OP_IFL
 - dcpu.h, [288](#)
- OP_IFN
 - dcpu.h, [288](#)
- OP_IFU
 - dcpu.h, [288](#)
- OP_MDI
 - dcpu.h, [287](#)
- OP_MLI
 - dcpu.h, [287](#)
- OP_MOD
 - dcpu.h, [287](#)
- OP_MUL
 - dcpu.h, [287](#)
- OP_NON_BASIC
 - dcpu.h, [287](#)
- OP_SBX
 - dcpu.h, [288](#)
- OP_SET
 - dcpu.h, [287](#)
- OP_SHL
 - dcpu.h, [288](#)
- OP_SHR
 - dcpu.h, [288](#)
- OP_STD
 - dcpu.h, [288](#)
- OP_STI
 - dcpu.h, [288](#)
- OP_SUB
 - dcpu.h, [287](#)
- OP_XOR
 - dcpu.h, [288](#)
- Open
 - DevNull, [84](#)
 - Driver, [95](#)
 - NLFS_File, [164](#)
- Option_t, [173](#)
- PanelControl, [174](#)
 - Activate, [175](#)
 - Draw, [175](#)
 - Init, [175](#)
 - ProcessEvent, [175](#)
- Panic
 - Kernel, [131](#)
- Pend
 - Semaphore, [189](#)
- PendTransaction
 - Semaphore, [189](#)
- Point
 - GraphicsDriver, [108](#)
- Pop
 - GlobalMessagePool, [105](#)
- Pop_Free_Block
 - NLFS, [160](#)
- Pop_Free_Node
 - NLFS, [160](#)
- Post
 - Semaphore, [191](#)
- PostTransaction
 - Semaphore, [191](#)
- Print_Dir_Details
 - NLFS, [160](#)
- Print_File_Details
 - NLFS, [160](#)
- Print_Free_Details
 - NLFS, [160](#)
- Print_Node_Details
 - NLFS, [160](#)
- PrintLn
 - SlipTerm, [206](#)
- Process
 - TimerList, [228](#)
 - TimerScheduler, [230](#)
- ProcessEvent
 - ButtonControl, [72](#)
 - CheckBoxControl, [74](#)
 - GamePanelControl, [104](#)
 - GroupBoxControl, [111](#)
 - GuiControl, [117](#)
 - GuiEventSurface, [121](#)
 - GuiWindow, [127](#)
 - LabelControl, [137](#)
 - NotificationControl, [173](#)
 - PanelControl, [175](#)
 - ProgressControl, [180](#)
 - SlickButtonControl, [195](#)
 - SlickGroupBoxControl, [198](#)
 - SlickProgressControl, [199](#)
 - StubControl, [208](#)
- ProcessQueue
 - EventFlag, [99](#)
 - Semaphore, [191](#)
- ProfileTimer, [176](#)
 - ComputeCurrentTicks, [177](#)
 - GetAverage, [178](#)
 - GetCurrent, [178](#)
 - Init, [178](#)
 - Start, [178](#)
- Profiler, [176](#)
 - Init, [176](#)
- ProgressControl, [178](#)
 - Activate, [179](#)

- Draw, [179](#)
- Init, [180](#)
- ProcessEvent, [180](#)
- Push
 - GlobalMessagePool, [105](#)
- Push_Free_Block
 - NLFS, [161](#)
- Push_Free_Node
 - NLFS, [161](#)
- Quantum, [180](#)
 - AddThread, [181](#)
 - RemoveThread, [181](#)
 - SetTimer, [181](#)
 - UpdateTimer, [181](#)
- RFI
 - DCPU, [80](#)
- RI
 - KernelSWI, [133](#)
 - KernelTimer, [134](#)
- Read
 - DevNull, [84](#)
 - Driver, [95](#)
 - KernelTimer, [134](#)
 - NLFS_File, [165](#)
- Read_Block
 - NLFS, [161](#)
 - NLFS_RAM, [169](#)
- Read_Block_Header
 - NLFS, [161](#)
 - NLFS_RAM, [169](#)
- Read_Node
 - NLFS, [161](#)
 - NLFS_RAM, [169](#)
- ReadBits
 - BitStreamer, [66](#)
- ReadData
 - Slip, [201](#)
- ReadPixel
 - GraphicsDriver, [108](#)
- Receive
 - MessageQueue, [147](#), [148](#)
- Rectangle
 - GraphicsDriver, [108](#)
- Redraw
 - GuiWindow, [127](#)
- Release
 - Mutex, [151](#)
- ReleaseTransaction
 - Mutex, [151](#)
- Remove
 - CircularLinkList, [75](#)
 - DoubleLinkList, [86](#)
 - DriverList, [97](#)
 - LinkList, [139](#)
 - Scheduler, [183](#)
 - ThreadList, [219](#)
 - TimerList, [229](#)
 - TimerScheduler, [230](#)
- RemoveControl
 - GuiWindow, [127](#)
- RemoveThread
 - Quantum, [181](#)
- RemoveWindow
 - GuiEventSurface, [121](#)
- Render
 - Tile_8x8, [222](#)
- RootSync
 - NLFS, [162](#)
- RunCommand
 - ShellSupport, [193](#)
- SLIP_CHANNEL_GRAPHICS
 - slip.h, [440](#)
- SLIP_CHANNEL_HID
 - slip.h, [440](#)
- SLIP_CHANNEL_NVM
 - slip.h, [440](#)
- SLIP_CHANNEL_RESET
 - slip.h, [440](#)
- SLIP_CHANNEL_TERMINAL
 - slip.h, [440](#)
- SLIP_CHANNEL_UNISCOPE
 - slip.h, [440](#)
- SAFE_UNLINK
 - mark3cfg.h, [374](#)
- Schedule
 - Scheduler, [185](#)
- Scheduler, [181](#)
 - Add, [182](#)
 - GetCurrentThread, [183](#)
 - GetNextThread, [183](#)
 - GetStopList, [183](#)
 - GetThreadList, [183](#)
 - IsEnabled, [183](#)
 - Remove, [183](#)
 - Schedule, [185](#)
 - SetScheduler, [185](#)
- Screen, [185](#)
 - Activate, [186](#)
 - Deactivate, [186](#)
- ScreenList, [186](#)
- ScreenManager, [187](#)
- Seek
 - NLFS_File, [165](#)
- Semaphore, [188](#)
 - GetCount, [189](#)
 - Init, [189](#)
 - Pend, [189](#)
 - PendTransaction, [189](#)
 - Post, [191](#)
 - PostTransaction, [191](#)
 - ProcessQueue, [191](#)
 - Timeout, [191](#)
 - TimeoutTransaction, [191](#)
- Send
 - MessageQueue, [148](#)

- SendEvent
 - GuiEventSurface, 121
- SendInterrupt
 - DCPU, 80
- Set
 - Atomic, 64
 - EventFlag, 99
 - Transaction, 233
- Set_Node_Name
 - NLFS, 162
- SetBuffers
 - WriteBuffer16, 240
- SetCallback
 - Timer, 225
 - WriteBuffer16, 241
- SetCode
 - Message, 146
- SetControlIndex
 - GuiControl, 117
- SetCurrent
 - Thread, 215
- SetData
 - Message, 146
 - Timer, 225
- SetDriver
 - GuiWindow, 127
 - Slip, 202
- SetEventFlagMask
 - Thread, 215
- SetEventFlagMode
 - Thread, 216
- SetExpiry
 - KernelTimer, 134
- SetFlagPointer
 - ThreadList, 219
- SetFlags
 - Timer, 225
- SetFocus
 - GuiWindow, 127
- SetHeight
 - GuiControl, 117
 - GuiWindow, 128
- SetID
 - Thread, 216
- SetIntervalMSeconds
 - Timer, 225
- SetIntervalSeconds
 - Timer, 225
- SetIntervalTicks
 - Timer, 225
- SetIntervalUSeconds
 - Timer, 226
- SetLeft
 - GuiControl, 117
 - GuiWindow, 128
- SetMemory
 - MemUtil, 144
- SetName
 - Driver, 95
 - Thread, 216
 - UnitTest, 237
- SetOwner
 - Thread, 216
 - Timer, 226
- SetPanic
 - Kernel, 132
- SetParentControl
 - GuiControl, 117
- SetParentWindow
 - GuiControl, 118
- SetPriority
 - Thread, 216
 - ThreadList, 219
- SetPriorityBase
 - Thread, 217
- SetQuantum
 - Thread, 217
- SetQueue
 - SlipMux, 205
- SetScheduler
 - Scheduler, 185
- SetTimer
 - Quantum, 181
- SetTolerance
 - Timer, 226
- SetTop
 - GuiControl, 118
 - GuiWindow, 128
- SetTransaction
 - EventFlag, 99
- SetVerbosity
 - SlipTerm, 206
- SetWidth
 - GuiControl, 118
 - GuiWindow, 128
- SetWindow
 - GraphicsDriver, 109
- SetZOrder
 - GuiControl, 118
- shell_support.h
 - fp_internal_command, 435
- ShellCommand_t, 192
- ShellSupport, 192
 - CheckForOption, 193
 - RunCommand, 193
 - TokensToCommandLine, 193
 - UnescapeToken, 194
- Sleep
 - Thread, 217
- SlickButtonControl, 194
 - Activate, 195
 - Draw, 195
 - Init, 195
 - ProcessEvent, 195
- SlickGroupBoxControl, 196
 - Activate, 197

- Draw, [198](#)
 - Init, [198](#)
 - ProcessEvent, [198](#)
- SlickProgressControl, [198](#)
 - Activate, [199](#)
 - Draw, [199](#)
 - Init, [199](#)
 - ProcessEvent, [199](#)
- Slip, [200](#)
 - DecodeByte, [201](#)
 - EncodeByte, [201](#)
 - GetDriver, [201](#)
 - ReadData, [201](#)
 - SetDriver, [202](#)
 - WriteData, [202](#)
 - WriteVector, [202](#)
- slip.h
 - SLIP_CHANNEL_GRAPHICS, [440](#)
 - SLIP_CHANNEL_HID, [440](#)
 - SLIP_CHANNEL_NVM, [440](#)
 - SLIP_CHANNEL_RESET, [440](#)
 - SLIP_CHANNEL_TERMINAL, [440](#)
 - SLIP_CHANNEL_UNISCOPE, [440](#)
- slip.h
 - SlipChannel, [440](#)
- slip_mux.cpp
 - SlipMux_CallBack, [442](#)
- SlipChannel
 - slip.h, [440](#)
- SlipDataVector, [202](#)
- SlipMux, [203](#)
 - GetDriver, [204](#)
 - GetQueue, [204](#)
 - GetSlip, [204](#)
 - Init, [204](#)
 - InstallHandler, [204](#)
 - MessageReceive, [205](#)
 - SetQueue, [205](#)
- SlipMux_CallBack
 - slip_mux.cpp, [442](#)
- SlipTerm, [205](#)
 - Init, [206](#)
 - m_ucVerbosity, [207](#)
 - PrintLn, [206](#)
 - SetVerbosity, [206](#)
 - StrLen, [206](#)
- Stamp
 - GraphicsDriver, [109](#)
- Start
 - Kernel, [132](#)
 - ProfileTimer, [178](#)
 - Timer, [226](#)
- Stop
 - Thread, [217](#)
 - Timer, [227](#)
- StrLen
 - SlipTerm, [206](#)
- StringLength
 - MemUtil, [144](#)
- StringSearch
 - MemUtil, [144](#)
- StubControl, [207](#)
 - Activate, [208](#)
 - Draw, [208](#)
 - Init, [208](#)
 - ProcessEvent, [208](#)
- Sub
 - Atomic, [64](#)
- SubtractExpiry
 - KernelTimer, [135](#)
- system_heap.h
 - HEAP_RAW_SIZE, [450](#)
 - HEAP_RAW_SIZE_1, [450](#)
- SystemHeap, [208](#)
 - Alloc, [209](#)
 - Free, [209](#)
- TestAndSet
 - Atomic, [64](#)
- Text
 - GraphicsDriver, [109](#)
- TextFX
 - GraphicsDriver, [109](#)
- TextFX_t, [210](#)
- Thread, [210](#)
 - ContextSwitchSWI, [213](#)
 - Exit, [213](#)
 - GetCurPriority, [213](#)
 - GetCurrent, [213](#)
 - GetEventFlagMask, [213](#)
 - GetEventFlagMode, [213](#)
 - GetID, [214](#)
 - GetName, [214](#)
 - GetOwner, [214](#)
 - GetPriority, [214](#)
 - GetQuantum, [214](#)
 - GetStackSlack, [214](#)
 - InheritPriority, [215](#)
 - Init, [215](#)
 - SetCurrent, [215](#)
 - SetEventFlagMask, [215](#)
 - SetEventFlagMode, [216](#)
 - SetID, [216](#)
 - SetName, [216](#)
 - SetOwner, [216](#)
 - SetPriority, [216](#)
 - SetPriorityBase, [217](#)
 - SetQuantum, [217](#)
 - Sleep, [217](#)
 - Stop, [217](#)
 - USleep, [217](#)
 - Yield, [217](#)
- ThreadList, [218](#)
 - Add, [219](#)
 - HighestWaiter, [219](#)
 - Remove, [219](#)
 - SetFlagPointer, [219](#)

- SetPriority, [219](#)
- ThreadPort, [221](#)
 - InitStack, [221](#)
- threadport.h
 - CS_ENTER, [468](#)
 - CS_EXIT, [468](#)
- Tile_8x8, [222](#)
 - LoadTile, [222](#)
 - Render, [222](#)
- TileDef_t, [223](#)
- TimeToExpiry
 - KernelTimer, [135](#)
- Timeout
 - Mutex, [151](#)
 - Semaphore, [191](#)
- TimeoutTransaction
 - EventFlag, [100](#)
 - Mutex, [151](#)
 - Semaphore, [191](#)
- Timer, [223](#)
 - SetCallback, [225](#)
 - SetData, [225](#)
 - SetFlags, [225](#)
 - SetIntervalMSeconds, [225](#)
 - SetIntervalSeconds, [225](#)
 - SetIntervalTicks, [225](#)
 - SetIntervalUSeconds, [226](#)
 - SetOwner, [226](#)
 - SetTolerance, [226](#)
 - Start, [226](#)
 - Stop, [227](#)
- TimerEvent_t, [227](#)
- TimerList, [227](#)
 - Add, [228](#)
 - Init, [228](#)
 - Process, [228](#)
 - Remove, [229](#)
- TimerScheduler, [229](#)
 - Add, [230](#)
 - Init, [230](#)
 - Process, [230](#)
 - Remove, [230](#)
- Token_t, [230](#)
- Tokenize
 - MemUtil, [145](#)
- TokensToCommandLine
 - ShellSupport, [193](#)
- TouchEvent_t, [231](#)
- Transaction, [231](#)
 - GetCode, [232](#)
 - GetData, [232](#)
 - Set, [233](#)
- TransactionQueue, [233](#)
 - Dequeue, [234](#)
 - Enqueue, [234](#)
 - Finish, [235](#)
 - GlobalQueueInit, [235](#)
- TriangleFill
 - GraphicsDriver, [109](#)
- TriangleWire
 - GraphicsDriver, [109](#)
- USleep
 - Thread, [217](#)
- UnBlock
 - BlockingObject, [69](#)
- UnLock
 - BlockingObject, [69](#)
- UnescapeToken
 - ShellSupport, [194](#)
- UnitTest, [235](#)
 - Complete, [237](#)
 - GetFailed, [237](#)
 - GetName, [237](#)
 - GetPassed, [237](#)
 - GetResult, [237](#)
 - GetTotal, [237](#)
 - SetName, [237](#)
- UpdateTimer
 - Quantum, [181](#)
- Wait
 - EventFlag, [100](#)
- WaitTransaction
 - EventFlag, [100](#)
- Write
 - DevNull, [85](#)
 - Driver, [96](#)
 - NLFS_File, [165](#)
- Write_Block
 - NLFS, [162](#)
 - NLFS_RAM, [170](#)
- Write_Block_Header
 - NLFS, [162](#)
 - NLFS_RAM, [170](#)
- Write_Node
 - NLFS, [162](#)
 - NLFS_RAM, [170](#)
- WriteBuffer16, [239](#)
 - SetBuffers, [240](#)
 - SetCallback, [241](#)
 - WriteData, [241](#)
 - WriteVector, [241](#)
- WriteData
 - Slip, [202](#)
 - WriteBuffer16, [241](#)
- WriteVector
 - Slip, [202](#)
 - WriteBuffer16, [241](#)
- Yield
 - Thread, [217](#)