

Mark3 Realtime Kernel

Generated by Doxygen 1.8.1.2

Tue Feb 5 2013 20:29:38

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	Messages	21
6.6.1	Message Objects	22
6.6.2	Global Message Pool	22
6.6.3	Message Queues	22
6.6.4	Messaging Example	22
6.7	Sleep	23
6.8	Round-Robin Quantum	23
7	Build System	25
7.1	Source Layout	25
7.2	Building the kernel	25
8	License	27
8.1	License	27
9	Profiling Results	29
9.1	Date Performed	29
9.2	Compiler Information	29
9.3	Profiling Results	29
10	Class Index	31
10.1	Class Hierarchy	31
11	Class Index	33
11.1	Class List	33
12	File Index	37
12.1	File List	37
13	Class Documentation	41
13.1	BlockHeap Class Reference	41
13.1.1	Detailed Description	41
13.1.2	Member Function Documentation	42
13.1.2.1	Alloc	42
13.1.2.2	Create	42
13.1.2.3	Free	42
13.1.2.4	IsFree	42
13.2	BlockingObject Class Reference	43
13.2.1	Detailed Description	43
13.2.2	Member Function Documentation	43
13.2.2.1	Block	43
13.2.2.2	UnBlock	43
13.3	ButtonControl Class Reference	44

13.3.1 Detailed Description	45
13.3.2 Member Function Documentation	45
13.3.2.1 Activate	45
13.3.2.2 Draw	45
13.3.2.3 Init	45
13.3.2.4 ProcessEvent	45
13.4 CheckBoxControl Class Reference	46
13.4.1 Detailed Description	46
13.4.2 Member Function Documentation	47
13.4.2.1 Activate	47
13.4.2.2 Draw	47
13.4.2.3 Init	47
13.4.2.4 ProcessEvent	47
13.5 CircularLinkedList Class Reference	47
13.5.1 Detailed Description	48
13.5.2 Member Function Documentation	48
13.5.2.1 Add	48
13.5.2.2 Remove	48
13.6 DCPU Class Reference	49
13.6.1 Detailed Description	50
13.6.2 Member Function Documentation	51
13.6.2.1 AddPlugin	51
13.6.2.2 GetOperand	51
13.6.2.3 GetRegisters	51
13.6.2.4 HWN	51
13.6.2.5 IAQ	51
13.6.2.6 Init	51
13.6.2.7 RFI	52
13.6.2.8 SendInterrupt	52
13.6.3 Member Data Documentation	52
13.6.3.1 m_clPluginList	52
13.7 DCPU_Registers Struct Reference	52
13.7.1 Detailed Description	53
13.8 DCPUPlugin Class Reference	53
13.8.1 Detailed Description	54
13.8.2 Member Function Documentation	54
13.8.2.1 Enumerate	54
13.8.2.2 GetDeviceNumber	54
13.8.2.3 Init	54
13.8.2.4 Interrupt	55

13.9 DevNull Class Reference	55
13.9.1 Detailed Description	55
13.9.2 Member Function Documentation	56
13.9.2.1 Close	56
13.9.2.2 Control	56
13.9.2.3 Open	56
13.9.2.4 Read	56
13.9.2.5 Write	57
13.10 DoubleLinkedList Class Reference	57
13.10.1 Detailed Description	58
13.10.2 Member Function Documentation	58
13.10.2.1 Add	58
13.10.2.2 Remove	58
13.11 DrawBitmap_t Struct Reference	58
13.11.1 Detailed Description	59
13.12 DrawCircle_t Struct Reference	59
13.12.1 Detailed Description	59
13.13 DrawEllipse_t Struct Reference	60
13.13.1 Detailed Description	60
13.14 DrawLine_t Struct Reference	60
13.14.1 Detailed Description	61
13.15 DrawMove_t Struct Reference	61
13.15.1 Detailed Description	61
13.16 DrawPoint_t Struct Reference	61
13.16.1 Detailed Description	62
13.17 DrawPoly_t Struct Reference	62
13.17.1 Detailed Description	62
13.18 DrawRectangle_t Struct Reference	62
13.18.1 Detailed Description	63
13.19 DrawStamp_t Struct Reference	63
13.19.1 Detailed Description	63
13.20 DrawText_t Struct Reference	64
13.20.1 Detailed Description	64
13.21 DrawVector_t Struct Reference	64
13.21.1 Detailed Description	64
13.22 DrawWindow_t Struct Reference	65
13.22.1 Detailed Description	65
13.23 Driver Class Reference	65
13.23.1 Detailed Description	66
13.23.2 Member Function Documentation	66

13.23.2.1 Close	66
13.23.2.2 Control	66
13.23.2.3 GetPath	67
13.23.2.4 Open	67
13.23.2.5 Read	67
13.23.2.6 SetName	67
13.23.2.7 Write	68
13.24DriverList Class Reference	68
13.24.1 Detailed Description	68
13.24.2 Member Function Documentation	69
13.24.2.1 Add	69
13.24.2.2 FindByPath	69
13.24.2.3 Init	69
13.24.2.4 Remove	69
13.25FixedHeap Class Reference	69
13.25.1 Detailed Description	70
13.25.2 Member Function Documentation	70
13.25.2.1 Alloc	70
13.25.2.2 Create	70
13.25.2.3 Free	70
13.26Font_t Struct Reference	71
13.26.1 Detailed Description	71
13.27GlobalMessagePool Class Reference	71
13.27.1 Detailed Description	72
13.27.2 Member Function Documentation	72
13.27.2.1 Pop	72
13.27.2.2 Push	72
13.28Glyph_t Struct Reference	72
13.28.1 Detailed Description	72
13.29GraphicsDriver Class Reference	73
13.29.1 Detailed Description	74
13.29.2 Member Function Documentation	74
13.29.2.1 Bitmap	74
13.29.2.2 Circle	74
13.29.2.3 DrawPixel	74
13.29.2.4 Ellipse	75
13.29.2.5 Line	75
13.29.2.6 Move	75
13.29.2.7 Point	75
13.29.2.8 ReadPixel	75

13.29.2.9 Rectangle	75
13.29.2.10 SetWindow	76
13.29.2.11 Stamp	76
13.29.2.12 Text	76
13.29.2.13 TriangleFill	76
13.29.2.14 TriangleWire	76
13.30 GroupBoxControl Class Reference	77
13.30.1 Detailed Description	77
13.30.2 Member Function Documentation	78
13.30.2.1 Activate	78
13.30.2.2 Draw	78
13.30.2.3 Init	78
13.30.2.4 ProcessEvent	78
13.31 GuiControl Class Reference	78
13.31.1 Detailed Description	81
13.31.2 Member Function Documentation	81
13.31.2.1 Activate	81
13.31.2.2 ClearStale	81
13.31.2.3 Draw	81
13.31.2.4 GetControllIndex	81
13.31.2.5 GetControlOffset	81
13.31.2.6 GetHeight	82
13.31.2.7 GetLeft	82
13.31.2.8 GetParentControl	82
13.31.2.9 GetParentWindow	82
13.31.2.10 GetTop	82
13.31.2.11 GetWidth	83
13.31.2.12 GetZOrder	83
13.31.2.13 Init	83
13.31.2.14 IsInFocus	83
13.31.2.15 IsStale	83
13.31.2.16 ProcessEvent	83
13.31.2.17 SetControllIndex	84
13.31.2.18 SetHeight	84
13.31.2.19 SetLeft	84
13.31.2.20 SetParentControl	84
13.31.2.21 SetParentWindow	84
13.31.2.22 SetTop	85
13.31.2.23 SetWidth	85
13.31.2.24 SetZOrder	85

13.31.3 Member Data Documentation	85
13.31.3.1 m_ucControlIndex	85
13.31.3.2 m_ucZOrder	85
13.32 GuiEvent_t Struct Reference	86
13.32.1 Detailed Description	86
13.33 GuiEventSurface Class Reference	86
13.33.1 Detailed Description	87
13.33.2 Member Function Documentation	87
13.33.2.1 AddWindow	87
13.33.2.2 CopyEvent	87
13.33.2.3 Init	87
13.33.2.4 ProcessEvent	87
13.33.2.5 RemoveWindow	88
13.33.2.6 SendEvent	88
13.34 GuiWindow Class Reference	88
13.34.1 Detailed Description	90
13.34.2 Member Function Documentation	90
13.34.2.1 AddControl	90
13.34.2.2 CycleFocus	90
13.34.2.3 GetDriver	90
13.34.2.4 GetHeight	91
13.34.2.5 GetLeft	91
13.34.2.6 GetMaxZOrder	91
13.34.2.7 GetTop	91
13.34.2.8 GetWidth	91
13.34.2.9 Init	92
13.34.2.10 IsInFocus	92
13.34.2.11 ProcessEvent	92
13.34.2.12 Redraw	92
13.34.2.13 RemoveControl	92
13.34.2.14 SetDriver	92
13.34.2.15 SetFocus	93
13.34.2.16 SetHeight	93
13.34.2.17 SetLeft	93
13.34.2.18 SetTop	93
13.34.2.19 SetWidth	93
13.34.3 Member Data Documentation	94
13.34.3.1 m_pclDriver	94
13.35 HeapConfig Class Reference	94
13.35.1 Detailed Description	94

13.36JoystickEvent_t Struct Reference	94
13.36.1 Detailed Description	95
13.37Kernel Class Reference	95
13.37.1 Detailed Description	96
13.37.2 Member Function Documentation	96
13.37.2.1 Init	96
13.37.2.2 Start	96
13.38KernelSWI Class Reference	96
13.38.1 Detailed Description	97
13.38.2 Member Function Documentation	97
13.38.2.1 DI	97
13.38.2.2 RI	97
13.39KernelTimer Class Reference	97
13.39.1 Detailed Description	98
13.39.2 Member Function Documentation	98
13.39.2.1 GetOvertime	98
13.39.2.2 Read	98
13.39.2.3 RI	99
13.39.2.4 SetExpiry	99
13.39.2.5 SubtractExpiry	99
13.39.2.6 TimeToExpiry	99
13.40KeyEvent_t Struct Reference	100
13.40.1 Detailed Description	100
13.41LabelControl Class Reference	100
13.41.1 Detailed Description	101
13.41.2 Member Function Documentation	101
13.41.2.1 Activate	101
13.41.2.2 Draw	101
13.41.2.3 Init	101
13.41.2.4 ProcessEvent	102
13.42LinkList Class Reference	102
13.42.1 Detailed Description	103
13.42.2 Member Function Documentation	103
13.42.2.1 Add	103
13.42.2.2 GetHead	103
13.42.2.3 GetTail	103
13.42.2.4 Remove	103
13.43LinkListNode Class Reference	103
13.43.1 Detailed Description	105
13.43.2 Member Function Documentation	105

13.43.2.1	GetNext	105
13.43.2.2	GetPrev	105
13.44	MemUtil Class Reference	105
13.44.1	Detailed Description	106
13.44.2	Member Function Documentation	106
13.44.2.1	Checksum16	106
13.44.2.2	Checksum8	106
13.44.2.3	CompareMemory	107
13.44.2.4	CompareStrings	107
13.44.2.5	CopyMemory	107
13.44.2.6	CopyString	107
13.44.2.7	DecimalToHex	108
13.44.2.8	DecimalToString	108
13.44.2.9	SetMemory	108
13.44.2.10	StringLength	108
13.44.2.11	StringSearch	108
13.45	Message Class Reference	109
13.45.1	Detailed Description	109
13.45.2	Member Function Documentation	110
13.45.2.1	GetCode	110
13.45.2.2	GetData	110
13.45.2.3	SetCode	110
13.45.2.4	SetData	110
13.46	MessageQueue Class Reference	110
13.46.1	Detailed Description	111
13.46.2	Member Function Documentation	111
13.46.2.1	GetCount	111
13.46.2.2	Receive	111
13.46.2.3	Receive	111
13.46.2.4	Send	112
13.47	MouseEvent_t Struct Reference	112
13.47.1	Detailed Description	113
13.48	Mutex Class Reference	113
13.48.1	Detailed Description	114
13.48.2	Member Function Documentation	114
13.48.2.1	Claim	114
13.48.2.2	Claim	114
13.48.2.3	Release	114
13.48.2.4	WakeMe	114
13.49	PanelControl Class Reference	115

13.49.1 Detailed Description	115
13.49.2 Member Function Documentation	115
13.49.2.1 Activate	115
13.49.2.2 Draw	115
13.49.2.3 Init	116
13.49.2.4 ProcessEvent	116
13.50 Profiler Class Reference	116
13.50.1 Detailed Description	117
13.50.2 Member Function Documentation	117
13.50.2.1 Init	117
13.51 ProfileTimer Class Reference	117
13.51.1 Detailed Description	118
13.51.2 Member Function Documentation	118
13.51.2.1 ComputeCurrentTicks	118
13.51.2.2 GetAverage	118
13.51.2.3 GetCurrent	118
13.51.2.4 Init	119
13.51.2.5 Start	119
13.52 ProgressControl Class Reference	119
13.52.1 Detailed Description	120
13.52.2 Member Function Documentation	120
13.52.2.1 Activate	120
13.52.2.2 Draw	120
13.52.2.3 Init	120
13.52.2.4 ProcessEvent	120
13.53 Quantum Class Reference	121
13.53.1 Detailed Description	121
13.53.2 Member Function Documentation	121
13.53.2.1 AddThread	121
13.53.2.2 RemoveThread	121
13.53.2.3 SetTimer	122
13.53.2.4 UpdateTimer	122
13.54 Scheduler Class Reference	122
13.54.1 Detailed Description	123
13.54.2 Member Function Documentation	123
13.54.2.1 Add	123
13.54.2.2 GetCurrentThread	123
13.54.2.3 GetNextThread	123
13.54.2.4 GetStopList	123
13.54.2.5 GetThreadList	124

13.54.2.6 IsEnabled	124
13.54.2.7 Remove	124
13.54.2.8 Schedule	124
13.54.2.9 SetScheduler	124
13.55Screen Class Reference	125
13.55.1 Detailed Description	126
13.55.2 Member Function Documentation	126
13.55.2.1 Activate	126
13.55.2.2 Deactivate	126
13.56ScreenList Class Reference	126
13.56.1 Detailed Description	126
13.57ScreenManager Class Reference	127
13.57.1 Detailed Description	127
13.58Semaphore Class Reference	127
13.58.1 Detailed Description	128
13.58.2 Member Function Documentation	128
13.58.2.1 GetCount	128
13.58.2.2 Init	128
13.58.2.3 Pend	129
13.58.2.4 Pend	129
13.58.2.5 SetExpired	129
13.58.2.6 WakeMe	129
13.59Shell Class Reference	129
13.59.1 Detailed Description	130
13.60ShellCommand Class Reference	130
13.60.1 Detailed Description	130
13.61Slip Class Reference	131
13.61.1 Detailed Description	131
13.61.2 Member Function Documentation	132
13.61.2.1 DecodeByte	132
13.61.2.2 EncodeByte	132
13.61.2.3 GetDriver	132
13.61.2.4 ReadData	132
13.61.2.5 SetDriver	133
13.61.2.6 WriteData	133
13.61.2.7 WriteVector	133
13.62SlipDataVector Struct Reference	133
13.62.1 Detailed Description	134
13.63SlipMux Class Reference	134
13.63.1 Detailed Description	135

13.63.2 Member Function Documentation	135
13.63.2.1 GetDriver	135
13.63.2.2 GetQueue	135
13.63.2.3 GetSlip	135
13.63.2.4 Init	135
13.63.2.5 InstallHandler	136
13.63.2.6 MessageReceive	136
13.63.2.7 SetQueue	136
13.64SlipTerm Class Reference	136
13.64.1 Detailed Description	137
13.64.2 Member Function Documentation	137
13.64.2.1 Init	137
13.64.2.2 PrintLn	137
13.64.2.3 PrintLn	137
13.64.2.4 SetVerbosity	137
13.64.2.5 StrLen	138
13.64.3 Member Data Documentation	138
13.64.3.1 m_ucVerbosity	138
13.65StubControl Class Reference	138
13.65.1 Detailed Description	139
13.65.2 Member Function Documentation	139
13.65.2.1 Activate	139
13.65.2.2 Draw	139
13.65.2.3 Init	139
13.65.2.4 ProcessEvent	139
13.66SystemHeap Class Reference	140
13.66.1 Detailed Description	140
13.67Thread Class Reference	140
13.67.1 Detailed Description	142
13.67.2 Member Function Documentation	142
13.67.2.1 ContextSwitchSWI	142
13.67.2.2 Exit	142
13.67.2.3 GetCurrent	143
13.67.2.4 GetID	143
13.67.2.5 GetOwner	143
13.67.2.6 GetPriority	143
13.67.2.7 GetQuantum	143
13.67.2.8 GetStackSlack	143
13.67.2.9 InheritPriority	144
13.67.2.10Init	144

13.67.2.11SetCurrent	144
13.67.2.12SetID	144
13.67.2.13SetName	145
13.67.2.14SetOwner	145
13.67.2.15SetPriority	145
13.67.2.16SetQuantum	145
13.67.2.17Sleep	145
13.67.2.18Yield	146
13.68ThreadList Class Reference	146
13.68.1 Detailed Description	147
13.68.2 Member Function Documentation	147
13.68.2.1 Add	147
13.68.2.2 Add	147
13.68.2.3 HighestWaiter	147
13.68.2.4 Remove	147
13.68.2.5 SetFlagPointer	148
13.68.2.6 SetPriority	148
13.69ThreadPort Class Reference	148
13.69.1 Detailed Description	149
13.69.2 Member Function Documentation	149
13.69.2.1 InitStack	149
13.70Timer Class Reference	149
13.70.1 Detailed Description	150
13.70.2 Member Function Documentation	150
13.70.2.1 SetCallback	150
13.70.2.2 SetData	151
13.70.2.3 SetFlags	151
13.70.2.4 SetIntervalMSeconds	151
13.70.2.5 SetIntervalSeconds	151
13.70.2.6 SetIntervalTicks	151
13.70.2.7 SetIntervalUSeconds	151
13.70.2.8 SetOwner	152
13.70.2.9 Stop	152
13.71TimerEvent_t Struct Reference	152
13.71.1 Detailed Description	152
13.72TimerList Class Reference	152
13.72.1 Detailed Description	153
13.72.2 Member Function Documentation	153
13.72.2.1 Add	153
13.72.2.2 Init	153

13.72.2.3 Process	154
13.72.2.4 Remove	154
13.73TimerScheduler Class Reference	154
13.73.1 Detailed Description	155
13.73.2 Member Function Documentation	155
13.73.2.1 Add	155
13.73.2.2 Init	155
13.73.2.3 Process	155
13.73.2.4 Remove	155
13.74TouchEvent_t Struct Reference	155
13.74.1 Detailed Description	156
13.75UnitTest Class Reference	156
13.75.1 Detailed Description	157
13.75.2 Member Function Documentation	157
13.75.2.1 Complete	157
13.75.2.2 GetFailed	157
13.75.2.3 GetName	157
13.75.2.4 GetPassed	158
13.75.2.5 GetResult	158
13.75.2.6 GetTotal	158
13.75.2.7 SetName	158
13.76WriteBuffer16 Class Reference	158
13.76.1 Detailed Description	159
13.76.2 Member Function Documentation	159
13.76.2.1 SetBuffers	159
13.76.2.2 SetCallback	159
13.76.2.3 WriteData	160
13.76.2.4 WriteVector	160
14 File Documentation	161
14.1 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp File Reference	161
14.1.1 Detailed Description	161
14.2 blocking.cpp	161
14.3 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h File Reference	162
14.3.1 Detailed Description	162
14.4 blocking.h	163
14.5 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.cpp File Reference	163
14.5.1 Detailed Description	163
14.6 control_button.cpp	164
14.7 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h File Reference	166

14.7.1 Detailed Description	167
14.8 control_button.h	167
14.9 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.cpp File Reference	168
14.9.1 Detailed Description	168
14.9.2 Variable Documentation	168
14.9.2.1 aucBox	168
14.9.2.2 aucCheck	169
14.10 control_checkbox.cpp	169
14.11 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.h File Reference .	171
14.11.1 Detailed Description	171
14.12 control_checkbox.h	171
14.13 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.cpp File Reference	172
14.13.1 Detailed Description	172
14.14 control_groupbox.cpp	173
14.15 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.h File Reference .	174
14.15.1 Detailed Description	174
14.16 control_groupbox.h	174
14.17 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h File Reference . . .	175
14.17.1 Detailed Description	175
14.18 control_label.h	175
14.19 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.cpp File Reference . .	176
14.19.1 Detailed Description	176
14.20 control_panel.cpp	176
14.21 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h File Reference . . .	177
14.21.1 Detailed Description	177
14.22 control_panel.h	177
14.23 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.cpp File Reference	178
14.23.1 Detailed Description	178
14.24 control_progress.cpp	178
14.25 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.h File Reference .	179
14.25.1 Detailed Description	180
14.26 control_progress.h	180
14.27 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.cpp File Reference	181
14.27.1 Detailed Description	181
14.28 dcpu.cpp	182
14.29 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h File Reference	192
14.29.1 Detailed Description	194
14.29.2 Macro Definition Documentation	194
14.29.2.1 DCPU_NORMAL_OPCODE_MASK	194
14.29.3 Enumeration Type Documentation	194

14.29.3.1 DCPU_OpBasic	194
14.29.3.2 DCPU_OpExtended	195
14.30dcpu.h	196
14.31/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/debug_tokens.h File Reference	200
14.31.1 Detailed Description	202
14.32debug_tokens.h	202
14.33/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h File Reference	203
14.33.1 Detailed Description	204
14.34draw.h	204
14.35/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp File Reference	206
14.35.1 Detailed Description	206
14.36driver.cpp	207
14.37/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h File Reference	208
14.37.1 Detailed Description	208
14.37.2 Intro	208
14.37.3 Driver Design	209
14.37.4 Driver API	209
14.38driver.h	209
14.39/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp File Reference	210
14.39.1 Detailed Description	210
14.40fixed_heap.cpp	211
14.41/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h File Reference	213
14.41.1 Detailed Description	213
14.42fixed_heap.h	213
14.43/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h File Reference	214
14.43.1 Detailed Description	214
14.44font.h	215
14.45/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.cpp File Reference	215
14.45.1 Detailed Description	215
14.46graphics.cpp	216
14.47/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h File Reference	227
14.47.1 Detailed Description	227
14.48graphics.h	227
14.49/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp File Reference	228
14.49.1 Detailed Description	228
14.50gui.cpp	228
14.51/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h File Reference	236
14.51.1 Detailed Description	237
14.51.2 Enumeration Type Documentation	237
14.51.2.1 GuiEventType_t	237

14.52gui.h	237
14.53/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.cpp File Reference	242
14.53.1 Detailed Description	243
14.54kernel.cpp	243
14.55/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h File Reference	244
14.55.1 Detailed Description	244
14.56kernel.h	244
14.57/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel_debug.h File Reference	244
14.57.1 Detailed Description	245
14.58kernel_debug.h	245
14.59/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.cpp File Reference	246
14.59.1 Detailed Description	246
14.60kernelswi.cpp	246
14.61/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h File Reference	247
14.61.1 Detailed Description	248
14.62kernelswi.h	248
14.63/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.cpp File Reference	248
14.63.1 Detailed Description	249
14.64kerneltimer.cpp	249
14.65/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.h File Reference	250
14.65.1 Detailed Description	251
14.66kerneltimer.h	251
14.67/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltypes.h File Reference	252
14.67.1 Detailed Description	252
14.68kerneltypes.h	252
14.69/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/keycodes.h File Reference	252
14.69.1 Detailed Description	253
14.70keycodes.h	253
14.71/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp File Reference	255
14.71.1 Detailed Description	255
14.72kprofile.cpp	255
14.73/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h File Reference	256
14.73.1 Detailed Description	257
14.74kprofile.h	257
14.75/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp File Reference	258
14.75.1 Detailed Description	258
14.76ll.cpp	258
14.77/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h File Reference	260
14.77.1 Detailed Description	260
14.78ll.h	261

14.79/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/manual.h File Reference	262
14.79.1 Detailed Description	262
14.80manual.h	262
14.81/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mark3cfg.h File Reference	262
14.81.1 Detailed Description	263
14.81.2 Macro Definition Documentation	263
14.81.2.1 KERNEL_USE_DRIVER	263
14.81.2.2 KERNEL_USE_DYNAMIC_THREADS	263
14.81.2.3 KERNEL_USE_MESSAGE	263
14.81.2.4 KERNEL_USE_MUTEX	264
14.81.2.5 KERNEL_USE_PROFILER	264
14.81.2.6 KERNEL_USE_QUANTUM	264
14.81.2.7 KERNEL_USE_SEMAPHORE	264
14.81.2.8 KERNEL_USE_THREADNAME	264
14.81.2.9 KERNEL_USE_TIMERS	264
14.82mark3cfg.h	265
14.83/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.cpp File Reference	265
14.83.1 Detailed Description	265
14.84memutil.cpp	266
14.85/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.h File Reference	269
14.85.1 Detailed Description	269
14.86memutil.h	269
14.87/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp File Reference	270
14.87.1 Detailed Description	270
14.88message.cpp	271
14.89/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h File Reference	272
14.89.1 Detailed Description	273
14.89.2 Using Messages, Queues, and the Global Message Pool	273
14.90message.h	274
14.91/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp File Reference	275
14.91.1 Detailed Description	275
14.92mutex.cpp	275
14.93/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h File Reference	278
14.93.1 Detailed Description	278
14.93.2 Initializing	279
14.93.3 Resource protection example	279
14.94mutex.h	279
14.95/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.cpp File Reference	280
14.95.1 Detailed Description	280
14.96profile.cpp	280

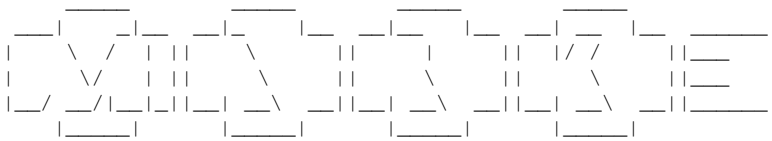
14.97/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.h File Reference	282
14.97.1 Detailed Description	282
14.98profile.h	283
14.99/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp File Reference	283
14.99.1 Detailed Description	284
14.100quantum.cpp	284
14.101/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h File Reference	285
14.101.1 Detailed Description	285
14.102quantum.h	286
14.103/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.cpp File Reference	286
14.103.1 Detailed Description	287
14.104scheduler.cpp	287
14.105/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h File Reference	288
14.105.1 Detailed Description	288
14.106scheduler.h	289
14.107/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp File Reference	289
14.107.1 Detailed Description	290
14.108screen.cpp	290
14.109/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h File Reference	291
14.109.1 Detailed Description	291
14.110screen.h	291
14.111/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp File Reference	292
14.111.1 Detailed Description	293
14.112semaphore.cpp	293
14.113/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h File Reference	295
14.113.1 Detailed Description	296
14.114semaphore.h	296
14.115/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp File Reference	297
14.115.1 Detailed Description	297
14.116slip.cpp	297
14.117/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h File Reference	301
14.117.1 Detailed Description	301
14.117.2 Enumeration Type Documentation	301
14.117.2.1 SlipChannel	301
14.118slip.h	302
14.119/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.cpp File Reference	302
14.119.1 Detailed Description	303
14.119.2 Function Documentation	303
14.119.2.1 SlipMux_CallBack	303
14.120slip_mux.cpp	303

14.121	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h File Reference	304
14.121.1	Detailed Description	305
14.122	lip_mux.h	305
14.123	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.cpp File Reference	306
14.123.1	Detailed Description	306
14.124	lipterm.cpp	306
14.125	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.h File Reference	307
14.125.1	Detailed Description	307
14.126	lipterm.h	307
14.127	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.cpp File Reference	308
14.127.1	Detailed Description	308
14.128	system_heap.cpp	308
14.129	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.h File Reference	310
14.129.1	Detailed Description	311
14.129.2	Macro Definition Documentation	311
14.129.2.1	HEAP_RAW_SIZE	311
14.129.2.2	HEAP_RAW_SIZE_1	312
14.130	system_heap.h	312
14.131	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap_config.h File Reference	314
14.131.1	Detailed Description	315
14.131.2	Macro Definition Documentation	315
14.131.2.1	HEAP_BLOCK_SIZE_1	315
14.132	system_heap_config.h	315
14.133	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.cpp File Reference	316
14.133.1	Detailed Description	316
14.134	hread.cpp	316
14.135	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.h File Reference	320
14.135.1	Detailed Description	320
14.136	hread.h	321
14.137	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp File Reference	322
14.137.1	Detailed Description	323
14.138	hreadlist.cpp	323
14.139	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.h File Reference	324
14.139.1	Detailed Description	324
14.140	hreadlist.h	324
14.141	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp File Reference	325
14.141.1	Detailed Description	326
14.142	hreadport.cpp	326
14.143	home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.h File Reference	328
14.143.1	Detailed Description	328

14.143.2Macro Definition Documentation	329
14.143.2.1CS_ENTER	329
14.143.2.2CS_EXIT	329
14.144threadport.h	329
14.145home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp File Reference	331
14.145.1Detailed Description	331
14.145.2Macro Definition Documentation	331
14.145.2.1TL_FUDGE_FACTOR	331
14.146merlist.cpp	332
14.147home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h File Reference	335
14.147.1Detailed Description	336
14.147.2Macro Definition Documentation	336
14.147.2.1TIMERLIST_FLAG_EXPIRED	336
14.148merlist.h	336
14.149home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.cpp File Reference	338
14.149.1Detailed Description	338
14.150acebuffer.cpp	338
14.151home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.h File Reference	339
14.151.1Detailed Description	339
14.152acebuffer.h	339
14.153home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp File Reference	340
14.153.1Detailed Description	340
14.154nit_test.cpp	340
14.155home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h File Reference	341
14.155.1Detailed Description	341
14.156nit_test.h	341
14.157home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.cpp File Reference	342
14.157.1Detailed Description	342
14.158writebuf16.cpp	342
14.159home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.h File Reference	344
14.159.1Detailed Description	344
14.160writebuf16.h	344

Chapter 1

The Mark3 Realtime Kernel



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

```
Copyright (c) 2012 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 K_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 K_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 K_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 `K_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 K_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, K_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
- 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 K_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 K_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 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.

6.6.1 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.

6.6.2 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.

6.6.3 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.

6.6.4 Messaging Example

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

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

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

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

    // Send the data to the message queue
    clMsgQ.Send(pclMesg);
}

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

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

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

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

6.7 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;
}
```

6.8 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 `K_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

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, Mark2 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.

7.1 Source Layout

One key aspect of Mark2 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	Mark2 Bootloader code for AVR
build	Makefile support for various platforms
doc	Documentation (including this)
drivers	Device driver code
example	Example applications
kernel	Basic Mark2 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

7.2 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.

Chapter 8

License

8.1 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 the <organization> 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 <COPYRIGHT holder>=""> 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 9

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.

9.1 Date Performed

Sun Jan 27 10:36:50 EST 2013

9.2 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`

9.3 Profiling Results

- Semaphore Initialization: 2356994244 cycles (averaged over 82 iterations)
- Semaphore Post (uncontested): 148 cycles (averaged over 82 iterations)
- Semaphore Pend (uncontested): 68 cycles (averaged over 82 iterations)
- Semaphore Flyback Time (Contested Pend): 1668 cycles (averaged over 82 iterations)
- Mutex Init: 4190211955 cycles (averaged over 82 iterations)
- Mutex Claim: 204 cycles (averaged over 82 iterations)
- Mutex Release: 47 cycles (averaged over 82 iterations)
- Thread Initialize: 7897 cycles (averaged over 82 iterations)
- Thread Start: 798 cycles (averaged over 82 iterations)
- Context Switch: 171 cycles (averaged over 82 iterations)
- Thread Schedule: 70 cycles (averaged over 82 iterations)

Chapter 10

Class Index

10.1 Class Hierarchy

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

BlockHeap	41
BlockingObject	43
Mutex	113
Semaphore	127
DCPU	49
DCPU_Registers	52
DrawBitmap_t	58
DrawCircle_t	59
DrawEllipse_t	60
DrawLine_t	60
DrawMove_t	61
DrawPoint_t	61
DrawPoly_t	62
DrawRectangle_t	62
DrawStamp_t	63
DrawText_t	64
DrawVector_t	64
DrawWindow_t	65
DriverList	68
FixedHeap	69
Font_t	71
GlobalMessagePool	71
Glyph_t	72
GuiEvent_t	86
GuiEventSurface	86
HeapConfig	94
JoystickEvent_t	94
Kernel	95
KernelSWI	96
KernelTimer	97
KeyEvent_t	100
LinkList	102
CircularLinkList	47
ThreadList	146
DoubleLinkList	57
Shell	129
TimerList	152

LinkedListNode	103
DCPUPlugin	53
Driver	65
DevNull	55
GraphicsDriver	73
GuiControl	78
ButtonControl	44
CheckBoxControl	46
GroupBoxControl	77
LabelControl	100
PanelControl	115
ProgressControl	119
StubControl	138
GuiWindow	88
Message	109
Screen	125
ShellCommand	130
Thread	140
Timer	149
MemUtil	105
MessageQueue	110
MouseEvent_t	112
Profiler	116
ProfileTimer	117
Quantum	121
Scheduler	122
ScreenList	126
ScreenManager	127
Slip	131
SlipDataVector	133
SlipMux	134
SlipTerm	136
SystemHeap	140
ThreadPort	148
TimerEvent_t	152
TimerScheduler	154
TouchEvent_t	155
UnitTest	156
WriteBuffer16	158

Chapter 11

Class Index

11.1 Class List

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

BlockHeap	
Single-block-size heap	41
BlockingObject	
Class implementing thread-blocking primitives	43
ButtonControl	44
CheckBoxControl	46
CircularLinkedList	
Circular-linked-list data type, inherited from the base LinkedList type	47
DCPU	
DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH	49
DCPU_Registers	
Structure defining the DCPU hardware registers	52
DCPUPlugin	
Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system	53
DevNull	
This class implements the "default" driver (/dev/null)	55
DoubleLinkedList	
Doubly-linked-list data type, inherited from the base LinkedList type	57
DrawBitmap_t	
Defines a bitmap	58
DrawCircle_t	
Defines a circle	59
DrawEllipse_t	
Defines a ellipse	60
DrawLine_t	
Defines a simple line	60
DrawMove_t	
Simple 2D copy/paste	61
DrawPoint_t	
Defines a pixel	61
DrawPoly_t	
Defines the structure of an arbitrary polygon	62
DrawRectangle_t	
Defines a rectangle	62
DrawStamp_t	
Defines a 1-bit 2D bitmap of arbitrary resolution	63

DrawText_t	Defines a bitmap-rendered string	64
DrawVector_t	Specifies a single 2D point	64
DrawWindow_t	Defines the active window - establishes boundaries for drawing on the current display	65
Driver	Base device-driver class used in hardware abstraction	65
DriverList	List of Driver objects used to keep track of all device drivers in the system	68
FixedHeap	Fixed-size-block heap allocator with multiple block sizes	69
Font_t	71
GlobalMessagePool	Implements a list of message objects shared between all threads	71
Glyph_t	72
GraphicsDriver	Defines the base graphics driver class, which is inherited by all other graphics drivers	73
GroupBoxControl	77
GuiControl	GUI Control Base Class	78
GuiEvent_t	86
GuiEventSurface	GUI Event Surface Object	86
GuiWindow	Basic Window Class	88
HeapConfig	Heap configuration object	94
JoystickEvent_t	Joystick UI event structure	94
Kernel	Class that encapsulates all of the kernel startup functions	95
KernelSWI	Class providing the software-interrupt required for context-switching in the kernel	96
KernelTimer	Hardware timer interface, used by all scheduling/timer subsystems	97
KeyEvent_t	Keyboard UI event structure definition	100
LabelControl	100
LinkList	Abstract-data-type from which all other linked-lists are derived	102
LinkListNode	Basic linked-list node data structure	103
MemUtil	String and Memory manipulation class	105
Message	Class to provide message-based IPC services in the kernel	109
MessageQueue	List of messages, used as the channel for sending and receiving messages between threads	110
MouseEvent_t	Mouse UI event structure	112
Mutex	Mutual-exclusion locks, based on BlockingObject	113
PanelControl	115
Profiler	System profiling timer interface	116
ProfileTimer	Profiling timer	117

ProgressControl	119
Quantum	
Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling	121
Scheduler	
Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping	122
Screen	125
ScreenList	126
ScreenManager	127
Semaphore	
Counting semaphore, based on BlockingObject base class	127
Shell	129
ShellCommand	130
Slip	
Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP)	131
SlipDataVector	
Data structure used for vector-based SLIP data transmission	133
SlipMux	
Static-class which implements a multiplexed stream of SLIP data over a single interface	134
SlipTerm	
Class implementing a simple debug terminal interface	136
StubControl	
Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented	138
SystemHeap	140
Thread	
Object providing fundamental multitasking support in the kernel	140
ThreadList	
This class is used for building thread-management facilities, such as schedulers, and blocking objects	146
ThreadPort	
Class defining the architecture specific functions required by the kernel	148
Timer	
Timer - an event-driven execution context based on a specified time interval	149
TimerEvent_t	
Timer UI event structure	152
TimerList	
TimerList class - a doubly-linked-list of timer objects	152
TimerScheduler	
"Static" Class used to interface a global TimerList with the rest of the kernel	154
TouchEvent_t	
Touch UI event structure	155
UnitTest	
Class used to implement a simple unit-testing framework	156
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	158

Chapter 12

File Index

12.1 File List

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

/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp	
Implementation of base class for blocking objects	161
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h	
Blocking object base class declarations	163
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/colordepth.h	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/colospace.h	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.cpp	
GUI Button Control Implementation	164
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h	
GUI Button Control	167
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.cpp	
Checkbox Control	169
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.h	
Checkbox Control	171
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.cpp	
GUI GroupBox Control Implementation	173
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.h	
GUI Group Box Control	174
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.cpp	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h	
GUI Label Control	175
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.cpp	
GUI Panel Control Implementation	176
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h	
GUI Panel Control	177
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.cpp	
GUI Progress Bar Control	178
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.h	
GUI Progress Bar Control	180
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.cpp	
Portable DCPU-16 CPU emulator	182
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h	
DCPU-16 emulator	196
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/debug_tokens.h	
Hex codes/translation tables used for efficient string tokenization	202
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h	
Raster graphics APIs Description: Implements basic drawing functionality	204

/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp	
Device driver/hardware abstraction layer	207
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h	
Driver abstraction framework	209
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp	
Fixed-block-size memory management	211
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h	
Fixed-block-size heaps	213
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h	
Font structure definitions	215
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fontport.h	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.cpp	
Generic graphics driver implementation	216
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h	
Graphics driver class declaration	227
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp	
Graphical User Interface classes and data structure definitions	228
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h	
Graphical User Interface classes and data structure declarations	237
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.cpp	
Kernel initialization and startup code	243
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h	
Kernel initialization and startup class	244
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel_debug.h	
Macros and functions used for assertions, kernel traces, etc	245
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.cpp	
Kernel Software interrupt implementation for ATMega328p	246
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h	
Kernel Software interrupt declarations	248
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.cpp	
Kernel Timer Implementation for ATMega328p	249
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.h	
Kernel Timer Class declaration	251
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltypes.h	
Basic data type primitives used throughout the OS	252
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/keycodes.h	
Standard ASCII keyboard codes	253
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp	
ATMega328p Profiling timer implementation	255
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h	
Profiling timer hardware interface	257
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp	
Core Linked-List implementation, from which all kernel objects are derived	258
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h	
Core linked-list declarations, used by all kernel list types	261
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/manual.h	
Ascii-format documentation, used by doxygen to create various printable and viewable forms	262
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mark3cfg.h	
Mark3 Kernel Configuration	265
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.cpp	
Implementation of memory, string, and conversion routines	266
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.h	
Utility class containing memory, string, and conversion routines	269
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp	
Inter-thread communications via message passing	271
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h	
Inter-thread communication via message-passing	274

/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp	
Mutual-exclusion object	275
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h	
Mutual exclusion class declaration	279
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.cpp	
Code profiling utilities	280
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.h	
High-precision profiling timers	283
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profiling_results.h	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp	
Thread Quantum Implementation for Round-Robin Scheduling	284
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h	
Thread Quantum declarations for Round-Robin Scheduling	286
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.cpp	
Strict-Priority + Round-Robin thread scheduler implementation	287
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h	
Thread scheduler function declarations	289
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp	
Higher level window management framework	290
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h	
Higher level window management framework	291
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp	
Semaphore Blocking-Object Implementation	293
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h	
Semaphore Blocking Object class declarations	296
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.cpp	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.h	??
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp	
Serial Line IP framing code	297
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h	
Serial Line IP framing code	302
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.cpp	
FunkenSlip Channel Multiplexer	303
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h	
FunkenSlip Channel Multiplexer	305
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.cpp	
Serial debug interface using SLIP protocol, and FunkenSlip multiplexing	306
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.h	
Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing	307
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.cpp	
Global system-heap implementation	308
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.h	
Global system-heap implementation	312
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap_config.h	
System heap configuration - defines the block sizes and counts used to fulfill system/service allocations	315
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.cpp	
Platform-Independent thread class Definition	316
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.h	
Platform independent thread class declarations	321
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp	
Thread linked-list definitions	323
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.h	
Thread linked-list declarations	324
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp	
ATMega328p Multithreading	326
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.h	
ATMega328p Multithreading support	329

/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp	
Timer data structure + scheduler implementations	332
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h	
Timer list and timer-scheduling declarations	336
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.cpp	
Kernel trace buffer class definition	338
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.h	
Kernel trace buffer class declaration	339
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp	
Unit test class definition	340
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h	
Unit test class declarations	341
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.cpp	
16 bit circular buffer implementation with callbacks	342
/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.h	
Thread-safe circular buffer implementation with 16-bit elements	344

Chapter 13

Class Documentation

13.1 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.

13.1.1 Detailed Description

Single-block-size heap.

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

13.1.2 Member Function Documentation

13.1.2.1 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](#).

13.1.2.2 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](#).

13.1.2.3 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](#).

13.1.2.4 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:

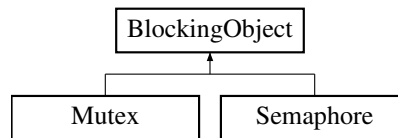
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp](#)

13.2 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_*)

Protected Attributes

- [ThreadList](#) *m_clBlockList*
ThreadList which is used to hold the list of threads blocked on a given object.

13.2.1 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 65 of file [blocking.h](#).

13.2.2 Member Function Documentation

13.2.2.1 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](#).

13.2.2.2 void [BlockingObject::UnBlock](#) ([Thread](#) * *pciThread_*) [protected]

Parameters

<code>pclThread_</code>	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

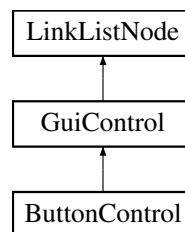
Definition at line 52 of file [blocking.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp](#)

13.3 ButtonControl Class Reference

Inheritance diagram for ButtonControl:



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 **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

13.3.1 Detailed Description

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

13.3.2 Member Function Documentation

13.3.2.1 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](#).

13.3.2.2 void ButtonControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

13.3.2.3 void ButtonControl::Init () [virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

13.3.2.4 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](#).

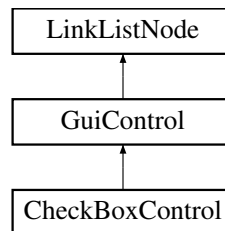
Definition at line 117 of file [control_button.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.cpp](#)

13.4 CheckBoxControl Class Reference

Inheritance diagram for CheckBoxControl:



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 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

13.4.1 Detailed Description

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

13.4.2 Member Function Documentation

13.4.2.1 `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](#).

13.4.2.2 `void CheckBoxControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

13.4.2.3 `void CheckBoxControl::Init () [virtual]`

Initialilize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

13.4.2.4 `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 112 of file [control_checkbox.cpp](#).

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

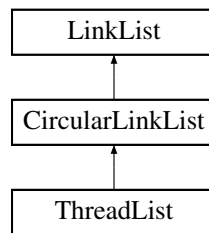
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_checkbox.cpp](#)

13.5 CircularLinkedList Class Reference

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

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

Inheritance diagram for CircularLinkedList:



Public Member Functions

- 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.
- 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

13.5.1 Detailed Description

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

Definition at line 201 of file [ll.h](#).

13.5.2 Member Function Documentation

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

Add the linked list node to this linked list.

Parameters

node_	Pointer to the node to add
-----------------------	----------------------------

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

Definition at line 89 of file [ll.cpp](#).

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

Add the linked list node to this linked list.

Parameters

node_	Pointer to the node to remove
-----------------------	-------------------------------

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

Definition at line 114 of file [ll.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp](#)

13.6 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.

13.6.1 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](#).

13.6.2 Member Function Documentation

13.6.2.1 void DCPU::AddPlugin (DCPUPugin * *pclPlugin_*)

Add a plugin to the CPU.

Parameters

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

Definition at line 944 of file [dcpu.cpp](#).

13.6.2.2 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 721 of file [dcpu.cpp](#).

13.6.2.3 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](#).

13.6.2.4 void DCPU::HWN () [private]

Returns the number of connected hardware devices to "a"

Definition at line 641 of file [dcpu.cpp](#).

13.6.2.5 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 623 of file [dcpu.cpp](#).

13.6.2.6 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 696 of file [dcpu.cpp](#).

13.6.2.7 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 608 of file [dcpu.cpp](#).

13.6.2.8 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 918 of file [dcpu.cpp](#).

13.6.3 Member Data Documentation

13.6.3.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.cpp](#)

13.7 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]
};

```

13.7.1 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:

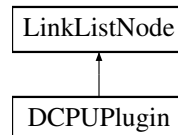
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h](#)

13.8 DCPUPlugin 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 DCPUPlugin:



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](#) *pclCPU_)
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

13.8.1 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](#).

13.8.2 Member Function Documentation

13.8.2.1 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](#).

13.8.2.2 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](#).

13.8.2.3 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](#).

13.8.2.4 void DCPUPugin::Interrupt (DCPU * *pclCPU_*) [inline]

Execute the hardware callback.

Parameters

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

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

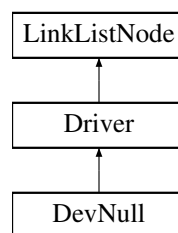
The documentation for this class was generated from the following file:

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.h](#)

13.9 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

13.9.1 Detailed Description

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

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

13.9.2 Member Function Documentation

13.9.2.1 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](#).

13.9.2.2 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](#).

13.9.2.3 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](#).

13.9.2.4 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 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](#).

13.9.2.5 `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:

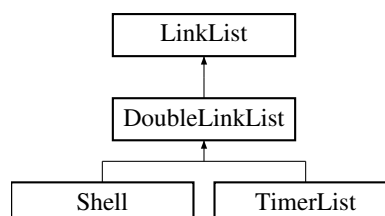
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp](#)

13.10 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

13.10.1 Detailed Description

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

Definition at line 170 of file [ll.h](#).

13.10.2 Member Function Documentation

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

Add the linked list node to this linked list.

Parameters

node_	Pointer to the node to add
-----------------------	----------------------------

Implements [LinkedList](#).

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

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

Add the linked list node to this linked list.

Parameters

node_	Pointer to the node to remove
-----------------------	-------------------------------

Implements [LinkedList](#).

Definition at line 64 of file [ll.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp](#)

13.11 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.

13.11.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.12 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.

13.12.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.13 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.

13.13.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.14 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.

13.14.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.15 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.

13.15.1 Detailed Description

Simple 2D copy/paste.

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

Definition at line 172 of file [draw.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.16 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.

13.16.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.17 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.

13.17.1 Detailed Description

Defines the structure of an arbitrary polygon.

Can be used to specify the

Definition at line 199 of file [draw.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.18 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.

13.18.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.19 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.

13.19.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.20 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.

13.20.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.21 DrawVector_t Struct Reference

Specifies a single 2D point.

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

Public Attributes

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

13.21.1 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 [188](#) of file [draw.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.22 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.

13.22.1 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 159 of file [draw.h](#).

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

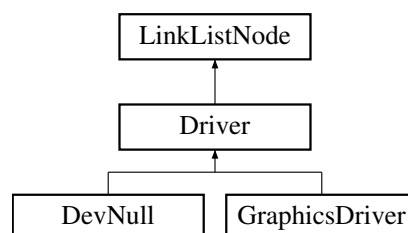
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.h](#)

13.23 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

13.23.1 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](#).

13.23.2 Member Function Documentation

13.23.2.1 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](#).

13.23.2.2 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](#).

13.23.2.3 `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](#).

13.23.2.4 `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](#).

13.23.2.5 `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 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](#).

13.23.2.6 `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](#).

13.23.2.7 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h](#)

13.24 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.

13.24.1 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](#).

13.24.2 Member Function Documentation

13.24.2.1 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](#).

13.24.2.2 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 97 of file [driver.cpp](#).

13.24.2.3 void DriverList::Init () [static]

Initialize the list of drivers.

Must be called prior to using the device driver library.

Definition at line 88 of file [driver.cpp](#).

13.24.2.4 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.cpp](#)

13.25 FixedHeap Class Reference

Fixed-size-block heap allocator with multiple block sizes.

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

Public Member Functions

- void [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.
- 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_paclHeaps](#)

Pointer to the configuration data used by the heap.

13.25.1 Detailed Description

Fixed-size-block heap allocator with multiple block sizes.

Definition at line 104 of file [fixed_heap.h](#).

13.25.2 Member Function Documentation

13.25.2.1 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

usSize_	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](#).

13.25.2.2 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

pvHeap_	Pointer to the data blob that will contain the heap
pclHeapConfig_	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](#).

13.25.2.3 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

<code>pvNode_</code>	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp](#)

13.26 Font_t Struct Reference

Public Attributes

- K_UCHAR **ucSize**
- K_UCHAR **ucFlags**
- K_UCHAR **ucStartChar**
- K_UCHAR **ucMaxChar**
- K_CHAR * **szName**
- const FONT_STORAGE_TYPE * **pucFontData**

13.26.1 Detailed Description

Definition at line 43 of file [font.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h](#)

13.27 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](#) [8]
Array of message objects that make up the message pool.
- static [DoubleLinkedList](#) [m_clList](#)
Linked list used to manage the [Message](#) objects.

13.27.1 Detailed Description

Implements a list of message objects shared between all threads.

Definition at line 157 of file [message.h](#).

13.27.2 Member Function Documentation

13.27.2.1 `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](#).

13.27.2.2 `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

<i>pclMessage_</i>	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp](#)

13.28 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.

13.28.1 Detailed Description

Definition at line 26 of file [font.h](#).

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

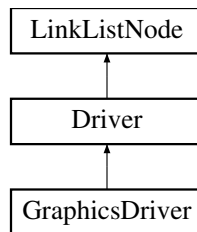
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.h](#)

13.29 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_)
Draw a filled polygon to the display.
- virtual void **Text** (**DrawText_t** *pstText_)
Draw a string of text to the display using a bitmap font.
- virtual K_USHORT **TextWidth** (**DrawText_t** *pstText_)
Get the width of the text.
- 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

13.29.1 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](#).

13.29.2 Member Function Documentation

13.29.2.1 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 300 of file [graphics.cpp](#).

13.29.2.2 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 176 of file [graphics.cpp](#).

13.29.2.3 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](#).

13.29.2.4 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 248 of file [graphics.cpp](#).

13.29.2.5 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 48 of file [graphics.cpp](#).

13.29.2.6 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 438 of file [graphics.cpp](#).

13.29.2.7 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 42 of file [graphics.cpp](#).

13.29.2.8 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](#).

13.29.2.9 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 131 of file [graphics.cpp](#).

13.29.2.10 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 882 of file [graphics.cpp](#).

13.29.2.11 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 399 of file [graphics.cpp](#).

13.29.2.12 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 499 of file [graphics.cpp](#).

13.29.2.13 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 655 of file [graphics.cpp](#).

13.29.2.14 void GraphicsDriver::TriangleWire (DrawPoly_t * *pstPoly_*) [virtual]

Draw a wireframe triangle to the display.

Parameters

<i>pstPoly_</i>	Pointer to the polygon to draw.
-----------------	---------------------------------

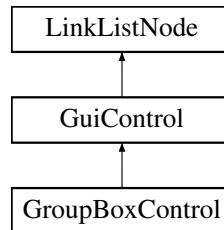
Definition at line 630 of file [graphics.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.cpp](#)

13.30 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

13.30.1 Detailed Description

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

13.30.2 Member Function Documentation

13.30.2.1 `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](#).

13.30.2.2 `void GroupBoxControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 30 of file [control_groupbox.cpp](#).

13.30.2.3 `virtual void GroupBoxControl::Init () [inline],[virtual]`

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

13.30.2.4 `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:

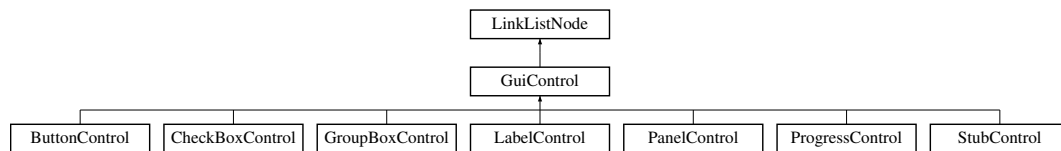
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_groupbox.cpp](#)

13.31 GuiControl Class Reference

GUI Control Base Class.

```
#include <gui.h>
```

Inheritance diagram for GuiControl:



Public Member Functions

- virtual void **Init** ()=0
Initialiize 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 **GetControlIndex** ()
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_ucControlIndex](#)
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

13.31.1 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 521 of file [gui.h](#).

13.31.2 Member Function Documentation

13.31.2.1 void GuiControl::Activate (bool *bActivate_*) [pure virtual]

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

Parameters

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

Implemented in [StubControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [GroupBoxControl](#), [ProgressControl](#), and [CheckBoxControl](#).

13.31.2.2 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 724 of file [gui.h](#).

13.31.2.3 void GuiControl::Draw () [pure virtual]

Redraw the control "cleanly".

Subclass specific.

Implemented in [StubControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [GroupBoxControl](#), [ProgressControl](#), and [CheckBoxControl](#).

13.31.2.4 K_UCHAR GuiControl::GetControlIndex () [inline]

Return the Control Index of the control.

Returns

The control index of the control

Definition at line 631 of file [gui.h](#).

13.31.2.5 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 543 of file [gui.cpp](#).

13.31.2.6 K_USHORT GuiControl::GetHeight () [inline]

Get the height of the control in pixels.

Returns

Height of the control in pixels

Definition at line 610 of file [gui.h](#).

13.31.2.7 K_USHORT GuiControl::GetLeft () [inline]

Return the leftmost pixel of the control.

Returns

Leftmost pixel of the control

Definition at line 603 of file [gui.h](#).

13.31.2.8 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 708 of file [gui.h](#).

13.31.2.9 GuiWindow * GuiControl::GetParentWindow () [inline], [protected]

Get the parent window of this control.

Returns

Pointer to the control's window

Definition at line 716 of file [gui.h](#).

13.31.2.10 K_USHORT GuiControl::GetTop () [inline]

Return the topmost pixel of the control.

Returns

Topmost pixel of the control

Definition at line 596 of file [gui.h](#).

13.31.2.11 `K_USHORT GuiControl::GetWidth () [inline]`

Get the width of the control in pixels.

Returns

Width of the control in pixels

Definition at line 617 of file [gui.h](#).

13.31.2.12 `K_UCHAR GuiControl::GetZOrder () [inline]`

Return the Z-order of the control.

Returns

Z-order of the control

Definition at line 624 of file [gui.h](#).

13.31.2.13 `void GuiControl::Init () [pure virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implemented in [StubControl](#), [ButtonControl](#), [PanelControl](#), [LabelControl](#), [ProgressControl](#), [CheckBoxControl](#), and [GroupBoxControl](#).

13.31.2.14 `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 660 of file [gui.h](#).

13.31.2.15 `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 638 of file [gui.h](#).

13.31.2.16 `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](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [GroupBoxControl](#), [ProgressControl](#), and [CheckBoxControl](#).

13.31.2.17 void `GuiControl::SetControlIndex (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 [589](#) of file [gui.h](#).

13.31.2.18 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 [567](#) of file [gui.h](#).

13.31.2.19 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 [560](#) of file [gui.h](#).

13.31.2.20 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>pclParent_</i>	Pointer to the control's parent control
-------------------	---

Definition at line [689](#) of file [gui.h](#).

13.31.2.21 void `GuiControl::SetParentWindow (GuiWindow * pclWindow_)` [`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 700 of file [gui.h](#).

13.31.2.22 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 553 of file [gui.h](#).

13.31.2.23 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 574 of file [gui.h](#).

13.31.2.24 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 581 of file [gui.h](#).

13.31.3 Member Data Documentation

13.31.3.1 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 753 of file [gui.h](#).

13.31.3.2 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 749 of file [gui.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp](#)

13.32 GuiEvent_t Struct Reference

Public Attributes

- K_UCHAR **ucEventType**
- K_UCHAR **ucTargetID**
- union {
 - [KeyEvent_t](#) **stKey**
 - [MouseEvent_t](#) **stMouse**
 - [TouchEvent_t](#) **stTouch**
 - [JoystickEvent_t](#) **stJoystick**
 - [TimerEvent_t](#) **stTimer**
- };

13.32.1 Detailed Description

Definition at line 183 of file [gui.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.33 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.*

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.

13.33.1 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 441 of file [gui.h](#).

13.33.2 Member Function Documentation

13.33.2.1 `void GuiEventSurface::AddWindow (GuiWindow * pclWindow_)`

Add a window to the event surface.

Parameters

<i>pclWindow_</i>	Pointer to the window object to add to the sruface
-------------------	--

Definition at line 409 of file [gui.cpp](#).

13.33.2.2 `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 530 of file [gui.cpp](#).

13.33.2.3 `void GuiEventSurface::Init ()` `[inline]`

Initialize an event surface before use.

Must be called prior to any other object methods.

Definition at line 448 of file [gui.h](#).

13.33.2.4 `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 461 of file [gui.cpp](#).

13.33.2.5 void GuiEventSurface::RemoveWindow (GuiWindow * *pclWindow_*)

Remove a window from the event surface.

Parameters

<i>pclWindow_</i>	Pointer to the window object to remove from the surface
-------------------	---

Definition at line 417 of file [gui.cpp](#).

13.33.2.6 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

<i>pstEvent_</i>	Pointer to an event to send
------------------	-----------------------------

Returns

true on success, false on failure

Definition at line 425 of file [gui.cpp](#).

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

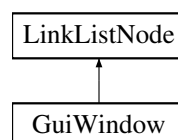
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp](#)

13.34 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.

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

13.34.1 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 219 of file [gui.h](#).

13.34.2 Member Function Documentation

13.34.2.1 `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 28 of file [gui.cpp](#).

13.34.2.2 `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 279 of file [gui.cpp](#).

13.34.2.3 `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 248 of file [gui.h](#).

13.34.2.4 K_USHORT GuiWindow::GetHeight () [inline]

Get the height of the window in pixels.

Returns

Height of the window in pixels

Definition at line 375 of file [gui.h](#).

13.34.2.5 K_USHORT GuiWindow::GetLeft () [inline]

Return the leftmost pixel of the window.

Returns

Leftmost pixel of the window

Definition at line 368 of file [gui.h](#).

13.34.2.6 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 62 of file [gui.cpp](#).

13.34.2.7 K_USHORT GuiWindow::GetTop () [inline]

Return the topmost pixel of the window.

Returns

Topmost pixel of the window

Definition at line 361 of file [gui.h](#).

13.34.2.8 K_USHORT GuiWindow::GetWidth () [inline]

Get the width of the window in pixels.

Returns

Width of the window in pixels

Definition at line 382 of file [gui.h](#).

13.34.2.9 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 227 of file [gui.h](#).

13.34.2.10 K_BOOL GuiWindow::IsInFocus (GuiControl * *pclControl_*) [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 319 of file [gui.h](#).

13.34.2.11 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 128 of file [gui.cpp](#).

13.34.2.12 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 86 of file [gui.cpp](#).

13.34.2.13 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 41 of file [gui.cpp](#).

13.34.2.14 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 240 of file [gui.h](#).

13.34.2.15 void GuiWindow::SetFocus (*GuiControl* * *pclControl_*)

Set the control used to accept "focus" events.

Such events include keyboard events.

Parameters

<i>pclControl_</i>	Pointer to the control object to set focus on.
--------------------	--

Definition at line 271 of file [gui.cpp](#).

13.34.2.16 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 347 of file [gui.h](#).

13.34.2.17 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 340 of file [gui.h](#).

13.34.2.18 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 333 of file [gui.h](#).

13.34.2.19 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 354 of file [gui.h](#).

13.34.3 Member Data Documentation

13.34.3.1 GraphicsDriver* GuiWindow::m_pclDriver [private]

Graphics driver for this window.

Definition at line 425 of file [gui.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.cpp](#)

13.35 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**

13.35.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h](#)

13.36 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.

```
};
```

13.36.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.37 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.

13.37.1 Detailed Description

Class that encapsulates all of the kernel startup functions.

Definition at line 40 of file [kernel.h](#).

13.37.2 Member Function Documentation

13.37.2.1 `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 43 of file [kernel.cpp](#).

13.37.2.2 `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 68 of file [kernel.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.cpp](#)

13.38 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.*

13.38.1 Detailed Description

Class providing the software-interrupt required for context-switching in the kernel.

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

13.38.2 Member Function Documentation

13.38.2.1 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](#).

13.38.2.2 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.cpp](#)

13.39 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_)
Retstore 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 ullInterval_)
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 ullInterval_)
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.

13.39.1 Detailed Description

Hardware timer interface, used by all scheduling/timer subsystems.

Definition at line 33 of file [kerneltimer.h](#).

13.39.2 Member Function Documentation

13.39.2.1 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 94 of file [kerneltimer.cpp](#).

13.39.2.2 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 57 of file [kerneltimer.cpp](#).

13.39.2.3 void KernelTimer::RI (K_UCHAR *bEnable_*) [static]

Restore the state of the kernel timer's expiry interrupt.

Parameters

<i>bEnable_</i>	1 enable, 0 disable
-----------------	---------------------

Definition at line 137 of file [kerneltimer.cpp](#).

13.39.2.4 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 100 of file [kerneltimer.cpp](#).

13.39.2.5 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 71 of file [kerneltimer.cpp](#).

13.39.2.6 K_ULONG KernelTimer::TimeToExpiry (void) [static]

Returns the number of ticks remaining before the next timer expiry.

Parameters

<i>Time</i>	before next expiry in platform-specific ticks
-------------	---

Definition at line 78 of file [kerneltimer.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltimer.cpp](#)

13.40 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 is 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.

13.40.1 Detailed Description

Keyboard UI event structure definition.

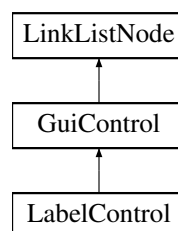
Definition at line 80 of file [gui.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.41 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_)
- 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 **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

13.41.1 Detailed Description

Definition at line 30 of file [control_label.h](#).

13.41.2 Member Function Documentation

13.41.2.1 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](#).

13.41.2.2 void LabelControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 26 of file [control_label.cpp](#).

13.41.2.3 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](#).

13.41.2.4 `virtual GuiReturn_t LabelControl::ProcessEvent (GuiEvent_t * pstEvent_) [inline],[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 39 of file [control_label.h](#).

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

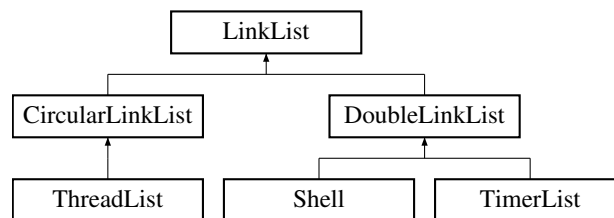
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.cpp](#)

13.42 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`
- `LinkListNode * m_pstTail`

13.42.1 Detailed Description

Abstract-data-type from which all other linked-lists are derived.

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

13.42.2 Member Function Documentation

13.42.2.1 void LinkedList::Add (LinkedListNode * 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](#).

13.42.2.2 LinkedListNode * LinkedList::GetHead () [inline]

Get the head node in the linked list.

Returns

Pointer to the head node in the list

Definition at line 154 of file [ll.h](#).

13.42.2.3 LinkedListNode * LinkedList::GetTail () [inline]

Get the tail node of the linked list.

Returns

Pointer to the tail node in the list

Definition at line 163 of file [ll.h](#).

13.42.2.4 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:

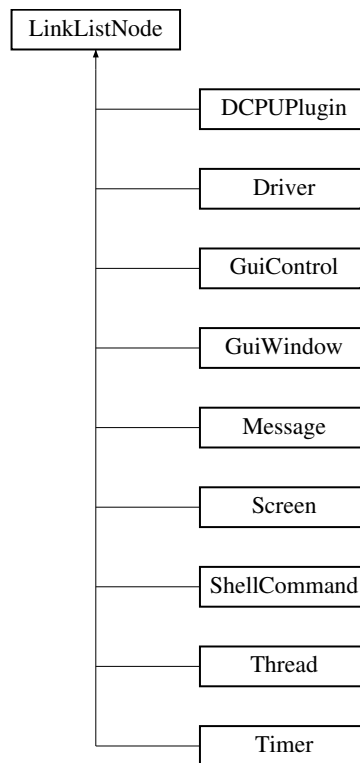
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h](#)

13.43 LinkedListNode Class Reference

Basic linked-list node data structure.

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

Inheritance diagram for LinkListNode:



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](#)
- [LinkListNode * prev](#)

Friends

- class **LinkList**
- class **DoubleLinkList**
- class **CircularLinkList**

13.43.1 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 75 of file [ll.h](#).

13.43.2 Member Function Documentation

13.43.2.1 `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 97 of file [ll.h](#).

13.43.2.2 `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 106 of file [ll.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp](#)

13.44 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 [DecimalToString](#) (K_UCHAR ucData_, char *szText_)
Convert an 8-bit unsigned binary value as a decimal string.
- 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.

13.44.1 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 37 of file [memutil.h](#).

13.44.2 Member Function Documentation

13.44.2.1 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 108 of file [memutil.cpp](#).

13.44.2.2 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 92 of file [memutil.cpp](#).

13.44.2.3 `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 235 of file [memutil.cpp](#).

13.44.2.4 `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 140 of file [memutil.cpp](#).

13.44.2.5 `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 166 of file [memutil.cpp](#).

13.44.2.6 `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 183 of file [memutil.cpp](#).

13.44.2.7 `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](#).

13.44.2.8 `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 61 of file [memutil.cpp](#).

13.44.2.9 `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 256 of file [memutil.cpp](#).

13.44.2.10 `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 125 of file [memutil.cpp](#).

13.44.2.11 `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 200 of file [memutil.cpp](#).

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

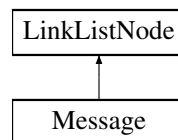
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.cpp](#)

13.45 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

13.45.1 Detailed Description

Class to provide message-based IPC services in the kernel.

Definition at line 99 of file [message.h](#).

13.45.2 Member Function Documentation

13.45.2.1 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](#).

13.45.2.2 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](#).

13.45.2.3 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](#).

13.45.2.4 void Message::SetData (void * *pvData_*) [inline]

Set the data pointer for the message before transmission.

Parameters

<i>pvData_</i>	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h](#)

13.46 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.

13.46.1 Detailed Description

List of messages, used as the channel for sending and receiving messages between threads.

Definition at line 201 of file [message.h](#).

13.46.2 Member Function Documentation

13.46.2.1 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 147 of file [message.cpp](#).

13.46.2.2 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 88 of file [message.cpp](#).

13.46.2.3 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

<code>uWaitTimeMS_</code>	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 108 of file [message.cpp](#).

13.46.2.4 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

<code>pclSrc_</code>	Pointer to the message object to add to the queue
----------------------	---

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

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.cpp](#)

13.47 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)
 }
};

13.47.1 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:

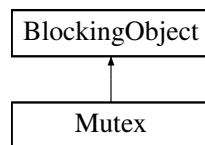
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.48 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 [WakeMe](#) ([Thread](#) *pclOwner_)
Wake a thread blocked on the mutex.
- void [SetExpired](#) (bool bExpired_)
- void [Release](#) ()
Release the mutex.

Private Member Functions

- K_UCHAR [WakeNext](#) ()
Wake the next thread waiting on the [Mutex](#).

Private Attributes

- K_UCHAR **m_bReady**
- K_UCHAR **m_ucMaxPri**
- [Thread](#) * **m_pclOwner**
- bool **m_bExpired**

Additional Inherited Members

13.48.1 Detailed Description

Mutual-exclusion locks, based on [BlockingObject](#).

Definition at line 68 of file [mutex.h](#).

13.48.2 Member Function Documentation

13.48.2.1 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 96 of file [mutex.cpp](#).

13.48.2.2 bool Mutex::Claim (K_ULONG ulWaitTimeMS_)

Parameters

<i>ulWaitTimeMS_</i>	
----------------------	--

Returns

true - mutex was claimed within the time period specified
false - mutex operation timed-out before the claim operation.

Definition at line 100 of file [mutex.cpp](#).

13.48.2.3 void Mutex::Release ()

Release the mutex.

When the mutex is released, another object can enter the mutex-protected region.

Definition at line 190 of file [mutex.cpp](#).

13.48.2.4 void Mutex::WakeMe (Thread * pOwner_)

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>pOwner_</i>	Thread to unblock from this object.
----------------	---

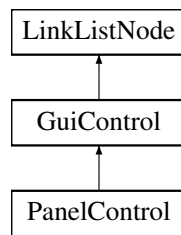
Definition at line 55 of file [mutex.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp](#)

13.49 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

13.49.1 Detailed Description

Definition at line 33 of file [control_panel.h](#).

13.49.2 Member Function Documentation

13.49.2.1 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](#).

13.49.2.2 void [PanelControl::Draw](#) () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 26 of file [control_panel.cpp](#).

13.49.2.3 `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](#).

13.49.2.4 `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/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.cpp](#)

13.50 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**

13.50.1 Detailed Description

System profiling timer interface.

Definition at line 37 of file [kprofile.h](#).

13.50.2 Member Function Documentation

13.50.2.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp](#)

13.51 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.

13.51.1 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](#).

13.51.2 Member Function Documentation

13.51.2.1 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](#).

13.51.2.2 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](#).

13.51.2.3 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](#).

13.51.2.4 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](#).

13.51.2.5 void ProfileTimer::Start (void)

Start a profiling session, if the timer is not already active.

Has no effect if the timer is already active.

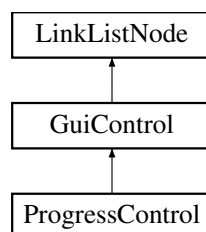
Definition at line 46 of file [profile.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.cpp](#)

13.52 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

13.52.1 Detailed Description

Definition at line 30 of file [control_progress.h](#).

13.52.2 Member Function Documentation

13.52.2.1 `virtual void ProgressControl::Activate (bool bActivate_) [inline], [virtual]`

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

Parameters

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

Implements [GuiControl](#).

Definition at line 36 of file [control_progress.h](#).

13.52.2.2 `void ProgressControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

13.52.2.3 `void ProgressControl::Init () [virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 27 of file [control_progress.cpp](#).

13.52.2.4 `GuiReturn_t ProgressControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

<code><i>pstEvent_</i></code>	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_progress.cpp](#)

13.53 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](#)

13.53.1 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](#).

13.53.2 Member Function Documentation

13.53.2.1 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](#).

13.53.2.2 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](#).

13.53.2.3 void Quantum::SetTimer (Thread * *pcThread_*) [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>pcThread_</i>	Pointer to the thread to set the Quantum timer on
------------------	---

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

13.53.2.4 void Quantum::UpdateTime (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/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp](#)

13.54 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 **pcThread_*)
Add a thread to the scheduler at its current priority level.
- static void [Remove](#) (Thread **pcThread_*)
Remove a thread from the scheduler at its current priority level.
- static void [SetScheduler](#) (K_UCHAR *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 Private Attributes

- static K_UCHAR [m_bEnabled](#)
Scheduler's state - enabled or disabled.
- 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.

13.54.1 Detailed Description

Priority-based round-robin [Thread](#) scheduling, using ThreadLists for housekeeping.

Definition at line 61 of file [scheduler.h](#).

13.54.2 Member Function Documentation

13.54.2.1 `void Scheduler::Add (Thread * pclThread_) [static]`

Add a thread to the scheduler at its current priority level.

Parameters

pclThread_	Pointer to the thread to add to the scheduler
----------------------------	---

Definition at line 77 of file [scheduler.cpp](#).

13.54.2.2 `static Thread* Scheduler::GetCurrentThread () [inline],[static]`

Return the pointer to the currently-running thread.

Returns

Pointer to the currently-running thread

Definition at line 118 of file [scheduler.h](#).

13.54.2.3 `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 126 of file [scheduler.h](#).

13.54.2.4 `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 144 of file [scheduler.h](#).

13.54.2.5 `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 136 of file [scheduler.h](#).

13.54.2.6 `K_UCHAR Scheduler::IsEnabled () [inline],[static]`

Return the current state of the scheduler - whether or not scheudling is enabled or disabled.

Returns

true - scheduler enabled, false - disabled

Definition at line 154 of file [scheduler.h](#).

13.54.2.7 `void Scheduler::Remove (Thread * pclThread_) [static]`

Remove a thread from the scheduler at its current priority level.

Parameters

<i>pclThread_</i>	Pointer to the thread to be removed from the scheduler
-------------------	--

Definition at line 84 of file [scheduler.cpp](#).

13.54.2.8 `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 60 of file [scheduler.cpp](#).

13.54.2.9 `void Scheduler::SetScheduler (K_UCHAR bEnable_) [inline],[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
-----------------------	--

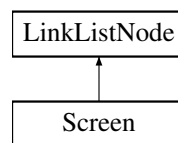
Definition at line 111 of file [scheduler.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.cpp](#)

13.55 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**

13.55.1 Detailed Description

Definition at line 31 of file [screen.h](#).

13.55.2 Member Function Documentation

13.55.2.1 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](#).

13.55.2.2 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp](#)

13.56 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.

13.56.1 Detailed Description

Definition at line 84 of file [screen.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.h](#)

13.57 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.

13.57.1 Detailed Description

Definition at line 109 of file [screen.h](#).

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

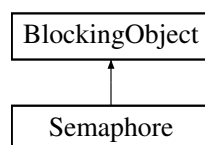
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[screen.h](#)
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[screen.cpp](#)

13.58 Semaphore Class Reference

Counting semaphore, based on [BlockingObject](#) base class.

```
#include <semaphore.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 [WakeMe](#) (Thread *pclChosenOne_)
- Wake a thread blocked on the semaphore.*
- void [SetExpired](#) (bool bExpired_)
- Set the semaphore expired flag on this object.*
- bool [GetExpired](#) ()

Private Member Functions

- K_UCHAR [WakeNext](#) ()
- Wake the next thread waiting on the semaphore.*

Private Attributes

- K_USHORT **m_usValue**
- K_USHORT **m_usMaxValue**
- bool **m_bExpired**

Additional Inherited Members

13.58.1 Detailed Description

Counting semaphore, based on [BlockingObject](#) base class.

Definition at line 37 of file [semaphore.h](#).

13.58.2 Member Function Documentation

13.58.2.1 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 213 of file [semaphore.cpp](#).

13.58.2.2 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 84 of file [semaphore.cpp](#).

13.58.2.3 void Semaphore::Pend ()

Decrement the semaphore count.

If the count is zero, the thread will block until the semaphore is pended.

Definition at line 143 of file [semaphore.cpp](#).

13.58.2.4 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 148 of file [semaphore.cpp](#).

13.58.2.5 void Semaphore::SetExpired (bool bExpired_) [inline]

Set the semaphore expired flag on this object.

\

Definition at line 112 of file [semaphore.h](#).

13.58.2.6 void Semaphore::WakeMe (Thread * pChosenOne_)

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.

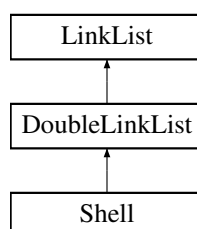
Definition at line 57 of file [semaphore.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp](#)

13.59 Shell Class Reference

Inheritance diagram for Shell:



Public Member Functions

- void **AddCommand** ([ShellCommand](#) *pclCommand_)
Add a new command to the list of commands supported in the shell.
- void **RunCommand** (const K_CHAR *pcCommand_)
Parse the command string and run the associated command.

Private Attributes

- int **m_argc**
- void * **m_argv** [MAX_ARGV]

Additional Inherited Members

13.59.1 Detailed Description

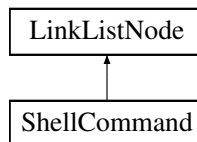
Definition at line 32 of file [shell_command.h](#).

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

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.h

13.60 ShellCommand Class Reference

Inheritance diagram for ShellCommand:



Public Member Functions

- **ShellCommand** (const K_CHAR *pcCommand_, ShellHandler *pfHandler_)
- void **SetCommand** (const K_CHAR *pcCommand_)
- void **SetHandler** (ShellHandler *pfHandler_)
- const K_CHAR * **GetCommand** ()
- int **RunHandler** (int argc_, void *argv_)

Private Attributes

- const K_CHAR * **m_pcCommand**
- ShellHandler **m_pfHandler**

Additional Inherited Members

13.60.1 Detailed Description

Definition at line 12 of file [shell_command.h](#).

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

- `/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/shell_command.h`

13.61 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_, 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_, 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](#)

13.61.1 Detailed Description

Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP).

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

13.61.2 Member Function Documentation

13.61.2.1 K_USHORT Slip::DecodeByte (K_UCHAR * *ucChar_*, 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](#).

13.61.2.2 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](#).

13.61.2.3 Driver* Slip::GetDriver () [inline]

Return the pointer to the driver attached to this object.

Returns

Pointer to the driver attached

Definition at line 81 of file [slip.h](#).

13.61.2.4 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](#).

13.61.2.5 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 74 of file [slip.h](#).

13.61.2.6 void Slip::WriteData (K_UCHAR *ucChannel_*, 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](#).

13.61.2.7 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp](#)

13.62 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.

13.62.1 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 55 of file [slip.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.h](#)

13.63 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](#) (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](#)

13.63.1 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](#).

13.63.2 Member Function Documentation

13.63.2.1 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](#).

13.63.2.2 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](#).

13.63.2.3 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](#).

13.63.2.4 void SlipMux::Init (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](#).

13.63.2.5 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>pfHanler_</i>	Pointer to the handler function to attach

Definition at line 76 of file [slip_mux.cpp](#).

13.63.2.6 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](#).

13.63.2.7 static void SlipMux::SetQueue (MessageQueue * pciMessageQueue_) [inline],[static]

Set the message queue that will receive the notification when the slip mux channel has received data.

Parameters

<i>pciMessageQueue_</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/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.cpp](#)

13.64 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](#)
- [Slip](#) [m_clSlip](#)
Slip object that this module interfaces with.

13.64.1 Detailed Description

Class implementing a simple debug terminal interface.

This is useful for printf style debugging.

Definition at line 40 of file [slipterm.h](#).

13.64.2 Member Function Documentation

13.64.2.1 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](#).

13.64.2.2 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](#).

13.64.2.3 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](#).

13.64.2.4 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](#).

13.64.2.5 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](#).

13.64.3 Member Data Documentation

13.64.3.1 K_UCHAR SlipTerm::m_ucVerbosity [private]

Verbosity level. Messages with a severity level greater than this Are not displayed.

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

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

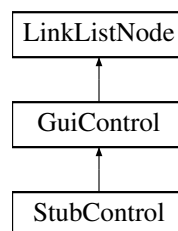
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.cpp](#)

13.65 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](#) ()
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.

Additional Inherited Members

13.65.1 Detailed Description

Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Definition at line 779 of file [gui.h](#).

13.65.2 Member Function Documentation

13.65.2.1 `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 785 of file [gui.h](#).

13.65.2.2 `virtual void StubControl::Draw () [inline], [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 783 of file [gui.h](#).

13.65.2.3 `virtual void StubControl::Init () [inline], [virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 782 of file [gui.h](#).

13.65.2.4 `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 784 of file [gui.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.66 SystemHeap Class Reference

Static Public Member Functions

- static void **Init** (void)
- static void * **Alloc** (K_USHORT usSize_)
- static void **Free** (void *pvData_)

Static Private Attributes

- static K_UCHAR **m_pucRawHeap** [HEAP_RAW_SIZE]
- static [HeapConfig](#) **m_pciSystemHeapConfig** [[HEAP_NUM_SIZES](#)+1]
- static [FixedHeap](#) **m_clSystemHeap**
- static bool **m_blnit**

13.66.1 Detailed Description

Definition at line 185 of file [system_heap.h](#).

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

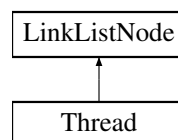
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[system_heap.h](#)
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[system_heap.cpp](#)

13.67 Thread Class Reference

Object providing fundamental multitasking support in the kernel.

```
#include <thread.h>
```

Inheritance diagram for Thread:



Public Member Functions

- void [Init](#) (K_UCHAR *paucStack_, K_USHORT usStackSize_, K_UCHAR ucPriority_, [ThreadEntry_t](#) pfEntry-Point_, 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 [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.
- [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)
- 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.

Static Public Member Functions

- static void [Sleep](#) (K_ULONG ulTimeMs_)
- Put the thread to sleep for the specified time (in milliseconds).

 - 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_UCHAR * [m_paucStackTop](#)
- Pointer to the top of the thread's stack.
- K_UCHAR * [m_paucStack](#)
- 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 entryptoint.
- const K_CHAR * [m_szName](#)
Thread name.
- [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

13.67.1 Detailed Description

Object providing fundamental multitasking support in the kernel.

Definition at line 64 of file [thread.h](#).

13.67.2 Member Function Documentation

13.67.2.1 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 277 of file [thread.cpp](#).

13.67.2.2 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 116 of file [thread.cpp](#).

13.67.2.3 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 131 of file [thread.h](#).

13.67.2.4 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 268 of file [thread.h](#).

13.67.2.5 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 122 of file [thread.h](#).

13.67.2.6 K_UCHAR Thread::GetPriority (void) [inline]

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 141 of file [thread.h](#).

13.67.2.7 K_USHORT Thread::GetQuantum (void) [inline]

Get the thread's round-robin execution quantum.

Returns

The thread's quantum

Definition at line 169 of file [thread.h](#).

13.67.2.8 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 178 of file [thread.cpp](#).

13.67.2.9 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 270 of file [thread.cpp](#).

13.67.2.10 void Thread::Init (K_UCHAR * *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 39 of file [thread.cpp](#).

13.67.2.11 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 179 of file [thread.h](#).

13.67.2.12 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 259 of file [thread.h](#).

13.67.2.13 `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 111 of file [thread.h](#).

13.67.2.14 `void Thread::SetOwner (ThreadList * pcNewList_) [inline]`

Set the thread's owner to the specified thread list.

Parameters

<i>pcNewList_</i>	Pointer to the threadlist to apply thread ownership
-------------------	---

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

13.67.2.15 `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 233 of file [thread.cpp](#).

13.67.2.16 `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 160 of file [thread.h](#).

13.67.2.17 `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

<code>ulTimeMs_</code>	Time to sleep (in ms)
------------------------	-----------------------

Definition at line 156 of file [thread.cpp](#).

13.67.2.18 `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 199 of file [thread.cpp](#).

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

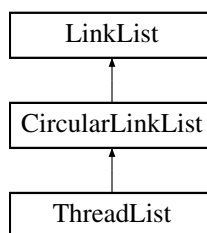
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.cpp](#)

13.68 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

13.68.1 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](#).

13.68.2 Member Function Documentation

13.68.2.1 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](#).

13.68.2.2 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](#).

13.68.2.3 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](#).

13.68.2.4 void ThreadList::Remove (LinkListNode * *node_*) [virtual]

Remove the specified thread from the threadlist.

Parameters

<code>node_</code>	Pointer to the thread to remove
--------------------	---------------------------------

Reimplemented from [CircularLinkedList](#).

Definition at line 71 of file [threadlist.cpp](#).

13.68.2.5 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

<code>pucFlag_</code>	Pointer to the bitmap flag
-----------------------	----------------------------

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

13.68.2.6 void ThreadList::SetPriority (K_UCHAR *ucPriority_*)

Set the priority of this threadlist (if used for a scheduler).

Parameters

<code>ucPriority_</code>	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp](#)

13.69 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](#)

13.69.1 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](#).

13.69.2 Member Function Documentation

13.69.2.1 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:

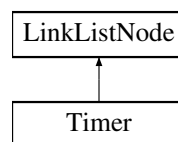
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp](#)

13.70 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 [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 [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 callback 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...

- 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)

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.

- Thread * [m_pclOwner](#)

Pointer to the owner thread.

- void * [m_pvData](#)

Pointer to the callback data.

Friends

- class **TimerList**

Additional Inherited Members

13.70.1 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 78 of file [timerlist.h](#).

13.70.2 Member Function Documentation

13.70.2.1 void Timer::SetCallback (TimerCallback_t pfCallback_) [inline]

Define the callback function to be executed on expiry of the timer.

Parameters

pfCallback_	Pointer to the callback function to call
-----------------------------	--

Definition at line 114 of file [timerlist.h](#).

13.70.2.2 void Timer::SetData (void * *pvData_*) [inline]

Define a pointer to be sent to the timer callback on timer expiry.

Parameters

<i>pvData_</i>	Pointer to data to pass as argument into the callback
----------------	---

Definition at line 123 of file [timerlist.h](#).

13.70.2.3 void Timer::SetFlags (K_UCHAR *ucFlags_*) [inline]

Set the timer's flags based on the bits in the *ucFlags_* argument.

Parameters

<i>Flags</i>	to assign to the timer object
--------------	-------------------------------

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

13.70.2.4 void Timer::SetIntervalMSeconds (K_ULONG *ulMSeconds_*)

Set the timer expiry interval in milliseconds (platform agnostic)

Parameters

<i>ulMSeconds_</i>	Time in milliseconds
--------------------	----------------------

Definition at line 270 of file [timerlist.cpp](#).

13.70.2.5 void Timer::SetIntervalSeconds (K_ULONG *ulSeconds_*)

! The next three cost us 330 bytes of flash on AVR...

Set the timer expiry interval in seconds (platform agnostic)

Parameters

<i>ulSeconds_</i>	Time in seconds
-------------------	-----------------

Definition at line 264 of file [timerlist.cpp](#).

13.70.2.6 void Timer::SetIntervalTicks (K_ULONG *ulTicks_*)

Set the timer expiry in system-ticks (platform specific!)

Parameters

<i>ulTicks_</i>	Time in ticks
-----------------	---------------

Definition at line 256 of file [timerlist.cpp](#).

13.70.2.7 void Timer::SetIntervalUSeconds (K_ULONG *ulUSeconds_*)

Set the timer expiry interval in microseconds (platform agnostic)

Parameters

<code>ulUSeconds_</code>	Time in microseconds
--------------------------	----------------------

Definition at line 276 of file [timerlist.cpp](#).

13.70.2.8 void Timer::SetOwner (Thread * *pclOwner_*) [inline]

Set the owner-thread of this timer object (all timers must be owned by a thread).

Parameters

<code>pclOwner_</code>	Owner thread of this timer object
------------------------	-----------------------------------

Definition at line 133 of file [timerlist.h](#).

13.70.2.9 void Timer::Stop (void)

Stop a timer already in progress.

Has no effect on timers that have already been stopped.

Definition at line 250 of file [timerlist.cpp](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp](#)

13.71 TimerEvent_t Struct Reference

[Timer](#) UI event structure.

```
#include <gui.h>
```

Public Attributes

- K_USHORT [usTicks](#)
Number of clock ticks (arbitrary) that have elapsed.

13.71.1 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:

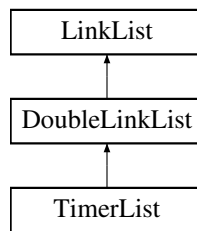
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.72 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

13.72.1 Detailed Description

[TimerList](#) class - a doubly-linked-list of timer objects.

Definition at line 198 of file [timerlist.h](#).

13.72.2 Member Function Documentation

13.72.2.1 void TimerList::Add ([Timer](#) * *pclListNode_*)

Add a timer to the [TimerList](#).

Parameters

<i>pclListNode_</i>	Pointer to the Timer to Add
---------------------	---

Definition at line 55 of file [timerlist.cpp](#).

13.72.2.2 void TimerList::Init (void)

Initialize the [TimerList](#) object.

Must be called before using the object.

Reimplemented from [LinkList](#).

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

13.72.2.3 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 110 of file [timerlist.cpp](#).

13.72.2.4 void TimerList::Remove (Timer * pclListNode_)

Remove a timer from the [TimerList](#), cancelling its expiry.

Parameters

pclListNode_	Pointer to the Timer to remove
------------------------------	--

Definition at line 95 of file [timerlist.cpp](#).

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

- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[timerlist.h](#)
- /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/[timerlist.cpp](#)

13.73 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](#) *pclListNode_)
Add a timer to the timer scheduler.
- static void [Remove](#) ([Timer](#) *pclListNode_)
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_clTimerList](#)
[TimerList](#) object manipulated by the [Timer Scheduler](#).

13.73.1 Detailed Description

"Static" Class used to interface a global [TimerList](#) with the rest of the kernel.

Definition at line 248 of file [timerlist.h](#).

13.73.2 Member Function Documentation

13.73.2.1 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 267 of file [timerlist.h](#).

13.73.2.2 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 257 of file [timerlist.h](#).

13.73.2.3 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 289 of file [timerlist.h](#).

13.73.2.4 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 278 of file [timerlist.h](#).

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

- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.cpp](#)

13.74 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.
 }
};

13.74.1 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.h](#)

13.75 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 [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.

13.75.1 Detailed Description

Class used to implement a simple unit-testing framework.

Definition at line 28 of file [unit_test.h](#).

13.75.2 Member Function Documentation

13.75.2.1 `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 72 of file [unit_test.h](#).

13.75.2.2 `K_USHORT UnitTest::GetFailed () [inline]`

Return the number of failed test points/iterations.

Returns

Failed test point/iteration count

Definition at line 108 of file [unit_test.h](#).

13.75.2.3 `const K_CHAR * UnitTest::GetName () [inline]`

Get the name of the tests associated with this object.

Returns

Name of the test

Definition at line 81 of file [unit_test.h](#).

13.75.2.4 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 99 of file [unit_test.h](#).

13.75.2.5 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 90 of file [unit_test.h](#).

13.75.2.6 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 117 of file [unit_test.h](#).

13.75.2.7 void UnitTest::SetName (const K_CHAR * szName_) [inline]

Set the name of the test object.

Parameters

szName_	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/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp](#)

13.76 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.*

13.76.1 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](#).

13.76.2 Member Function Documentation

13.76.2.1 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](#).

13.76.2.2 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](#).

13.76.2.3 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](#).

13.76.2.4 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.h](#)
- [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.cpp](#)

File Documentation

Implementation of base class for blocking objects.

Macros

- ### 14.1.1 Detailed Description

Definition in file [blocking.cpp](#).

[illegible]

```

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     Scheduler::Remove(pclThread_);
00044     m_clBlockList.Add(pclThread_);
00045
00046     // Set the "current" list location to the blocklist for this thread
00047     pclThread_>SetCurrent(&m_clBlockList);
00048 }
00049 }
00050
00051 //-----
00052 void BlockingObject::UnBlock(Thread *pclThread_)
00053 {
00054     KERNEL_ASSERT( pclThread_ );
00055     KERNEL_TRACE_1( STR_THREAD_UNBLOCK_1, (K_USHORT)pclThread_>GetID() );
00056
00057     // Remove the thread from its current thread list (the "owner" list)
00058     pclThread_>GetCurrent()->Remove(pclThread_);
00059
00060     // Put the thread back in its active owner's list. This is usually
00061     // the ready-queue at the thread's original priority.
00062     Scheduler::Add(pclThread_);
00063
00064     // Tag the thread's current list location to its owner
00065     pclThread_>SetCurrent(pclThread_>GetOwner());
00066 }
00067
00068 #endif

```

14.3 /home/moslevin/googlecode/mark3/trunk/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"

```

Classes

- class [BlockingObject](#)

Class implementing thread-blocking primitives.

14.3.1 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.

Definition in file [control_button.cpp](#).

14.6 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( GuiEvent_t
    *pstEvent_ )
00118 {
00119     K_USHORT usXOffset, usYOffset;
00120
00121     GetControlOffset(&usXOffset, &usYOffset);
00122
00123     GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");
00124
00125     switch (pstEvent_->ucEventType)
00126     {
00127     case EVENT_TYPE_KEYBOARD:
00128     {
00129         // If this is a space bar or an enter key, behave like a mouse
00130         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             {
00140                 m_bState = false;
00141                 if (m_pfCallback)
00142                 {
00143                     m_pfCallback(m_pvCallbackData);
00144                 }
00145                 SetState();
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
00155             coordinates.
00156             // If so, de-activate the control
00157             if (pstEvent_->stMouse.bLeftState)
00158             {
00159                 if ((pstEvent_->stMouse.usX < GetLeft() +
00160                     usXOffset) ||
00161                     (pstEvent_->stMouse.usX >= GetLeft() +
00162                     usXOffset + GetWidth()-1) ||
00163                     (pstEvent_->stMouse.usY < GetTop() + usYOffset
00164                     ) ||
00165                     (pstEvent_->stMouse.usY >= GetTop() +
00166                     usYOffset + GetHeight() - 1))

```

```

00162         {
00163             m_bState = false;
00164             SetStale();
00165         }
00166     }
00167     // left button state is now up, and the control was previously
    active.
00168     // Run the event callback for the mouse, and go from there.
00169     else
00170     {
00171         if ((pstEvent_>stMouse.usX >= GetLeft() +
usXOffset) &&
00172             (pstEvent_>stMouse.usX < GetLeft() +
usXOffset + GetWidth()-1) &&
00173             (pstEvent_>stMouse.usY >= GetTop() +
usYOffset) &&
00174             (pstEvent_>stMouse.usY < GetTop() + usYOffset
+ GetHeight() - 1))
00175         {
00176             m_bState = false;
00177             SetStale();
00178             if (m_pfCallback)
00179             {
00180                 m_pfCallback(m_pvCallbackData);
00181             }
00182         }
00183     }
00184 }
00185 else if (!m_bState)
00186 {
00187     // If we registered a down-click in the bounding box, set the
    state of the
00188     // control to activated.
00189     if (pstEvent_>stMouse.bLeftState)
00190     {
00191         if ((pstEvent_>stMouse.usX >= GetLeft() +
usXOffset) &&
00192             (pstEvent_>stMouse.usX < GetLeft() +
usXOffset + GetWidth()-1) &&
00193             (pstEvent_>stMouse.usY >= GetTop() +
usYOffset) &&
00194             (pstEvent_>stMouse.usY < GetTop() + usYOffset
+ GetHeight() - 1))
00195         {
00196             m_bState = true;
00197             SetStale();
00198         }
00199     }
00200 }
00201 }
00202 if (!IsInFocus())
00203 {
00204     GetParentWindow()->SetFocus(this);
00205     SetStale();
00206 }
00207 }
00208 }
00209 break;
00210 }
00211 }
00212 }
00213 }
00214 //-----
00215 void ButtonControl::Activate( bool bActivate_ )
00216 {
00217     // When we de-activate the control, simply disarm the control and force
    // a redraw
00218     if (!bActivate_)
00219     {
00220         {
00221             m_bState = false;
00222         }
00223         SetStale();
00224     }

```

14.7 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_button.h File Reference

GUI Button Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class [ButtonControl](#)

Typedefs

- typedef void(* [ButtonCallback](#))(void *pvData_)

14.7.1 Detailed Description

GUI Button Control. Basic pushbutton control with an up/down state.

Definition in file [control_button.h](#).

14.8 control_button.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_BUTTON_H__
00023 #define __CONTROL_BUTTON_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 ButtonControl : public GuiControl
00033 {
00034 public:
00035
00036     virtual void Init();
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *
00039         pstEvent_ );
00039     virtual void Activate( bool bActivate_ );
00040
00041     void SetBGColor( COLOR eColor_ )      { m_uBGColor = eColor_; }
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

```

14.9 /home/moslevin/googlecode/mark3/trunk/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** []

14.9.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file [control_checkbox.cpp](#).

14.9.2 Variable Documentation

14.9.2.1 const K_UCHAR aucBox[] [static]

Initial value:

```

{ 0xFF,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0xFF }

```

Definition at line 31 of file [control_checkbox.cpp](#).

14.9.2.2 `const K_UCHAR aucCheck[]` `[static]`

Initial value:

```
{ 0,
  0,
  0x3C,
  0x3C,
  0x3C,
  0x3C,
  0,
  0 }
```

Definition at line 42 of file `control_checkbox.cpp`.

14.10 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  { 0xFF,
00033    0x81,
00034    0x81,
00035    0x81,
00036    0x81,
00037    0x81,
00038    0x81,
00039    0xFF };
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         stRect.uLineColor = m_uBackColor;
00071         stRect.uFillColor = m_uBackColor;
00072         stRect.usTop = usY + GetTop() + ((GetHeight() - 4)
>> 1);
00073         stRect.usLeft = usX + GetLeft();
00074         stRect.usRight = stRect.usLeft + 8;
00075         stRect.usBottom = stRect.usTop + 8;
00076         stRect.bFill = true;
00077         pclDriver->Rectangle(&stRect);
00078     }
00079
00080     {
00081         DrawStamp_t stStamp;
00082         stStamp.uColor = m_uBoxColor;
00083         stStamp.usY = usY + GetTop() + ((GetHeight() - 4) >>
1);
00084         stStamp.usX = usX + GetLeft();
00085         stStamp.usWidth = 8;
00086         stStamp.usHeight = 8;
00087         stStamp.pucData = (K_UCHAR*)aucBox;
00088         pclDriver->Stamp(&stStamp);
00089
00090         if (m_bChecked)
00091         {
00092             stStamp.pucData = (K_UCHAR*)aucCheck;
00093             pclDriver->Stamp(&stStamp);
00094         }
00095     }
00096
00097     // Draw the caption
00098     {
00099         DrawText_t stText;
00100         stText.usLeft = usX + GetLeft() + TEXT_X_OFFSET;
00101         stText.usTop = usY + GetTop();
00102         stText.uColor = m_uFontColor;
00103         stText.pstFont = m_pstFont;
00104         stText.pcString = m_szCaption;
00105
00106         usTextWidth = pclDriver->TextWidth(&stText);
00107         pclDriver->Text(&stText);
00108     }
00109 }
00110
00111 //-----
00112 GuiReturn_t CheckBoxControl::ProcessEvent(
GuiEvent_t *pstEvent_)
00113 {
00114     K_USHORT usXOffset, usYOffset;
00115
00116     GetControlOffset(&usXOffset, &usYOffset);
00117
00118     GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");
00119
00120     switch (pstEvent_>ucEventType)
00121     {
00122         case EVENT_TYPE_KEYBOARD:
00123         {
00124             // If this is a space bar or an enter key, behave like a mouse
click.
00125             if ((KEYCODE_SPACE == pstEvent_>stKey.ucKeyCode) ||
(KEYCODE_RETURN == pstEvent_>stKey.ucKeyCode))
00126             {
00127                 if (pstEvent_>stKey.bKeyState)
00128                 {
00129                     m_bChecked = true;
00130                 }
00131                 else
00132                 {
00133                     m_bChecked = false;
00134                 }
00135                 SetStale();
00136             }
00137         }
00138         break;
00139         case EVENT_TYPE_MOUSE:
00140         {
00141             // Is this control currently in the "active"/pressed state?
00142             if (m_bChecked)
00143             {
00144                 // Check to see if the movement is out-of-bounds based on the
coordinates.
00145                 // If so, de-activate the control
00146                 if (pstEvent_>stMouse.bLeftState)
00147                 {
00148                     if ((pstEvent_>stMouse.usX >= GetLeft() +
usXOffset) &&

```

```

00150             (pstEvent_>stMouse.usX < GetLeft() +
usXOffset + GetWidth()-1) &&
00151             (pstEvent_>stMouse.usY >= GetTop() +
usYOffset) &&
00152             (pstEvent_>stMouse.usY < GetTop() + usYOffset
+ GetHeight() - 1))
00153         {
00154             m_bChecked = false;
00155             SetStale();
00156         }
00157     }
00158 }
00159 else if (!m_bChecked)
00160 {
00161     // If we registered a down-click in the bounding box, set the
state of the
00162     // control to activated.
00163     if (pstEvent_>stMouse.bLeftState)
00164     {
00165         if ((pstEvent_>stMouse.usX >= GetLeft() +
usXOffset) &&
00166             (pstEvent_>stMouse.usX < GetLeft() +
usXOffset + GetWidth()-1) &&
00167             (pstEvent_>stMouse.usY >= GetTop() +
usYOffset) &&
00168             (pstEvent_>stMouse.usY < GetTop() + usYOffset
+ GetHeight() - 1))
00169         {
00170             m_bChecked = true;
00171             SetStale();
00172         }
00173     }
00174 }
00175
00176 if (!IsInFocus())
00177 {
00178     GetParentWindow()->SetFocus(this);
00179     SetStale();
00180 }
00181 }
00182 break;
00183 }
00184 }

```

14.11 /home/moslevin/googlecode/mark3/trunk/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](#)

14.11.1 Detailed Description

Checkbox Control. A binary On/Off switch control

Definition in file [control_checkbox.h](#).

14.12 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

```

14.13 /home/moslevin/googlecode/mark3/trunk/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)

14.13.1 Detailed Description

GUI GroupBox Control Implementation.

14.14 control_groupbox.cpp

Generated on Tue Feb 5 2013 20:29:32 for Mark3 Realtime Kernel by Doxygen


```

00031 public:
00032     virtual void Init() { m_uLineColor = COLOR_BLACK;
00033                           m_uFontColor = COLOR_GREY25;
00034                           m_uPanelColor = COLOR_GREY75;
00035                           SetAcceptFocus(false); }
00036     virtual void Draw();
00037     virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) {};
00038     virtual void Activate( bool bActivate_ ) {}
00039
00040     void SetPanelColor( COLOR eColor_ ) { m_uPanelColor = eColor_; }
00041     void SetLineColor( COLOR eColor_ ) { m_uLineColor = eColor_; }
00042     void SetFontColor( COLOR eColor_ ) { m_uFontColor = eColor_; }
00043     void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00044     void SetCaption( const K_CHAR *pcCaption_ ) { m_pcCaption = pcCaption_; }
00045 private:
00046     COLOR m_uPanelColor;
00047     COLOR m_uLineColor;
00048     COLOR m_uFontColor;
00049
00050     Font_t *m_pstFont;
00051     const K_CHAR *m_pcCaption;
00052 };
00053
00054 #endif
00055

```

14.17 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_label.h File Reference

GUI Label Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "font.h"
```

Classes

- class `LabelControl`

14.17.1 Detailed Description

GUI Label Control. A label control is a static text element, specified by a font, a color, and a string to overlay at a given location.

Definition in file [control_label.h](#).

14.18 control_label.h

```
00001 /*=====
00002 
00003 |_____|_____|_____|_____|_____|_____|
00004 | \ / | \ / | \ / | \ / | \ / | \ / |
00005 |  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 =====*/
00022 #ifndef __CONTROL_LABEL_H__
00023 #define __CONTROL_LABEL_H__
00024 
00025 #include "gui.h"
```



```

00024
00025 //-----
00026 void PanelControl::Draw()
00027 {
00028     GUI_DEBUG_PRINT( "PanelControl::Draw()\n");
00029     GraphicsDriver *pclDriver = GetParentWindow()
->GetDriver();
00030     DrawRectangle_t stRectangle;
00031     K_USHORT usX, usY;
00032
00033     GetControlOffset(&usX, &usY);
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 }

```

14.21 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_panel.h File Reference

GUI Panel Control.

```
#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
```

Classes

- class `PanelControl`

14.21.1 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](#).

14.22 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

```



```

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(
GuiEvent_t *pstEvent_)
00103 {
00104     return GUI_EVENT_OK;
00105 }

```

14.25 /home/moslevin/googlecode/mark3/trunk/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](#)

14.25.1 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.h](#).

14.26 control_progress.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_PROGRESS_H__
00023 #define __CONTROL_PROGRESS_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 class ProgressControl : 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 SetBackColor( COLOR eColor_ ) { m_uBackColor = eColor_; }
00039     void SetProgressColor( COLOR eColor_ ) { m_uProgressColor = eColor_; }
00040     void SetBorderColor( COLOR eColor_ ) { m_uBorderColor = eColor_; }
00041
00042     void SetProgress( K_UCHAR ucProgress_ );
00043
00044 private:
00045     COLOR m_uBackColor;
00046     COLOR m_uProgressColor;
00047     COLOR m_uBorderColor;
00048     K_UCHAR m_ucProgress;
00049 };
00050
00051 #endif
00052
```

14.27 /home/moslevin/googlecode/mark3/trunk/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.

14.27.1 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 c1MyDCPU;

    // Read program code into ausROM buffer here

    // Initialize the DCPU emulator
    c1MyDCPU.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)
{
    c1MyCPU.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 = c1MyCPU.GetRegisters();
```

ToDo: Implement the hardware bus interface - provide functions to register devices/drivers/etc. with the VM prior to initialization. Hook into the interrupt system to provide communication from the VM back to the host processor.

Definition in file [dcpu.cpp](#).

14.28 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 =====*/
00118 #include "dcpu.h"
00119 #include "kerneltypes.h"
00120 #include "ll.h"
00121
00122 #define CORE_DEBUG 0
00123
00124 //-----
00125 #if CORE_DEBUG
00126     #define DBG_PRINT(...)      printf(__VA_ARGS__)
00127 #else
00128     #define DBG_PRINT(...)
00129 #endif
00130
00131 //-----
00135 static const K_UCHAR aucBasicOpcodeCycles[] =
00136 {
00137     0,      // OP_NON_BASIC = 0
00138     1,      // OP_SET
00139     2,      // OP_ADD
```

```

00140     2,      // OP_SUB
00141     2,      // OP_MUL
00142     2,      // OP_MLI
00143     3,      // OP_DIV
00144     3,      // OP_DVI,
00145     3,      // OP_MOD,
00146     3,      // OP_MDI,
00147     1,      // OP_AND,
00148     1,      // OP_BOR,
00149     1,      // OP_XOR,
00150     1,      // OP_SHR,
00151     1,      // OP_ASR,
00152     1,      // OP_SHL,
00153     2,      // OP_IFB,
00154     2,      // OP_IFC,
00155     2,      // OP_IFE,
00156     2,      // OP_IFN,
00157     2,      // OP_IFG,
00158     2,      // OP_IFA,
00159     2,      // OP_IFL,
00160     2,      // OP_IFU,
00161     0,      // OP_18,
00162     0,      // OP_19,
00163     3,      // OP_ADX,
00164     3,      // OP_SBX,
00165     0,      // OP_1C,
00166     0,      // OP_1D,
00167     2,      // OP_STI,
00168     2,      // OP_STD
00169 };
00170
00171 //-----
00175 static const K_UCHAR aucExtendedOpcodeCycles[] =
00176 {
00177     0,      // "RESERVED",
00178     3,      // "JSR",
00179     0,      // "UNDEFINED"
00180     0,      // "UNDEFINED"
00181     0,      // "UNDEFINED"
00182     0,      // "UNDEFINED"
00183     0,      // "UNDEFINED"
00184     0,      // "UNDEFINED"
00185     4,      // "INT",
00186     1,      // "IAG",
00187     1,      // "IAS",
00188     3,      // "RFI",
00189     2,      // "IAQ",
00190     0,      // "UNDEFINED"
00191     0,      // "UNDEFINED"
00192     0,      // "UNDEFINED"
00193     2,      // "HWN",
00194     4,      // "HWQ",
00195     4,      // "HWI",
00196     0,      // "UNDEFINED"
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 };
00210
00211 //-----
00212 void DCPU::SET()
00213 {
00214     DBG_PRINT("SET\n");
00215     *b = *a;
00216 }
00217
00218 //-----
00219 void DCPU::ADD()
00220 {
00221     K_ULONG ulTemp;
00222     DBG_PRINT("ADD\n");
00223
00224     ulTemp = (K_ULONG)*a + (K_ULONG)*b;
00225     if (ulTemp >= 65536)
00226     {
00227         m_stRegisters.EX = 0x0001;
00228     }
00229     else

```

```

00230     {
00231         m_stRegisters.EX = 0;
00232     }
00233
00234     *b = *b + *a;
00235 }
00236
00237 //-----
00238 void DCPU::SUB()
00239 {
00240     K_LONG lTemp;
00241     DBG_PRINT("SUB\n");
00242
00243     lTemp = (K_LONG)*b - (K_LONG)*a;
00244     if (lTemp < 0)
00245     {
00246         m_stRegisters.EX = 0xFFFF;
00247     }
00248     else
00249     {
00250         m_stRegisters.EX = 0;
00251     }
00252
00253     *b = *b - *a;
00254 }
00255
00256 //-----
00257 void DCPU::MUL()
00258 {
00259     K_ULONG ulTemp;
00260
00261     DBG_PRINT("MUL\n");
00262     ulTemp = ((K_ULONG)*a * (K_ULONG)*b);
00263     m_stRegisters.EX = (K_USHORT)(ulTemp >> 16);
00264     *b = (K_USHORT)(ulTemp & 0x0000FFFF);
00265 }
00266
00267 //-----
00268 void DCPU::MLI()
00269 {
00270     K_LONG lTemp;
00271
00272     DBG_PRINT("MLI\n");
00273     lTemp = ((K_LONG)*a * (K_SHORT)*b);
00274     m_stRegisters.EX = (K_USHORT)(lTemp >> 16);
00275     *b = (K_USHORT)(lTemp & 0x0000FFFF);
00276 }
00277
00278 //-----
00279 void DCPU::DIV()
00280 {
00281     K_USHORT usTemp;
00282
00283     DBG_PRINT("DIV\n");
00284     if (*a == 0)
00285     {
00286         *b = 0;
00287         m_stRegisters.EX = 0;
00288     }
00289     else
00290     {
00291         usTemp = (K_USHORT)((K_ULONG)*b << 16) / (K_ULONG)*a;
00292         *b = *b / *a;
00293         m_stRegisters.EX = usTemp;
00294     }
00295 }
00296
00297 //-----
00298 void DCPU::DVI()
00299 {
00300     K_USHORT usTemp;
00301
00302     DBG_PRINT("DVI\n");
00303     if (*a == 0)
00304     {
00305         *b = 0;
00306         m_stRegisters.EX = 0;
00307     }
00308     else
00309     {
00310         usTemp = (K_USHORT)((K_LONG)*a * (K_SHORT)*b) << 16 / (K_LONG)*a;
00311         *b = (K_USHORT)((K_SHORT)*b / *a);
00312         m_stRegisters.EX = usTemp;
00313     }
00314 }
00315 }

```



```

00316
00317 //-----
00318 void DCPU::MOD()
00319 {
00320     DBG_PRINT("MOD\n");
00321     if (*a == 0)
00322     {
00323         *b = 0;
00324     }
00325     else
00326     {
00327         *b = *b % *a;
00328     }
00329 }
00330
00331 //-----
00332 void DCPU::MDI()
00333 {
00334     DBG_PRINT("MDI\n");
00335     if (*b == 0)
00336     {
00337         *a = 0;
00338     }
00339     else
00340     {
00341         *b = (K_USHORT) (*((K_SHORT*)b) % *((K_SHORT*)a));
00342     }
00343 }
00344
00345 //-----
00346 void DCPU::AND()
00347 {
00348     DBG_PRINT("AND\n");
00349     *b = *b & *a;
00350 }
00351
00352 //-----
00353 void DCPU::BOR()
00354 {
00355     DBG_PRINT("BOR\n");
00356     *b = *b | *a;
00357 }
00358
00359 //-----
00360 void DCPU::XOR()
00361 {
00362     DBG_PRINT("XOR\n");
00363     *b = *b ^ *a;
00364 }
00365
00366 //-----
00367 void DCPU::SHR()
00368 {
00369     K_USHORT usTemp = (K_USHORT) (((K_ULONGLONG) *b) << 16) >> (K_ULONGLONG) *a;
00370
00371     DBG_PRINT("SHR\n");
00372     *b = *b >> *a;
00373     m_stRegisters.EX = usTemp;
00374 }
00375
00376 //-----
00377 void DCPU::ASR()
00378 {
00379     K_USHORT usTemp = (K_USHORT) (((K_LONG) *b) << 16) >> (K_LONG) *a;
00380
00381     DBG_PRINT("ASR\n");
00382     *b = (K_USHORT) (*((K_SHORT*)b) >> *((K_SHORT*)a));
00383     m_stRegisters.EX = usTemp;
00384 }
00385 //-----
00386 void DCPU::SHL()
00387 {
00388     K_USHORT usTemp = (K_USHORT) (((K_ULONGLONG) *b) << (K_ULONGLONG) *a) >> 16;
00389
00390     DBG_PRINT("SHL\n");
00391     *b = *b << *a;
00392     m_stRegisters.EX = usTemp;
00393 }
00394
00395 //-----
00396 bool DCPU::IFB()
00397 {
00398     DBG_PRINT("IFB\n");
00399     if ((*b & *a) != 0)
00400     {
00401         return true;
00402     }

```

```

00403     return false;
00404 }
00405
00406 //-----
00407 bool DCPU::IFC()
00408 {
00409     DBG_PRINT("IFC\n");
00410     if ((*b & *a) == 0)
00411     {
00412         return true;
00413     }
00414     return false;
00415 }
00416
00417 //-----
00418 bool DCPU::IFE()
00419 {
00420     DBG_PRINT("IFE\n");
00421     if (*b == *a)
00422     {
00423         return true;
00424     }
00425     return false;
00426 }
00427
00428 //-----
00429 bool DCPU::IFN()
00430 {
00431     DBG_PRINT("IFN\n");
00432     if (*b != *a)
00433     {
00434         return true;
00435     }
00436     return false;
00437 }
00438
00439 //-----
00440 bool DCPU::IFG()
00441 {
00442     DBG_PRINT("IFG\n");
00443     if (*b > *a)
00444     {
00445         return true;
00446     }
00447     return false;
00448 }
00449
00450 //-----
00451 bool DCPU::IFA()
00452 {
00453     DBG_PRINT("IFA\n");
00454     if (*(K_SHORT*)b > *(K_SHORT*)a)
00455     {
00456         return true;
00457     }
00458     return false;
00459 }
00460
00461 //-----
00462 bool DCPU::IFL()
00463 {
00464     DBG_PRINT("IFL\n");
00465     if (*b < *a)
00466     {
00467         return true;
00468     }
00469     return false;
00470 }
00471
00472 //-----
00473 bool DCPU::IFU()
00474 {
00475     DBG_PRINT("IFU\n");
00476     if (*(K_SHORT*)b < *(K_SHORT*)a)
00477     {
00478         return true;
00479     }
00480     return false;
00481 }
00482
00483 //-----
00484 void DCPU::ADX()
00485 {
00486     K_ULONG ulTemp;
00487     DBG_PRINT("ADX\n");
00488     ulTemp = (K_ULONG)*b + (K_ULONG)*a + (K_ULONG)m_stRegisters.
EX;

```

```

00489     if (ulTemp >= 0x10000)
00490     {
00491         m_stRegisters.EX = 1;
00492     }
00493     else
00494     {
00495         m_stRegisters.EX = 0;
00496     }
00497
00498     *b = ((K_USHORT)(ulTemp & 0x0000FFFF));
00499 }
00500
00501 //-----
00502 void DCPU::SBX()
00503 {
00504     K_LONG lTemp;
00505     DBG_PRINT("SBX\n");
00506     lTemp = (K_LONG)*b - (K_LONG)*a + (K_LONG)m_stRegisters.EX;
00507     if (lTemp < 0 )
00508     {
00509         m_stRegisters.EX = 0xFFFF;
00510     }
00511     else
00512     {
00513         m_stRegisters.EX = 0;
00514     }
00515
00516     *b = ((K_USHORT)(lTemp & 0x0000FFFF));
00517 }
00518
00519 //-----
00520 void DCPU::STI()
00521 {
00522     DBG_PRINT("STI\n");
00523     *b = *a;
00524     m_stRegisters.I++;
00525     m_stRegisters.J++;
00526 }
00527
00528 //-----
00529 void DCPU::STD()
00530 {
00531     DBG_PRINT("STD\n");
00532     *b = *a;
00533     m_stRegisters.I--;
00534     m_stRegisters.J--;
00535 }
00536
00537 //-----
00538 void DCPU::JSR()
00539 {
00540     DBG_PRINT("JSR\n");
00541     m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
00542     m_stRegisters.PC = *b;
00543 }
00544
00545 //-----
00546 void DCPU::INT()
00547 {
00548     DBG_PRINT("INT\n");
00549
00550     if (m_stRegisters.IA == 0)
00551     {
00552         // If IA is not set, return out.
00553         return;
00554     }
00555
00556     // Either acknowledge the interrupt immediately, or queue it.
00557     if (m_bInterruptQueueing == false)
00558     {
00559         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
00560         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;
00561
00562         m_stRegisters.A = *a;
00563         m_stRegisters.PC = m_stRegisters.IA;
00564         m_bInterruptQueueing = true;
00565     }
00566     else
00567     {
00568         // Add interrupt message to the queue
00569         m_ausInterruptQueue[ ++m_ucQueueLevel
] = *a;
00570     }
00571 }

```

```

00572
00573 //-----
00574 void DCPU::ProcessInterruptQueue()
00575 {
00576     // If there's an interrupt address specified, queueing is disabled, and
00577     // the queue isn't empty
00578     if (m_stRegisters.IA && !m_bInterruptQueueing
        && m_ucQueueLevel)
00579     {
00580         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
00581         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;
00582
00583         m_stRegisters.A = m_ausInterruptQueue[
m_ucQueueLevel-- ];
00584         m_stRegisters.PC = m_stRegisters.IA;
00585
00586         m_bInterruptQueueing = true;
00587     }
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
00620 }
00621
00622 //-----
00623 void DCPU::IAQ()
00624 {
00625     DBG_PRINT("IAQ\n");
00626
00627     if (*a)
00628     {
00629         m_bInterruptQueueing = true;
00630     }
00631     else
00632     {
00633         m_bInterruptQueueing = false;
00634     }
00635 }
00636
00637
00638 }
00639
00640 //-----
00641 void DCPU::HWN()
00642 {
00643     LinkListNode *pclNode;
00644
00645     DBG_PRINT("HWN\n");
00646     m_usTempA = 0;
00647     pclNode = m_clPluginList.GetHead();
00648     while (pclNode)
00649     {
00650         m_usTempA++;
00651         pclNode = pclNode->GetNext();
00652     }
00653
00654     *a = m_usTempA;
00655 }
00656
00657
00658 //-----
00659 void DCPU::HWQ()

```

```

00660 {
00661     DBG_PRINT("HWQ\n");
00662     DCPUPugin *pclPlugin;
00663     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead
00664     ();
00665     while (pclPlugin)
00666     {
00667         if (pclPlugin->GetDeviceNumber() == *a)
00668         {
00669             pclPlugin->Enumerate(&m_stRegisters);
00670             break;
00671         }
00672         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00673     }
00674 }
00675
00676 //-----
00677 void DCPU::HWI()
00678 {
00679     DBG_PRINT("HWI\n");
00680
00681     DCPUPugin *pclPlugin;
00682     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead
00683     ();
00684     while (pclPlugin)
00685     {
00686         if (pclPlugin->GetDeviceNumber() == *a)
00687         {
00688             pclPlugin->Interrupt(this);
00689             break;
00690         }
00691         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00692     }
00693 }
00694
00695 //-----
00696 void DCPU::Init( K_USHORT *pusRAM_, K_USHORT usRAMSize_,
00697                 const K_USHORT *pusROM_, K_USHORT usROMSize_ )
00698 {
00699     m_stRegisters.PC = 0;
00700     m_stRegisters.SP = usRAMSize_ ;
00701     m_stRegisters.A = 0;
00702     m_stRegisters.B = 0;
00703     m_stRegisters.C = 0;
00704     m_stRegisters.X = 0;
00705     m_stRegisters.Y = 0;
00706     m_stRegisters.Z = 0;
00707     m_stRegisters.I = 0;
00708     m_stRegisters.J = 0;
00709     m_stRegisters.EX = 0;
00710     m_stRegisters.IA = 0;
00711     m_ulCycleCount = 0;
00712
00713     m_pusROM = (K_USHORT*)pusROM_;
00714     m_usROMSize = usROMSize_;
00715
00716     m_pusRAM = pusRAM_;
00717     m_usRAMSize = usRAMSize_;
00718 }
00719
00720 //-----
00721 K_UCHAR DCPU::GetOperand( K_UCHAR ucOpType_, K_USHORT **
00722     pusResult_ )
00723 {
00724     K_UCHAR ucRetVal = 0;
00725     switch (ucOpType_)
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_ -
00730             ARG_A ];
00731             break;
00732
00733         case ARG_BRACKET_A: case ARG_BRACKET_B: case ARG_BRACKET_C: case
00734         ARG_BRACKET_X:
00735         case ARG_BRACKET_Y: case ARG_BRACKET_Z: case ARG_BRACKET_I: case
00736         ARG_BRACKET_J:
00737             *pusResult_ = &m_pusRAM[ m_stRegisters.
00738             ausRegisters[ ucOpType_ - ARG_BRACKET_A ] ];
00739             break;
00740
00741         case ARG_WORD_A: case ARG_WORD_B: case ARG_WORD_C: case ARG_WORD_X:
00742         case ARG_WORD_Y: case ARG_WORD_Z: case ARG_WORD_I: case ARG_WORD_J:
00743             {
00744                 K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++

```

```

    };
00740     usTemp += m_stRegisters.ausRegisters[ ucOpType_ -
ARG_WORD_A ];
00741     *pusResult_ = &m_pusRAM[ usTemp ];
00742     ucRetVal = 1;
00743 }
00744     break;
00745     case ARG_PUSH_POP_SP:
00746         if (*pusResult_ == a)
00747         {
00748             a = &m_pusRAM[ m_stRegisters.SP++ ];
00749         }
00750         else
00751         {
00752             b = &m_pusRAM[ --m_stRegisters.SP ];
00753         }
00754         break;
00755     case ARG_PEEK_SP:
00756         *pusResult_ = &m_pusRAM[ m_stRegisters.SP ];
00757         break;
00758     case ARG_WORD_SP:
00759     {
00760         K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++
];
00761         usTemp += m_stRegisters.SP;
00762         *pusResult_ = &m_pusRAM[ usTemp ];
00763         ucRetVal++;
00764     }
00765     break;
00766     case ARG_SP:
00767         *pusResult_ = &(m_stRegisters.SP);
00768         break;
00769     case ARG_PC:
00770         *pusResult_ = &(m_stRegisters.PC);
00771         break;
00772     case ARG_EX:
00773         *pusResult_ = &(m_stRegisters.EX);
00774         break;
00775     case ARG_NEXT_WORD:
00776         *pusResult_ = &m_pusRAM[ m_pusROM[ m_stRegisters
.PC++ ] ];
00777         ucRetVal++;
00778         break;
00779     case ARG_NEXT_LITERAL:
00780         *pusResult_ = &m_pusROM[ m_stRegisters.PC++ ];
00781         ucRetVal++;
00782         break;
00783
00784     case ARG_LITERAL_0:
00785         *pusResult_ = &m_usTempA;
00786         m_usTempA = (K_USHORT) (-1);
00787         break;
00788     case ARG_LITERAL_1: case ARG_LITERAL_2: case ARG_LITERAL_3: case
ARG_LITERAL_4:
00789     case ARG_LITERAL_5: case ARG_LITERAL_6: case ARG_LITERAL_7: case
ARG_LITERAL_8:
00790     case ARG_LITERAL_9: case ARG_LITERAL_A: case ARG_LITERAL_B: case
ARG_LITERAL_C:
00791     case ARG_LITERAL_D: case ARG_LITERAL_E: case ARG_LITERAL_F: case
ARG_LITERAL_10:
00792     case ARG_LITERAL_11: case ARG_LITERAL_12: case ARG_LITERAL_13: case
ARG_LITERAL_14:
00793     case ARG_LITERAL_15: case ARG_LITERAL_16: case ARG_LITERAL_17: case
ARG_LITERAL_18:
00794     case ARG_LITERAL_19: case ARG_LITERAL_1A: case ARG_LITERAL_1B: case
ARG_LITERAL_1C:
00795     case ARG_LITERAL_1D: case ARG_LITERAL_1E: case ARG_LITERAL_1F:
00796         *pusResult_ = &m_usTempA;
00797         m_usTempA = (K_USHORT) (ucOpType_ - ARG_LITERAL_1);
00798         break;
00799     default:
00800         break;
00801 }
00802     return ucRetVal;
00803 }
00804
00805 //-----
00806 void DCPU::RunOpcode()
00807 {
00808     // Fetch the opcode @ the current program counter
00809     K_USHORT usWord = m_pusROM[ m_stRegisters.PC++ ];
00810     K_UCHAR ucOp = (K_UCHAR)DCPU_NORMAL_OPCODE_MASK(
usWord);
00811     K_UCHAR ucA = (K_UCHAR)DCPU_A_MASK(usWord);
00812     K_UCHAR ucB = (K_UCHAR)DCPU_B_MASK(usWord);
00813     K_UCHAR ucSize = 1;
00814

```

```

00815 // Decode the opcode
00816 if (ucOp)
00817 {
00818     bool bRunNext = true;
00819
00820     a = &m_usTempA;
00821     b = 0;
00822
00823     // If this is a "basic" opcode, decode "a" and "b"
00824     ucSize += GetOperand( ucA , &a );
00825     ucSize += GetOperand( ucB, &b );
00826
00827     // Add the cycles to the runtime clock
00828     m_ulCycleCount += (K_ULONG)aucBasicOpcodeCycles
[ ucOp ];
00829     m_ulCycleCount += (ucSize - 1);
00830
00831     // Execute the instruction once we've decoded the opcode and
00832     // processed the arguments.
00833     switch (DCPU_NORMAL_OPCODE_MASK(usWord))
00834     {
00835         case OP_SET: SET(); break;
00836         case OP_ADD: ADD(); break;
00837         case OP_SUB: SUB(); break;
00838         case OP_MUL: MUL(); break;
00839         case OP_MLI: MLI(); break;
00840         case OP_DIV: DIV(); break;
00841         case OP_DVI: DVI(); break;
00842         case OP_MOD: MOD(); break;
00843         case OP_MDI: MDI(); break;
00844         case OP_AND: AND(); break;
00845         case OP_BOR: BOR(); break;
00846         case OP_XOR: XOR(); break;
00847         case OP_SHR: SHR(); break;
00848         case OP_ASR: ASR(); break;
00849         case OP_SHL: SHL(); break;
00850         case OP_IFB: bRunNext = IFB(); break;
00851         case OP_IFC: bRunNext = IFC(); break;
00852         case OP_IFE: bRunNext = IFE(); break;
00853         case OP_IFN: bRunNext = IFN(); break;
00854         case OP_IFG: bRunNext = IFG(); break;
00855         case OP_IFA: bRunNext = IFA(); break;
00856         case OP_IFL: bRunNext = IFL(); break;
00857         case OP_IFU: bRunNext = IFU(); break;
00858         case OP_ADX: ADX(); break;
00859         case OP_SBX: SBX(); break;
00860         case OP_STI: STI(); break;
00861         case OP_STD: STD(); break;
00862         default: break;
00863     }
00864
00865     // If we're not supposed to run the next instruction (i.e. skip it
00866     // due to failed condition), adjust the PC.
00867     if (!bRunNext)
00868     {
00869         // Skipped branches take an extra cycle
00870         m_ulCycleCount++;
00871
00872         // Skip the next opcode
00873         usWord = m_pusROM[ m_stRegisters.PC++ ];
00874         if (DCPU_NORMAL_OPCODE_MASK(usWord))
00875         {
00876             DBG_PRINT( "Skipping Basic Opcode: %X\n",
DCPU_NORMAL_OPCODE_MASK(usWord));
00877             // If this is a "basic" opcode, decode "a" and "b" - we do this
to make sure our
00878             // PC gets adjusted properly.
00879             GetOperand( DCPU_A_MASK(usWord), &a );
00880             GetOperand( DCPU_B_MASK(usWord), &b );
00881         }
00882         else
00883         {
00884             DBG_PRINT( "Skipping Extended Opcode: %X\n",
DCPU_EXTENDED_OPCODE_MASK(usWord));
00885             GetOperand( DCPU_A_MASK(usWord), &a );
00886         }
00887     }
00888 }
00889 else
00890 {
00891     // Extended opcode. These only have a single argument, stored in the
00892     // "a" field.
00893     GetOperand( ucA, &a );
00894     m_ulCycleCount++;
00895
00896     // Execute the "extended" instruction now that the opcode has been
00897     // decoded, and the arguments processed.

```

```

00898         switch (ucB)
00899         {
00900             case OP_EX_JSR:    JSR(); break;
00901             case OP_EX_INT:    INT(); break;
00902             case OP_EX_IAG:    IAG(); break;
00903             case OP_EX_IAS:    IAS(); break;
00904             case OP_EX_RFI:    RFI(); break;
00905             case OP_EX_IAQ:    IAQ(); break;
00906             case OP_EX_HWN:    HWN(); break;
00907             case OP_EX_HWQ:    HWQ(); break;
00908             case OP_EX_HWI:    HWI(); break;
00909             default:          break;
00910         }
00911     }
00912
00913     // Process an interrupt from the queue (if there is one)
00914     ProcessInterruptQueue();
00915 }
00916
00917 //-----
00918 void DCPU::SendInterrupt( K_USHORT usMessage_ )
00919 {
00920     if (m_stRegisters.IA == 0)
00921     {
00922         // If IA is not set, return out.
00923         return;
00924     }
00925
00926     // Either acknowledge the interrupt immediately, or queue it.
00927     if (m_bInterruptQueueing == false)
00928     {
00929         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.PC;
00930         m_pusRAM[ --m_stRegisters.SP ] = m_stRegisters
.A;
00931
00932         m_stRegisters.A = usMessage_;
00933         m_stRegisters.PC = m_stRegisters.IA;
00934         m_bInterruptQueueing = true;
00935     }
00936     else
00937     {
00938         // Add interrupt message to the queue
00939         m_ausInterruptQueue[ ++m_ucQueueLevel
] = usMessage_;
00940     }
00941 }
00942
00943 //-----
00944 void DCPU::AddPlugin( DCPUPugin *pclPlugin_ )
00945 {
00946     m_clPluginList.Add( (LinkListNode*)pclPlugin_
);
00947 }

```

14.29 /home/moslevin/googlecode/mark3/trunk/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 [DCPUPugin](#)
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.

14.29.1 Detailed Description

DCPU-16 emulator.

Definition in file [dcpu.h](#).

14.29.2 Macro Definition Documentation

14.29.2.1 `#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](#).

14.29.3 Enumeration Type Documentation

14.29.3.1 `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](#).

14.29.3.2 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](#).

14.30 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 | \
00064       ((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 } DCPU_Argument;
00251
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 DCPUPugin : 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

```

14.31 /home/moslevin/googlecode/mark3/trunk/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="kernels.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="semaphore.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" */

14.31.1 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](#).

14.32 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="semaphore.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_ENTRYPPOINT_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

```

14.33 /home/moslevin/googlecode/mark3/trunk/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 a 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 [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.


```

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 //-----
00159 typedef struct
00160 {
00161     K_USHORT usLeft;
00162     K_USHORT usRight;
00163     K_USHORT usTop;
00164     K_USHORT usBottom;
00165 } DrawWindow_t;
00166
00167 //-----
00172 typedef struct
00173 {
00174     K_USHORT usSrcX;
00175     K_USHORT usSrcY;
00176     K_USHORT usDstX;
00177     K_USHORT usDstY;

```

```

00178     K_USHORT  usCopyHeight;
00179     K_USHORT  usCopyWidth;
00180 } DrawMove_t;
00181
00182 //-----
00188 typedef struct
00189 {
00190     K_USHORT usX;
00191     K_USHORT usY;
00192 } DrawVector_t;
00193
00194 //-----
00199 typedef struct
00200 {
00201     K_USHORT    usNumPoints;
00202     COLOR       uColor;
00203     K_BOOL      bFill;
00204     DrawVector_t *pstVector;
00205 } DrawPoly_t;
00206
00207 #endif //__DRAW_H_

```

14.35 /home/moslevin/googlecode/mark3/trunk/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**

14.35.1 Detailed Description

Device driver/hardware abstraction layer.

Definition in file [driver.cpp](#).

14.36 driver.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 #include "driver.h"
00025
00026 //-----
00027 #if defined __FILE_ID__
00028 #undef __FILE_ID__
00029 #endif
00030 #define __FILE_ID__ DRIVER_CPP
00031
00032 //-----
00033 #if KERNEL_USE_DRIVER
00034
00035 DoubleLinkedList DriverList::m_clDriverList
00036 ;
00037
00038 class DevNull : public Driver
00039 {
00040 public:
00041     virtual void Init() { SetName("/dev/null"); };
00042     virtual K_UCHAR Open() { return 0; };
00043     virtual K_UCHAR Close() { return 0; };
00044
00045     virtual K_USHORT Read( K_USHORT usBytes_,
00046         K_UCHAR *pucData_) { return 0; };
00047
00048     virtual K_USHORT Write( K_USHORT usBytes_,
00049         K_UCHAR *pucData_) { return 0; };
00050
00051     virtual K_USHORT Control( K_USHORT usEvent_,
00052         void *pvDataIn_,
00053         K_USHORT usSizeIn_,
00054         void *pvDataOut_,
00055         K_USHORT usSizeOut_ ) { return 0; };
00056
00057 };
00058
00059 //-----
00060 static DevNull clDevNull;
00061
00062 //-----
00063 static K_UCHAR DrvCmp( const K_CHAR *szStr1_, const K_CHAR *szStr2_ )
00064 {
00065     K_CHAR *szTmp1 = (K_CHAR*) szStr1_;
00066     K_CHAR *szTmp2 = (K_CHAR*) szStr2_;
00067
00068     while (*szTmp1 && *szTmp2)
00069     {
00070         if (*szTmp1++ != *szTmp2++)
00071         {
00072             return 0;
00073         }
00074     }
00075
00076     // Both terminate at the same length
00077     if (!(*szTmp1) && !(*szTmp2))
00078     {
00079         return 1;
00080     }
00081
00082     return 0;
00083 }
00084
00085 //-----
00086 void DriverList::Init()
00087 {
00088     // Ensure we always have at least one entry - a default in case no match
00089     // is found (/dev/null)
00090     clDevNull.Init();
00091     Add(&clDevNull);
00092 }

```

```

00094 }
00095
00096 //-----
00097 Driver *DriverList::FindByPath( const K_CHAR *
    m_pcPath )
00098 {
00099     KERNEL_ASSERT( m_pcPath );
00100     Driver *pclTemp = static_cast<Driver*>(m_clDriverList
        .GetHead());
00101
00102     while (pclTemp)
00103     {
00104         if(DrvCmp(m_pcPath, pclTemp->GetPath()))
00105         {
00106             return pclTemp;
00107         }
00108         pclTemp = static_cast<Driver*>(pclTemp->GetNext());
00109     }
00110     return &clDevNull;
00111 }
00112
00113 #endif

```

14.37 /home/moslevin/googlecode/mark3/trunk/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.

14.37.1 Detailed Description

[Driver](#) abstraction framework.

14.37.2 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.

14.37.3 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.

14.37.4 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 he object to operate on.

Once a driver has been added to the table, drivers are opened by NAME using `DriverList::FindBy-Name("/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:

```
pclI2C = DriverList::FindByName("/dev/i2c");
pclUART = DriverList::FindByName("/dev/tty0");
pclSPI = DriverList::FindByName("/dev/spi");

pclI2C->Write(12, "Hello World!");
pclUART->Write(12, "Hello World!");
pclSPI->Write(12, "Hello World!");
```

Definition in file [driver.h](#).

14.38 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();
00264     static void Add( Driver *pclDriver_ ) { m_clDriverList
00265 .Add(pclDriver_); }
00274     static void Remove( Driver *pclDriver_ ) { m_clDriverList
00275 .Remove(pclDriver_); }
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

```

14.39 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.cpp File Reference

Fixed-block-size memory management.

```

#include "kerneltypes.h"
#include "fixed_heap.h"
#include "threadport.h"

```

14.39.1 Detailed Description

Fixed-block-size memory management. This allows a user to create heaps containing multiple lists, each list containing a linked-list of blocks that are each the same size. As a result of the linked-list format, these heaps are very fast - requiring only a linked list pop/push to allocated/free memory. Array traversal is required to allow for the

optimal heap to be used. Blocks are chosen from the first heap with free blocks large enough to fulfill the request.

Only simple malloc/free functionality is supported in this implementation, no complex vector-allocate or reallocation functions are supported.

Heaps are protected by critical section, and are thus thread-safe.

When creating a heap, a user supplies an array of heap configuration objects, which determines how many objects of what size are available.

The configuration objects are defined from smallest list to largest, the memory to back the heap is supplied as a pointer to a "blob" of memory which will be used to create the underlying heap objects that make up the heap internal data structures. This blob must be large enough to contain all of the requested heap objects, with all of the additional metadata required to manage the objects.

Multiple heaps can be created using this library (heaps are not singleton).

Definition in file [fixed_heap.cpp](#).

14.40 fixed_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  =====*/
00043  #include "kerneltypes.h"
00044  #include "fixed_heap.h"
00045  #include "threadport.h"
00046
00047  //-----
00048  void *BlockHeap::Create( void *pvHeap_, K_USHORT usSize_,
00049                          K_USHORT usBlockSize_ )
00049  {
00050      K_USHORT usNodeCount = usSize_ /
00051                          (usBlockSize_ + sizeof(LinkListNode
00052                          ) + sizeof(void*));
00053      void *pvNode = pvHeap_;
00054      void *pvMaxNode = (void*) ((K_ULONG)pvHeap_ + (K_ULONG)usSize_);
00055      m_clList.Init();
00056
00057      // Create a heap (linked-list nodes + byte pool) in the middle of
00058      // the data blob
00059      for (K_USHORT i = 0; i < usNodeCount; i++ )
00060      {
00061          // Create a pointer back to the source list.
00062          BlockHeap **pclTemp = (BlockHeap**) (pvNode + sizeof(
00063          LinkListNode));
00064          *pclTemp = (BlockHeap*)(this);
00065
00066          // Add the node to the block list
00067          m_clList.Add( (LinkListNode*)pvNode );
00068
00069          // Move the pointer in the pool to point to the next block to allocate
00070          pvNode += (usBlockSize_ + sizeof(LinkListNode) + sizeof(
00071          BlockHeap*));
00072
00073          // Bail if we would be going past the end of the allocated space...
00074          if ((K_ULONG)pvNode >= (K_ULONG)pvMaxNode)
00075          {
00076              break;
00077          }
00078      }
00079      m_usBlocksFree = usNodeCount;
00080
00081      // Return pointer to end of heap (used for heap-chaining)
00082      return pvNode;
00083  }
00084
00085  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*)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*) (pvData_ -
00106 sizeof(LinkListNode) - sizeof(void*));
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 *
00114 pclHeapConfig_ )
00115 {
00116     K_USHORT i = 0;
00117     void *pvTemp = pvHeap_;
00118     while( pclHeapConfig_[i].m_usBlockSize != 0)
00119     {
00120         pvTemp = pclHeapConfig_[i].m_clHeap.Create
00121             (pvTemp,
00122              (pclHeapConfig_[i].m_usBlockSize + sizeof(LinkListNode
00123 ) + sizeof(void*)) *
00124              pclHeapConfig_[i].m_usBlockCount,
00125              pclHeapConfig_[i].m_usBlockSize );
00126         i++;
00127     }
00128     m_paclHeaps = pclHeapConfig_;
00129 }
00130
00131 //-----
00132 void *FixedHeap::Alloc( K_USHORT usSize_ )
00133 {
00134     void *pvRet = 0;
00135     K_USHORT i = 0;
00136
00137     // Go through all heaps, trying to find the smallest one that
00138     // has a free item to satisfy the allocation
00139     while (m_paclHeaps[i].m_usBlockSize != 0)
00140     {
00141         CS_ENTER();
00142         if ((m_paclHeaps[i].m_usBlockSize >= usSize_) && m_paclHeaps
00143 [i].m_clHeap.IsFree() )
00144         {
00145             // Found a match
00146             pvRet = m_paclHeaps[i].m_clHeap.Alloc();
00147         }
00148         CS_EXIT();
00149
00150         // Return an object if found
00151         if (pvRet)
00152         {
00153             return pvRet;
00154         }
00155         i++;
00156     }
00157
00158     // Or null on no-match
00159     return pvRet;
00160 }
00161
00162 //-----
00163 void FixedHeap::Free( void *pvNode_ )
00164 {
00165     // Compute the pointer to the block-heap this block belongs to, and
00166     // return it.
00167     CS_ENTER();

```

```

00165     BlockHeap **pclHeap = (BlockHeap**) (pvNode_ - sizeof(
    BlockHeap*));
00166     (*pclHeap)->Free(pvNode_);
00167     CS_EXIT();
00168 }
00169
00170

```

14.41 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed_heap.h File Reference

Fixed-block-size heaps.

```

#include "kerneltypes.h"
#include "ll.h"

```

Classes

- class [BlockHeap](#)
Single-block-size heap.
- class [HeapConfig](#)
Heap configuration object.
- class [FixedHeap](#)
Fixed-size-block heap allocator with multiple block sizes.

14.41.1 Detailed Description

Fixed-block-size heaps.

Definition in file [fixed_heap.h](#).

14.42 fixed_heap.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 __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

```

14.43 /home/moslevin/googlecode/mark3/trunk/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.*

14.43.1 Detailed Description

Font structure definitions.

Definition in file [font.h](#).

14.46 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  //-----
00027  void GraphicsDriver::ClearScreen()
00028  {
00029      DrawPoint_t stPoint;
00030      stPoint.uColor = COLOR_BLACK;
00031
00032      for (stPoint.usX = 0; stPoint.usX < m_usResX; stPoint.usX++)
00033      {
00034          for (stPoint.usY = 0; stPoint.usY < m_usResY; stPoint.usY++)
00035          {
00036              // Pixel Write
00037              DrawPixel(&stPoint);
00038          }
00039      }
00040  }
00041  //-----
00042  void GraphicsDriver::Point(DrawPoint_t *
pstPoint_)
00043  {
00044      DrawPixel(pstPoint_);
00045  }
00046
00047  //-----
00048  void GraphicsDriver::Line(DrawLine_t *pstLine_)
00049  {
00050      // Bresenham Line drawing algorithm, adapted from:
00051      // www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html
00052
00053      DrawPoint_t stPoint;
00054      K_SHORT sX1 = (K_SHORT)pstLine_>usX1;
00055      K_SHORT sX2 = (K_SHORT)pstLine_>usX2;
00056      K_SHORT sY1 = (K_SHORT)pstLine_>usY1;
00057      K_SHORT sY2 = (K_SHORT)pstLine_>usY2;
00058      K_SHORT sDeltaY = sY2 - sY1;
00059      K_SHORT sDeltaX = sX2 - sX1;
00060      K_CHAR cStepx, cStepy;
00061      stPoint.uColor = pstLine_>uColor;
00062
00063      if (sDeltaY < 0)
00064      {
00065          sDeltaY = -sDeltaY;
00066          cStepy = -1;
00067      }
00068      else
00069      {
00070          cStepy = 1;
00071      }
00072
00073      if (sDeltaX < 0)
00074      {
00075          sDeltaX = -sDeltaX;
00076          cStepx = -1;
00077      }
00078      else
00079      {
00080          cStepx = 1;
00081      }
00082
00083      // Scale by a factor of 2 in each direction
00084      sDeltaY <<= 1;
00085      sDeltaX <<= 1;
00086
00087      stPoint.usX = sX1;
00088      stPoint.usY = sY1;

```



```

00089     DrawPixel(&stPoint);
00090
00091     if (sDeltaX > sDeltaY)
00092     {
00093         K_SHORT sFraction = sDeltaY - (sDeltaX >> 1);
00094
00095         while (sX1 != sX2)
00096         {
00097             if (sFraction >= 0)
00098             {
00099                 sY1 += cStepy;
00100                 sFraction -= sDeltaX;
00101             }
00102             sX1 += cStepx;
00103             sFraction += sDeltaY;
00104
00105             stPoint.usX = sX1;
00106             stPoint.usY = sY1;
00107             DrawPixel(&stPoint);
00108         }
00109     }
00110     else
00111     {
00112         K_SHORT sFraction = sDeltaX - (sDeltaY >> 1);
00113         while (sY1 != sY2)
00114         {
00115             if (sFraction >= 0)
00116             {
00117                 sX1 += cStepx;
00118                 sFraction -= sDeltaY;
00119             }
00120             sY1 += cStepy;
00121             sFraction += sDeltaX;
00122
00123             stPoint.usX = sX1;
00124             stPoint.usY = sY1;
00125             DrawPixel(&stPoint);
00126         }
00127     }
00128 }
00129
00130 //-----
00131 void GraphicsDriver::Rectangle(DrawRectangle_t
                                *pstRectangle_)
00132 {
00133     DrawPoint_t stPoint;
00134
00135     // if drawing a background fill color (optional)
00136     if (pstRectangle_>bFill == true)
00137     {
00138         stPoint.uColor = pstRectangle_>uFillColor;
00139         for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <=
pstRectangle_>usRight; stPoint.usX++)
00140         {
00141             for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <=
pstRectangle_>usBottom; stPoint.usY++)
00142             {
00143                 DrawPixel(&stPoint);
00144             }
00145         }
00146     }
00147
00148     // Draw four orthogonal lines...
00149     stPoint.uColor = pstRectangle_>uLineColor;
00150     stPoint.usY = pstRectangle_>usTop;
00151     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <=
pstRectangle_>usRight; stPoint.usX++)
00152     {
00153         DrawPixel(&stPoint);
00154     }
00155
00156     stPoint.usY = pstRectangle_>usBottom;
00157     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <=
pstRectangle_>usRight; stPoint.usX++)
00158     {
00159         DrawPixel(&stPoint);
00160     }
00161
00162     stPoint.usX = pstRectangle_>usLeft;
00163     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <=
pstRectangle_>usBottom; stPoint.usY++)
00164     {
00165         DrawPixel(&stPoint);
00166     }
00167
00168     stPoint.usX = pstRectangle_>usRight;
00169     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <=

```

```

        pstRectangle_>usBottom; stPoint.usY++)
00170     {
00171         DrawPixel(&stPoint);
00172     }
00173 }
00174
00175 //-----
00176 void GraphicsDriver::Circle(DrawCircle_t *
    pstCircle_)
00177 {
00178     DrawPoint_t stPoint;
00179     K_SHORT sX;
00180     K_SHORT sY;
00181     K_ULONG ulRadSquare;
00182
00183     K_ULONG ulXSquare;
00184     K_ULONG ulYSquare;
00185
00186     // Get the radius squared value...
00187     ulRadSquare = (K_ULONG)pstCircle_>usRadius;
00188     ulRadSquare *= ulRadSquare;
00189
00190     // Look at the upper-right quarter of the circle
00191     for (sX = 0; sX <= (K_SHORT)pstCircle_>usRadius; sX++)
00192     {
00193         ulXSquare = (K_ULONG)sX;
00194         ulXSquare *= ulXSquare;
00195         for (sY = 0; sY <= (K_SHORT)pstCircle_>usRadius; sY++)
00196         {
00197             ulYSquare = (K_ULONG)sY;
00198             ulYSquare *= ulYSquare;
00199
00200             // if filled...
00201             if (pstCircle_>bFill == true)
00202             {
00203                 stPoint.uColor = pstCircle_>uFillColor;
00204                 if (ulXSquare + ulYSquare <= ulRadSquare)
00205                 {
00206                     // Draw the fill color at the appropriate locations
00207                     (quadrature...)
00208                     stPoint.usX = pstCircle_>usX + sX;
00209                     stPoint.usY = pstCircle_>usY + sY;
00210                     DrawPixel(&stPoint);
00211                     stPoint.usX = pstCircle_>usX - sX;
00212                     stPoint.usY = pstCircle_>usY + sY;
00213                     DrawPixel(&stPoint);
00214                     stPoint.usX = pstCircle_>usX + sX;
00215                     stPoint.usY = pstCircle_>usY - sY;
00216                     DrawPixel(&stPoint);
00217                     stPoint.usX = pstCircle_>usX - sX;
00218                     stPoint.usY = pstCircle_>usY - sY;
00219                     DrawPixel(&stPoint);
00220                 }
00221             }
00222             // Check for edge...
00223             if (
                ((ulXSquare + ulYSquare) >= (ulRadSquare-pstCircle_>usRadius
00224 )) &&
                ((ulXSquare + ulYSquare) <= (ulRadSquare+pstCircle_>usRadius
00225 ))
00226             )
00227             {
00228                 stPoint.uColor = pstCircle_>uLineColor;
00229
00230                 // Draw the fill color at the appropriate locations
00231                 (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 void GraphicsDriver::Ellipse(DrawEllipse_t
    *pstEllipse_)
00250 {

```

```

00250     DrawPoint_t stPoint;
00251     K_SHORT sX;
00252     K_SHORT sY;
00253     K_ULONG ulRadius;
00254     K_ULONG ulHSquare;
00255     K_ULONG ulVSquare;
00256     K_ULONG ulXSquare;
00257     K_ULONG ulYSquare;
00258
00259     ulHSquare = (K_ULONG)pstEllipse->usWidth;
00260     ulHSquare *= ulHSquare;
00261
00262     ulVSquare = (K_ULONG)pstEllipse->usHeight;
00263     ulVSquare *= ulVSquare;
00264
00265     ulRadius = ulHSquare * ulVSquare;
00266
00267     for (sX = 0; sX <= (K_SHORT)pstEllipse->usWidth; sX++)
00268     {
00269         ulXSquare = (K_ULONG)sX;
00270         ulXSquare *= ulXSquare;
00271         ulXSquare *= ulHSquare;
00272
00273         for (sY = 0; sY <= (K_SHORT)pstEllipse->usHeight; sY++)
00274         {
00275             ulYSquare = (K_ULONG)sY;
00276             ulYSquare *= ulYSquare;
00277             ulYSquare *= ulVSquare;
00278
00279             if ((ulXSquare + ulYSquare) <= ulRadius)
00280             {
00281                 // Draw the fill color at the appropriate locations
00282                 (quadrature...)
00283                 stPoint.usX = pstEllipse->usX + sX;
00284                 stPoint.usY = pstEllipse->usY + sY;
00285                 DrawPixel(&stPoint);
00286                 stPoint.usX = pstEllipse->usX - sX;
00287                 stPoint.usY = pstEllipse->usY + sY;
00288                 DrawPixel(&stPoint);
00289                 stPoint.usX = pstEllipse->usX + sX;
00290                 stPoint.usY = pstEllipse->usY - sY;
00291                 DrawPixel(&stPoint);
00292                 stPoint.usX = pstEllipse->usX - sX;
00293                 stPoint.usY = pstEllipse->usY - sY;
00294                 DrawPixel(&stPoint);
00295             }
00296         }
00297     }
00298
00299     //-----
00300 void GraphicsDriver::Bitmap(DrawBitmap_t *
    pstBitmap_)
00301 {
00302     K_USHORT usRow;
00303     K_USHORT usCol;
00304
00305     K_USHORT usIndex;
00306
00307     K_UCHAR ucRed = 0;
00308     K_UCHAR ucBlue = 0;
00309     K_UCHAR ucGreen = 0;
00310
00311     DrawPoint_t stPoint;
00312
00313     usIndex = 0;
00314     for (usRow = pstBitmap_->usY; usRow < (pstBitmap_->usY + pstBitmap_->
    usHeight); usRow++)
00315     {
00316         for (usCol = pstBitmap_->usX; usCol < (pstBitmap_->usX +
    pstBitmap_->usWidth); usCol++)
00317         {
00318
00319             stPoint.usX = usCol;
00320             stPoint.usY = usRow;
00321
00322             // Build the color based on the bitmap value... This algorithm
00323             // is slow, but it automatically converts any 8/16/24 bit bitmap
00324             into the // current colorspace defined...
00325             switch(pstBitmap_->ucBPP)
00326             {
00327                 case 1:
00328                 {
00329                     // 3:2:3, RGB
00330                     ucRed = ((pstBitmap_->pucData[usIndex]) & 0xE0)
00331                     << 1;

```

```

00331         ucGreen  = ((pstBitmap->pucData[usIndex]) & 0x18)
00332     << 3;
00333         ucBlue   = ((pstBitmap->pucData[usIndex]) & 0x07)
00334     << 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 // Convert the R,G,B values into the correct colorspace for display
00369 #if DRAW_COLOR_2BIT
00370 //1-bit
00371 ucRed >>= 7;
00372 ucGreen >>= 7;
00373 ucBlue >>= 7;
00374 #elif DRAW_COLOR_8BIT
00375 //3:2:3 R:G:B
00376 ucRed >>= 5;
00377 ucGreen >>= 6;
00378 ucBlue >>= 5;
00379 #elif DRAW_COLOR_16BIT
00380 //5:6:5 R:G:B
00381 ucRed >>= 3;
00382 ucGreen >>= 2;
00383 ucBlue >>= 3;
00384 #elif DRAW_COLOR_24BIT
00385 // No conversion required
00386 #endif
00387 // Build the color.
00388 stPoint.uColor = RGB_COLOR(ucRed,ucGreen,ucBlue);
00389 // Draw the point.
00390 DrawPixel(&stPoint);
00391 // Stamps are opaque, don't fill in the BG
00392 usIndex += m_ucBPP / 8;
00393 }
00394 }
00395 }
00396 }
00397
00398 //-----
00399 void GraphicsDriver::Stamp(DrawStamp_t *
00400     pstStamp_)
00401 {
00402     K_USHORT usRow;
00403     K_USHORT usCol;
00404     K_USHORT usShift;
00405     K_USHORT usIndex;
00406     DrawPoint_t stPoint;
00407
00408     usIndex = 0;
00409     for (usRow = pstStamp_->usY; usRow < (pstStamp_->usY + pstStamp_->
00410         usHeight); usRow++)
00411     {
00412         usShift = 0x80;
00413         for (usCol = pstStamp_->usX; usCol < (pstStamp_->usX + pstStamp_-
00414             >usWidth); usCol++)

```

```

00413         // If the packed bit in the bitmap is a "1", draw the color.
00414         if (pstStamp_>pucData[usIndex] & usShift)
00415         {
00416             stPoint.usX = usCol;
00417             stPoint.usY = usRow;
00418             stPoint.uColor = pstStamp_>uColor;
00419             DrawPixel(&stPoint);
00420         }
00421         // Stamps are opaque, don't fill in the BG
00422
00423         // Shift to the next bit in the field
00424         usShift >>= 1;
00425
00426         // Rollover - next bit in the bitmap.
00427         // This obviously works best for stamps that are multiples of 8x8
00428         if (usShift == 0)
00429         {
00430             usShift = 0x80;
00431             usIndex++;
00432         }
00433     }
00434 }
00435 }
00436
00437 //-----
00438 void GraphicsDriver::Move( DrawMove_t *pstMove_ )
00439 {
00440     DrawPoint_t stPoint;
00441     K_LONG sX;
00442     K_LONG sY;
00443     K_LONG sXInc = 0;
00444     K_LONG sYInc = 0;
00445
00446     K_BOOL bLeftToRight = false;
00447     K_BOOL bTopToBottom = false;
00448
00449     if (pstMove_>usSrcX > pstMove_>usDstX)
00450     {
00451         bLeftToRight = true;
00452     }
00453     if (pstMove_>usSrcY > pstMove_>usDstY)
00454     {
00455         bTopToBottom = true;
00456     }
00457
00458     if (bLeftToRight)
00459     {
00460         sXInc++;
00461     }
00462     else
00463     {
00464         sXInc--;
00465         pstMove_>usSrcX += pstMove_>usCopyWidth - 1;
00466         pstMove_>usDstX += pstMove_>usCopyWidth - 1;
00467     }
00468
00469     if (bTopToBottom)
00470     {
00471         sYInc++;
00472     }
00473     else
00474     {
00475         sYInc--;
00476         pstMove_>usSrcY += pstMove_>usCopyHeight - 1;
00477         pstMove_>usDstY += pstMove_>usCopyHeight - 1;
00478     }
00479
00480     // Hideously inefficient memory move...
00481     for (sX = 0; sX < pstMove_>usCopyWidth; sX++)
00482     {
00483         for (sY = 0; sY < pstMove_>usCopyHeight; sY++)
00484         {
00485             // Read from source (value read into the point struct)
00486             stPoint.usY = (K_USHORT) ((K_LONG)pstMove_>usSrcY + ((
K_LONG)sY * sYInc));
00487             stPoint.usX = (K_USHORT) ((K_LONG)pstMove_>usSrcX + ((
K_LONG)sX * sXInc));
00488             ReadPixel(&stPoint);
00489
00490             // Copy to dest
00491             stPoint.usY = (K_USHORT) ((K_LONG)pstMove_>usDstY + ((
K_LONG)sY * sYInc));
00492             stPoint.usX = (K_USHORT) ((K_LONG)pstMove_>usDstX + ((
K_LONG)sX * sXInc));
00493             DrawPixel(&stPoint);
00494         }
00495     }

```

```

00496 }
00497
00498 //-----
00499 void GraphicsDriver::Text(DrawText_t *pstText_)
00500 {
00501     K_USHORT usX, usY;
00502     K_USHORT usStartX;
00503     K_USHORT usStartY;
00504     K_USHORT usCharOffsetX;
00505     K_USHORT usCharIndex = 0;
00506     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00507     DrawPoint_t stPoint;
00508
00509     // set the color for this element.
00510     stPoint.uColor = pstText_>uColor;
00511
00512     usCharOffsetX = 0;
00513
00514     // Draw every character in the string, one at a time
00515     while (pstText_>pcString[usCharIndex] != 0)
00516     {
00517         K_USHORT usOffset = 0;
00518
00519         K_UCHAR ucWidth;
00520         K_UCHAR ucHeight;
00521         K_UCHAR ucVOffset;
00522         K_UCHAR ucBitmask;
00523
00524         // Read the glyphs from memory until we arrive at the one we wish to
00525         print
00526         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00527         {
00528             // Glyphs are variable-sized for efficiency - to look up a
00529             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             // Header information: glyph size and vertical offset
00539             ucWidth = Font_ReadByte(usOffset++, pucData);
00540             ucHeight = Font_ReadByte(usOffset++, pucData);
00541             ucVOffset = Font_ReadByte(usOffset++, pucData);
00542
00543             usStartY = pstText_>usTop + (K_USHORT)ucVOffset;
00544             usStartX = pstText_>usLeft;
00545
00546             // Draw the font from left->right, top->bottom
00547             for ( usY = usStartY;
00548                  usY < usStartY + (K_USHORT)ucHeight;
00549                  usY++ )
00550             {
00551                 K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
00552                 ucBitmask = 0x80;
00553
00554                 for ( usX = usCharOffsetX + usStartX;
00555                      usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
00556                      usX++ )
00557                 {
00558                     if (!ucBitmask)
00559                     {
00560                         ucBitmask = 0x80;
00561                         usOffset++;
00562                         ucTempChar = Font_ReadByte(usOffset, pucData);
00563                     }
00564
00565                     if (ucTempChar & ucBitmask)
00566                     {
00567                         // Update the location
00568                         stPoint.usX = usX;
00569                         stPoint.usY = usY;
00570
00571                         // Draw the point.
00572                         DrawPixel(&stPoint);
00573                     }
00574
00575                     ucBitmask >>= 1;
00576                 }
00577
00578                 usOffset++;
00579             }
00580
00581             // Next character

```

```

00581         usCharIndex++;
00582         usCharOffsetX += (K_USHORT)ucWidth + 1;
00583     }
00584 }
00585
00586 //-----
00587 K_USHORT GraphicsDriver::TextWidth(DrawText_t *pstText_)
00588 {
00589     K_USHORT usCharOffsetX;
00590     K_USHORT usCharIndex = 0;
00591     K_USHORT usX;
00592     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00593
00594     usCharOffsetX = 0;
00595
00596     // Draw every character in the string, one at a time
00597     while (pstText_>pcString[usCharIndex] != 0)
00598     {
00599         K_USHORT usOffset = 0;
00600
00601         K_UCHAR ucWidth;
00602         K_UCHAR ucHeight;
00603
00604         // Read the glyphs from memory until we arrive at the one we wish to
00605         print for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00606         {
00607             // Glyphs are variable-sized for efficiency - to look up a
00608             particular // glyph, we must traverse all preceding glyphs in the list
00609             ucWidth = Font_ReadByte(usOffset, pucData);
00610             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00611
00612             // Adjust the offset to point to the next glyph
00613             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00614                 + (sizeof(Glyph_t) - 1);
00615         }
00616
00617         // Header information: glyph size and vertical offset
00618         ucWidth = Font_ReadByte(usOffset, pucData);
00619         usOffset += (sizeof(Glyph_t) - 1);
00620
00621         // Next character
00622         usCharIndex++;
00623         usCharOffsetX += (K_USHORT)ucWidth + 1;
00624     }
00625
00626     return usCharOffsetX;
00627 }
00628
00629 //-----
00630 void GraphicsDriver::TriangleWire(DrawPoly_t
00631 *pstPoly_)
00632 {
00633     DrawLine_t stLine;
00634
00635     stLine.uColor = pstPoly_>uColor;
00636
00637     stLine.usX1 = pstPoly_>pstVector[0].usX;
00638     stLine.usY1 = pstPoly_>pstVector[0].usY;
00639     stLine.usX2 = pstPoly_>pstVector[1].usX;
00640     stLine.usY2 = pstPoly_>pstVector[1].usY;
00641     Line(&stLine);
00642
00643     stLine.usX1 = pstPoly_>pstVector[1].usX;
00644     stLine.usY1 = pstPoly_>pstVector[1].usY;
00645     stLine.usX2 = pstPoly_>pstVector[2].usX;
00646     stLine.usY2 = pstPoly_>pstVector[2].usY;
00647     Line(&stLine);
00648
00649     stLine.usX1 = pstPoly_>pstVector[2].usX;
00650     stLine.usY1 = pstPoly_>pstVector[2].usY;
00651     stLine.usX2 = pstPoly_>pstVector[0].usX;
00652     stLine.usY2 = pstPoly_>pstVector[0].usY;
00653     Line(&stLine);
00654 }
00655 //-----
00656 void GraphicsDriver::TriangleFill(DrawPoly_t
00657 *pstPoly_)
00658 {
00659     // Drawing a raster-filled triangle:
00660     K_UCHAR ucMaxEdge = 0;
00661     K_UCHAR ucMinEdge1 = 0, ucMinEdge2 = 0;
00662     K_SHORT sMax = 0;
00663     K_SHORT sTemp;
00664
00665     K_SHORT sDeltaX1, sDeltaX2;

```

```

00664     K_SHORT sDeltaY1, sDeltaY2;
00665     K_CHAR cStepX1, cStepX2;
00666     K_CHAR cStepY;
00667     K_SHORT sX1, sX2, sX3, sY1, sY2, sY3;
00668     K_SHORT sTempX1, sTempY1, sTempX2, sTempY2;
00669     K_SHORT sFraction1;
00670     K_SHORT sFraction2;
00671     K_SHORT i;
00672     DrawPoint_t stPoint;
00673
00674     // Figure out which line segment is the longest
00675     sTemp = (K_SHORT)pstPoly->pstVector[0].usY - (K_SHORT)pstPoly->
pstVector[1].usY;
00676     if( sTemp < 0 ) { sTemp = -sTemp; }
00677     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 0; ucMinEdge1 = 1;
ucMinEdge2 = 2; }
00678
00679     sTemp = (K_SHORT)pstPoly->pstVector[1].usY - (K_SHORT)pstPoly->
pstVector[2].usY;
00680     if( sTemp < 0 ) { sTemp = -sTemp; }
00681     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 1; ucMinEdge1 = 2;
ucMinEdge2 = 0; }
00682
00683     sTemp = (K_SHORT)pstPoly->pstVector[2].usY - (K_SHORT)pstPoly->
pstVector[0].usY;
00684     if( sTemp < 0 ) { sTemp = -sTemp; }
00685     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 2; ucMinEdge1 = 0;
ucMinEdge2 = 1; }
00686
00687     // Label the vectors and copy into temporary signed buffers
00688     sX1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usX;
00689     sX2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usX;
00690     sX3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usX;
00691
00692     sY1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usY;
00693     sY2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usY;
00694     sY3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usY;
00695
00696     // Figure out whether or not we're drawing up-down or down-up
00697     sDeltaY1 = sY1 - sY2;
00698     if (sDeltaY1 < 0) { cStepY = -1; sDeltaY1 = -sDeltaY1; } else { cStepY = 1;
}
00699
00700     sDeltaX1 = sX1 - sX2;
00701     if (sDeltaX1 < 0) { cStepX1 = -1; sDeltaX1 = -sDeltaX1; } else { cStepX1 =
1; }
00702
00703     sDeltaY2 = sY1 - sY3;
00704     if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1;
}
00705
00706     sDeltaX2 = sX1 - sX3;
00707     if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 =
1; }
00708
00709     sDeltaX1 <=<=1;
00710     sDeltaX2 <=<=1;
00711     sDeltaY1 <=<=1;
00712     sDeltaY2 <=<=1;
00713
00714     sFraction1 = sDeltaX1;// - (sDeltaY1 >> 1);
00715     sFraction2 = sDeltaX2;// - (sDeltaY2 >> 1);
00716
00717     sTempY1 = sY1;
00718     sTempY2 = sY1;
00719     sTempX1 = sX1;
00720     sTempX2 = sX1;
00721
00722     stPoint.uColor = pstPoly->uColor;
00723
00724     if( sDeltaY2 != 0 )
00725     {
00726         while (sTempY2 != sY3)
00727         {
00728             stPoint.usY = sTempY2;
00729             if( sTempX1 < sTempX2 ) {
00730                 for( i = sTempX1; i <= sTempX2; i++) {
00731                     stPoint.usX = i;
00732                     Point(&stPoint);
00733                 }
00734             } else {
00735                 for( i = sTempX2; i <= sTempX1; i++ ) {
00736                     stPoint.usX = i;
00737                     Point(&stPoint);
00738                 }
00739             }
00740

```



```

00741         while (sFraction2 >= sDeltaY2)
00742         {
00743             sTempX2 -= cStepX2;
00744             sFraction2 -= sDeltaY2;
00745         }
00746         sTempY2 -= cStepY;
00747         sFraction2 += sDeltaX2;
00748
00749         while (sFraction1 >= sDeltaY1)
00750         {
00751             sTempX1 -= cStepX1;
00752             sFraction1 -= sDeltaY1;
00753         }
00754         sTempY1 -= cStepY;
00755         sFraction1 += sDeltaX1;
00756     }
00757 }
00758
00759 sDeltaY2 = sY3 - sY2;
00760 sDeltaX2 = sX3 - sX2;
00761
00762 if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 =
00763 1; }
00764 if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY =
00765 1; }
00766
00767 sDeltaX2 <=1;
00768 sDeltaY2 <=1;
00769
00770 sFraction2 = sDeltaX2; // - (sDeltaY2 >> 1);
00771
00772 sTempY2 = sY3;
00773 sTempX2 = sX3;
00774
00775 if( sDeltaY2 != 0)
00776 {
00777     while (sTempY2 != sY2)
00778     {
00779         stPoint.usY = sTempY2;
00780         if( sTempX1 < sTempX2 ) {
00781             for( i = sTempX1; i <= sTempX2; i++) {
00782                 stPoint.usX = i;
00783                 Point(&stPoint);
00784             }
00785         } else {
00786             for( i = sTempX2; i <= sTempX1; i++ ) {
00787                 stPoint.usX = i;
00788                 Point(&stPoint);
00789             }
00790         }
00791     }
00792     while (sFraction2 >= sDeltaY2)
00793     {
00794         sTempX2 -= cStepX2;
00795         sFraction2 -= sDeltaY2;
00796     }
00797     sTempY2 -= cStepY;
00798     sFraction2 += sDeltaX2;
00799
00800     while (sFraction1 >= sDeltaY1)
00801     {
00802         sTempX1 -= cStepX1;
00803         sFraction1 -= sDeltaY1;
00804     }
00805     sTempY1 -= cStepY;
00806     sFraction1 += sDeltaX1;
00807 }
00808
00809 //-----
00810 void GraphicsDriver::Polygon(DrawPoly_t *pstPoly_)
00811 {
00812     K_USHORT i,j,k;
00813     K_BOOL bState = false;
00814
00815     DrawPoly_t stTempPoly;
00816     DrawVector_t astTempVec[3];
00817
00818     if (pstPoly_>usNumPoints < 3)
00819     {
00820         return;
00821     }
00822
00823     stTempPoly.uColor = pstPoly_>uColor;
00824     stTempPoly.bFill = pstPoly_>bFill;
00825     stTempPoly.pstVector = astTempVec;

```

```

00826     stTempPoly.usNumPoints = 3;
00827
00828     astTempVec[0].usX = pstPoly_>pstVector[0].usX;
00829     astTempVec[1].usX = pstPoly_>pstVector[1].usX;
00830     astTempVec[0].usY = pstPoly_>pstVector[0].usY;
00831     astTempVec[1].usY = pstPoly_>pstVector[1].usY;
00832
00833     j = 2;
00834     astTempVec[2].usX = pstPoly_>pstVector[pstPoly_>usNumPoints
- 1].usX;
00835     astTempVec[2].usY = pstPoly_>pstVector[pstPoly_>usNumPoints
- 1].usY;
00836
00837     k = pstPoly_>usNumPoints - 2;
00838
00839     if( pstPoly_>bFill )
00840     {
00841         TriangleFill(&stTempPoly);
00842     }
00843     else
00844     {
00845         TriangleWire(&stTempPoly);
00846     }
00847
00848     // Filled polygon/wireframe polygon using triangle decomp.
00849     for(i = 0; i < pstPoly_>usNumPoints - 3; i++)
00850     {
00851         astTempVec[0].usX = astTempVec[1].usX;
00852         astTempVec[1].usX = astTempVec[2].usX;
00853         astTempVec[0].usY = astTempVec[1].usY;
00854         astTempVec[1].usY = astTempVec[2].usY;
00855
00856         if( !bState )
00857         {
00858             bState = true;
00859             astTempVec[2].usX = pstPoly_>pstVector[j].usX;
00860             astTempVec[2].usY = pstPoly_>pstVector[j].usY;
00861             j++;
00862         }
00863         else
00864         {
00865             bState = false;
00866             astTempVec[2].usX = pstPoly_>pstVector[k].usX;
00867             astTempVec[2].usY = pstPoly_>pstVector[k].usY;
00868             k--;
00869         }
00870         if( pstPoly_>bFill )
00871         {
00872             TriangleFill(&stTempPoly);
00873         }
00874         else
00875         {
00876             TriangleWire(&stTempPoly);
00877         }
00878     }
00879 }
00880
00881 //-----
00882 void GraphicsDriver::SetWindow(DrawWindow_t
*pstWindow_)
00883 {
00884     if ((pstWindow_>usLeft <= pstWindow_>usRight) &&
00885         (pstWindow_>usRight < m_usResX) &&
00886         (pstWindow_>usLeft < m_usResX))
00887     {
00888         m_usLeft = pstWindow_>usLeft;
00889         m_usRight = pstWindow_>usRight;
00890     }
00891
00892     if ((pstWindow_>usTop <= pstWindow_>usBottom) &&
00893         (pstWindow_>usTop < m_usTop) &&
00894         (pstWindow_>usBottom < m_usBottom))
00895     {
00896         m_usTop = pstWindow_>usTop;
00897         m_usBottom = pstWindow_>usBottom;
00898     }
00899
00900 }
00901
00902 //-----
00903 void GraphicsDriver::ClearWindow()
00904 {
00905     m_usLeft = 0;
00906     m_usTop = 0;
00907     m_usRight = m_usResX - 1;
00908     m_usBottom = m_usResY - 1;
00909 }

```

14.47 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h File Reference

Graphics driver class declaration.

```
#include "driver.h"
#include "draw.h"
```

Classes

- class [GraphicsDriver](#)

Defines the base graphics driver class, which is inherited by all other graphics drivers.

14.47.1 Detailed Description

Graphics driver class declaration.

Definition in file [graphics.h](#).

14.48 graphics.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 __GRAPHICSX_H__
00020  #define __GRAPHICSX_H__
00021
00022  #include "driver.h"
00023  #include "draw.h"
00024
00025  //-----
00032  class GraphicsDriver : public Driver
00033  {
00034  public:
00035  //-----
00036  /*
00037       The base graphics driver does not implement the set of
00038       virtual methods inherited from the Driver class. This
00039       is left to the actual hardware implementation.
00040  */
00041  //-----
00042
00049       virtual void DrawPixel(DrawPoint_t *pstPoint_) {};
00050
00058       virtual void ReadPixel(DrawPoint_t *pstPoint_) {};
00059
00060  //-----
00061  /*
00062       Raster operations defined using per-pixel rendering.
00063       Can be overridden in inheriting classes.
00064  */
00065  //-----
00071       virtual void ClearScreen();
00072
00078       virtual void Point(DrawPoint_t *pstPoint_);
00079
00085       virtual void Line(DrawLine_t *pstLine_);
00086
00092       virtual void Rectangle(DrawRectangle_t *
                                pstRectangle_);
```

```

00093
00099     virtual void Circle(DrawCircle_t *pstCircle_);
00100
00106     virtual void Ellipse(DrawEllipse_t *pstEllipse_);
00107
00113     virtual void Bitmap(DrawBitmap_t *pstBitmap_);
00114
00120     virtual void Stamp(DrawStamp_t *pstStamp_);
00121
00131     virtual void Move(DrawMove_t *pstMove_ );
00132
00138     virtual void TriangleWire(DrawPoly_t *pstPoly_);
00139
00145     virtual void TriangleFill(DrawPoly_t *pstPoly_);
00146
00152     virtual void Polygon(DrawPoly_t *pstPoly_);
00153
00159     virtual void Text(DrawText_t *pstText_);
00160
00167     virtual K_USHORT TextWidth(DrawText_t *pstText_);
00168
00174     void SetWindow( DrawWindow_t *pstWindow_ );
00175
00181     void ClearWindow();
00182 protected:
00183
00184     K_USHORT m_usResX;
00185     K_USHORT m_usResY;
00186
00187     K_USHORT m_usLeft;
00188     K_USHORT m_usTop;
00189     K_USHORT m_usRight;
00190     K_USHORT m_usBottom;
00191
00192     K_UCHAR m_ucBPP;
00193 };
00194
00195 #endif
00196

```

14.49 /home/moslevin/googlecode/mark3/trunk/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"
#include <stdio.h>

```

14.49.1 Detailed Description

Graphical User Interface classes and data structure definitions.

Definition in file [gui.cpp](#).

14.50 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 =====*/
00019 #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 #include <stdio.h>
00027 //-----
00028 void GuiWindow::AddControl( GuiControl *
    pclControl_, GuiControl *pclParent_ )
00029 {
00030     GUI_DEBUG_PRINT("GuiWindow::AddControl\n");
00031
00032     m_clControlList.Add(static_cast<LinkListNode*>(
    pclControl_));
00033     m_pclInFocus = pclControl_;
00034     m_ucControlCount++;
00035
00036     pclControl_>SetParentWindow(this);
00037     pclControl_>SetParentControl(pclParent_);
00038 }
00039
00040 //-----
00041 void GuiWindow::RemoveControl( GuiControl *
    pclControl_ )
00042 {
00043     GUI_DEBUG_PRINT("GuiWindow::RemoveControl\n");
00044
00045     if (pclControl_>GetPrev())
00046     {
00047         m_pclInFocus = static_cast<GuiControl*>(
    pclControl_>GetPrev());
00048     }
00049     else if (pclControl_>GetNext())
00050     {
00051         m_pclInFocus = static_cast<GuiControl*>(
    pclControl_>GetNext());
00052     }
00053     else
00054     {
00055         m_pclInFocus = NULL;
00056     }
00057     m_clControlList.Remove(static_cast<LinkListNode*>(
    pclControl_));
00058     m_ucControlCount--;
00059 }
00060
00061 //-----
00062 K_UCHAR GuiWindow::GetMaxZOrder()
00063 {
00064     GUI_DEBUG_PRINT("GuiWindow::GetMaxZOrder\n");
00065
00066     LinkListNode *pclTempNode;
00067     K_UCHAR ucZ = 0;
00068     K_UCHAR ucTempZ;
00069
00070     pclTempNode = m_clControlList.GetHead();
00071
00072     while (pclTempNode)
00073     {
00074         ucTempZ = (static_cast<GuiControl*>(pclTempNode))>GetZOrder(
    );
00075         if (ucTempZ > ucZ)
00076         {
00077             ucZ = ucTempZ;
00078         }
00079         pclTempNode = pclTempNode->GetNext();
00080     }
00081
00082     return ucZ;
00083 }
00084
00085 //-----
00086 void GuiWindow::Redraw( K_BOOL bRedrawAll_ )
00087 {
00088     GUI_DEBUG_PRINT("GuiWindow::Redraw\n");
00089
00090     K_UCHAR ucControlsLeft = m_ucControlCount;
00091     K_UCHAR ucCurrentZ = 0;
00092     K_UCHAR ucMaxZ;
00093

```

```

00094     ucMaxZ = GetMaxZOrder();
00095
00096     // While there are still controls left to process (and we're less than
00097     // the maximum Z-order, just a sanity check.), redraw each object that
00098     // has its stale flag set, or all controls if the bRedrawAll_ parameter
00099     // is true.
00100     while (ucControlsLeft && (ucCurrentZ <= ucMaxZ))
00101     {
00102         LinkListNode *pclTempNode;
00103
00104         pclTempNode = m_clControlList.GetHead();
00105         while (pclTempNode)
00106         {
00107             GuiControl* pclTempControl = static_cast<GuiControl
00108             *>(pclTempNode);
00109             if (pclTempControl->GetZOrder() == ucCurrentZ)
00110             {
00111                 if ((bRedrawAll_) || (pclTempControl->IsStale()))
00112                 {
00113                     pclTempControl->Draw();
00114                     pclTempControl->ClearStale();
00115                 }
00116                 ucControlsLeft--;
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 //-----
00128 void GuiWindow::ProcessEvent( GuiEvent_t *
00129     pstEvent_ )
00130 {
00131     GUI_DEBUG_PRINT("GuiWindow::ProcessEvent\n");
00132
00133     // If the event is for broadcast - send it to all controls,
00134     // without regard to order.
00135     if ((TARGET_ID_BROADCAST == pstEvent_->ucTargetID)
00136         || (TARGET_ID_BROADCAST_Z == pstEvent_->ucTargetID)
00137     )
00138     {
00139         GUI_DEBUG_PRINT(" TARGET_ID_BROADCAST(_Z)\n");
00140
00141         LinkListNode *pclTempNode;
00142         pclTempNode = m_clControlList.GetHead();
00143         while (pclTempNode)
00144         {
00145             GuiReturn_t eRet;
00146             eRet = (static_cast<GuiControl*>(pclTempNode))->
00147             ProcessEvent(pstEvent_);
00148             if (GUI_EVENT_CONSUMED == eRet)
00149             {
00150                 break;
00151             }
00152             pclTempNode = pclTempNode->GetNext();
00153         }
00154         // Send the event only to the currently-selected object.
00155         else if (TARGET_ID_FOCUS == pstEvent_->ucTargetID)
00156         {
00157             GUI_DEBUG_PRINT(" TARGET_ID_FOCUS\n");
00158             GuiReturn_t eReturn = GUI_EVENT_OK;
00159
00160             // Try to let the control process the event on its own
00161             if (m_pclInFocus)
00162             {
00163                 eReturn = m_pclInFocus->ProcessEvent(
00164                 pstEvent_);
00165             }
00166             // If the event was not consumed, use default logic to process the
00167             event
00168             if (GUI_EVENT_CONSUMED != eReturn)
00169             {
00170                 if (EVENT_TYPE_KEYBOARD == pstEvent_->
00171                 ucEventType)
00172                 {
00173                     if (KEYCODE_TAB == pstEvent_->stKey.ucKeyCode)
00174                     {
00175                         if (pstEvent_->stKey.bKeyState)
00176                         {

```

```

00174         CycleFocus(true);
00175     }
00176 }
00177 }
00178 else if (EVENT_TYPE_JOYSTICK == pstEvent_>
ucEventType)
00179 {
00180     if (pstEvent_>stJoystick.bUp || pstEvent_>stJoystick.bLeft
)
00181     {
00182         // Cycle focus *backwards*
00183         CycleFocus(false);
00184     }
00185     else if (pstEvent_>stJoystick.bRight || pstEvent_>
stJoystick.bDown)
00186     {
00187         // Cycle focus *forewards*
00188         CycleFocus(true);
00189     }
00190 }
00191 }
00192 }
00193 else if (TARGET_ID_HIGH_Z == pstEvent_>ucTargetID)
00194 {
00195     GUI_DEBUG_PRINT(" TARGET_ID_HIGH_Z\n");
00196
00197     K_USHORT usTargetX, usTargetY;
00198     K_USHORT usOffsetX, usOffsetY;
00199     K_UCHAR ucMaxZ = 0;
00200
00201     LinkListNode *pclTempNode;
00202     pclTempNode = m_clControlList.GetHead();
00203
00204     switch (pstEvent_>ucEventType)
00205     {
00206     case EVENT_TYPE_MOUSE:
00207     case EVENT_TYPE_TOUCH:
00208     {
00209         GuiControl *pclTargetControl = NULL;
00210
00211         // Read the target X/Y coordinates out of the event struct
00212         if (EVENT_TYPE_TOUCH == pstEvent_>ucEventType)
00213         {
00214             usTargetX = pstEvent_>stTouch.usX;
00215             usTargetY = pstEvent_>stTouch.usY;
00216         }
00217         else
00218         {
00219             usTargetX = pstEvent_>stMouse.usX;
00220             usTargetY = pstEvent_>stMouse.usY;
00221         }
00222
00223         // Go through every control on the window, checking to see if
the
00224         // event falls within the bounding box
00225         while (pclTempNode)
00226         {
00227             GuiReturn_t eRet;
00228             GuiControl *pclControl = (static_cast<GuiControl
*>(pclTempNode));
00229
00230             pclControl->GetControlOffset(&usOffsetX, &
usOffsetY);
00231
00232             // Compare event coordinates to bounding box (with offsets)
00233             if ( ((usTargetX >= (usOffsetX + pclControl->GetLeft
()) &&
00234                 (usTargetX <= (usOffsetX + pclControl->GetLeft
() + pclControl->GetWidth() - 1))) &&
00235                 ((usTargetY >= (usOffsetY + pclControl->GetTop()
) &&
00236                 (usTargetY <= (usOffsetY + pclControl->GetTop()
+ pclControl->GetHeight() - 1))) )
00237             {
00238                 // If this control is higher in Z-Order, set this as
the newest
00239                 // candidate control to accept the event
00240                 if (pclControl->GetZOrder() >= ucMaxZ)
00241                 {
00242                     pclTargetControl = pclControl;
00243                     ucMaxZ = pclControl->GetZOrder();
00244                 }
00245             }
00246
00247             pclTempNode = pclTempNode->GetNext();
00248         }
00249     }

```

```

00250         // If a suitable control was found on the event surface, pass
00251         the event off // for processing.
00252         if (pclTargetControl)
00253         {
00254             // If the selected control is different from the current
00255             in-focus // control, then deactivate that control.
00256             if (m_pclInFocus && (m_pclInFocus
!= pclTargetControl))
00257             {
00258                 m_pclInFocus->Activate(false);
00259                 m_pclInFocus = NULL;
00260             }
00261             (static_cast<GuiControl*>(pclTargetControl))->
ProcessEvent(pstEvent_);
00262         }
00263     }
00264     break;
00265     default:
00266         break;
00267 }
00268 }
00269 }
00270 //-----
00271 void GuiWindow::SetFocus( GuiControl *pclControl_
)
00272 {
00273     GUI_DEBUG_PRINT("GuiWindow::SetFocus\n");
00274
00275     m_pclInFocus = pclControl_;
00276 }
00277
00278 //-----
00279 void GuiWindow::CycleFocus( bool bForward_ )
00280 {
00281     GUI_DEBUG_PRINT("GuiWindow::CycleFocus\n");
00282
00283     // Set starting point and cached copy of current nodes
00284     LinkListNode *pclTempNode = static_cast<GuiControl*>{
m_clControlList.GetHead()};
00285     LinkListNode *pclStartNode = m_pclInFocus;
00286
00287     if (bForward_)
00288     {
00289         // If there isn't a current focus node, set the focus to the beginning
00290         // of the list
00291         if (!m_pclInFocus)
00292         {
00293             m_pclInFocus = static_cast<GuiControl*>{
pclTempNode};
00294             if (!m_pclInFocus)
00295             {
00296                 return;
00297             }
00298             pclTempNode = static_cast<GuiControl*>(m_pclInFocus
);
00299             pclStartNode = NULL;
00300         }
00301         else
00302         {
00303             // Deactivate the control that's losing focus
00304             static_cast<GuiControl*>(m_pclInFocus)->
Activate(false);
00305
00306             // Otherwise start with the next node
00307             pclStartNode = pclStartNode->GetNext();
00308         }
00309
00310         // Go through the whole control list and find the next one to accept
00311         // the focus
00312         while (pclTempNode && pclTempNode != pclStartNode)
00313         {
00314             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00315             {
00316                 m_pclInFocus = static_cast<GuiControl*>{
pclTempNode};
00317                 m_pclInFocus->Activate(true);
00318                 SetFocus(m_pclInFocus);
00319                 return;
00320             }
00321             pclTempNode = pclTempNode->GetNext();
00322         }
00323
00324         pclTempNode = static_cast<GuiControl*>(m_clControlList
.GetHead());
00325         while (pclTempNode && pclTempNode != pclStartNode)

```



```

00326     {
00327         if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00328         {
00329             m_pclInFocus = static_cast<GuiControl*>(
00330 pclTempNode);
00331             m_pclInFocus->Activate(true);
00332             SetFocus(m_pclInFocus);
00333             return;
00334         }
00335         pclTempNode = pclTempNode->GetNext();
00336     }
00337     else
00338     {
00339         pclTempNode = static_cast<GuiControl*>(m_clControlList
00340 .GetTail());
00341         pclStartNode = m_pclInFocus;
00342         // If there isn't a current focus node, set the focus to the end
00343         // of the list
00344         if (!m_pclInFocus)
00345         {
00346             m_pclInFocus = static_cast<GuiControl*>(
00347 pclTempNode);
00348             if (!m_pclInFocus)
00349             {
00350                 return;
00351             }
00352             pclTempNode = static_cast<GuiControl*>(m_pclInFocus
00353 );
00354             pclStartNode = NULL;
00355         }
00356         else
00357         {
00358             // Deactivate the control that's losing focus
00359             static_cast<GuiControl*>(m_pclInFocus)->
00360 Activate(false);
00361             // Otherwise start with the previous node
00362             pclStartNode = pclStartNode->GetPrev();
00363         }
00364         // Go through the whole control list and find the next one to accept
00365         // the focus
00366         while (pclTempNode && pclTempNode != pclStartNode)
00367         {
00368             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00369             {
00370                 m_pclInFocus = static_cast<GuiControl*>(
00371 pclTempNode);
00372                 m_pclInFocus->Activate(true);
00373                 SetFocus(m_pclInFocus);
00374                 return;
00375             }
00376             pclTempNode = pclTempNode->GetPrev();
00377         }
00378         pclTempNode = static_cast<GuiControl*>(m_clControlList
00379 .GetTail());
00380         while (pclTempNode && pclTempNode != pclStartNode)
00381         {
00382             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00383             {
00384                 m_pclInFocus = static_cast<GuiControl*>(
00385 pclTempNode);
00386                 m_pclInFocus->Activate(true);
00387                 SetFocus(m_pclInFocus);
00388                 return;
00389             }
00390             pclTempNode = pclTempNode->GetPrev();
00391         }
00392     }
00393 }
00394 //-----
00395 GuiWindow *GuiEventSurface::FindWindowByName
00396 ( const K_CHAR *szName_ )
00397 {
00398     LinkListNode *pclTempNode = static_cast<LinkListNode
00399 *>(m_clWindowList.GetHead());
00400     while (pclTempNode)
00401     {
00402         if (MemUtil::CompareStrings(szName_,
00403 static_cast<GuiWindow*>(pclTempNode)->GetName()))
00404         {
00405             return static_cast<GuiWindow*>(pclTempNode);
00406         }
00407         pclTempNode = pclTempNode->GetNext();
00408     }
00409     return NULL;
00410 }

```

```

00402         pclTempNode = pclTempNode->GetNext();
00403     }
00404
00405     return NULL;
00406 }
00407
00408 //-----
00409 void GuiEventSurface::AddWindow( GuiWindow *
    pclWindow_ )
00410 {
00411     GUI_DEBUG_PRINT("GuiEventSurface::AddWindow\n");
00412
00413     m_clWindowList.Add(static_cast<LinkListNode*>(pclWindow_))
    ;
00414 }
00415
00416 //-----
00417 void GuiEventSurface::RemoveWindow( GuiWindow
    *pclWindow_ )
00418 {
00419     GUI_DEBUG_PRINT("GuiEventSurface::RemoveWindow\n");
00420
00421     m_clWindowList.Remove(static_cast<LinkListNode*>(
    pclWindow_));
00422 }
00423
00424 //-----
00425 K_BOOL GuiEventSurface::SendEvent( GuiEvent_t
    *pstEvent_ )
00426 {
00427     GUI_DEBUG_PRINT("GuiEventSurface::SendEvent\n");
00428
00429     // Allocate a message from the global message pool
00430     Message *pclMessage = GlobalMessagePool::Pop()
    ;
00431
00432     // No messages available? Return a failure
00433     if (!pclMessage)
00434     {
00435         return false;
00436     }
00437
00438     // Allocate a copy of the event from the heap
00439     GuiEvent_t *pstEventCopy = static_cast<GuiEvent_t*>(
    SystemHeap::Alloc(sizeof(GuiEvent_t)));
00440
00441     // If the allocation fails, push the message back to the global pool and
    bail
00442     if (!pstEventCopy)
00443     {
00444         GlobalMessagePool::Push(pclMessage);
00445         return false;
00446     }
00447
00448     // Copy the source event into the destination event buffer
00449     CopyEvent(pstEventCopy, pstEvent_);
00450
00451     // Set the new event as the message payload
00452     pclMessage->SetData(static_cast<void*>(pstEventCopy));
00453
00454     // Send the event to the message queue
00455     m_clMessageQueue.Send(pclMessage);
00456
00457     return true;
00458 }
00459
00460 //-----
00461 K_BOOL GuiEventSurface::ProcessEvent()
00462 {
00463     GUI_DEBUG_PRINT("GuiEventSurface::ProcessEvent\n");
00464
00465     // read the event from the queue (blocking call)
00466     Message *pclMessage = m_clMessageQueue.Receive
    ();
00467     GuiEvent_t stLocalEvent;
00468
00469     // If we failed to get something from the queue,
    // bail out
00470     if (!pclMessage)
00471     {
00472         return false;
00473     }
00474
00475     // Copy the event data from the message into a local copy
00476     CopyEvent(&stLocalEvent,
    static_cast<GuiEvent_t*>(pclMessage->GetData()));
00477
00478
00479

```

```

00480     // Free the message and event as soon as possible, since
00481     // they are shared system resources
00482     SystemHeap::Free(pclMessage->GetData());
00483     GlobalMessagePool::Push(pclMessage);
00484
00485     // Special case check - target ID is the highest Z-ordered window(s) ONLY.
00486     if (stLocalEvent.ucTargetID == TARGET_ID_BROADCAST_Z)
00487     {
00488         LinkListNode* pclTempNode = m_clWindowList.
GetHead();
00489         LinkListNode* pclHighestNode = pclTempNode;
00490         K_UCHAR ucMaxZ = 0;
00491
00492         while (pclTempNode)
00493         {
00494             if (ucMaxZ < (static_cast<GuiWindow*>(pclTempNode))->GetZOrder() )
00495             {
00496                 ucMaxZ = static_cast<GuiWindow*>(pclTempNode)->
GetZOrder();
00497             }
00498             pclTempNode = pclTempNode->GetNext();
00499         }
00500
00501         // Iterate through all windows again - may have multiple windows
00502         // at the same z-order.
00503         pclTempNode = m_clWindowList.GetHead();
00504         while (pclTempNode)
00505         {
00506             if (ucMaxZ == (static_cast<GuiWindow*>(pclTempNode))->GetZOrder())
00507             {
00508                 (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent (&
stLocalEvent);
00509             }
00510             pclTempNode = pclTempNode->GetNext();
00511         }
00512     }
00513     // Broadcast the event - sending it to *all* windows. Let the individual
00514     // windows figure out what to do with the events.
00515     else
00516     {
00517         LinkListNode* pclTempNode = m_clWindowList.
GetHead();
00518         while (pclTempNode)
00519         {
00520             (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent (&
stLocalEvent);
00521             pclTempNode = pclTempNode->GetNext();
00522         }
00523     }
00524
00525     // Return out
00526     return true;
00527 }
00528
00529 //-----
00530 void GuiEventSurface::CopyEvent( GuiEvent_t
*pstDst_, GuiEvent_t *pstSrc_ )
00531 {
00532     GUI_DEBUG_PRINT("GuiEventSurface::CopyEvent\n");
00533     K_UCHAR *pucDst_ = (K_UCHAR*)pstDst_;
00534     K_UCHAR *pucSrc_ = (K_UCHAR*)pstSrc_;
00535     K_UCHAR i;
00536     for (i = 0; i < sizeof(GuiEvent_t); i++)
00537     {
00538         *pucDst_++ = *pucSrc_++;
00539     }
00540 }
00541
00542 //-----
00543 void GuiControl::GetControlOffset( K_USHORT *pusX_,
K_USHORT *pusY_ )
00544 {
00545     GUI_DEBUG_PRINT("GuiControl::GetControlOffset\n");
00546     GuiControl *pclTempControl = m_pclParentControl
;
00547     *pusX_ = 0;
00548     *pusY_ = 0;
00549     while (pclTempControl)
00550     {
00551         *pusX_ += pclTempControl->GetLeft();
00552         *pusY_ += pclTempControl->GetTop();
00553         pclTempControl = pclTempControl->GetParentControl();
00554     }
00555
00556     if (m_pclParentWindow)
00557     {
00558         *pusX_ += m_pclParentWindow->GetLeft();

```

```

00559         *pusY_ += m_pclParentWindow->GetTop();
00560     }
00561 }

```

14.51 /home/moslevin/googlecode/mark3/trunk/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](#)
- 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` }

14.51.1 Detailed Description

Graphical User Interface classes and data structure declarations.

Definition in file [gui.h](#).

14.51.2 Enumeration Type Documentation

14.51.2.1 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](#).

14.52 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 } EVENT_TYPE_COUNT;
00073
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
00107     union
00108     {
00109         K_UCHAR ucFlags;
00110         struct
00111         {
00112             unsigned int bLeftState:1;
00113             unsigned int bRightState:1;
00114             unsigned int bMiddleState:1;
00115             unsigned int bScrollUp:1;
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 //-----
00141 typedef struct
00142 {
00143     union
00144     {
00145         K_USHORT usRawData;
00146         struct
00147         {
00148             unsigned int bUp:1;
00149             unsigned int bDown:1;
00150             unsigned int bLeft:1;
00151             unsigned int bRight:1;
00152             unsigned int bButton1:1;
00153             unsigned int bButton2:1;
00154             unsigned int bButton3:1;
00155             unsigned int bButton4:1;
00156             unsigned int bButton5:1;
00157             unsigned int bButton6:1;
00158             unsigned int bButton7:1;
00159             unsigned int bButton8:1;
00160             unsigned int bButton9:1;
00161             unsigned int bButton10:1;
00162             unsigned int bSelect:1;
00163             unsigned int bStart:1;
00164         };
00165     };
00166 } JoystickEvent_t;
00167
00168 //-----
00169 typedef struct
00170 {
00171     K_USHORT usTicks;
00172 } TimerEvent_t;
00173
00174 //-----
00175 typedef struct
00176 {
00177     K_UCHAR ucEventType;
00178     K_UCHAR ucTargetID;
00179     union
00180     {
00181         KeyEvent_t      stKey;
00182         MouseEvent_t    stMouse;
00183         TouchEvent_t    stTouch;
00184         JoystickEvent_t stJoystick;
00185         TimerEvent_t    stTimer;
00186     };
00187 } GuiEvent_t;
00188
00189 //-----
00190 typedef enum
00191 {
00192     GUI_EVENT_OK = 0,
00193     GUI_EVENT_CONSUMED,
00194     GUI_EVENT_CANCEL,
00195     GUI_EVENT_RETRY,
00196 } GuiReturn_t;
00197
00198 //---
00199 GUI_EVENT_COUNT
00200 } GuiReturn_t;
00201
00202 class GuiControl;
00203
00204 //-----
00205 class GuiWindow : public LinkListNode
00206 {
00207 public:
00208     void Init()
00209     {
00210         m_ucControlCount = 0;
00211         m_pclDriver = NULL;
00212         m_szName = "";
00213     }

```

```

00233
00240 void SetDriver( GraphicsDriver *pclDriver_ ) {
00241     m_pclDriver = pclDriver_; }
00241
00248 GraphicsDriver *GetDriver() { return m_pclDriver
; }
00249
00261 void AddControl( GuiControl *pclControl_, GuiControl
*pclParent_ );
00262
00270 void RemoveControl( GuiControl *pclControl_ );
00271
00279 K_UCHAR GetMaxZOrder();
00280
00289 void Redraw( K_BOOL bRedrawAll_ );
00290
00297 void ProcessEvent( GuiEvent_t *pstEvent_ );
00298
00307 void SetFocus( GuiControl *pclControl_ );
00308
00319 K_BOOL IsInFocus( GuiControl *pclControl_ )
00320 {
00321     if (m_pclInFocus == pclControl_)
00322     {
00323         return true;
00324     }
00325     return false;
00326 }
00327
00333 void SetTop( K_USHORT usTop_ ) { m_usTop = usTop_; }
00334
00340 void SetLeft( K_USHORT usLeft_ ) { m_usLeft = usLeft_;
}
00341
00347 void SetHeight( K_USHORT usHeight_ ) { m_usHeight =
usHeight_; }
00348
00354 void SetWidth( K_USHORT usWidth_ ) { m_usWidth =
usWidth_; }
00355
00361 K_USHORT GetTop() { return m_usTop; }
00362
00368 K_USHORT GetLeft() { return m_usLeft; }
00369
00375 K_USHORT GetHeight() { return m_usHeight; }
00376
00382 K_USHORT GetWidth() { return m_usWidth; }
00383
00387 K_UCHAR GetZOrder() { return m_ucZ; }
00388
00392 void SetZOrder( K_UCHAR ucZ_ ) { m_ucZ = ucZ_; }
00393
00401 void CycleFocus( bool bForward_ );
00402
00406 void SetName( const K_CHAR *szName_ ) { m_szName = szName_;
}
00407
00411 const K_CHAR *GetName() { return m_szName; }
00412
00413 private:
00414     K_USHORT m_usTop;
00415     K_USHORT m_usLeft;
00416     K_USHORT m_usHeight;
00417     K_USHORT m_usWidth;
00418
00419     K_UCHAR m_ucZ;
00420     const K_CHAR *m_szName;
00421
00422     DoubleLinkedList m_clControlList;
00423     GuiControl *m_pclInFocus;
00424     K_UCHAR m_ucControlCount;
00425     GraphicsDriver *m_pclDriver;
00426 };
00427
00428 //-----
00441 class GuiEventSurface
00442 {
00443 public:
00448     void Init() { m_clMessageQueue.Init(); }
00449
00455     void AddWindow( GuiWindow *pclWindow_ );
00456
00462     void RemoveWindow( GuiWindow *pclWindow_ );
00463
00471     K_BOOL SendEvent( GuiEvent_t *pstEvent_ );
00472
00477     K_BOOL ProcessEvent();

```



```

00478
00482     K_UCHAR GetEventCount() { return m_clMessageQueue
    .GetCount(); }
00483
00487     GuiWindow *FindWindowByName( const K_CHAR *szName_
    );
00488
00489
00490 private:
00497     void CopyEvent( GuiEvent_t *pstDst_, GuiEvent_t
    *pstSrc_ );
00498
00499 private:
00503     DoubleLinkedList m_clWindowList;
00504
00508     MessageQueue m_clMessageQueue;
00509 };
00510
00511 //-----
00521 class GuiControl : public LinkListNode
00522 {
00523 public:
00530     virtual void Init() = 0;
00531
00537     virtual void Draw() = 0;
00538
00546     virtual GuiReturn_t ProcessEvent( GuiEvent_t *
    pstEvent_ ) = 0;
00547
00553     void SetTop( K_USHORT usTop_ )          { m_usTop = usTop_; }
00554
00560     void SetLeft( K_USHORT usLeft_ )         { m_usLeft = usLeft_;
    }
00561
00567     void SetHeight( K_USHORT usHeight_ ) { m_usHeight =
    usHeight_; }
00568
00574     void SetWidth( K_USHORT usWidth_ )      { m_usWidth =
    usWidth_; }
00575
00581     void SetZOrder( K_UCHAR ucZ_ )          { m_ucZOrder =
    ucZ_; }
00582
00589     void SetControlIndex( K_UCHAR ucIdx_ ) { m_ucControlIndex
    = ucIdx_; }
00590
00596     K_USHORT GetTop()                      { return m_usTop; }
00597
00603     K_USHORT GetLeft()                     { return m_usLeft; }
00604
00610     K_USHORT GetHeight()                   { return m_usHeight; }
00611
00617     K_USHORT GetWidth()                    { return m_usWidth; }
00618
00624     K_UCHAR GetZOrder()                    { return m_ucZOrder; }
00625
00631     K_UCHAR GetControlIndex()              { return m_ucControlIndex
    ; }
00632
00638     K_BOOL IsStale()                       { return m_bStale; }
00639
00651     void GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ );
00652
00660     K_BOOL IsInFocus()
00661     {
00662         return m_pclParentWindow->IsInFocus(this);
00663     }
00664
00672     virtual void Activate( bool bActivate_ ) = 0;
00673
00674 protected:
00675     friend class GuiWindow;
00676     friend class GuiEventSurface;
00677
00689     void SetParentControl( GuiControl *pclParent_ ) {
    m_pclParentControl = pclParent_; }
00690
00700     void SetParentWindow( GuiWindow *pclWindow_ ) {
    m_pclParentWindow = pclWindow_; }
00701
00708     GuiControl *GetParentControl()
00709     { return m_pclParentControl; }
00709
00716     GuiWindow *GetParentWindow()
00717     { return m_pclParentWindow; }
00717
00724     void ClearStale()                      { m_bStale

```

```

    = false; }
00725
00729     void SetStale()                                { m_bStale
= true; }
00730
00734     void SetAcceptFocus( bool bFocus_ )            {
m_bAcceptsFocus = bFocus_; }
00735
00739     bool AcceptsFocus()                             { return
m_bAcceptsFocus; }
00740 private:
00742     K_BOOL    m_bStale;
00743
00745     K_BOOL    m_bAcceptsFocus;
00746
00749     K_UCHAR   m_ucZOrder;
00750
00753     K_UCHAR   m_ucControlIndex;
00754
00756     K_USHORT  m_usTop;
00757
00759     K_USHORT  m_usLeft;
00760
00762     K_USHORT  m_usWidth;
00763
00765     K_USHORT  m_usHeight;
00766
00768     GuiControl *m_pclParentControl;
00769
00771     GuiWindow  *m_pclParentWindow;
00772 };
00773
00774 //-----
00779 class StubControl : public GuiControl
00780 {
00781 public:
00782     virtual void Init() { }
00783     virtual void Draw() { }
00784     virtual GuiReturn_t ProcessEvent( GuiEvent_t *
pstEvent_ ) { return GUI_EVENT_OK; }
00785     virtual void Activate( bool bActivate_ ) { }
00786 };
00787
00788 #endif
00789

```

14.53 /home/moslevin/googlecode/mark3/trunk/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"

```

Macros

- `#define __FILE_ID__ KERNEL_CPP`

14.55 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.h File Reference

[Kernel](#) initialization and startup class.

```
#include "kerneltypes.h"
```

Classes

- class [Kernel](#)

Class that encapsulates all of the kernel startup functions.

14.55.1 Detailed Description

[Kernel](#) initialization and startup class. The [Kernel](#) namespace provides functions related to initializing and starting up the kernel.

The [Kernel::Init\(\)](#) function must be called before any of the other functions in the kernel can be used.

Once the initial kernel configuration has been completed (i.e. first threads have been added to the scheduler), the [Kernel::Start\(\)](#) function can then be called, which will transition code execution from the "main()" context to the threads in the scheduler.

Definition in file [kernel.h](#).

14.56 kernel.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 =====*/
00032 #ifndef __KERNEL_H__
00033 #define __KERNEL_H__
00034
00035 #include "kerneltypes.h"
00036 //-----
00040 class Kernel
00041 {
00042 public:
00051     static void Init(void);
00052
00065     static void Start(void);
00066 };
00067
00068 #endif
00069
```

14.57 /home/moslevin/googlecode/mark3/trunk/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"
```

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)

14.57.1 Detailed Description

Macros and functions used for assertions, kernel traces, etc.

Definition in file [kernel_debug.h](#).

14.58 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
00027 //-----
00028 #if KERNEL_USE_DEBUG
00029
00030 //-----
00031 #define __FILE_ID__          STR_UNDEFINED
00032
00033 //-----
00034 #define KERNEL_TRACE( x ) \
00035 { \
00036     K_USHORT ausMsg__[5]; \
00037     ausMsg__[0] = 0xACDC; \
00038     ausMsg__[1] = __FILE_ID__; \
00039     ausMsg__[2] = __LINE__; \
00040     ausMsg__[3] = TraceBuffer::Increment(); \
00041     ausMsg__[4] = (K_USHORT)(x); \
00042     TraceBuffer::Write(ausMsg__, 5); \
00043 };
00044
00045 //-----
00046 #define KERNEL_TRACE_1( x, arg1 ) \
00047 { \
00048     K_USHORT ausMsg__[6]; \
00049     ausMsg__[0] = 0xACDC; \
00050     ausMsg__[1] = __FILE_ID__; \
00051     ausMsg__[2] = __LINE__; \
00052     ausMsg__[3] = TraceBuffer::Increment(); \
00053     ausMsg__[4] = (K_USHORT)(x); \
00054     ausMsg__[5] = arg1; \
00055     TraceBuffer::Write(ausMsg__, 6); \
00056 }
00057
```



```

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)
00072 {
00073     EIFR &= ~(1 << INTF0); // Clear the interrupt flag for INT0
00074 }
00075
00076 //-----
00077 void KernelSWI::Trigger(void)
00078 {
00079     //if(Thread_IsSchedulerEnabled())
00080     {
00081         PORTD &= ~0x04;
00082         PORTD |= 0x04;
00083     }
00084 }

```

14.61 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.h File Reference

Kernel Software interrupt declarations.

```
#include "kerneltypes.h"
```


- #define **TIMER_IMSK** (1 << OCIE1A)
- #define **TIMER_IFR** (1 << OCF1A)

14.63.1 Detailed Description

Kernel Timer Implementation for ATmega328p.

Definition in file [kerneltimer.cpp](#).

14.64 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
00024 #include <avr/io.h>
00025 #include <avr/interrupt.h>
00026
00027 #define TCCR1B_INIT      ((1 << WGM12) | (1 << CS12))
00028 #define TIMER_IMSK      (1 << OCIE1A)
00029 #define TIMER_IFR       (1 << OCF1A)
00030
00031 //-----
00032 void KernelTimer::Config(void)
00033 {
00034     TCCR1B = TCCR1B_INIT;
00035 }
00036
00037 //-----
00038 void KernelTimer::Start(void)
00039 {
00040     TCNT1 = 0;
00041     TIFR1 &= ~TIMER_IFR;
00042     TIMSK1 |= TIMER_IMSK;
00043     TCCR1B |= (1 << CS12);    // Enable count...
00044 }
00045
00046 //-----
00047 void KernelTimer::Stop(void)
00048 {
00049     TIFR1 &= ~TIMER_IFR;
00050     TIMSK1 &= ~TIMER_IMSK;
00051     TCCR1B &= ~(1 << CS12);    // Disable count...
00052     TCNT1 = 0;
00053     OCR1A = 0;
00054 }
00055
00056 //-----
00057 K_USHORT KernelTimer::Read(void)
00058 {
00059     volatile K_USHORT usRead1;
00060     volatile K_USHORT usRead2;
00061
00062     do {
00063         usRead1 = TCNT1;
00064         usRead2 = TCNT1;
00065     } while (usRead1 != usRead2);
00066
00067     return usRead1;
00068 }
00069
00070 //-----
00071 K_ULONG KernelTimer::SubtractExpiry(K_ULONG
ulInterval_)
00072 {
00073     OCR1A -= (K_USHORT)ulInterval_;
00074     return (K_ULONG)OCR1A;

```

```

00075 }
00076
00077 //-----
00078 K_ULONG KernelTimer::TimeToExpiry(void)
00079 {
00080     K_USHORT usRead = KernelTimer::Read();
00081     K_USHORT usOCR1A = OCR1A;
00082
00083     if (usRead >= usOCR1A)
00084     {
00085         return 0;
00086     }
00087     else
00088     {
00089         return (K_ULONG) (usOCR1A - usRead);
00090     }
00091 }
00092
00093 //-----
00094 K_ULONG KernelTimer::GetOvertime(void)
00095 {
00096     return KernelTimer::Read();
00097 }
00098
00099 //-----
00100 K_ULONG KernelTimer::SetExpiry(K_ULONG ulInterval_)
00101 {
00102     K_USHORT usSetInterval;
00103     if (ulInterval_ > 65535)
00104     {
00105         usSetInterval = 65535;
00106     }
00107     else
00108     {
00109         usSetInterval = (K_USHORT)ulInterval_ ;
00110     }
00111     OCR1A = usSetInterval;
00112     return (K_ULONG)usSetInterval;
00113 }
00114
00115 //-----
00116 void KernelTimer::ClearExpiry(void)
00117 {
00118     OCR1A = 65535;           // Clear the compare value
00119 }
00120
00121 //-----
00122 K_UCHAR KernelTimer::DI(void)
00123 {
00124     K_UCHAR bEnabled = ((TIMSK1 & (TIMER_IMSK)) != 0);
00125     TIFR1 &= ~TIMER_IFR;    // Clear interrupt flags
00126     TIMSK1 &= ~TIMER_IMSK;  // Disable interrupt
00127     return bEnabled;
00128 }
00129
00130 //-----
00131 void KernelTimer::EI(void)
00132 {
00133     KernelTimer::RI(0);
00134 }
00135
00136 //-----
00137 void KernelTimer::RI(K_UCHAR bEnable_)
00138 {
00139     if (bEnable_)
00140     {
00141         TIMSK1 |= (1 << OCIE1A);    // Enable interrupt
00142     }
00143     else
00144     {
00145         TIMSK1 &= ~(1 << OCIE1A);
00146     }
00147 }

```

14.65 /home/moslevin/googlecode/mark3/trunk/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))

14.65.1 Detailed Description

[Kernel Timer](#) Class declaration.

Definition in file [kerneltimer.h](#).

14.66 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_

```

14.67 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kerneltypes.h File Reference

Basic data type primitives used throughout the OS.

Macros

- `#define K_BOOL` unsigned char
- `#define K_CHAR` char
- `#define K_UCHAR` unsigned char
- `#define K_USHORT` unsigned short
- `#define K_SHORT` short
- `#define K_ULONG` unsigned long
- `#define K_LONG` long

14.67.1 Detailed Description

Basic data type primitives used throughout the OS.

Definition in file [kerneltypes.h](#).

14.68 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 #ifndef __KERNELTYPES_H__
00020 #define __KERNELTYPES_H__
00021
00022 #if defined(bool)
00023     #define K_BOOL          bool
00024 #else
00025     #define K_BOOL          unsigned char
00026 #endif
00027
00028 #define K_CHAR              char
00029 #define K_UCHAR             unsigned char
00030 #define K_USHORT            unsigned short
00031 #define K_SHORT             short
00032 #define K_ULONG             unsigned long
00033 #define K_LONG              long
00034
00035 #endif

```

14.69 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/keycodes.h File Reference

Standard ASCII keyboard codes.

```
#include "kerneltypes.h"
```

Enumerations

```

• enum KEYCODE {
    KEYCODE_LBUTTON = 0x01, KEYCODE_RBUTTON, KEYCODE_CANCEL, KEYCODE_MBUTTON,
    KEYCODE_BACK = 0x08, KEYCODE_TAB, KEYCODE_CLEAR = 0x0C, KEYCODE_RETURN,
    KEYCODE_SHIFT = 0x10, KEYCODE_CONTROL, KEYCODE_MENU, KEYCODE_PAUSE,
    KEYCODE_CAPITAL, KEYCODE_ESCAPE = 0x1B, KEYCODE_SPACE, KEYCODE_PRIOR,
    KEYCODE_NEXT, KEYCODE_END, KEYCODE_HOME, KEYCODE_LEFT,
    KEYCODE_UP, KEYCODE_RIGHT, KEYCODE_DOWN, KEYCODE_SELECT,
    KEYCODE_PRINT, KEYCODE_EXECUTE, KEYCODE_SNAPSHOT, KEYCODE_INSERT,
    KEYCODE_DELETE, KEYCODE_HELP = 0x2F, KEYCODE_0, KEYCODE_1,
    KEYCODE_2, KEYCODE_3, KEYCODE_4, KEYCODE_5,
    KEYCODE_6, KEYCODE_7, KEYCODE_8, KEYCODE_9,
    KEYCODE_A, KEYCODE_B, KEYCODE_C, KEYCODE_D,
    KEYCODE_E, KEYCODE_F, KEYCODE_G, KEYCODE_H,
    KEYCODE_I, KEYCODE_J, KEYCODE_K, KEYCODE_L,
    KEYCODE_M, KEYCODE_N, KEYCODE_O, KEYCODE_P,
    KEYCODE_Q, KEYCODE_R, KEYCODE_S, KEYCODE_T,
    KEYCODE_U, KEYCODE_V, KEYCODE_W, KEYCODE_X,
    KEYCODE_Y, KEYCODE_Z, KEYCODE_NUMPAD0 = 0x60, KEYCODE_NUMPAD1,
    KEYCODE_NUMPAD2, KEYCODE_NUMPAD3, KEYCODE_NUMPAD4, KEYCODE_NUMPAD5,
    KEYCODE_NUMPAD6, KEYCODE_NUMPAD7, KEYCODE_NUMPAD8, KEYCODE_NUMPAD9,
    KEYCODE_SEPARATOR = 0x6C, KEYCODE_SUBTRACT, KEYCODE_DECIMAL, KEYCODE_DIVIDE,
    KEYCODE_F1, KEYCODE_F2, KEYCODE_F3, KEYCODE_F4,
    KEYCODE_F5, KEYCODE_F6, KEYCODE_F7, KEYCODE_F8,
    KEYCODE_F9, KEYCODE_F10, KEYCODE_F11, KEYCODE_F12,
    KEYCODE_F13, KEYCODE_F14, KEYCODE_F15, KEYCODE_F16,
    KEYCODE_F17, KEYCODE_F18, KEYCODE_F19, KEYCODE_F20,
    KEYCODE_F21, KEYCODE_F22, KEYCODE_F23, KEYCODE_F24,
    KEYCODE_NUMLOCK = 0x90, KEYCODE_SCROLL, KEYCODE_LSHIFT = 0xA0, KEYCODE_RSHIFT,
    KEYCODE_LCONTROL, KEYCODE_RCONTROL, KEYCODE_LMENU, KEYCODE_RMENU,
    KEYCODE_PLAY = 0xFA, KEYCODE_ZOOM }

```

14.69.1 Detailed Description

Standard ASCII keyboard codes.

Definition in file [keycodes.h](#).

14.70 keycodes.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 __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,
00105     KEYCODE_DECIMAL,
00106     KEYCODE_DIVIDE,
00107     KEYCODE_F1,
00108     KEYCODE_F2,
00109     KEYCODE_F3,
00110     KEYCODE_F4,
00111     KEYCODE_F5,
00112     KEYCODE_F6,
00113     KEYCODE_F7,
00114     KEYCODE_F8,
00115     KEYCODE_F9,
00116     KEYCODE_F10,
```

```

00117     KEYCODE_F11,
00118     KEYCODE_F12,
00119     KEYCODE_F13,
00120     KEYCODE_F14,
00121     KEYCODE_F15,
00122     KEYCODE_F16,
00123     KEYCODE_F17,
00124     KEYCODE_F18,
00125     KEYCODE_F19,
00126     KEYCODE_F20,
00127     KEYCODE_F21,
00128     KEYCODE_F22,
00129     KEYCODE_F23,
00130     KEYCODE_F24,
00131     KEYCODE_NUMLOCK = 0x90,
00132     KEYCODE_SCROLL,
00133     KEYCODE_LSHIFT = 0xA0,
00134     KEYCODE_RSHIFT,
00135     KEYCODE_LCONTROL,
00136     KEYCODE_RCONTROL,
00137     KEYCODE_LMENU,
00138     KEYCODE_RMENU,
00139     KEYCODE_PLAY = 0xFA,
00140     KEYCODE_ZOOM
00141 } KEYCODE;
00142
00143 #endif // _KEYCODES_H_

```

14.71 [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.cpp](#) File Reference

ATMega328p Profiling timer implementation.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "profile.h"
#include "kprofile.h"
#include "threadport.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Functions

- **ISR** (TIMER0_OVF_vect)

14.71.1 Detailed Description

ATMega328p Profiling timer implementation.

Definition in file [kprofile.cpp](#).

14.72 kprofile.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 "mark3cfg.h"
00022 #include "profile.h"
00023 #include "kprofile.h"
00024 #include "threadport.h"
00025 #include <avr/io.h>
00026 #include <avr/interrupt.h>
00027
00028 #if KERNEL_USE_PROFILER
00029 K_ULONG Profiler::m_ulEpoch;
00030
00031 //-----
00032 void Profiler::Init()
00033 {
00034     TCCR0A = 0;
00035     TCCR0B = 0;
00036     TIFR0 = 0;
00037     TIMSK0 = 0;
00038     m_ulEpoch = 0;
00039 }
00040
00041 //-----
00042 void Profiler::Start()
00043 {
00044     TIFR0 = 0;
00045     TCNT0 = 0;
00046     TCCR0B |= (1 << CS01);
00047     TIMSK0 |= (1 << TOIE0);
00048 }
00049
00050 //-----
00051 void Profiler::Stop()
00052 {
00053     TIFR0 = 0;
00054     TCCR0B &= ~(1 << CS01);
00055     TIMSK0 &= ~(1 << TOIE0);
00056 }
00057 //-----
00058 K_USHORT Profiler::Read()
00059 {
00060     K_USHORT usRet;
00061     CS_ENTER();
00062     TCCR0B &= ~(1 << CS01);
00063     usRet = TCNT0;
00064     TCCR0B |= (1 << CS01);
00065     CS_EXIT();
00066     return usRet;
00067 }
00068
00069 //-----
00070 void Profiler::Process()
00071 {
00072     CS_ENTER();
00073     m_ulEpoch++;
00074     CS_EXIT();
00075 }
00076
00077 //-----
00078 ISR(TIMER0_OVF_vect)
00079 {
00080     Profiler::Process();
00081 }
00082
00083 #endif

```

14.73 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.h File Reference

Profiling timer hardware interface.

```

#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)

14.73.1 Detailed Description

Profiling timer hardware interface.

Definition in file [kprofile.h](#).

14.74 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
00080     static K_ULONG m_ulEpoch;
00081 };
00082
00083 #endif //KERNEL_USE_PROFILER
00084
00085 #endif
00086

```

14.75 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.cpp File Reference

Core Linked-List implementation, from which all kernel objects are derived.

```
#include "kerneltypes.h"
#include "ll.h"
#include "kernel_debug.h"
```

Macros

- #define `__FILE_ID__` `LL_CPP`

14.75.1 Detailed Description

Core Linked-List implementation, from which all kernel objects are derived.

Definition in file [ll.cpp](#).

14.76 ll.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 "kernel_debug.h"
00025
00026 //-----
00027 #if defined __FILE_ID__
00028     #undef __FILE_ID__
00029 #endif
00030 #define __FILE_ID__      LL_CPP
00031
00032 //-----
00033 void LinkListNode::ClearNode()
00034 {
00035     next = NULL;
00036     prev = NULL;
00037 }
00038
00039 //-----
00040 void DoubleLinkedList::Add(LinkListNode *node_)
00041 {
00042     KERNEL_ASSERT( node_ );
00043
00044     // Add a node to the end of the linked list.
00045     if (!m_pstHead)
00046     {
00047         // If the list is empty, initilize the nodes
00048         m_pstHead = node_;
00049         m_pstTail = node_;
00050
00051         m_pstHead->prev = NULL;
00052         m_pstTail->next = NULL;
00053         return;
00054     }
00055
00056     // Move the tail node, and assign it to the new node just passed in
00057     m_pstTail->next = node_;
00058     node_->prev = m_pstTail;
```

```

00059     node_>next = NULL;
00060     m_pstTail = node_;
00061 }
00062
00063 //-----
00064 void DoubleLinkedList::Remove(LinkListNode *
node_)
00065 {
00066     KERNEL_ASSERT( node_ );
00067     if (node_>prev)
00068     {
00069         node_>prev->next = node_>next;
00070     }
00071     if (node_>next)
00072     {
00073         node_>next->prev = node_>prev;
00074     }
00075     if (node_ == m_pstHead)
00076     {
00077         m_pstHead = node_>next;
00078     }
00079     if (node_ == m_pstTail)
00080     {
00081         m_pstTail = node_>prev;
00082     }
00083 }
00084
00085     node_>ClearNode();
00086 }
00087
00088 //-----
00089 void CircularLinkedList::Add(LinkListNode *node_
)
00090 {
00091     KERNEL_ASSERT( node_ );
00092
00093     // Add a node to the end of the linked list.
00094     if (!m_pstHead)
00095     {
00096         // If the list is empty, initilize the nodes
00097         m_pstHead = node_;
00098         m_pstTail = node_;
00099
00100         m_pstHead->prev = m_pstHead;
00101         m_pstHead->next = m_pstHead;
00102         return;
00103     }
00104
00105     // Move the tail node, and assign it to the new node just passed in
00106     m_pstTail->next = node_;
00107     node_>prev = m_pstTail;
00108     node_>next = m_pstHead;
00109     m_pstTail = node_;
00110     m_pstHead->prev = node_;
00111 }
00112
00113 //-----
00114 void CircularLinkedList::Remove(LinkListNode
*node_)
00115 {
00116     KERNEL_ASSERT( node_ );
00117
00118     // Check to see if this is the head of the list...
00119     if ((node_ == m_pstHead) && (m_pstHead == m_pstTail))
00120     {
00121         // Clear the head and tail pointers - nothing else left.
00122         m_pstHead = NULL;
00123         m_pstTail = NULL;
00124         return;
00125     }
00126
00127     // This is a circularly linked list - no need to check for connection,
00128     // just remove the node.
00129     node_>next->prev = node_>prev;
00130     node_>prev->next = node_>next;
00131
00132     if (node_ == m_pstHead)
00133     {
00134         m_pstHead = m_pstHead->next;
00135     }
00136     if (node_ == m_pstTail)
00137     {
00138         m_pstTail = m_pstTail->prev;
00139     }
00140     node_>ClearNode();
00141 }
00142

```

```

00143 //-----
00144 void CircularLinkedList::PivotForward()
00145 {
00146     if (m_pstHead)
00147     {
00148         m_pstHead = m_pstHead->next;
00149         m_pstTail = m_pstTail->next;
00150     }
00151 }
00152
00153 //-----
00154 void CircularLinkedList::PivotBackward()
00155 {
00156     if (m_pstHead)
00157     {
00158         m_pstHead = m_pstHead->prev;
00159         m_pstTail = m_pstTail->prev;
00160     }
00161 }

```

14.77 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.h File Reference

Core linked-list declarations, used by all kernel list types.

```
#include "kerneltypes.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)`
- `#define SAFE_UNLINK (0)`
"Safe unlinking" performs extra checks on data to make sure that there are no consistencies when performing node operations.

14.77.1 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](#).

- `#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_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.

14.81.1 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.

As of 7.6.2012 on AVR, these are the costs associated with the various features:

Base [Kernel](#): 2888 bytes Tickless Timers: 1194 bytes Semaphores: 224 bytes [Message](#) Queues: 332 bytes (+ Semaphores) Mutexes: 290 bytes [Thread](#) Sleep: 162 bytes (+ Semaphores/Timers) Round-Robin: 304 bytes (+ Timers) Drivers: 144 bytes Dynamic Threads: 68 bytes [Thread](#) Names: 8 bytes Profiling Timers: 624 bytes

Definition in file [mark3cfg.h](#).

14.81.2 Macro Definition Documentation

14.81.2.1 `#define KERNEL_USE_DRIVER (1)`

Enabling device drivers provides a posix-like filesystem interface for peripheral device drivers.

When enabled, the size of the filesystem table is specified in `DRIVER_TABLE_SIZE`. Permissions are enforced for driver access by thread ID and group when `DRIVER_USE_PERMS` are enabled.

Definition at line 118 of file [mark3cfg.h](#).

14.81.2.2 `#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 133 of file [mark3cfg.h](#).

14.81.2.3 `#define KERNEL_USE_MESSAGE (1)`

Enable inter-thread messaging using named mailboxes.

If per-thread mailboxes are defined, each thread is allocated a default mailbox of a depth specified by `THREAD_MAILBOX_SIZE`.

Definition at line 88 of file [mark3cfg.h](#).

14.81.2.4 `#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](#).

Enabling per-thread mutex automatically allocates a mutex for each thread.

Definition at line 99 of file [mark3cfg.h](#).

14.81.2.5 `#define KERNEL_USE_PROFILER (1)`

Provides extra classes for profiling the performance of code.

Useful for debugging and development, but uses an additional timer.

Definition at line 139 of file [mark3cfg.h](#).

14.81.2.6 `#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 68 of file [mark3cfg.h](#).

14.81.2.7 `#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. By also enabling per-thread semaphores, each thread will receive it's own built-in semaphore.

Definition at line 80 of file [mark3cfg.h](#).

14.81.2.8 `#define KERNEL_USE_THREADNAME (1)`

Provide [Thread](#) method to allow the user to set a name for each thread in the system.

Adds to the size of the thread member data.

Definition at line 125 of file [mark3cfg.h](#).

14.81.2.9 `#define KERNEL_USE_TIMERS (1)`

The following options is related to all kernel time-tracking.

-timers provide a way for events to be periodically triggered in a lightweight manner. These can be periodic, or one-shot.

-Thread [Quantum](#) (used for round-robin scheduling) is dependent on this module, as is [Thread](#) Sleep functionality.

Definition at line 56 of file [mark3cfg.h](#).

14.82 mark3cfg.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 =====*/
00044 #ifndef __MARK3CFG_H__
00045 #define __MARK3CFG_H__
00046
00056 #define KERNEL_USE_TIMERS (1)
00057
00067 #if KERNEL_USE_TIMERS
00068 #define KERNEL_USE_QUANTUM (1)
00069 #else
00070 #define KERNEL_USE_QUANTUM (0)
00071 #endif
00072
00080 #define KERNEL_USE_SEMAPHORE (1)
00081
00087 #if KERNEL_USE_SEMAPHORE
00088 #define KERNEL_USE_MESSAGE (1)
00089 #else
00090 #define KERNEL_USE_MESSAGE (0)
00091 #endif
00092
00099 #define KERNEL_USE_MUTEX (1)
00100
00105 #if KERNEL_USE_TIMERS && KERNEL_USE_SEMAPHORE
00106 #define KERNEL_USE_SLEEP (1)
00107 #else
00108 #define KERNEL_USE_SLEEP (0)
00109 #endif
00110
00111
00118 #define KERNEL_USE_DRIVER (1)
00119
00125 #define KERNEL_USE_THREADNAME (1)
00126
00133 #define KERNEL_USE_DYNAMIC_THREADS (1)
00134
00139 #define KERNEL_USE_PROFILER (1)
00140
00145 #define KERNEL_USE_DEBUG (0)
00146
00147
00148 #endif

```

14.83 /home/moslevin/googlecode/mark3/trunk/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"

```

14.83.1 Detailed Description

Implementation of memory, string, and conversion routines.

Definition in file [memutil.cpp](#).

14.84 memutil.cpp

```

00001  /*=====
00002
00003  _____
00004  | \ / | | | | \ / | | | | \ / | | | | \ / | | | |
00005  |  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  =====*/
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  {
00031      K_UCHAR ucTmp = ucData_;
00032      K_UCHAR ucMax;
00033
00034      KERNEL_ASSERT( szText_ );
00035
00036      if (ucTmp >= 0x10)
00037      {
00038          ucMax = 2;
00039      }
00040      else
00041      {
00042          ucMax = 1;
00043      }
00044
00045      ucTmp = ucData_;
00046      szText_[ucMax] = 0;
00047      while (ucMax--)
00048      {
00049          if ((ucTmp & 0x0F) <= 9)
00050          {
00051              szText_[ucMax] = '0' + (ucTmp & 0x0F);
00052          }
00053          else
00054          {
00055              szText_[ucMax] = 'A' + ((ucTmp & 0x0F) - 10);
00056          }
00057          ucTmp >>= 4;
00058      }
00059
00060  //-----
00061  void MemUtil::DecimalToString( K_UCHAR ucData_, char *
00062      szText_ )
00063  {
00064      K_UCHAR ucTmp = ucData_;
00065      K_UCHAR ucMax;
00066
00067      KERNEL_ASSERT(szText_);
00068
00069      // Find max index to print...
00070      if (ucData_ >= 100)
00071      {
00072          ucMax = 3;
00073      }
00074      else if (ucData_ >= 10)
00075      {
00076          ucMax = 2;
00077      }
00078      else
00079      {
00080          ucMax = 1;
00081      }
00082
00083      szText_[ucMax] = 0;
00084      while (ucMax--)
00085      {
00086          szText_[ucMax] = '0' + (ucTmp % 10);
00087          ucTmp /= 10;
00088      }
00089
00090  //-----

```

```

00091 // Basic checksum routines
00092 K_UCHAR MemUtil::Checksum8( const void *pvSrc_, K_USHORT
    usLen_ )
00093 {
00094     K_UCHAR ucRet = 0;
00095     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00096     KERNEL_ASSERT(pvSrc_);
00097
00098     // 8-bit CRC, computed byte at a time
00099     while (usLen_--)
00100     {
00101         ucRet += *pcData++;
00102     }
00103     return ucRet;
00104 }
00105
00106 //-----
00107 K_USHORT MemUtil::Checksum16( const void *pvSrc_, K_USHORT
    usLen_ )
00108 {
00109     K_USHORT usRet = 0;
00110     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00111     KERNEL_ASSERT(pvSrc_);
00112
00113     // 16-bit CRC, computed byte at a time
00114     while (usLen_--)
00115     {
00116         usRet += *pcData++;
00117     }
00118     return usRet;
00119 }
00120
00121 //-----
00122 // Basic string routines
00123 K_USHORT MemUtil::StringLength( const char *szStr_ )
00124 {
00125     K_UCHAR *pcData = (K_UCHAR*)szStr_;
00126     K_USHORT usLen = 0;
00127     KERNEL_ASSERT(szStr_);
00128
00129     while (*pcData++)
00130     {
00131         usLen++;
00132     }
00133     return usLen;
00134 }
00135
00136 //-----
00137 bool MemUtil::CompareStrings( const char *szStr1_,
    const char *szStr2_ )
00138 {
00139     char *szTmp1 = (char*) szStr1_;
00140     char *szTmp2 = (char*) szStr2_;
00141     KERNEL_ASSERT(szStr1_);
00142     KERNEL_ASSERT(szStr2_);
00143
00144     while (*szTmp1 && *szTmp2)
00145     {
00146         if (*szTmp1++ != *szTmp2++)
00147         {
00148             return false;
00149         }
00150     }
00151
00152     // Both terminate at the same length
00153     if (!(*szTmp1) && !(*szTmp2))
00154     {
00155         return true;
00156     }
00157     return false;
00158 }
00159
00160 //-----
00161 void MemUtil::CopyMemory( void *pvDst_, const void *pvSrc_,
    K_USHORT usLen_ )
00162 {
00163     char *szDst = (char*) pvDst_;
00164     char *szSrc = (char*) pvSrc_;
00165     KERNEL_ASSERT(pvDst_);
00166     KERNEL_ASSERT(pvSrc_);
00167 }

```

```

00174     // Run through the strings verifying that each character matches
00175     // and the lengths are the same.
00176     while (usLen_--)
00177     {
00178         *szDst++ = *szSrc++;
00179     }
00180 }
00181
00182 //-----
00183 void MemUtil::CopyString( char *szDst_, const char *szSrc_ )
00184 {
00185     char *szDst = (char*) szDst_;
00186     char *szSrc = (char*) szSrc_;
00187
00188     KERNEL_ASSERT(szDst_);
00189     KERNEL_ASSERT(szSrc_);
00190
00191     // Run through the strings verifying that each character matches
00192     // and the lengths are the same.
00193     while (*szSrc)
00194     {
00195         *szDst++ = *szSrc++;
00196     }
00197 }
00198
00199 //-----
00200 K_SHORT MemUtil::StringSearch( const char *szBuffer_,
                                const char *szPattern_ )
00201 {
00202     char *szTmpPat = (char*)szPattern_;
00203     K_SHORT i16Idx = 0;
00204     K_SHORT i16Start;
00205     KERNEL_ASSERT( szBuffer_ );
00206     KERNEL_ASSERT( szPattern_ );
00207
00208     // Run through the big buffer looking for a match of the pattern
00209     while (szBuffer_[i16Idx])
00210     {
00211         // Reload the pattern
00212         i16Start = i16Idx;
00213         szTmpPat = (char*)szPattern_;
00214         while (*szTmpPat && szBuffer_[i16Idx])
00215         {
00216             if (*szTmpPat != szBuffer_[i16Idx])
00217             {
00218                 break;
00219             }
00220             szTmpPat++;
00221             i16Idx++;
00222         }
00223         // Made it to the end of the pattern, it's a match.
00224         if (*szTmpPat == '\0')
00225         {
00226             return i16Start;
00227         }
00228         i16Idx++;
00229     }
00230
00231     return -1;
00232 }
00233
00234 //-----
00235 bool MemUtil::CompareMemory( const void *pvMem1_, const
                              void *pvMem2_, K_USHORT usLen_ )
00236 {
00237     char *szTmp1 = (char*) pvMem1_;
00238     char *szTmp2 = (char*) pvMem2_;
00239
00240     KERNEL_ASSERT(pvMem1_);
00241     KERNEL_ASSERT(pvMem2_);
00242
00243     // Run through the strings verifying that each character matches
00244     // and the lengths are the same.
00245     while (usLen_--)
00246     {
00247         if (*szTmp1++ != *szTmp2++)
00248         {
00249             return false;
00250         }
00251     }
00252     return true;
00253 }
00254
00255 //-----
00256 void MemUtil::SetMemory( void *pvDst_, K_UCHAR ucVal_,
                          K_USHORT usLen_ )
00257 {

```

```

00258     char *szDst = (char*)pvDst_;
00259
00260     KERNEL_ASSERT(pvDst_);
00261
00262     while (usLen-->0)
00263     {
00264         *szDst++ = ucVal_;
00265     }
00266 }
00267
00268
00269

```

14.85 /home/moslevin/googlecode/mark3/trunk/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

- class [MemUtil](#)

String and Memory manipulation class.

14.85.1 Detailed Description

Utility class containing memory, string, and conversion routines.

Definition in file [memutil.h](#).

14.86 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 //-----
00037 class MemUtil
00038 {
00039
00040 public:
00041
00042     //-----
00051     static void DecimalToHex( K_UCHAR ucData_, char *szText_ );
00052
00053     //-----
00062     static void DecimalToString( K_UCHAR ucData_, char *szText_
00063 );

```

```

00063
00064 //-----
00074 static K_UCHAR Checksum8( const void *pvSrc_, K_USHORT usLen_ );
00075
00076 //-----
00086 static K_USHORT Checksum16( const void *pvSrc_, K_USHORT usLen_ )
;
00087
00088 //-----
00098 static K_USHORT StringLength( const char *szStr_ );
00099
00100 //-----
00110 static bool CompareStrings( const char *szStr1_, const char *
szStr2_ );
00111
00112 //-----
00122 static void CopyMemory( void *pvDst_, const void *pvSrc_,
K_USHORT usLen_ );
00123
00124 //-----
00133 static void CopyString( char *szDst_, const char *szSrc_ );
00134
00135 //-----
00145 static K_SHORT StringSearch( const char *szBuffer_, const char
*szPattern_ );
00146
00147 //-----
00159 static bool CompareMemory( const void *pvMem1_, const void *
pvMem2_, K_USHORT usLen_ );
00160
00161 //-----
00171 static void SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT
usLen_ );
00172
00173 };
00174
00175
00176 #endif //__MEMUTIL_H__
00177
00178
00179
00180

```

14.87 /home/moslevin/googlecode/mark3/trunk/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

14.87.1 Detailed Description

Inter-thread communications via message passing.

Definition in file [message.cpp](#).

14.88 message.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 "message.h"
00026 #include "threadport.h"
00027 #include "kernel_debug.h"
00028
00029 //-----
00030 #if defined __FILE_ID__
00031     #undef __FILE_ID__
00032 #endif
00033 #define __FILE_ID__      MESSAGE_CPP
00034
00035
00036 #if KERNEL_USE_MESSAGE
00037
00038 #if KERNEL_USE_TIMERS
00039     #include "timerlist.h"
00040 #endif
00041
00042 Message GlobalMessagePool::m_aclMessagePool
00043 [8];
00044 DoubleLinkList GlobalMessagePool::m_clList
00045 ;
00046
00047 //-----
00048 void GlobalMessagePool::Init ()
00049 {
00050     K_UCHAR i;
00051     for (i = 0; i < 8; i++)
00052     {
00053         GlobalMessagePool::m_aclMessagePool[
00054 i].Init ();
00055         GlobalMessagePool::m_clList.Add(&(
00056 GlobalMessagePool::m_aclMessagePool[i]));
00057     }
00058 }
00059
00060 //-----
00061 void GlobalMessagePool::Push( Message *
00062 pclMessage_ )
00063 {
00064     KERNEL_ASSERT( pclMessage_ );
00065
00066     CS_ENTER();
00067
00068     GlobalMessagePool::m_clList.Add(pclMessage_);
00069
00070     CS_EXIT();
00071 }
00072
00073 //-----
00074 Message *GlobalMessagePool::Pop ()
00075 {
00076     Message *pclRet;
00077     CS_ENTER();
00078
00079     pclRet = static_cast<Message*>( GlobalMessagePool::m_clList.GetHead
00080 () );
00081     GlobalMessagePool::m_clList.Remove( static_cast<LinkListNode*>(
00082 pclRet ) );
00083
00084     CS_EXIT();
00085     return pclRet;
00086 }
00087
00088 //-----
00089 void MessageQueue::Init ()
00090 {
00091     m_clSemaphore.Init(0, 100);
00092 }

```

```

00086
00087 //-----
00088 Message *MessageQueue::Receive()
00089 {
00090     Message *pclRet;
00091
00092     // Block the current thread on the counting semaphore
00093     m_clSemaphore.Pend();
00094
00095     CS_ENTER();
00096
00097     // Pop the head of the message queue and return it
00098     pclRet = static_cast<Message*>( m_clLinkedList.GetHead
00099     () );
00100     m_clLinkedList.Remove( static_cast<Message*>(pclRet));
00101     CS_EXIT();
00102
00103     return pclRet;
00104 }
00105
00106 #if KERNEL_USE_TIMERS
00107 //-----
00108 Message *MessageQueue::Receive( K_ULONG
00109     ulTimeWaitMS_ )
00110 {
00111     Message *pclRet;
00112
00113     // Block the current thread on the counting semaphore
00114     if (!m_clSemaphore.Pend(ulTimeWaitMS_))
00115     {
00116         return NULL;
00117     }
00118
00119     CS_ENTER();
00120
00121     // Pop the head of the message queue and return it
00122     pclRet = static_cast<Message*>( m_clLinkedList.GetHead
00123     () );
00124     m_clLinkedList.Remove( static_cast<Message*>(pclRet));
00125     CS_EXIT();
00126
00127     return pclRet;
00128 }
00129 #endif
00130 //-----
00131 void MessageQueue::Send( Message *pclSrc_ )
00132 {
00133     KERNEL_ASSERT( pclSrc_ );
00134
00135     CS_ENTER();
00136
00137     // Add the message to the head of the linked list
00138     m_clLinkedList.Add( pclSrc_ );
00139
00140     // Post the semaphore, waking the blocking thread for the queue.
00141     m_clSemaphore.Post();
00142
00143     CS_EXIT();
00144 }
00145
00146 //-----
00147 K_USHORT MessageQueue::GetCount()
00148 {
00149     return m_clSemaphore.GetCount();
00150 }
00151 #endif //KERNEL_USE_MESSAGE

```

14.89 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.h File Reference

Inter-thread communication via message-passing.


```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "semaphore.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.

14.89.1 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.

14.89.2 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](#).

14.90 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 "semaphore.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[8];
00191
00193     static DoubleLinkedList 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     DoubleLinkedList m_clLinkList;
00265 };
00266
00267 #endif //KERNEL_USE_MESSAGE
00268
00269 #endif
```

14.91 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.cpp File Reference

Mutual-exclusion object.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "mutex.h"
#include "kernel_debug.h"
```

Macros

- #define **FILE ID** MUTEX CPP

Functions

- void **TimedMutex Callback** (Thread *pclOwner , void *pvData)

14.91.1 Detailed Description

Mutual-exclusion object.

Definition in file [mutex.cpp](#).

14.92 mutex.cpp

[illegible]

```

00024 #include "mutex.h"
00025 #include "kernel_debug.h"
00026 //-----
00027 #if defined __FILE_ID__
00028     #undef __FILE_ID__
00029 #endif
00030 #define __FILE_ID__      MUTEX_CPP
00031
00032
00033 #if KERNEL_USE_MUTEX
00034
00035 #if KERNEL_USE_TIMERS
00036
00037 //-----
00038 void TimedMutex_Callback(Thread *pclOwner_, void *pvData_)
00039 {
00040     Mutex *pclMutex = static_cast<Mutex*>(pvData_);
00041
00042     // Indicate that the semaphore has expired on the thread
00043     pclMutex->SetExpired(true);
00044
00045     // Wake up the thread that was blocked on this semaphore.
00046     pclMutex->WakeMe(pclOwner_);
00047
00048     if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread
00049         ()->GetPriority())
00049     {
00050         Thread::Yield();
00051     }
00052 }
00053
00054 //-----
00055 void Mutex::WakeMe(Thread *pclOwner_)
00056 {
00057     // Remove from the semaphore waitlist and back to its ready list.
00058     Unblock(pclOwner_);
00059 }
00060
00061 #endif
00062
00063 //-----
00064 K_UCHAR Mutex::WakeNext()
00065 {
00066     Thread *pclChosenOne = NULL;
00067
00068     // Get the highest priority waiter thread
00069     pclChosenOne = m_clBlockList.HighestWaiter();
00070
00071     // Unblock the thread
00072     Unblock(pclChosenOne);
00073
00074     // The chosen one now owns the mutex
00075     m_pclOwner = pclChosenOne;
00076
00077     // Signal a context switch if it's a greater than or equal to the current
00078     priority
00079     if (pclChosenOne->GetPriority() >= Scheduler::GetCurrentThread
00080         ()->GetPriority())
00079     {
00080         return 1;
00081     }
00082     return 0;
00083 }
00084
00085 //-----
00086 void Mutex::Init()
00087 {
00088     // Reset the data in the mutex
00089     m_bReady = 1;           // The mutex is free.
00090     m_ucMaxPri = 0;         // Set the maximum priority inheritance state
00091     m_pclOwner = NULL;      // Clear the mutex owner
00092 }
00093
00094 //-----
00095 #if KERNEL_USE_TIMERS
00096 void Mutex::Claim()
00097 {
00098     Claim(0);
00099 }
00100 bool Mutex::Claim(K_ULONG ulWaitTimeMS_)
00101 #else
00102 void Mutex::Claim()
00103 #endif
00104 {
00105     KERNEL_TRACE_1( STR_MUTEX_CLAIM_1, (K_USHORT)g_pstCurrent->GetID() );
00106
00107     K_UCHAR bSchedule = 0;

```

```

00108     Thread *pclThread;
00109
00110     #if KERNEL_USE_TIMERS
00111         Timer clTimer;
00112
00113         m_bExpired = false;
00114     #endif
00115
00116     // Disable the scheduler while claiming the mutex - we're dealing with all
00117     // sorts of private thread data, can't have a thread switch while messing
00118     // with internal data structures.
00119     Scheduler::SetScheduler(0);
00120
00121     // Get the current thread pointer
00122     pclThread = Scheduler::GetCurrentThread();
00123
00124     // Check to see if the mutex is claimed or not
00125     if (m_bReady != 0)
00126     {
00127         // Mutex isn't claimed, claim it.
00128         m_bReady = 0;
00129         m_ucMaxPri = pclThread->GetPriority();
00130         m_pclOwner = pclThread;
00131     }
00132     else
00133     {
00134         // The mutex is claimed already - we have to block now. Move the
00135         // current thread to the list of threads waiting on the mutex.
00136         #if KERNEL_USE_TIMERS
00137             if (ulWaitTimeMS_)
00138             {
00139                 clTimer.Start(0, ulWaitTimeMS_, (TimerCallback_t)
TimedMutex_Callback, (void*)this);
00140             }
00141         #endif
00142
00143         Block(pclThread);
00144
00145         // Check if priority inheritance is necessary. We do this in order
00146         // to ensure that we don't end up with priority inversions in case
00147         // multiple threads are waiting on the same resource.
00148         if (m_ucMaxPri <= pclThread->GetPriority())
00149         {
00150             m_ucMaxPri = pclThread->GetPriority();
00151         }
00152
00153         Thread *pclTemp = static_cast<Thread*>(
m_clBlockList.GetHead());
00154         while (pclTemp)
00155         {
00156             pclTemp->InheritPriority(m_ucMaxPri);
00157             if (pclTemp == static_cast<Thread*>(m_clBlockList.GetTail()))
00158             {
00159                 break;
00160             }
00161             pclTemp = static_cast<Thread*>(pclTemp->GetNext
());
00162         }
00163     }
00164
00165     // Switch Threads when we exit the critical section.
00166     bSchedule = 1;
00167
00168     // Done with thread data -reenable the scheduler
00169     Scheduler::SetScheduler(1);
00170
00171     if (bSchedule)
00172     {
00173         // Switch threads if this thread acquired the mutex
00174         Thread::Yield();
00175     }
00176
00177     #if KERNEL_USE_TIMERS
00178         if (ulWaitTimeMS_)
00179         {
00180             clTimer.Stop();
00181         }
00182         return (m_bExpired == 0);
00183     #endif
00184 }
00185
00186 //-----
00187 void Mutex::Release()

```

```

00191 {
00192     KERNEL_TRACE_1( STR_MUTEX_RELEASE_1, (K_USHORT)g_pstCurrent->GetID() )
00193 ;
00194     K_UCHAR bSchedule = 0;
00195     Thread *pclThread;
00196
00197     // Disable the scheduler while we deal with internal data structures.
00198     Scheduler::SetScheduler(0);
00199     pclThread = Scheduler::GetCurrentThread();
00200
00201     // Restore the thread's original priority
00202     if (pclThread->GetCurPriority() != pclThread->GetPriority())
00203     {
00204         pclThread->SetPriority(pclThread->GetPriority());
00205
00206         // In this case, we want to reschedule
00207         bSchedule = 1;
00208     }
00209
00210     // No threads are waiting on this semaphore?
00211     if (m_clBlockList.GetHead() == NULL)
00212     {
00213         // Re-initialize the mutex to its default values
00214         m_bReady = 1;
00215         m_ucMaxPri = 0;
00216         m_pclOwner = NULL;
00217     }
00218     else
00219     {
00220         // Wake the highest priority Thread pending on the mutex
00221         if (WakeNext())
00222         {
00223             // Switch threads if it's higher or equal priority than the current
00224             thread
00225                 bSchedule = 1;
00226         }
00227
00228         // Must enable the scheduler again in order to switch threads.
00229         Scheduler::SetScheduler(1);
00230         if (bSchedule)
00231         {
00232             // Switch threads if a higher-priority thread was woken
00233             Thread::Yield();
00234         }
00235     }
00236
00237 #endif //KERNEL_USE_MUTEX

```

14.93 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.h File Reference

Mutual exclusion class declaration.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "timerlist.h"

```

Classes

- class [Mutex](#)

Mutual-exclusion locks, based on [BlockingObject](#).

14.93.1 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.

14.93.2 Initializing

Initializing a mutex object by calling:

```
clMutex.Init();
```

14.93.3 Resource protection example

```
clMutex.Claim();
...
<resource protected block>
...
clMutex.Release();
```

Definition in file [mutex.h](#).

14.94 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
00058 #if KERNEL_USE_MUTEX
00059
00060 #if KERNEL_USE_TIMERS
00061 #include "timerlist.h"
00062 #endif
00063
00064 //-----
00068 class Mutex : public BlockingObject
00069 {
00070 public:
00071     void Init();
00072
00073     void Claim();
00074
00075 #if KERNEL_USE_TIMERS
00076     bool Claim(K_ULONG ulWaitTimeMS_);
00077
00078     void WakeMe( Thread *pclOwner_ );
00079
00080     void SetExpired( bool bExpired_ ) { m_bExpired = bExpired_; }
00081 #endif
00082
00083     void Release();
00084
00085 private:
00086     K_UCHAR WakeNext();
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
```



```

00036 //-----
00037 void ProfileTimer::Init()
00038 {
00039     m_ulCumulative = 0;
00040     m_ulCurrentIteration = 0;
00041     m_usIterations = 0;
00042     m_bActive = 0;
00043 }
00044
00045 //-----
00046 void ProfileTimer::Start()
00047 {
00048     if (!m_bActive)
00049     {
00050         CS_ENTER();
00051         m_ulCurrentIteration = 0;
00052         m_ulInitialEpoch = Profiler::GetEpoch
00053             ();
00054         m_usInitial = Profiler::Read();
00055         CS_EXIT();
00056         m_bActive = 1;
00057     }
00058 }
00059 //-----
00060 void ProfileTimer::Stop()
00061 {
00062     if (m_bActive)
00063     {
00064         K_USHORT usFinal;
00065         K_ULONG ulEpoch;
00066         CS_ENTER();
00067         usFinal = Profiler::Read();
00068         ulEpoch = Profiler::GetEpoch();
00069         // Compute total for current iteration...
00070         m_ulCurrentIteration = ComputeCurrentTicks
00071             (usFinal, ulEpoch);
00072         m_ulCumulative += m_ulCurrentIteration
00073             ;
00074         m_usIterations++;
00075         CS_EXIT();
00076         m_bActive = 0;
00077     }
00078 }
00079 //-----
00079 K_ULONG ProfileTimer::GetAverage()
00080 {
00081     if (m_usIterations)
00082     {
00083         return m_ulCumulative / (K_ULONG)m_usIterations
00084             ;
00085     }
00086     return 0;
00087 }
00088 //-----
00089 K_ULONG ProfileTimer::GetCurrent()
00090 {
00091     if (m_bActive)
00092     {
00093         K_USHORT usCurrent;
00094         K_ULONG ulEpoch;
00095         CS_ENTER();
00096         usCurrent = Profiler::Read();
00097         ulEpoch = Profiler::GetEpoch();
00098         CS_EXIT();
00099         return ComputeCurrentTicks(usCurrent, ulEpoch);
00100     }
00101     return m_ulCurrentIteration;
00102 }
00103 }
00104
00105 //-----
00106 K_ULONG ProfileTimer::ComputeCurrentTicks(
00107     K_USHORT usCurrent_, K_ULONG ulEpoch_)
00108 {
00109     K_ULONG ulTotal;
00110     K_ULONG ulOverflows;
00111     ulOverflows = ulEpoch_ - m_ulInitialEpoch;
00112     // More than one overflow...
00113 }

```

```

00114     if (ulOverflows > 1)
00115     {
00116         ulTotal = ((K_ULONG) (ulOverflows-1) * TICKS_PER_OVERFLOW)
00117             + (K_ULONG) (TICKS_PER_OVERFLOW - m_usInitial) +
00118             (K_ULONG) usCurrent_;
00119     }
00120     // Only one overflow, or one overflow that has yet to be processed
00121     else if (ulOverflows || (usCurrent_ < m_usInitial))
00122     {
00123         ulTotal = (K_ULONG) (TICKS_PER_OVERFLOW - m_usInitial) +
00124             (K_ULONG) usCurrent_;
00125     }
00126     // No overflows, none pending.
00127     else
00128     {
00129         ulTotal = (K_ULONG) (usCurrent_ - m_usInitial);
00130     }
00131
00132     return ulTotal;
00133 }
00134
00135 #endif

```

14.97 /home/moslevin/googlecode/mark3/trunk/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.

14.97.1 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](#).

14.98 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:
00079     void Init();
00080
00087     void Start();
00088
00095     void Stop();
00096
00104     K_ULONG GetAverage();
00105
00114     K_ULONG GetCurrent();
00115
00116 private:
00117
00126     K_ULONG ComputeCurrentTicks(K_USHORT usCount_, K_ULONG
        ulEpoch_);
00127
00128     K_ULONG m_ulCumulative;
00129     K_ULONG m_ulCurrentIteration;
00130     K_USHORT m_usInitial;
00131     K_ULONG m_ulInitialEpoch;
00132     K_USHORT m_usIterations;
00133     K_UCHAR m_bActive;
00134 };
00135
00136 #endif // KERNEL_USE_PROFILE
00137
00138 #endif

```

14.99 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.cpp File Reference

[Thread Quantum](#) Implementation for Round-Robin Scheduling.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "timerlist.h"
#include "thread.h"
#include "quantum.h"
#include "kernel_debug.h"

```

Macros

- #define __FILE_ID__ QUANTUM_CPP


```

    );
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() !=
00078         pclThread_>GetCurrent()->GetTail() )
00079     {
00080         Quantum::SetTimer(pclThread_);
00081         TimerScheduler::Add(&m_clQuantumTimer);
00082         m_bActive = 1;
00083     }
00084 }
00085
00086 //-----
00087 void Quantum::RemoveThread(void)
00088 {
00089     if (!m_bActive)
00090     {
00091         return;
00092     }
00093
00094     // Cancel the current timer
00095     TimerScheduler::Remove(&m_clQuantumTimer);
00096     m_bActive = 0;
00097 }
00098
00099 //-----
00100 void Quantum::UpdateTimer(void)
00101 {
00102     // If we have to re-add the quantum timer (more than 2 threads at the
00103     // high-priority level...)
00104     if (bAddQuantumTimer)
00105     {
00106         // Trigger a thread yield - this will also re-schedule the
00107         // thread *and* reset the round-robin scheduler.
00108         Thread::Yield();
00109         bAddQuantumTimer = false;
00110     }
00111 }
00112
00113 #endif //KERNEL_USE_QUANTUM

```

14.101 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/quantum.h File Reference

[Thread Quantum](#) declarations for Round-Robin Scheduling.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "timerlist.h"

```

Classes

- class [Quantum](#)

Static-class used to implement [Thread](#) quantum functionality, which is a key part of round-robin scheduling.

14.101.1 Detailed Description

[Thread Quantum](#) declarations for Round-Robin Scheduling.

Definition in file [quantum.h](#).

14.102 quantum.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 __KQUANTUM_H__
00023  #define __KQUANTUM_H__
00024
00025  #include "kerneltypes.h"
00026  #include "mark3cfg.h"
00027
00028  #include "thread.h"
00029  #include "timerlist.h"
00030
00031  #if KERNEL_USE_QUANTUM
00032  class Timer;
00033
00039  class Quantum
00040  {
00041  public:
00050      static void UpdateTimer();
00051
00058      static void AddThread( Thread *pclThread_ );
00059
00065      static void RemoveThread();
00066
00067  private:
00079      static void SetTimer( Thread *pclThread_ );
00080
00081      static Timer m_clQuantumTimer;
00082      static K_UCHAR m_bActive;
00083  };
00084
00085  #endif //KERNEL_USE_QUANTUM
00086
00087  #endif

```

14.103 /home/moslevin/googlecode/mark3/trunk/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 "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] = {-1,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3}

14.103.1 Detailed Description

Strict-Priority + Round-Robin thread scheduler implementation.

Definition in file [scheduler.cpp](#).

14.104 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 "tl.h"
00024 #include "scheduler.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__ SCHEDULER_CPP
00032
00033 //-----
00034 Thread *g_pstNext;
00035 Thread *g_pstCurrent;
00036
00037 //-----
00038 K_UCHAR Scheduler::m_bEnabled;
00039 ThreadList Scheduler::m_clStopList;
00040 ThreadList Scheduler::m_aclPriorities[
    NUM_PRIORITIES];
00041 K_UCHAR Scheduler::m_ucPriFlag;
00042
00043 K_UCHAR g_ucFlag;
00044 //-----
00045 static const K_UCHAR aucCLZ[16] = {-1,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3};
00046
00047 //-----
00048 void Scheduler::Init()
00049 {
00050     m_ucPriFlag = 0;
00051     for (int i = 0; i < NUM_PRIORITIES; i++)
00052     {
00053         m_aclPriorities[i].SetPriority(i);
00054         m_aclPriorities[i].SetFlagPointer(&
00055 m_ucPriFlag);
00056     }
00057     g_ucFlag = m_ucPriFlag;
00058 }
00059 //-----
00060 void Scheduler::Schedule()
00061 {
00062     K_UCHAR ucPri = 0;
00063
00064     // Figure out what priority level has ready tasks (8 priorities max)
00065     ucPri = aucCLZ[m_ucPriFlag >> 4];
00066     if (ucPri == 0xFF) { ucPri = aucCLZ[m_ucPriFlag & 0x0F]; }
00067     else { ucPri += 4; }
00068
00069     // Get the thread node at this priority.
00070     g_pstNext = (Thread*)( m_aclPriorities[ucPri].GetHead(
    ) );
00071     g_ucFlag = m_ucPriFlag;

```

```

00072
00073     KERNEL_TRACE_1( STR_SCHEDULE_1, (K_USHORT)g_pstNext->GetID() );
00074 }
00075
00076 //-----
00077 void Scheduler::Add(Thread *pclThread_)
00078 {
00079     m_aclPriorities[pclThread_->GetPriority()].Add
00080     (pclThread_);
00081     g_ucFlag = m_ucPriFlag;
00082 }
00083 //-----
00084 void Scheduler::Remove(Thread *pclThread_)
00085 {
00086     m_aclPriorities[pclThread_->GetPriority()].Remove
00087     (pclThread_);
00088     g_ucFlag = m_ucPriFlag;
00089 }

```

14.105 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.h File Reference

[Thread](#) scheduler function declarations.

```

#include "kerneltypes.h"
#include "thread.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**

14.105.1 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](#).

14.106 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
00052 extern Thread *g_pstNext;
00053 extern Thread *g_pstCurrent;
00054
00055 #define NUM_PRIORITIES (8)
00056 //-----
00061 class Scheduler
00062 {
00063 public:
00069     static void Init();
00070
00078     static void Schedule();
00079
00087     static void Add(Thread *pclThread_);
00088
00097     static void Remove(Thread *pclThread_);
00098
00111     static void SetScheduler(K_UCHAR bEnable_) { m_bEnabled
= bEnable_; }
00112
00118     static Thread *GetCurrentThread() { return
g_pstCurrent; }
00119
00126     static Thread *GetNextThread() { return g_pstNext; }
00127
00136     static ThreadList *GetThreadList(K_UCHAR ucPriority_
){ return &m_aclPriorities[ucPriority_]; }
00137
00144     static ThreadList *GetStopList() { return &m_clStopList
; }
00145
00154     static K_UCHAR IsEnabled() { return m_bEnabled; }
00155
00156 private:
00158     static K_UCHAR m_bEnabled;
00159
00161     static ThreadList m_clStopList;
00162
00164     static ThreadList m_aclPriorities[NUM_PRIORITIES];
00165
00167     static K_UCHAR m_ucPriFlag;
00168 };
00169 #endif
00170

```

14.107 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.cpp File Reference

Higher level window management framework.

```
#include "kerneltypes.h"
#include "screen.h"
#include "gui.h"
#include "memutil.h"
```

14.107.1 Detailed Description

Higher level window management framework.

Definition in file [screen.cpp](#).

14.108 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 =====*/
00019 #include "kerneltypes.h"
00020 #include "screen.h"
00021 #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
00056     return NULL;
00057 }
00058
```



```

00075
00076 private:
00077
00078     virtual void Create() = 0;
00079     virtual void Destroy() = 0;
00080
00081 };
00082
00083 //-----
00084 class ScreenList
00085 {
00086 public:
00087     ScreenList() { m_clList.Init()
00088     ; }
00089
00092     void Add( Screen *pclScreen_ ) { m_clList.Add(
00093     pclScreen_); }
00094
00097     void Remove( Screen *pclScreen_ ) { m_clList.Remove
00098     (pclScreen_); }
00099
00102     Screen *GetHead() { return static_cast<
00103     Screen*>(m_clList.GetHead()); }
00104 private:
00105     DoubleLinkedList m_clList;
00106 };
00107
00108 //-----
00109 class ScreenManager
00110 {
00111 public:
00112
00113     ScreenManager() { m_pclSurface = NULL; }
00114
00118     void AddScreen( Screen *pclScreen_ ) { m_clScreenList
00119     .Add(pclScreen_);
00120
00121     pclScreen_>SetManager
00122     (this); }
00123
00124     void RemoveScreen( Screen *pclScreen_ ) {
00125     m_clScreenList.Remove(pclScreen_);
00126
00127     pclScreen_>SetManager
00128     (NULL); }
00129
00130     void SetEventSurface( GuiEventSurface *
00131     pclSurface_ ) { m_pclSurface = pclSurface_; }
00132
00133     GuiWindow *FindWindowByName( const K_CHAR *
00134     m_szName_ );
00135
00136     Screen *FindScreenByName( const K_CHAR *m_szName_ );
00137 private:
00138     ScreenList m_clScreenList;
00139     GuiEventSurface *m_pclSurface;
00140 };
00141
00142 #endif

```

14.111 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.cpp

File Reference

[Semaphore](#) Blocking-Object Implemenation.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "semaphore.h"
#include "blocking.h"
#include "kernel_debug.h"
#include "timerlist.h"

```

Macros

- #define __FILE_ID__ SEMAPHORE_CPP

Functions

- void **TimedSemaphore_Callback** (Thread *pclOwner_, void *pvData_)

14.111.1 Detailed Description

Semaphore Blocking-Object Implemenation.

Definition in file [semaphore.cpp](#).

14.112 semaphore.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 "semaphore.h"
00026 #include "blocking.h"
00027 #include "kernel_debug.h"
00028 //-----
00029 #if defined __FILE_ID__
00030 #undef __FILE_ID__
00031 #endif
00032 #define __FILE_ID__ SEMAPHORE_CPP
00033
00034 #if KERNEL_USE_SEMAPHORE
00035
00036 #if KERNEL_USE_TIMERS
00037 #include "timerlist.h"
00038
00039 //-----
00040 void TimedSemaphore_Callback(Thread *pclOwner_, void *pvData_)
00041 {
00042     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_
00043 );
00044
00045     // Indicate that the semaphore has expired on the thread
00046     pclSemaphore->SetExpired(true);
00047
00048     // Wake up the thread that was blocked on this semaphore.
00049     pclSemaphore->WakeMe(pclOwner_);
00050
00051     if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread
00052 (->GetPriority())
00053     {
00054         Thread::Yield();
00055     }
00056 }
00057 //-----
00058 void Semaphore::WakeMe(Thread *pclChosenOne_)
00059 {
00060     // Remove from the semaphore waitlist and back to its ready list.
00061     Unblock(pclChosenOne_);
00062 }
00063 #endif // KERNEL_USE_TIMERS
00064
00065 //-----

```

```

00066 K_UCHAR Semaphore::WakeNext()
00067 {
00068     Thread *pClChosenOne;
00069
00070     pClChosenOne = m_clBlockList.HighestWaiter();
00071
00072     // Remove from the semaphore waitlist and back to its ready list.
00073     Unblock(pClChosenOne);
00074
00075     // Call a task switch only if higher priority thread
00076     if (pClChosenOne->GetPriority() > Scheduler::GetCurrentThread
        ()->GetPriority())
00077     {
00078         return 1;
00079     }
00080     return 0;
00081 }
00082
00083 //-----
00084 void Semaphore::Init(K_USHORT usInitVal_, K_USHORT usMaxVal_)
00085 {
00086     // Copy the paramters into the object - set the maximum value for this
00087     // semaphore to implement either binary or counting semaphores, and set
00088     // the initial count. Clear the wait list for this object.
00089     m_usValue = usInitVal_;
00090     m_usMaxValue = usMaxVal_;
00091 #if KERNEL_USE_TIMERS
00092     m_bExpired = false;
00093 #endif
00094 }
00095
00096 //-----
00097 void Semaphore::Post()
00098 {
00099     KERNEL_TRACE_1( STR_SEMAPHORE_POST_1, (K_USHORT)g_pstCurrent->GetID()
        );
00100
00101     K_UCHAR bThreadWake = 0;
00102
00103     // Increment the semaphore count - we can mess with threads so ensure this
00104     // is in a critical section. We don't just disable the scheudler since
00105     // we want to be able to do this from within an interrupt context as well.
00106     CS_ENTER();
00107
00108     // If nothing is waiting for the semaphore
00109     if (m_clBlockList.GetHead() == NULL)
00110     {
00111         // Check so see if we've reached the maximum value in the semaphore
00112         if (m_usValue < m_usMaxValue)
00113         {
00114             // Increment the count value
00115             m_usValue++;
00116         }
00117     }
00118     else
00119     {
00120         // Otherwise, there are threads waiting for the semaphore to be
00121         // posted, so wake the next one (highest priority goes first).
00122         bThreadWake = WakeNext();
00123     }
00124
00125     CS_EXIT();
00126
00127     // if bThreadWake was set, it means that a higher-priority thread was
00128     // woken. Trigger a context switch to ensure that this thread gets
00129     // to execute next.
00130     if (bThreadWake)
00131     {
00132         Thread::Yield();
00133     }
00134 }
00135
00136 #if !KERNEL_USE_TIMERS
00137 //-----
00138     // No timers, no timed pend
00139     void Semaphore::Pend()
00140 #else
00141 //-----
00142     // Redirect the untimed pend API to the timed pend, with a null timeout.
00143     void Semaphore::Pend()
00144     {
00145         Pend(0);
00146     }
00147 //-----
00148     bool Semaphore::Pend( K_ULONG ulWaitTimeMS_ )
00149 #endif
00150 {

```

```

00151     KERNEL_TRACE_1( STR_SEMAPHORE_PEND_1, (K_USHORT)g_pstCurrent->GetID()
00152 );
00153     // Decrement the semaphore count - if 0, wait.
00154     K_UCHAR bThreadWait = 0;
00155     Thread *pclThread;
00156
00157 #if KERNEL_USE_TIMERS
00158     Timer clSemTimer;
00159
00160     m_bExpired = false;
00161 #endif
00162
00163     // Once again, messing with thread data - ensure
00164     // we're doing all of these operations from within a thread-safe context.
00165     CS_ENTER();
00166
00167     // Get the current thread pointer.
00168     pclThread = Scheduler::GetCurrentThread();
00169
00170     // Check to see if we need to take any action based on the semaphore count
00171     if (m_usValue != 0)
00172     {
00173         // The semaphore count is non-zero, we can just decrement the count
00174         // and go along our merry way.
00175         m_usValue--;
00176     }
00177     else
00178     {
00179         // The semaphore count is zero - we need to block the current thread
00180         // and wait until the semaphore is posted from elsewhere.
00181 #if KERNEL_USE_TIMERS
00182         if (ulWaitTimeMS_)
00183         {
00184             clSemTimer.Start(0, ulWaitTimeMS_, TimedSemaphore_Callback
00185 , (void*)this);
00186         }
00187 #endif
00188         Block(pclThread);
00189         bThreadWait = 1;
00190     }
00191
00192     // If bThreadWait was set, it means that the current thread is blocked.
00193     // We need to call a context switch to ensure the highest-priority
00194     // ready thread gets to run next.
00195     if (bThreadWait)
00196     {
00197         // Switch Threads immediately
00198         Thread::Yield();
00199     }
00200     CS_EXIT();
00201
00202 #if KERNEL_USE_TIMERS
00203     if (ulWaitTimeMS_)
00204     {
00205         clSemTimer.Stop();
00206     }
00207     return (m_bExpired == 0);
00208 #endif
00209 }
00210
00211 //-----
00212 K_USHORT Semaphore::GetCount()
00213 {
00214     K_USHORT usRet;
00215     CS_ENTER();
00216     usRet = m_usValue;
00217     CS_EXIT();
00218     return usRet;
00219 }
00220
00221 #endif

```

14.113 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.h File Reference

[Semaphore](#) Blocking Object class declarations.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "blocking.h"
#include "threadlist.h"
```

Classes

- class [Semaphore](#)

Counting semaphore, based on [BlockingObject](#) base class.

14.113.1 Detailed Description

[Semaphore](#) Blocking Object class declarations.

Definition in file [semaphore.h](#).

14.114 semaphore.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 __SEMAPHORE_H__
00023 #define __SEMAPHORE_H__
00024
00025 #include "kerneltypes.h"
00026 #include "mark3cfg.h"
00027
00028 #include "blocking.h"
00029 #include "threadlist.h"
00030
00031 #if KERNEL_USE_SEMAPHORE
00032
00033 //-----
00037 class Semaphore : public BlockingObject
00038 {
00039 public:
00049     void Init(K_USHORT usInitVal_, K_USHORT usMaxVal_);
00050
00056     void Post();
00057
00064     void Pend();
00065
00066
00078     K_USHORT GetCount();
00079
00080 #if KERNEL_USE_TIMERS
00081
00092     bool Pend(K_ULONG ulWaitTimeMS_);
00093
00104     void WakeMe(Thread *pclChosenOne_);
00105
00112     void SetExpired(bool bExpired_) { m_bExpired = bExpired_; }
00113
00114     bool GetExpired() { return m_bExpired; }
00115 #endif
00116
00117 private:
00118
00124     K_UCHAR WakeNext();
00125
00126     K_USHORT m_usValue;
00127     K_USHORT m_usMaxValue;
```



```
00128
00129 #if KERNEL_USE_TIMERS
00130     bool m_bExpired;
00131 #endif
00132
00133 };
00134
00135 #endif //KERNEL_USE_SEMAPHORE
00136
00137 #endif
```

14.115 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.cpp File Reference

Serial Line IP framing code.

```
#include "kerneltypes.h"
#include "slip.h"
#include "driver.h"
```

Macros

- #define FRAMING_BYTE (192)
Byte indicating end-of-frame.
- #define FRAMING_ENC_BYTE (219)
Byte used to indicate substitution.
- #define FRAMING_SUB_BYTE (220)
Byte to substitute for framing byte.
- #define FRAMING_SUB_ENC_BYTE (221)
Byte to substitute for the substitute-byte.
- #define ACchar (69)
Acknowledgement character.
- #define NACchar (96)
Non-acknowledgement character.

14.115.1 Detailed Description

Serial Line IP framing code.

Definition in file [slip.cpp](#).

14.116 slip.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 "slip.h"
00016 #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_, 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_,
00105                          K_USHORT usLen_ )
00106 {
00107     K_USHORT usReadCount;
00108     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_, K_CHAR *aucBuf_,
00165                     K_USHORT usLen_)
00166 {
00167     K_UCHAR aucTmp[2];
00168     K_USHORT usCRC = 0;
00169
00170     // Lightweight protocol built on-top of SLIP.
00171     // 1) Channel ID (8-bit)
00172     // 2) Data Size (16-bit)
00173     // 3) Data blob
00174     // 4) CRC16 (16-bit)
00175     aucTmp[0] = FRAMING_BYTE;
00176     while( !m_pclDriver->Write(1, aucTmp) ) {}
00177
00178     if (!usLen_) // Read to end-of-line (\0)
00179     {
00180         K_UCHAR *pucBuf = (K_UCHAR*)aucBuf_;
00181         while (*pucBuf != '\0')
00182         {
00183             usLen_++;
00184             pucBuf++;
00185         }
00186     }
00187
00188     WriteByte(ucChannel_);
00189     usCRC = ucChannel_;
00190
00191     WriteByte((K_UCHAR) (usLen_ >> 8));
00192     usCRC += (usLen_ >> 8);
00193
00194     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_, K_USHORT usLen_)
00224 {
00225     K_UCHAR aucTmp[2];
00226     K_USHORT usCRC = 0;
00227     K_UCHAR i, j;
00228     K_USHORT usTotalLen = 0;
00229
00230     // Calculate the total length of all message fragments
00231     for (i = 0; i < usLen_; i++)
00232     {
00233         usTotalLen += astData_[i].ucSize;
00234     }
00235
00236     // Send a FRAMING_BYTE to start framing a message
00237     aucTmp[0] = FRAMING_BYTE;
00238     while( !m_pclDriver->Write(1, aucTmp) ) {}
00239
00240     // Write a the channel
00241     WriteByte(ucChannel_);
00242     usCRC = ucChannel_;
00243
00244     // Write the length
00245     WriteByte((K_UCHAR) (usTotalLen >> 8));
00246     usCRC += (usTotalLen >> 8);
00247
00248     WriteByte((K_UCHAR) (usTotalLen & 0x00FF));
00249     usCRC += (usTotalLen & 0x00FF);
00250
00251     // Write the message fragments
00252     for (i = 0; i < usLen_; i++)
00253     {
00254         K_UCHAR *aucBuf = astData_[i].pucData;
00255         for (j = 0; j < astData_[i].ucSize; j++)
00256         {
00257             WriteByte(*aucBuf);
00258             usCRC += (K_USHORT)*aucBuf;
00259             aucBuf++;
00260         }
00261     }
00262
00263     // Write the CRC
00264     WriteByte((K_UCHAR) (usCRC >> 8));
00265     WriteByte((K_UCHAR) (usCRC & 0x00FF));
00266
00267     // Write the end-of-message
00268     aucTmp[0] = FRAMING_BYTE;
00269     while( !m_pclDriver->Write(1, aucTmp) ) {}
00270 }

```

14.117 /home/moslevin/googlecode/mark3/trunk/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](#) }

14.117.1 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, this is the packet data CRC16 is 2 byte, Providing an error detection mechanism

Definition in file [slip.h](#).

14.117.2 Enumeration Type Documentation

14.117.2.1 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 37 of file [slip.h](#).

14.118 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  =====*/
00030  #include "kerneltypes.h"
00031  #include "driver.h"
00032
00033  #ifndef __SLIP_H__
00034  #define __SLIP_H__
00035
00036  //-----
00037  typedef enum
00038  {
00039      SLIP_CHANNEL_TERMINAL = 0,
00040      SLIP_CHANNEL_UNISCOPE,
00041      SLIP_CHANNEL_NVM,
00042      SLIP_CHANNEL_RESET,
00043      SLIP_CHANNEL_GRAPHICS,
00044      SLIP_CHANNEL_HID,
00045  } SLIP_CHANNEL_COUNT
00046  } SlipChannel;
00047
00048  //-----
00055  typedef struct
00056  {
00057      K_UCHAR ucSize;
00058      K_UCHAR *pucData;
00059  }SlipDataVector;
00060
00061  //-----
00066  class Slip
00067  {
00068  public:
00074      void SetDriver( Driver *pclDriver_ ){ m_pclDriver =
pclDriver_; }
00075
00081      Driver *GetDriver() { return m_pclDriver; }
00082
00094      static K_USHORT EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ );
00095
00110      static K_USHORT DecodeByte( K_UCHAR *ucChar_, K_UCHAR *aucBuf_ );
00111
00124      void WriteData( K_UCHAR ucChannel_, K_CHAR *aucBuf_, K_USHORT
usLen_ );
00125
00138      K_USHORT ReadData( K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT
usLen_ );
00139
00152      void WriteVector( K_UCHAR ucChannel_, SlipDataVector
*astData_, K_USHORT usLen_ );
00153
00159      void SendAck();
00160
00166      void SendNack();
00167
00168  private:
00169      void WriteByte(K_UCHAR ucData_);
00170      Driver *m_pclDriver;
00171  };
00172
00173  #endif

```

14.119 /home/moslevin/googlecode/mark3/trunk/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"
```

Functions

- static void SlipMux_CallBack (Driver *pclDriver_)

14.119.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel, and provides an abstraction to attach handlers for each event type.

Definition in file [slip_mux.cpp](#).

14.119.2 Function Documentation

14.119.2.1 static void SlipMux_Callback (Driver * *pclDriver_*) [static]

Parameters

<i>pclDriver_</i>	Pointer to the driver data for the port triggering the callback
-------------------	---

Definition at line 43 of file [slip_mux.cpp](#).

14.120 slip_mux.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 "driver.h"
00024  #include "drvUART.h"
00025  #include "slip.h"
00026  #include "slip_mux.h"
00027  #include "message.h"
00028
00029  //-----
00030  MessageQueue *SlipMux::m_pclMessageQueue;
00031  K_UCHAR SlipMux::m_aucData[SLIP_BUFFER_SIZE];
00032  Driver *SlipMux::m_pclDriver;
00033  Slip_Channel SlipMux::m_apfChannelHandlers[SLIP_CHANNEL_COUNT] = {0};
00034  Semaphore SlipMux::m_clSlipSem;
00035  Slip SlipMux::m_clSlip;
00036
00037  //-----
00043  static void SlipMux_Callback( Driver *pclDriver_)
00044  {
00045      Message *pclMsg = GlobalMessagePool::Pop();
00046      if (pclMsg)
00047      {
00048          pclDriver_>Control(CMD_SET_RX_DISABLE, 0, 0, 0, 0);
00049

```

```

00050         // Send a message to the queue, letting it know that there's a
00051         // pending slip message that needs to be processed
00052         pclMsg->SetCode(SLIP_RX_MESSAGE_ID);
00053         pclMsg->SetData(NULL);
00054         SlipMux::GetQueue()->Send(pclMsg);
00055     }
00056 }
00057
00058 //-----
00059 void SlipMux::Init(K_CHAR *pcDriverPath_, K_USHORT usRxSize_,
00060                  K_UCHAR *aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_)
00061 {
00062     m_pclDriver = DriverList::FindByPath(pcDriverPath_);
00063     m_pclMessageQueue = NULL;
00064     m_clSlip.SetDriver(m_pclDriver);
00065     m_clSlipSem.Init(0, 1);
00066     m_pclDriver->Control(CMD_SET_BUFFERS, (void*)aucRx_, usRxSize_, (
00067         void*)aucTx_, usTxSize_);
00068     m_pclDriver->Control(CMD_SET_RX_CALLBACK, (void*)SlipMux_CallBack
00069         , 0, 0, 0);
00069     {
00070         K_UCHAR ucEscape = 192;
00071         m_pclDriver->Control(CMD_SET_RX_ESCAPE, (void*)&ucEscape, 1, 0,
00072             NULL);
00073     }
00074
00075 //-----
00076 void SlipMux::InstallHandler( K_UCHAR ucChannel_,
00077                               Slip_Channel pfHandler_ )
00078 {
00079     if (pfHandler_)
00080     {
00081         m_apfChannelHandlers[ucChannel_] = pfHandler_;
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,
00091         SLIP_BUFFER_SIZE );
00091     if (usLen && (m_apfChannelHandlers[ucChannel] != NULL))
00092     {
00093         m_apfChannelHandlers[ucChannel]( m_pclDriver, ucChannel, &(m_aucData[3]
00094             ), usLen);
00095     }
00096
00097     // Re-enable the driver once we're done.
00097     m_pclDriver->Control( CMD_SET_RX_ENABLE, 0, 0, 0, 0 );
00098 }
00099

```

14.121 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip_mux.h File Reference

FunkenSlip Channel Multiplexer.

```

#include "kerneltypes.h"
#include "driver.h"
#include "semaphore.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_USH-ORT usLen_)

14.121.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel

Definition in file [slip_mux.h](#).

14.122 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 "semaphore.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:
00046     static void Init(K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *
aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_);
00047
00048     static void InstallHandler( K_UCHAR ucChannel_, Slip_Channel
pfHandler_ );
00049
00050     static void MessageReceive();
00051
00052     static Driver *GetDriver(){ return m_pclDriver; }
00053
00054     static MessageQueue *GetQueue(){ return
m_pclMessageQueue; }
00055
00056     static void SetQueue( MessageQueue *pclMessageQueue_ )
{ m_pclMessageQueue = pclMessageQueue_; }
00057
00058     static Slip *GetSlip(){ return &m_slip; }
00059
00060 private:
00061     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

```

14.123 /home/moslevin/googlecode/mark3/trunk/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"

```

14.123.1 Detailed Description

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

Definition in file [slipterm.cpp](#).

14.124 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
00029         ("/dev/tty" ) );
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_cSlip.WriteVector(SLIP_CHANNEL_TERMINAL
00053         , astData, 2);
00054     }
00055 }
00056 //-----
00057 void SlipTerm::PrintLn( K_UCHAR ucSeverity_, const char *
00058 szLine_ )
00059 {
00060     if (ucSeverity_ <= m_ucVerbosity)
00061     {
00062         PrintLn( szLine_ );
00063     }
00064 }
```

14.125 [/home/moslevin/googlecode/mark3/trunk/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__

14.125.1 Detailed Description

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

Definition in file [slipterm.h](#).14.126 **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"
```



```

00028 K_UCHAR    SystemHeap::m_pucRawHeap[ HEAP_RAW_SIZE ];
00029 HeapConfig  SystemHeap::m_pclSystemHeapConfig[ HEAP_NUM_SIZES
+ 1];
00030 FixedHeap    SystemHeap::m_clSystemHeap;
00031 bool        SystemHeap::m_bInit;
00032
00033 //-----
00034 void SystemHeap::Init(void)
00035 {
00036     #if HEAP_NUM_SIZES > 0
00037         m_pclSystemHeapConfig[0].m_usBlockSize = HEAP_BLOCK_SIZE_1
;
00038         m_pclSystemHeapConfig[0].m_usBlockCount = HEAP_BLOCK_COUNT_1
;
00039     #endif
00040     #if HEAP_NUM_SIZES > 1
00041         m_pclSystemHeapConfig[1].m_usBlockSize = HEAP_BLOCK_SIZE_2;
00042         m_pclSystemHeapConfig[1].m_usBlockCount = HEAP_BLOCK_COUNT_2;
00043     #endif
00044     #if HEAP_NUM_SIZES > 2
00045         m_pclSystemHeapConfig[2].m_usBlockSize = HEAP_BLOCK_SIZE_3;
00046         m_pclSystemHeapConfig[2].m_usBlockCount = HEAP_BLOCK_COUNT_3;
00047     #endif
00048     #if HEAP_NUM_SIZES > 3
00049         m_pclSystemHeapConfig[3].m_usBlockSize = HEAP_BLOCK_SIZE_4;
00050         m_pclSystemHeapConfig[3].m_usBlockCount = HEAP_BLOCK_COUNT_4;
00051     #endif
00052     #if HEAP_NUM_SIZES > 4
00053         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].m_usBlockSize
= 0;
00122     m_pclSystemHeapConfig[HEAP_NUM_SIZES].m_usBlockCount
= 0;
00123
00124     m_clSystemHeap.Create((void*)m_pucRawHeap, 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

```

14.129 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system_heap.h

File Reference

Global system-heap implementation.

```

#include "system_heap_config.h"
#include "fixed_heap.h"

```

Classes

- class [SystemHeap](#)

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(LinkedListNode) + sizeof(void*)) * HEAP_BLOCK_COUNT_2)`
- `#define HEAP_RAW_SIZE_3 ((HEAP_BLOCK_SIZE_3 + sizeof(LinkedListNode) + 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`

14.129.1 Detailed Description

Global system-heap implementation. Provides a basic malloc()/free() allocation scheme.

Definition in file [system_heap.h](#).

14.129.2 Macro Definition Documentation

14.129.2.1 `#define HEAP_RAW_SIZE`

Value:

```
HEAP_RAW_SIZE_1 + \
HEAP_RAW_SIZE_2 + \
HEAP_RAW_SIZE_3 + \
HEAP_RAW_SIZE_4 + \
HEAP_RAW_SIZE_5 + \
HEAP_RAW_SIZE_6 + \
HEAP_RAW_SIZE_7 + \
HEAP_RAW_SIZE_8 + \
HEAP_RAW_SIZE_9 + \
HEAP_RAW_SIZE_10 + \
HEAP_RAW_SIZE_11 + \
HEAP_RAW_SIZE_12 + \
HEAP_RAW_SIZE_13 + \
HEAP_RAW_SIZE_14 + \
HEAP_RAW_SIZE_15 + \
HEAP_RAW_SIZE_16 + \
HEAP_RAW_SIZE_17 + \
HEAP_RAW_SIZE_18 + \
HEAP_RAW_SIZE_19 + \
HEAP_RAW_SIZE_20 + \
HEAP_RAW_SIZE_21
```

Definition at line 161 of file [system_heap.h](#).

```
14.129.2.2 #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.

(don't touch this!!!)

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

14.130 system_heap.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 __SYSTEM_HEAP_H__
00022 #define __SYSTEM_HEAP_H__
00023
00024 #include "system_heap_config.h"
00025 #include "fixed_heap.h"
00026
00027 #if USE_SYSTEM_HEAP
00028
00029 //-----
00034 #if HEAP_NUM_SIZES > 0
00035     #define HEAP_RAW_SIZE_1 ((HEAP_BLOCK_SIZE_1 + sizeof(LinkListNode) +
00036                             sizeof(void*)) * 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) +
00042                             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) +
00048                             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) +
00054                             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) +
00060                             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) +
00066                             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) +
00072                             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) +
00078         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) +
00084         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) +
00090         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) +
00096         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) +
00102         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) +
00108         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) +
00114         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) +
00120         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) +
00126         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) +
00132         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) +
00138         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) +
00144         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 //-----
00185 class SystemHeap
00186 {
00187 public:
00188     static void  Init(void);
00189     static void* Alloc(K_USHORT usSize_);
00190     static void  Free(void *pvData_);
00191
00192 private:
00193     static K_UCHAR m_pucRawHeap[ HEAP_RAW_SIZE ];
00194     static HeapConfig m_pclSystemHeapConfig[ HEAP_NUM_SIZES
        + 1 ];
00195     static FixedHeap m_clSystemHeap;
00196     static bool m_bInit;
00197 };
00198
00199 #endif // USE_SYSTEM_HEAP
00200
00201 #endif // __SYSTEM_HEAP_H__

```

14.131 /home/moslevin/googlecode/mark3/trunk/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)

14.131.1 Detailed Description

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

Definition in file [system_heap_config.h](#).

14.131.2 Macro Definition Documentation

```
14.131.2.1 #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](#).

14.132 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

```

14.133 /home/moslevin/googlecode/mark3/trunk/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 "semaphore.h"
#include "quantum.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.

14.133.1 Detailed Description

Platform-Independent thread class Definition.

Definition in file [thread.cpp](#).

14.134 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 "semaphore.h"
00030 #include "quantum.h"
00031 #include "kernel_debug.h"
00032 //-----
00033 #if defined __FILE_ID__
00034 #undef __FILE_ID__
00035 #endif
00036 #define __FILE_ID__      THREAD_CPP
00037
00038 //-----
00039 void Thread::Init( K_UCHAR *paucStack_,
00040                  K_USHORT usStackSize_,
00041                  K_UCHAR ucPriority_,
00042                  ThreadEntry_t pfEntryPoint_,
00043                  void *pvArg_ )
00044 {
00045     static K_UCHAR ucThreadID = 0;
00046
00047     KERNEL_ASSERT( paucStack_ );
00048     KERNEL_ASSERT( pfEntryPoint_ );
00049
00050     m_ucThreadID = ucThreadID++;
00051
00052     KERNEL_TRACE_1( STR_STACK_SIZE_1, usStackSize_ );
00053     KERNEL_TRACE_1( STR_PRIORITY_1, (K_UCHAR)ucPriority_ );
00054     KERNEL_TRACE_1( STR_THREAD_ID_1, (K_USHORT)m_ucThreadID );
00055     KERNEL_TRACE_1( STR_ENTRYPOINT_1, (K_USHORT)pfEntryPoint_ );
00056
00057     // Initialize the thread parameters to their initial values.
00058     m_paucStack = paucStack_;
00059     m_paucStackTop = TOP_OF_STACK(paucStack_,
00060 usStackSize_);
00061
00062     m_usStackSize = usStackSize_;
00063
00064 #if KERNEL_USE_QUANTUM
00065     m_usQuantum = 4;
00066 #endif
00067
00068     m_ucPriority = ucPriority_;
00069     m_ucCurPriority = m_ucPriority;
00070     m_pfEntryPoint = pfEntryPoint_;
00071     m_pvArg = pvArg_;
00072
00073 #if KERNEL_USE_THREADNAME
00074     m_szName = NULL;
00075 #endif
00076
00077     // Call CPU-specific stack initialization
00078     ThreadPort::InitStack(this);
00079
00080     // Add to the global "stop" list.
00081     CS_ENTER();
00082     m_pclOwner = Scheduler::GetThreadList(
00083 m_ucPriority);
00084     m_pclCurrent = Scheduler::GetStopList();
00085     m_pclCurrent->Add(this);
00086     CS_EXIT();
00087 }
00088
00089 //-----
00088 void Thread::Start(void)
00089 {
00090     // Remove the thread from the scheduler's "stopped" list, and add it
00091     // to the scheduler's ready list at the proper priority.
00092     KERNEL_TRACE_1( STR_THREAD_START_1, (K_USHORT)m_ucThreadID );
00093
00094     CS_ENTER();
00095     Scheduler::GetStopList()->Remove(this);
00096     Scheduler::Add(this);

```

```

00097     m_pclOwner = Scheduler::GetThreadList (
m_ucPriority);
00098     m_pclCurrent = m_pclOwner;
00099     if (m_ucPriority >= Scheduler::GetCurrentThread
()->GetCurPriority())
00100     {
00101 #if KERNEL_USE_QUANTUM
00102     // Deal with the thread Quantum
00103     Quantum::RemoveThread();
00104     Quantum::AddThread(this);
00105 #endif
00106     }
00107     if (m_ucPriority > Scheduler::GetCurrentThread
()->GetPriority())
00108     {
00109         Thread::Yield();
00110     }
00111     CS_EXIT();
00112 }
00113
00114 #if KERNEL_USE_DYNAMIC_THREADS
00115 //-----
00116 void Thread::Exit ()
00117 {
00118     K_UCHAR bReschedule = 0;
00119
00120     KERNEL_TRACE_1( STR_THREAD_EXIT_1, m_ucThreadID );
00121
00122     CS_ENTER();
00123
00124     // If this thread is the actively-running thread, make sure we run the
00125     // scheduler again.
00126     if (this == Scheduler::GetCurrentThread())
00127     {
00128         bReschedule = 1;
00129     }
00130
00131     // Remove the thread from scheduling
00132     Scheduler::Remove(this);
00133
00134     CS_EXIT();
00135
00136     if (bReschedule)
00137     {
00138         // Choose a new "next" thread if we must
00139         Thread::Yield();
00140     }
00141 }
00142 #endif
00143
00144 #if KERNEL_USE_SLEEP
00145 //-----
00146 static void ThreadSleepCallback( Thread *pclOwner_,
void *pvData_ )
00147 {
00148     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_
);
00149
00150     // Post the semaphore, which will wake the sleeping thread.
00151     pclSemaphore->Post();
00152 }
00153
00154 //-----
00155 void Thread::Sleep(K_ULONG ulTimeMs_)
00156 {
00157     Timer clTimer;
00158     Semaphore clSemaphore;
00159
00160     // Create a semaphore that this thread will block on
00161     clSemaphore.Init(0, 1);
00162
00163     // Create a one-shot timer that will call a callback that posts the
00164     // semaphore, waking our thread.
00165     clTimer.SetIntervalMSeconds(ulTimeMs_);
00166     clTimer.SetCallback(ThreadSleepCallback);
00167     clTimer.SetData((void*)&clSemaphore);
00168     clTimer.SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00169
00170     // Add the new timer to the timer scheduler, and block the thread
00171     TimerScheduler::Add(&clTimer);
00172     clSemaphore.Pend();
00173 }
00174 #endif // KERNEL_USE_SLEEP
00175 //-----
00176 K_USHORT Thread::GetStackSlack()
00177 {

```

```

00180     K_USHORT usCount = 0;
00181
00182     CS_ENTER();
00183
00185     for (usCount = 0; usCount < m_usStackSize; usCount++)
00186     {
00187         if (m_paucStack[usCount] != 0xFF)
00188         {
00189             break;
00190         }
00191     }
00192
00193     CS_EXIT();
00194
00195     return usCount;
00196 }
00197
00198 //-----
00199 void Thread::Yield()
00200 {
00201     CS_ENTER();
00202
00203     // Run the scheduler
00204     Scheduler::Schedule();
00205
00206     // Only switch contexts if the new task is different than the old task
00207     if (Scheduler::GetCurrentThread() !=
        Scheduler::GetNextThread())
00208     {
00209         #if KERNEL_USE_QUANTUM
00210             // new thread scheduled. Stop current quantum timer (if it exists),
00211             // and restart it for the new thread (if required).
00212             Quantum::RemoveThread();
00213             Quantum::AddThread(g_pstNext);
00214         #endif
00215
00216         Thread::ContextSwitchSWI();
00217     }
00218
00219     CS_EXIT();
00220 }
00221
00222 //-----
00223 void Thread::SetPriorityBase(K_UCHAR ucPriority_)
00224 {
00225     GetCurrent()->Remove(this);
00226
00227     SetCurrent(Scheduler::GetThreadList(
        m_ucPriority));
00228
00229     GetCurrent()->Add(this);
00230 }
00231
00232 //-----
00233 void Thread::SetPriority(K_UCHAR ucPriority_)
00234 {
00235     K_UCHAR bSchedule = 0;
00236     CS_ENTER();
00237     // If this is the currently running thread, it's a good idea to reschedule
00238     // Or, if the new priority is a higher priority than the current thread's.
00239     if ((g_pstCurrent == this) || (ucPriority_ > g_pstCurrent->GetPriority
        ()))
00240     {
00241         bSchedule = 1;
00242     }
00243     CS_EXIT();
00244
00245     Scheduler::Remove(this);
00246
00247     m_ucCurPriority = ucPriority_;
00248     m_ucPriority = ucPriority_;
00249
00250     CS_ENTER();
00251     Scheduler::Add(this);
00252     CS_EXIT();
00253
00254     if (bSchedule)
00255     {
00256         CS_ENTER();
00257         Scheduler::Schedule();
00258     }
00259     #if KERNEL_USE_QUANTUM
00260         // new thread scheduled. Stop current quantum timer (if it exists),
00261         // and restart it for the new thread (if required).
00262         Quantum::RemoveThread();
00263         Quantum::AddThread(g_pstNext);
00264     #endif
00265     CS_EXIT();

```

```

00265         Thread::ContextSwitchSWI();
00266     }
00267 }
00268
00269 //-----
00270 void Thread::InheritPriority(K_UCHAR ucPriority_)
00271 {
00272     SetOwner(Scheduler::GetThreadList(
00273         ucPriority_));
00274     m_ucCurPriority = ucPriority_;
00275 }
00276 //-----
00277 void Thread::ContextSwitchSWI()
00278 {
00279     // Call the context switch interrupt if the scheduler is enabled.
00280     if (Scheduler::IsEnabled() == 1)
00281     {
00282         KERNEL_TRACE_1( STR_CONTEXT_SWITCH_1, (K_USHORT)g_pstNext->GetID()
00283     );
00284         KernelSWI::Trigger();
00285     }
00286 }
00287

```

14.135 /home/moslevin/googlecode/mark3/trunk/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.

Macros

- #define [THREAD_QUANTUM_DEFAULT](#) (4)
Suggested default thread quantum.

Typedefs

- typedef void(* [ThreadEntry_t](#))(void *pvArg_)
Function pointer type used for thread entryptoint functions.

14.135.1 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 of the processor. The [Thread](#) is the fundmanetal 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](#).

14.136 thread.h

```

00001  /*-----
00002  00003  00004  00005  00006  00007  00008  00009  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  //-----
00049  #define THREAD_QUANTUM_DEFAULT      (4)
00050
00051  //-----
00055  typedef void (*ThreadEntry_t)(void *pvArg_);
00056
00057  //-----
00058  class ThreadPort;
00059
00060  //-----
00064  class Thread : public LinkListNode
00065  {
00066  public:
00086      void Init(K_UCHAR *paucStack_,
00087               K_USHORT usStackSize_,
00088               K_UCHAR ucPriority_,
00089               ThreadEntry_t pfEntryPoint_,
00090               void *pvArg_ );
00091
00099      void Start();
00100
00101  #if KERNEL_USE_THREADNAME
00102
00111      void SetName(const K_CHAR *szName_) { m_szName = szName_; };
00112  #endif
00113
00122      ThreadList *GetOwner(void) { return m_pclOwner;
00123      };
00131      ThreadList *GetCurrent(void) { return m_pclCurrent
00132      ; };
00141      K_UCHAR GetPriority(void) { return m_ucPriority; };
00142
00150      K_UCHAR GetCurPriority(void) { return m_ucCurPriority; };
00151
00152  #if KERNEL_USE_QUANTUM
00153
00160      void SetQuantum( K_USHORT usQuantum_ ) { m_usQuantum =
00161      usQuantum_; }
00169      K_USHORT GetQuantum(void) { return m_usQuantum; };
00170  #endif
00171
00179      void SetCurrent( ThreadList *pclNewList_ ) {
00180      m_pclCurrent = pclNewList_; };

```

```

00188     void SetOwner( ThreadList *pclNewList_ ) { m_pclOwner
      = pclNewList_; };
00189
00190
00203     void SetPriority(K_UCHAR ucPriority_);
00204
00214     void InheritPriority(K_UCHAR ucPriority_);
00215
00216 #if KERNEL_USE_DYNAMIC_THREADS
00217
00228     void Exit();
00229 #endif
00230
00231 #if KERNEL_USE_SLEEP
00232
00240     static void Sleep(K_ULONG ulTimeMs_);
00241 #endif
00242
00250     static void Yield(void);
00251
00259     void SetID( K_UCHAR ucID_ ) { m_ucThreadID = ucID_; }
00260
00268     K_UCHAR GetID() { return m_ucThreadID; }
00269
00270
00283     K_USHORT GetStackSlack();
00284
00285     friend class ThreadPort;
00286
00287 private:
00295     static void ContextSwitchSWI(void);
00296
00297     void SetPriorityBase(K_UCHAR ucPriority_);
00298
00300     K_UCHAR *m_paucStackTop;
00301
00303     K_UCHAR *m_paucStack;
00304
00306     K_USHORT m_usStackSize;
00307
00308 #if KERNEL_USE_QUANTUM
00309
00310     K_USHORT m_usQuantum;
00311 #endif
00312
00314     K_UCHAR m_ucThreadID;
00315
00317     K_UCHAR m_ucPriority;
00318
00320     K_UCHAR m_ucCurPriority;
00321
00323     ThreadEntry_t m_pfEntryPoint;
00324
00326     void *m_pvArg;
00327
00328 #if KERNEL_USE_THREADNAME
00329
00330     const K_CHAR *m_szName;
00331 #endif
00332
00334     ThreadList *m_pclCurrent;
00335
00337     ThreadList *m_pclOwner;
00338 };
00339
00340 #endif

```

14.137 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.cpp File Reference

[Thread](#) linked-list definitions.

```

#include "kerneltypes.h"
#include "ll.h"
#include "threadlist.h"
#include "thread.h"
#include "kernel_debug.h"

```

Macros

- `#define __FILE_ID__ THREADLIST_CPP`

14.137.1 Detailed Description

[Thread](#) linked-list definitions.

Definition in file [threadlist.cpp](#).

14.138 threadlist.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 "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     // Set the threadlist's priority level, flag pointer, and then add the
00064     // thread to the threadlist
00065     SetPriority(ucPriority_);
00066     SetFlagPointer(pucFlag_);
00067     Add(node_);
00068 }
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
00110         pclTemp = static_cast<Thread*>(pclTemp->GetNext());
00111     }
00112     return pclChosen;
00113 }

```

14.139 /home/moslevin/googlecode/mark3/trunk/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.

14.139.1 Detailed Description

[Thread](#) linked-list declarations.

Definition in file [threadlist.h](#).

14.140 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 =
00041          NULL; };
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
00084          ucPriority_);
00092      void Remove(LinkListNode *node_);
00093
00101      Thread *HighestWaiter();
00102  private:
00103
00105      K_UCHAR m_ucPriority;
00106
00108      K_UCHAR *m_pucFlag;
00109  };
00110
00111  #endif
00112

```

14.141 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.cpp File Reference

ATMega328p Multithreading.

```

#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>

```

Functions

- 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.


```

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 of the stack.
00085 pclThread->m_paucStackTop = (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
00101     switch SWI
00102     KernelTimer::Config(); // configure the
00103     kernel timer
00104
00105     Scheduler::SetScheduler(1); // enable the
00106     scheduler
00107     Scheduler::Schedule(); // run the
00108     scheduler - determine the first thread to run
00109
00110     Thread_Switch(); // Set the next scheduled thread to
00111     the current thread
00112
00113     KernelTimer::Start(); // enable the kernel
00114     timer
00115     KernelSWI::Start(); // enable the task
00116     switch SWI
00117
00118     // Restore the context...
00119     Thread_RestoreContext(); // restore the context
00120     of the first running thread
00121     ASM("reti"); // return from interrupt - will return
00122     to the first scheduled thread
00123 }
00124
00125 //-----
00126 //-----
00127 ISR(INT0_vect) __attribute__ ( ( signal, naked ) );
00128 ISR(INT0_vect)
00129 {
00130     Thread_SaveContext(); // Push the context
00131     (registers) of the current task
00132     Thread_Switch(); // Switch to the next task
00133     Thread_RestoreContext(); // Pop the context
00134     (registers) of the next task
00135     ASM("reti"); // Return to the next task
00136 }
00137
00138 //-----
00139 //-----
00140 ISR(TIMER1_COMPA_vect)
00141 {
00142     #if KERNEL_USE_TIMERS
00143     TimerScheduler::Process();
00144     #endif
00145     #if KERNEL_USE_QUANTUM
00146     Quantum::UpdateTimer();
00147     #endif
00148 }

```

14.143 /home/moslevin/googlecode/mark3/trunk/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 !
- #define [CS_EXIT](#)()
Exit critical section (restore status register)
- #define [ENABLE_INTS](#)() [ASM](#)("sei");
Initiate a contex switch without using the SWI.
- #define [DISABLE_INTS](#)() [ASM](#)("cli");

14.143.1 Detailed Description

ATMega328p Multithreading support.

Definition in file [threadport.h](#).

14.143.2 Macro Definition Documentation

14.143.2.1 #define CS_ENTER()

Value:

```
{ \
volatile K_UCHAR x; \
x = _SFR_IO8(SR_); \
ASM("cli");
```

These macros *must* be used in pairs !

Enter critical section (copy status register, disable interrupts)

Definition at line 142 of file [threadport.h](#).

14.143.2.2 #define CS_EXIT()

Value:

```
_SFR_IO8(SR_) = x;\
}
```

Exit critical section (restore status register)

Definition at line 149 of file [threadport.h](#).

14.144 threadport.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 __THREADPORT_H_
00022 #define __THREADPORT_H_
00023
00024 #include "kerneltypes.h"
00025 #include "thread.h"
00026
00027 #include <avr/io.h>
00028 #include <avr/interrupt.h>
00029
00030 //-----
00032 #define ASM(x)      asm volatile(x);
00033
00034 #define SR_          0x3F
00035
00036 #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
00044 #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_

```

14.145 /home/moslevin/googlecode/mark3/trunk/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`
- `#define TL_FUDGE_FACTOR (9)`

Number of ticks to account for overhead when performing Time->tick computations.

14.145.1 Detailed Description

[Timer](#) data structure + scheduler implementations.

Definition in file [timerlist.cpp](#).

14.145.2 Macro Definition Documentation

14.145.2.1 `#define TL_FUDGE_FACTOR (9)`

Number of ticks to account for overhead when performing Time->tick computations.

This must be calibrated on a per-device basis. This value is currently Set up for a 16-bit timer, with a 256 prescaler, 16MHz clock, on an ATmega328p (i.e. ARDUINO UNO).

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

14.146 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  //-----
00044  #define TL_FUDGE_FACTOR      (9)
00045
00046  TimerList TimerScheduler::m_clTimerList;
00047  //-----
00048  void TimerList::Init(void)
00049  {
00050      m_bTimerActive = 0;
00051      m_ulNextWakeup = 0;
00052  }
00053
00054  //-----
00055  void TimerList::Add(Timer *pclListNode_)
00056  {
00057      K_LONG lDelta;
00058      K_UCHAR bStart = 0;
00059      CS_ENTER();
00060
00061      if (GetHead() == NULL)
00062      {
00063          bStart = 1;
00064      }
00065
00066      pclListNode_>ClearNode();
00067      DoubleLinkedList::Add(pclListNode_);
00068
00069      // Set the initial timer value
00070      pclListNode_>m_ulTimeLeft = pclListNode_>m_ulInterval
00071  ;
00072      if (!bStart)
00073      {
00074          // If the new interval is less than the amount of time remaining...
00075          lDelta = KernelTimer::TimeToExpiry() -
00076      pclListNode_>m_ulInterval;
00077          if (lDelta > 0)
00078          {
00079              // Set the new expiry time on the timer.
00080              m_ulNextWakeup = KernelTimer::SubtractExpiry
00081      ((K_ULONG)lDelta);
00082          }
00083          else
00084          {
00085              m_ulNextWakeup = pclListNode_>m_ulInterval;
00086              KernelTimer::SetExpiry(m_ulNextWakeup
00087      );
00088          }
00089          // Set the timer as active.
00090          pclListNode_>m_ucFlags |= TIMERLIST_FLAG_ACTIVE
00091  ;
00092      }
00093      CS_EXIT();
00094  }

```

```

00094 //-----
00095 void TimerList::Remove(Timer *pclLinkListNode_)
00096 {
00097     CS_ENTER();
00098
00099     DoubleLinkedList::Remove(pclLinkListNode_);
00100
00101     if (this->GetHead() == NULL)
00102     {
00103         KernelTimer::Stop();
00104     }
00105
00106     CS_EXIT();
00107 }
00108
00109 //-----
00110 void TimerList::Process(void)
00111 {
00112     K_ULONG ulNewExpiry;
00113     K_ULONG ulOvertime;
00114     K_UCHAR bContinue;
00115
00116     Timer *pclNode;
00117     Timer *pclPrev;
00118
00119     // Clear the timer and its expiry time - keep it running though
00120     KernelTimer::ClearExpiry();
00121
00122     do
00123     {
00124         ulNewExpiry = MAX_TIMER_TICKS;
00125         pclNode = static_cast<Timer*>(GetHead());
00126         pclPrev = NULL;
00127         bContinue = 0;
00128
00129         // Subtract the elapsed time interval from each active timer.
00130         while (pclNode)
00131         {
00132             // Active timers only...
00133             if (pclNode->m_ucFlags & TIMERLIST_FLAG_ACTIVE
00134
00135                 {
00136                     // Did the timer expire?
00137                     if (pclNode->m_ulTimeLeft <= m_ulNextWakeup
00138
00139                         {
00140                             // Yes - set the "callback" flag - we'll execute the
00141                             callbacks later
00142                             pclNode->m_ucFlags |= TIMERLIST_FLAG_CALLBACK;
00143
00144                             if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT
00145
00146                                 {
00147                                     // If this was a one-shot timer, deactivate the timer.
00148                                     pclNode->m_ucFlags |= TIMERLIST_FLAG_EXPIRED;
00149
00150                                     pclNode->m_ucFlags &= ~TIMERLIST_FLAG_ACTIVE;
00151
00152                                     }
00153                                 else
00154                                 {
00155                                     // Reset the interval timer.
00156                                     // I think we're good though...
00157                                     pclNode->m_ulTimeLeft = pclNode->
00158                                     m_ulInterval;
00159
00160                                     // If the time remaining is less than the expiry, set
00161                                     the new expiry.
00162                                     if (pclNode->m_ulTimeLeft < ulNewExpiry)
00163                                     {
00164                                         ulNewExpiry = pclNode->m_ulTimeLeft;
00165                                     }
00166                                 }
00167                             else
00168                             {
00169                                 // Not expiring, but determine how K_LONG to run the next
00170                                 timer interval for.
00171                                 pclNode->m_ulTimeLeft -= m_ulNextWakeup;
00172
00173                                 if (pclNode->m_ulTimeLeft < ulNewExpiry)
00174                                 {
00175                                     ulNewExpiry = pclNode->m_ulTimeLeft;
00176                                 }
00177                             }
00178                         }
00179                     }
00180             }
00181         }
00182     } while (pclNode);
00183 }

```

```

00171         }
00172         pclNode = static_cast<Timer*>(pclNode->GetNext());

00173     }
00174
00175     // Process the expired timers callbacks.
00176     pclNode = static_cast<Timer*>(GetHead());
00177     while (pclNode)
00178     {
00179         pclPrev = NULL;
00180
00181         // If the timer expired, run the callbacks now.
00182         if (pclNode->m_ucFlags & TIMERLIST_FLAG_CALLBACK
00183     )
00184     {
00185         // Run the callback. these callbacks must be very fast...
00186         pclNode->m_pfCallback( pclNode->m_pclOwner
00187 , pclNode->m_pvData );
00188         pclNode->m_ucFlags &= ~TIMERLIST_FLAG_CALLBACK
00189 ;
00190
00191         // If this was a one-shot timer, let's remove it.
00192         if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT
00193     )
00194     {
00195         pclPrev = pclNode;
00196     }
00197     }
00198     pclNode = static_cast<Timer*>(pclNode->GetNext());
00199
00200     // Remove one-shot-timers
00201     if (pclPrev)
00202     {
00203         Remove(pclPrev);
00204     }
00205
00206     // Check to see how much time has elapsed since the time we
00207     // acknowledged the interrupt...
00208     ulOvertime = KernelTimer::GetOvertime();
00209
00210     if( ulOvertime >= ulNewExpiry ) {
00211         m_ulNextWakeup = ulOvertime;
00212         bContinue = 1;
00213     }
00214
00215     // If it's taken longer to go through this loop than would take us to
00216     // the next expiry, re-run the timing loop
00217     } while (bContinue);
00218
00219     // This timer elapsed, but there's nothing more to do...
00220     // Turn the timer off.
00221     if (ulNewExpiry >= MAX_TIMER_TICKS)
00222     {
00223         KernelTimer::Stop();
00224     }
00225     else
00226     {
00227         // Update the timer with the new "Next Wakeup" value, plus whatever
00228         // overtime has accumulated since the last time we called this handler
00229         m_ulNextWakeup = KernelTimer::SetExpiry
00230 (ulNewExpiry + ulOvertime);
00231     }
00232 }
00233
00234 //-----
00235 void Timer::Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_,
00236 TimerCallback_t pfCallback_, void *pvData_ )
00237 {
00238     SetIntervalMSeconds(ulIntervalMs_);
00239     m_pfCallback = pfCallback_;
00240     m_pvData = pvData_;
00241     if (!bRepeat_)
00242     {
00243         m_ucFlags = TIMERLIST_FLAG_ONE_SHOT;
00244     }
00245     else
00246     {
00247         m_ucFlags = 0;
00248     }
00249     m_pclOwner = Scheduler::GetCurrentThread
00250 ();
00251     TimerScheduler::Add(this);
00252 }
00253
00254 //-----

```

```

00250 void Timer::Stop()
00251 {
00252     TimerScheduler::Remove(this);
00253 }
00254
00255 //-----
00256 void Timer::SetIntervalTicks( K_ULONG ulTicks_ )
00257 {
00258     m_ulInterval = ulTicks_;
00259 }
00260
00261 //-----
00263 //-----
00264 void Timer::SetIntervalSeconds( K_ULONG ulSeconds_ )
00265 {
00266     m_ulInterval = SECONDS_TO_TICKS(ulSeconds_) - TL_FUDGE_FACTOR
00267 ;
00268 }
00269 //-----
00270 void Timer::SetIntervalMSeconds( K_ULONG ulMSeconds_ )
00271 {
00272     m_ulInterval = MSECONDS_TO_TICKS(ulMSeconds_) - TL_FUDGE_FACTOR
00273 ;
00274 }
00275 //-----
00276 void Timer::SetIntervalUSeconds( K_ULONG ulUSeconds_ )
00277 {
00278     m_ulInterval = USECONDS_TO_TICKS(ulUSeconds_) - TL_FUDGE_FACTOR
00279 ;
00280 }
00281 #endif //KERNEL_USE_TIMERS

```

14.147 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/timerlist.h File Reference

[Timer](#) list and timer-scheduling declarations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "thread.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)


```

00047 //-----
00048 #define MAX_TIMER_TICKS (0x7FFFFFFF)
00049
00050 //-----
00051 /*
00052     Ugly macros to support a wide resolution of delays.
00053     Given a 16-bit timer @ 16MHz & 256 cycle prescaler, this gives us...
00054     Max time, SECONDS_TO_TICKS: 68719s
00055     Max time, MSECONDS_TO_TICKS: 6871.9s
00056     Max time, USECONDS_TO_TICKS: 6.8719s
00057     With a 16us tick resolution.
00058 */
00059 //-----
00060 #define SECONDS_TO_TICKS(x) (((K_ULONG)x) * TIMER_FREQ)
00061 #define MSECONDS_TO_TICKS(x) (((((K_ULONG)x) * (TIMER_FREQ/100)) +
5) / 10))
00062 #define USECONDS_TO_TICKS(x) (((((K_ULONG)x) * TIMER_FREQ) + 50000)
/ 1000000))
00063
00064 //-----
00065 #define MIN_TICKS (3)
00066 //-----
00067 typedef void (*TimerCallback_t)(Thread *pclOwner_, void *pvData_);
00068
00069 //-----
00070 class TimerList;
00071 class TimerScheduler;
00072 class Quantum;
00073 class Timer : public LinkListNode
00074 {
00075 public:
00076     Timer() { m_ulInterval = 0; m_ulTimeLeft = 0;
m_ucFlags = 0; }
00077
00078     void Start( K_UCHAR bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t
pfCallback_, void *pvData_ );
00079
00080     void Stop();
00081
00082     void SetFlags( K_UCHAR ucFlags_ ) { m_ucFlags = ucFlags_; }
00083 ;
00084
00085     void SetCallback( TimerCallback_t pfCallback_ ) { m_pfCallback
= pfCallback_; };
00086
00087     void SetData( void *pvData_ ) { m_pvData = pvData_; };
00088
00089     void SetOwner( Thread *pclOwner_ ) { m_pclOwner =
pclOwner_; };
00090
00091     void SetIntervalTicks(K_ULONG ulTicks_);
00092
00093     void SetIntervalSeconds(K_ULONG ulSeconds_);
00094
00095     void SetIntervalMSeconds(K_ULONG ulMSeconds_);
00096
00097     void SetIntervalUSeconds(K_ULONG ulUSeconds_);
00098
00099 private:
00100     friend class TimerList;
00101
00102     K_UCHAR m_ucFlags;
00103
00104     TimerCallback_t m_pfCallback;
00105
00106     K_ULONG m_ulInterval;
00107
00108     K_ULONG m_ulTimeLeft;
00109
00110     Thread *m_pclOwner;
00111
00112     void *m_pvData;
00113 };
00114
00115 //-----
00116 class TimerList : public DoubleLinkedList
00117 {
00118 public:
00119     void Init();
00120
00121     void Add(Timer *pclListNode_);
00122
00123     void Remove(Timer *pclListNode_);
00124
00125     void Process();
00126
00127

```



```

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

```

14.151 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/tracebuffer.h File Reference

Kernel trace buffer class declaration.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "writebuf16.h"
```

14.151.1 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, subsets of kernel trace information can be extracted and analyzed to provide information about runtime performance, thread-scheduling, and other nifty things in real-time.

Definition in file [tracebuffer.h](#).

14.152 tracebuffer.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  #ifndef __TRACEBUFFER_H__
00016  #define __TRACEBUFFER_H__
00017
00018  #include "kerneltypes.h"
00019  #include "mark3cfg.h"
00020  #include "writebuf16.h"
00021
00022  #if KERNEL_USE_DEBUG
00023
00024  #define TRACE_BUFFER_SIZE
00025
00026  (16)

```

```

00034
00038 class TraceBuffer
00039 {
00040 public:
00046     static void Init();
00047
00055     static K_USHORT Increment();
00056
00065     static void Write( K_USHORT *pusData_, K_USHORT usSize_ );
00066
00075     void SetCallback( WriteBufferCallback pfCallback_ )
00076     { m_clBuffer.SetCallback( pfCallback_ ); }
00077 private:
00078
00079     static WriteBuffer16 m_clBuffer;
00080     static volatile K_USHORT m_usIndex;
00081     static K_USHORT m_ausBuffer[ (TRACE_BUFFER_SIZE / sizeof( K_USHORT )) ];
00082 };
00083
00084 #endif //KERNEL_USE_DEBUG
00085
00086 #endif

```

14.153 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.cpp File Reference

Unit test class definition.

```

#include "kerneltypes.h"
#include "unit_test.h"

```

14.153.1 Detailed Description

Unit test class definition.

Definition in file [unit_test.cpp](#).

14.154 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  =====*/
00019  #include "kerneltypes.h"
00020  #include "unit_test.h"
00021
00022  //-----
00023  UnitTest::UnitTest()
00024  {
00025      m_bIsActive = false;
00026      m_usIterations = 0;
00027      m_usPassed = 0;
00028      m_bComplete = false;
00029  }
00030
00031  //-----
00032  void UnitTest::Pass()
00033  {
00034      if (m_bComplete)
00035      {
00036          return;
00037      }

```

```
00038
00039     if (m_bIsActive)
00040     {
00041         m_bIsActive = false;
00042         m_usIterations++;
00043         m_usPassed++;
00044         m_bStatus = true;
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 }
```

14.155 [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit_test.h](#) File Reference

Unit test class declarations.

```
#include "kerneltypes.h"
```

Classes

- class `UnitTest`
Class used to implement a simple unit-testing framework.

14.155.1 Detailed Description

Unit test class declarations.

Definition in file [unit_test.h](#).

14.156 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 class UnitTest
00026 {

```



```

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
00122         m_pfCallback( &m_pusData[ m_usSize >>
00123 1], m_usSize >> 1 );
00124     }
00125     else
00126     {
00127         // 50% point - process the front-half of the buffer
00128         m_pfCallback( m_pusData, m_usSize >> 1
00129 );
00130     }
00131 }

```

14.159 /home/moslevin/googlecode/mark3/trunk/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.

14.159.1 Detailed Description

Thread-safe circular buffer implementation with 16-bit elements.

Definition in file [writebuf16.h](#).

14.160 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/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.cpp, 228](#)
[cpp, 161](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/blocking.h, 162, 163](#) [236, 237](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 242, 243](#)
[_button.cpp, 163, 164](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 244](#)
[_button.h, 166, 167](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernel-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control_debug.h, 244, 245](#)
[_checkbox.cpp, 168, 169](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 246](#)
[_checkbox.h, 171](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 247, 248](#)
[_groupbox.cpp, 172, 173](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 248, 249](#)
[_groupbox.h, 174](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 250, 251](#)
[_label.h, 175](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kernelswi-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 252](#)
[_panel.cpp, 176](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/keycodes.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 252, 253](#)
[_panel.h, 177](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 255](#)
[_progress.cpp, 178](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/kprofile.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/control.h, 256, 257](#)
[_progress.h, 179, 180](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.-cpp, 258](#)
[cpp, 181, 182](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/ll.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/dcpu.-h, 260, 261](#)
[h, 192, 196](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/manual.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/debug.h, 262](#)
[_tokens.h, 200, 202](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mark3cfg.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/draw.-h, 262, 265](#)
[h, 203, 204](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.epp, 265, 266](#)
[cpp, 206, 207](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/memutil.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/driver.h, 269](#)
[h, 208, 209](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed- cpp, 270, 271](#)
[_heap.cpp, 210, 211](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/message.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/fixed- h, 272, 274](#)
[_heap.h, 213](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/font.- cpp, 275](#)
[h, 214, 215](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/mutex.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h, 278, 279](#)
[cpp, 215, 216](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/graphics.h, 227](#)
[h, 227](#) [/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/profile.-](#)
[/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/gui.- h, 282, 283](#)

/home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.-
 cpp, 283, 284
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/writebuf16.-
 h, 285, 286
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.-
 cpp, 286, 287
 Activate
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.-
 h, 288, 289
 ButtonControl, 45
 CheckBoxControl, 47
 GroupBoxControl, 78
 GuiControl, 81
 LabelControl, 101
 PanelControl, 115
 ProgressControl, 120
 Screen, 126
 SubControl, 139
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/scheduler.-
 cpp, 289, 290
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/screen.-
 h, 291
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.-
 cpp, 292, 293
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/semaphore.-
 h, 295, 296
 Add
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.-
 cpp, 297
 CircularLinkList, 48
 DoubleLinkList, 58
 DriverList, 69
 LinkList, 103
 Scheduler, 123
 ThreadList, 147
 TimerList, 153
 TimerScheduler, 155
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.-
 h, 301, 302
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.-
 _mux.cpp, 302, 303
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slip.-
 _mux.h, 304, 305
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.-
 cpp, 306
 AddControl
 GuiWindow, 90
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/slipterm.-
 h, 307
 AddPlugin
 DCPU, 51
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system.-
 _heap.cpp, 308
 AddThread
 Quantum, 121
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system.-
 _heap.h, 310, 312
 AddWindow
 GuiEventSurface, 87
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/system.-
 _heap_config.h, 314, 315
 Alloc
 BlockHeap, 42
 FixedHeap, 70
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/thread.-
 aucBox
 cpp, 316
 control_checkbox.cpp, 168
 h, 320, 321
 aucCheck
 control_checkbox.cpp, 168
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.-
 cpp, 322, 323
 Bitmap
 GraphicsDriver, 74
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadlist.-
 h, 324
 Block
 BlockingObject, 43
 BlockHeap, 41
 Alloc, 42
 Create, 42
 Free, 42
 IsFree, 42
 BlockingObject, 43
 Block, 43
 OnBlock, 43
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.-
 cpp, 325, 326
 ButtonControl, 44
 Activate, 45
 Draw, 45
 Init, 45
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.-
 h, 328, 329
 ProcessEvent, 45
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.-
 cpp, 331, 332
 CS_ENTER
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.-
 h, 335, 336
 ButtonControl, 44
 Activate, 45
 Draw, 45
 Init, 45
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/threadport.-
 _test.cpp, 340
 /home/moslevin/googlecode/mark3/trunk/embedded/stage/src/unit.-
 _test.h, 341

- threadport.h, 329
- CS_EXIT
 - threadport.h, 329
- CheckBoxControl, 46
 - Activate, 47
 - Draw, 47
 - Init, 47
 - ProcessEvent, 47
- Checksum16
 - MemUtil, 106
- Checksum8
 - MemUtil, 106
- Circle
 - GraphicsDriver, 74
- CircularLinkedList, 47
 - Add, 48
 - Remove, 48
- Claim
 - Mutex, 114
- ClearStale
 - GuiControl, 81
- Close
 - DevNull, 56
 - Driver, 66
- CompareMemory
 - MemUtil, 106
- CompareStrings
 - MemUtil, 107
- Complete
 - UnitTest, 157
- ComputeCurrentTicks
 - ProfileTimer, 118
- ContextSwitchSWI
 - Thread, 142
- Control
 - DevNull, 56
 - Driver, 66
- control_checkbox.cpp
 - aucBox, 168
 - aucCheck, 168
- CopyEvent
 - GuiEventSurface, 87
- CopyMemory
 - MemUtil, 107
- CopyString
 - MemUtil, 107
- Create
 - BlockHeap, 42
 - FixedHeap, 70
- CycleFocus
 - GuiWindow, 90
- DCPU, 49
 - AddPlugin, 51
 - GetOperand, 51
 - GetRegisters, 51
 - HWN, 51
 - IAQ, 51
 - Init, 51
 - m_clPluginList, 52
 - RFI, 52
 - SendInterrupt, 52
- DCPU_OpBasic
 - dcpu.h, 194
- DCPU_OpExtended
 - dcpu.h, 195
- DCPU_Registers, 52
- DCPUPlugin, 53
 - Enumerate, 54
 - GetDeviceNumber, 54
 - Init, 54
 - Interrupt, 55
- DI
 - KernelSWI, 97
- dcpu.h
 - OP_18, 195
 - OP_19, 195
 - OP_1C, 195
 - OP_1D, 195
 - OP_ADD, 194
 - OP_ADX, 195
 - OP_AND, 194
 - OP_ASR, 195
 - OP_BOR, 194
 - OP_DIV, 194
 - OP_DVI, 194
 - OP_EX_13, 196
 - OP_EX_14, 196
 - OP_EX_15, 196
 - OP_EX_16, 196
 - OP_EX_17, 196
 - OP_EX_18, 196
 - OP_EX_19, 196
 - OP_EX_1A, 196
 - OP_EX_1B, 196
 - OP_EX_1C, 196
 - OP_EX_1D, 196
 - OP_EX_1E, 196
 - OP_EX_1F, 196
 - OP_EX_2, 195
 - OP_EX_3, 195
 - OP_EX_4, 195
 - OP_EX_5, 195
 - OP_EX_6, 195
 - OP_EX_7, 195
 - OP_EX_D, 195
 - OP_EX_E, 195
 - OP_EX_F, 195
 - OP_EX_HWI, 196
 - OP_EX_HWN, 195
 - OP_EX_HWQ, 195
 - OP_EX_IAG, 195
 - OP_EX_IAQ, 195
 - OP_EX_IAS, 195
 - OP_EX_INT, 195
 - OP_EX_JSR, 195
 - OP_EX_RFI, 195

- OP_IFA, [195](#)
- OP_IFB, [195](#)
- OP_IFC, [195](#)
- OP_IFE, [195](#)
- OP_IFG, [195](#)
- OP_IFL, [195](#)
- OP_IFN, [195](#)
- OP_IFU, [195](#)
- OP_MDI, [194](#)
- OP_MLI, [194](#)
- OP_MOD, [194](#)
- OP_MUL, [194](#)
- OP_NON_BASIC, [194](#)
- OP_SBX, [195](#)
- OP_SET, [194](#)
- OP_SHL, [195](#)
- OP_SHR, [195](#)
- OP_STD, [195](#)
- OP_STI, [195](#)
- OP_SUB, [194](#)
- OP_XOR, [195](#)
- dcpu.h
 - DCPU_OpBasic, [194](#)
 - DCPU_OpExtended, [195](#)
- Deactivate
 - Screen, [126](#)
- DecimalToHex
 - MemUtil, [107](#)
- DecimalToString
 - MemUtil, [108](#)
- DecodeByte
 - Slip, [132](#)
- DevNull, [55](#)
 - Close, [56](#)
 - Control, [56](#)
 - Open, [56](#)
 - Read, [56](#)
 - Write, [57](#)
- DoubleLinkedList, [57](#)
 - Add, [58](#)
 - Remove, [58](#)
- Draw
 - ButtonControl, [45](#)
 - CheckBoxControl, [47](#)
 - GroupBoxControl, [78](#)
 - GuiControl, [81](#)
 - LabelControl, [101](#)
 - PanelControl, [115](#)
 - ProgressControl, [120](#)
 - StubControl, [139](#)
- DrawBitmap_t, [58](#)
- DrawCircle_t, [59](#)
- DrawEllipse_t, [60](#)
- DrawLine_t, [60](#)
- DrawMove_t, [61](#)
- DrawPixel
 - GraphicsDriver, [74](#)
- DrawPoint_t, [61](#)
- DrawPoly_t, [62](#)
- DrawRectangle_t, [62](#)
- DrawStamp_t, [63](#)
- DrawText_t, [64](#)
- DrawVector_t, [64](#)
- DrawWindow_t, [65](#)
- Driver, [65](#)
 - Close, [66](#)
 - Control, [66](#)
 - GetPath, [67](#)
 - Open, [67](#)
 - Read, [67](#)
 - SetName, [67](#)
 - Write, [67](#)
- DriverList, [68](#)
 - Add, [69](#)
 - FindByPath, [69](#)
 - Init, [69](#)
 - Remove, [69](#)
- EVENT_TYPE_COUNT
 - gui.h, [237](#)
- EVENT_TYPE_JOYSTICK
 - gui.h, [237](#)
- EVENT_TYPE_KEYBOARD
 - gui.h, [237](#)
- EVENT_TYPE_MOUSE
 - gui.h, [237](#)
- EVENT_TYPE_TIMER
 - gui.h, [237](#)
- EVENT_TYPE_TOUCH
 - gui.h, [237](#)
- Ellipse
 - GraphicsDriver, [74](#)
- EncodeByte
 - Slip, [132](#)
- Enumerate
 - DCPUPlugin, [54](#)
- Exit
 - Thread, [142](#)
- FindByPath
 - DriverList, [69](#)
- FixedHeap, [69](#)
 - Alloc, [70](#)
 - Create, [70](#)
 - Free, [70](#)
- Font_t, [71](#)
- Free
 - BlockHeap, [42](#)
 - FixedHeap, [70](#)
- GetAverage
 - ProfileTimer, [118](#)
- GetCode
 - Message, [110](#)
- GetControlIndex
 - GuiControl, [81](#)
- GetControlOffset

- GuiControl, [81](#)
- GetCount
 - MessageQueue, [111](#)
 - Semaphore, [128](#)
- GetCurrent
 - ProfileTimer, [118](#)
 - Thread, [142](#)
- GetCurrentThread
 - Scheduler, [123](#)
- GetData
 - Message, [110](#)
- GetDeviceNumber
 - DCPUPlugin, [54](#)
- GetDriver
 - GuiWindow, [90](#)
 - Slip, [132](#)
 - SlipMux, [135](#)
- GetFailed
 - UnitTest, [157](#)
- GetHead
 - LinkList, [103](#)
- GetHeight
 - GuiControl, [82](#)
 - GuiWindow, [91](#)
- GetID
 - Thread, [143](#)
- GetLeft
 - GuiControl, [82](#)
 - GuiWindow, [91](#)
- GetMaxZOrder
 - GuiWindow, [91](#)
- GetName
 - UnitTest, [157](#)
- GetNext
 - LinkListNode, [105](#)
- GetNextThread
 - Scheduler, [123](#)
- GetOperand
 - DCPU, [51](#)
- GetOvertime
 - KernelTimer, [98](#)
- GetOwner
 - Thread, [143](#)
- GetParentControl
 - GuiControl, [82](#)
- GetParentWindow
 - GuiControl, [82](#)
- GetPassed
 - UnitTest, [157](#)
- GetPath
 - Driver, [67](#)
- GetPrev
 - LinkListNode, [105](#)
- GetPriority
 - Thread, [143](#)
- GetQuantum
 - Thread, [143](#)
- GetQueue
 - SlipMux, [135](#)
- GetRegisters
 - DCPU, [51](#)
- GetResult
 - UnitTest, [158](#)
- GetSlip
 - SlipMux, [135](#)
- GetStackSlack
 - Thread, [143](#)
- GetStopList
 - Scheduler, [123](#)
- GetTail
 - LinkList, [103](#)
- GetThreadList
 - Scheduler, [124](#)
- GetTop
 - GuiControl, [82](#)
 - GuiWindow, [91](#)
- GetTotal
 - UnitTest, [158](#)
- GetWidth
 - GuiControl, [82](#)
 - GuiWindow, [91](#)
- GetZOrder
 - GuiControl, [83](#)
- GlobalMessagePool, [71](#)
 - Pop, [72](#)
 - Push, [72](#)
- Glyph_t, [72](#)
- GraphicsDriver, [73](#)
 - Bitmap, [74](#)
 - Circle, [74](#)
 - DrawPixel, [74](#)
 - Ellipse, [74](#)
 - Line, [75](#)
 - Move, [75](#)
 - Point, [75](#)
 - ReadPixel, [75](#)
 - Rectangle, [75](#)
 - SetWindow, [76](#)
 - Stamp, [76](#)
 - Text, [76](#)
 - TriangleFill, [76](#)
 - TriangleWire, [76](#)
- GroupBoxControl, [77](#)
 - Activate, [78](#)
 - Draw, [78](#)
 - Init, [78](#)
 - ProcessEvent, [78](#)
- gui.h
 - EVENT_TYPE_COUNT, [237](#)
 - EVENT_TYPE_JOYSTICK, [237](#)
 - EVENT_TYPE_KEYBOARD, [237](#)
 - EVENT_TYPE_MOUSE, [237](#)
 - EVENT_TYPE_TIMER, [237](#)
 - EVENT_TYPE_TOUCH, [237](#)
- gui.h
 - GuiEventType_t, [237](#)

- GuiControl, 78
 - Activate, 81
 - ClearStale, 81
 - Draw, 81
 - GetControllIndex, 81
 - GetControlOffset, 81
 - GetHeight, 82
 - GetLeft, 82
 - GetParentControl, 82
 - GetParentWindow, 82
 - GetTop, 82
 - GetWidth, 82
 - GetZOrder, 83
 - Init, 83
 - IsInFocus, 83
 - IsStale, 83
 - m_ucControllIndex, 85
 - m_ucZOrder, 85
 - ProcessEvent, 83
 - SetControllIndex, 84
 - SetHeight, 84
 - SetLeft, 84
 - SetParentControl, 84
 - SetParentWindow, 84
 - SetTop, 85
 - SetWidth, 85
 - SetZOrder, 85
- GuiEvent_t, 86
- GuiEventSurface, 86
 - AddWindow, 87
 - CopyEvent, 87
 - Init, 87
 - ProcessEvent, 87
 - RemoveWindow, 87
 - SendEvent, 88
- GuiEventType_t
 - gui.h, 237
- GuiWindow, 88
 - AddControl, 90
 - CycleFocus, 90
 - GetDriver, 90
 - GetHeight, 91
 - GetLeft, 91
 - GetMaxZOrder, 91
 - GetTop, 91
 - GetWidth, 91
 - Init, 91
 - IsInFocus, 92
 - m_pclDriver, 94
 - ProcessEvent, 92
 - Redraw, 92
 - RemoveControl, 92
 - SetDriver, 92
 - SetFocus, 93
 - SetHeight, 93
 - SetLeft, 93
 - SetTop, 93
 - SetWidth, 93
- HEAP_BLOCK_SIZE_1
 - system_heap_config.h, 315
- HEAP_RAW_SIZE
 - system_heap.h, 311
- HEAP_RAW_SIZE_1
 - system_heap.h, 311
- HWN
 - DCPU, 51
- HeapConfig, 94
- HighestWaiter
 - ThreadList, 147
- IAQ
 - DCPU, 51
- InheritPriority
 - Thread, 144
- Init
 - ButtonControl, 45
 - CheckBoxControl, 47
 - DCPU, 51
 - DCPUPlugin, 54
 - DriverList, 69
 - GroupBoxControl, 78
 - GuiControl, 83
 - GuiEventSurface, 87
 - GuiWindow, 91
 - Kernel, 96
 - LabelControl, 101
 - PanelControl, 116
 - Profiler, 117
 - ProfileTimer, 119
 - ProgressControl, 120
 - Semaphore, 128
 - SlipMux, 135
 - SlipTerm, 137
 - StubControl, 139
 - Thread, 144
 - TimerList, 153
 - TimerScheduler, 155
- InitStack
 - ThreadPort, 149
- InstallHandler
 - SlipMux, 135
- Interrupt
 - DCPUPlugin, 55
- IsEnabled
 - Scheduler, 124
- IsFree
 - BlockHeap, 42
- IsInFocus
 - GuiControl, 83
 - GuiWindow, 92
- IsStale
 - GuiControl, 83
- JoystickEvent_t, 94
- KERNEL_USE_DRIVER
 - mark3cfg.h, 263

- KERNEL_USE_MESSAGE
 - mark3cfg.h, 263
- KERNEL_USE_MUTEX
 - mark3cfg.h, 264
- KERNEL_USE_PROFILER
 - mark3cfg.h, 264
- KERNEL_USE_QUANTUM
 - mark3cfg.h, 264
- KERNEL_USE_TIMERS
 - mark3cfg.h, 264
- Kernel, 95
 - Init, 96
 - Start, 96
- KernelSWI, 96
 - DI, 97
 - RI, 97
- KernelTimer, 97
 - GetOvertime, 98
 - RI, 98
 - Read, 98
 - SetExpiry, 99
 - SubtractExpiry, 99
 - TimeToExpiry, 99
- KeyEvent_t, 100
- LabelControl, 100
 - Activate, 101
 - Draw, 101
 - Init, 101
 - ProcessEvent, 102
- Line
 - GraphicsDriver, 75
- LinkList, 102
 - Add, 103
 - GetHead, 103
 - GetTail, 103
 - Remove, 103
- LinkListNode, 103
 - GetNext, 105
 - GetPrev, 105
- m_clPluginList
 - DCPU, 52
- m_pclDriver
 - GuiWindow, 94
- m_ucControlIndex
 - GuiControl, 85
- m_ucVerbosity
 - SlipTerm, 138
- m_ucZOrder
 - GuiControl, 85
- mark3cfg.h
 - KERNEL_USE_DRIVER, 263
 - KERNEL_USE_MESSAGE, 263
 - KERNEL_USE_MUTEX, 264
 - KERNEL_USE_PROFILER, 264
 - KERNEL_USE_QUANTUM, 264
 - KERNEL_USE_TIMERS, 264
- MemUtil, 105
 - Checksum16, 106
 - Checksum8, 106
 - CompareMemory, 106
 - CompareStrings, 107
 - CopyMemory, 107
 - CopyString, 107
 - DecimalToHex, 107
 - DecimalToString, 108
 - SetMemory, 108
 - StringLength, 108
 - StringSearch, 108
- Message, 109
 - GetCode, 110
 - GetData, 110
 - SetCode, 110
 - SetData, 110
- MessageQueue, 110
 - GetCount, 111
 - Receive, 111
 - Send, 112
- MessageReceive
 - SlipMux, 136
- MouseEvent_t, 112
- Move
 - GraphicsDriver, 75
- Mutex, 113
 - Claim, 114
 - Release, 114
 - WakeMe, 114
- OP_18
 - dcpu.h, 195
- OP_19
 - dcpu.h, 195
- OP_1C
 - dcpu.h, 195
- OP_1D
 - dcpu.h, 195
- OP_ADD
 - dcpu.h, 194
- OP_ADX
 - dcpu.h, 195
- OP_AND
 - dcpu.h, 194
- OP_ASR
 - dcpu.h, 195
- OP_BOR
 - dcpu.h, 194
- OP_DIV
 - dcpu.h, 194
- OP_DVI
 - dcpu.h, 194
- OP_EX_13
 - dcpu.h, 196
- OP_EX_14
 - dcpu.h, 196
- OP_EX_15
 - dcpu.h, 196
- OP_EX_16

- dcpu.h, [196](#)
- OP_EX_17
 - dcpu.h, [196](#)
- OP_EX_18
 - dcpu.h, [196](#)
- OP_EX_19
 - dcpu.h, [196](#)
- OP_EX_1A
 - dcpu.h, [196](#)
- OP_EX_1B
 - dcpu.h, [196](#)
- OP_EX_1C
 - dcpu.h, [196](#)
- OP_EX_1D
 - dcpu.h, [196](#)
- OP_EX_1E
 - dcpu.h, [196](#)
- OP_EX_1F
 - dcpu.h, [196](#)
- OP_EX_2
 - dcpu.h, [195](#)
- OP_EX_3
 - dcpu.h, [195](#)
- OP_EX_4
 - dcpu.h, [195](#)
- OP_EX_5
 - dcpu.h, [195](#)
- OP_EX_6
 - dcpu.h, [195](#)
- OP_EX_7
 - dcpu.h, [195](#)
- OP_EX_D
 - dcpu.h, [195](#)
- OP_EX_E
 - dcpu.h, [195](#)
- OP_EX_F
 - dcpu.h, [195](#)
- OP_EX_HWI
 - dcpu.h, [196](#)
- OP_EX_HWN
 - dcpu.h, [195](#)
- OP_EX_HWQ
 - dcpu.h, [195](#)
- OP_EX_IAG
 - dcpu.h, [195](#)
- OP_EX_IAQ
 - dcpu.h, [195](#)
- OP_EX_IAS
 - dcpu.h, [195](#)
- OP_EX_INT
 - dcpu.h, [195](#)
- OP_EX_JSR
 - dcpu.h, [195](#)
- OP_EX_RFI
 - dcpu.h, [195](#)
- OP_IFA
 - dcpu.h, [195](#)
- OP_IFB
 - dcpu.h, [195](#)
- OP_IFC
 - dcpu.h, [195](#)
- OP_IFE
 - dcpu.h, [195](#)
- OP_IFG
 - dcpu.h, [195](#)
- OP_IFL
 - dcpu.h, [195](#)
- OP_IFN
 - dcpu.h, [195](#)
- OP_IFU
 - dcpu.h, [195](#)
- OP_MDI
 - dcpu.h, [194](#)
- OP_MLI
 - dcpu.h, [194](#)
- OP_MOD
 - dcpu.h, [194](#)
- OP_MUL
 - dcpu.h, [194](#)
- OP_NON_BASIC
 - dcpu.h, [194](#)
- OP_SBX
 - dcpu.h, [195](#)
- OP_SET
 - dcpu.h, [194](#)
- OP_SHL
 - dcpu.h, [195](#)
- OP_SHR
 - dcpu.h, [195](#)
- OP_STD
 - dcpu.h, [195](#)
- OP_STI
 - dcpu.h, [195](#)
- OP_SUB
 - dcpu.h, [194](#)
- OP_XOR
 - dcpu.h, [195](#)
- Open
 - DevNull, [56](#)
 - Driver, [67](#)
- PanelControl, [115](#)
 - Activate, [115](#)
 - Draw, [115](#)
 - Init, [116](#)
 - ProcessEvent, [116](#)
- Pend
 - Semaphore, [129](#)
- Point
 - GraphicsDriver, [75](#)
- Pop
 - GlobalMessagePool, [72](#)
- PrintLn
 - SlipTerm, [137](#)
- Process
 - TimerList, [154](#)
 - TimerScheduler, [155](#)

- ProcessEvent
 - ButtonControl, 45
 - CheckBoxControl, 47
 - GroupBoxControl, 78
 - GuiControl, 83
 - GuiEventSurface, 87
 - GuiWindow, 92
 - LabelControl, 102
 - PanelControl, 116
 - ProgressControl, 120
 - StubControl, 139
- ProfileTimer, 117
 - ComputeCurrentTicks, 118
 - GetAverage, 118
 - GetCurrent, 118
 - Init, 119
 - Start, 119
- Profiler, 116
 - Init, 117
- ProgressControl, 119
 - Activate, 120
 - Draw, 120
 - Init, 120
 - ProcessEvent, 120
- Push
 - GlobalMessagePool, 72
- Quantum, 121
 - AddThread, 121
 - RemoveThread, 121
 - SetTimer, 121
 - UpdateTimer, 122
- RFI
 - DCPU, 52
- RI
 - KernelSWI, 97
 - KernelTimer, 98
- Read
 - DevNull, 56
 - Driver, 67
 - KernelTimer, 98
- ReadData
 - Slip, 132
- ReadPixel
 - GraphicsDriver, 75
- Receive
 - MessageQueue, 111
- Rectangle
 - GraphicsDriver, 75
- Redraw
 - GuiWindow, 92
- Release
 - Mutex, 114
- Remove
 - CircularLinkList, 48
 - DoubleLinkList, 58
 - DriverList, 69
 - LinkList, 103
 - Scheduler, 124
 - ThreadList, 147
 - TimerList, 154
 - TimerScheduler, 155
- RemoveControl
 - GuiWindow, 92
- RemoveThread
 - Quantum, 121
- RemoveWindow
 - GuiEventSurface, 87
- SLIP_CHANNEL_GRAPHICS
 - slip.h, 301
- SLIP_CHANNEL_HID
 - slip.h, 301
- SLIP_CHANNEL_NVM
 - slip.h, 301
- SLIP_CHANNEL_RESET
 - slip.h, 301
- SLIP_CHANNEL_TERMINAL
 - slip.h, 301
- SLIP_CHANNEL_UNISCOPE
 - slip.h, 301
- Schedule
 - Scheduler, 124
- Scheduler, 122
 - Add, 123
 - GetCurrentThread, 123
 - GetNextThread, 123
 - GetStopList, 123
 - GetThreadList, 124
 - IsEnabled, 124
 - Remove, 124
 - Schedule, 124
 - SetScheduler, 124
- Screen, 125
 - Activate, 126
 - Deactivate, 126
- ScreenList, 126
- ScreenManager, 127
- Semaphore, 127
 - GetCount, 128
 - Init, 128
 - Pend, 129
 - SetExpired, 129
 - WakeMe, 129
- Send
 - MessageQueue, 112
- SendEvent
 - GuiEventSurface, 88
- SendInterrupt
 - DCPU, 52
- SetBuffers
 - WriteBuffer16, 159
- SetCallback
 - Timer, 150
 - WriteBuffer16, 159
- SetCode
 - Message, 110

- SetControllIndex
 - GuiControl, [84](#)
- SetCurrent
 - Thread, [144](#)
- SetData
 - Message, [110](#)
 - Timer, [150](#)
- SetDriver
 - GuiWindow, [92](#)
 - Slip, [133](#)
- SetExpired
 - Semaphore, [129](#)
- SetExpiry
 - KernelTimer, [99](#)
- SetFlagPointer
 - ThreadList, [148](#)
- SetFlags
 - Timer, [151](#)
- SetFocus
 - GuiWindow, [93](#)
- SetHeight
 - GuiControl, [84](#)
 - GuiWindow, [93](#)
- SetID
 - Thread, [144](#)
- SetIntervalMSeconds
 - Timer, [151](#)
- SetIntervalSeconds
 - Timer, [151](#)
- SetIntervalTicks
 - Timer, [151](#)
- SetIntervalUSeconds
 - Timer, [151](#)
- SetLeft
 - GuiControl, [84](#)
 - GuiWindow, [93](#)
- SetMemory
 - MemUtil, [108](#)
- SetName
 - Driver, [67](#)
 - Thread, [145](#)
 - UnitTest, [158](#)
- SetOwner
 - Thread, [145](#)
 - Timer, [152](#)
- SetParentControl
 - GuiControl, [84](#)
- SetParentWindow
 - GuiControl, [84](#)
- SetPriority
 - Thread, [145](#)
 - ThreadList, [148](#)
- SetQuantum
 - Thread, [145](#)
- SetQueue
 - SlipMux, [136](#)
- SetScheduler
 - Scheduler, [124](#)
- SetTimer
 - Quantum, [121](#)
- SetTop
 - GuiControl, [85](#)
 - GuiWindow, [93](#)
- SetVerbosity
 - SlipTerm, [137](#)
- SetWidth
 - GuiControl, [85](#)
 - GuiWindow, [93](#)
- SetWindow
 - GraphicsDriver, [76](#)
- SetZOrder
 - GuiControl, [85](#)
- Shell, [129](#)
- ShellCommand, [130](#)
- Sleep
 - Thread, [145](#)
- Slip, [131](#)
 - DecodeByte, [132](#)
 - EncodeByte, [132](#)
 - GetDriver, [132](#)
 - ReadData, [132](#)
 - SetDriver, [133](#)
 - WriteData, [133](#)
 - WriteVector, [133](#)
- slip.h
 - SLIP_CHANNEL_GRAPHICS, [301](#)
 - SLIP_CHANNEL_HID, [301](#)
 - SLIP_CHANNEL_NVM, [301](#)
 - SLIP_CHANNEL_RESET, [301](#)
 - SLIP_CHANNEL_TERMINAL, [301](#)
 - SLIP_CHANNEL_UNISCOPE, [301](#)
- slip.h
 - SlipChannel, [301](#)
- slip_mux.cpp
 - SlipMux_CallBack, [303](#)
- SlipChannel
 - slip.h, [301](#)
- SlipDataVector, [133](#)
- SlipMux, [134](#)
 - GetDriver, [135](#)
 - GetQueue, [135](#)
 - GetSlip, [135](#)
 - Init, [135](#)
 - InstallHandler, [135](#)
 - MessageReceive, [136](#)
 - SetQueue, [136](#)
- SlipMux_CallBack
 - slip_mux.cpp, [303](#)
- SlipTerm, [136](#)
 - Init, [137](#)
 - m_ucVerbosity, [138](#)
 - PrintLn, [137](#)
 - SetVerbosity, [137](#)
 - StrLen, [138](#)
- Stamp
 - GraphicsDriver, [76](#)

- Start
 - Kernel, 96
 - ProfileTimer, 119
- Stop
 - Timer, 152
- StrLen
 - SlipTerm, 138
- StringLength
 - MemUtil, 108
- StringSearch
 - MemUtil, 108
- StubControl, 138
 - Activate, 139
 - Draw, 139
 - Init, 139
 - ProcessEvent, 139
- SubtractExpiry
 - KernelTimer, 99
- system_heap.h
 - HEAP_RAW_SIZE, 311
 - HEAP_RAW_SIZE_1, 311
- SystemHeap, 140
- TL_FUDGE_FACTOR
 - timerlist.cpp, 331
- Text
 - GraphicsDriver, 76
- Thread, 140
 - ContextSwitchSWI, 142
 - Exit, 142
 - GetCurrent, 142
 - GetID, 143
 - GetOwner, 143
 - GetPriority, 143
 - GetQuantum, 143
 - GetStackSlack, 143
 - InheritPriority, 144
 - Init, 144
 - SetCurrent, 144
 - SetID, 144
 - SetName, 145
 - SetOwner, 145
 - SetPriority, 145
 - SetQuantum, 145
 - Sleep, 145
 - Yield, 146
- ThreadList, 146
 - Add, 147
 - HighestWaiter, 147
 - Remove, 147
 - SetFlagPointer, 148
 - SetPriority, 148
- ThreadPort, 148
 - InitStack, 149
- threadport.h
 - CS_ENTER, 329
 - CS_EXIT, 329
- TimeToExpiry
 - KernelTimer, 99
- Timer, 149
 - SetCallback, 150
 - SetData, 150
 - SetFlags, 151
 - SetIntervalMSeconds, 151
 - SetIntervalSeconds, 151
 - SetIntervalTicks, 151
 - SetIntervalUSeconds, 151
 - SetOwner, 152
 - Stop, 152
- TimerEvent_t, 152
- TimerList, 152
 - Add, 153
 - Init, 153
 - Process, 154
 - Remove, 154
- TimerScheduler, 154
 - Add, 155
 - Init, 155
 - Process, 155
 - Remove, 155
- timerlist.cpp
 - TL_FUDGE_FACTOR, 331
- TouchEvent_t, 155
- TriangleFill
 - GraphicsDriver, 76
- TriangleWire
 - GraphicsDriver, 76
- UnBlock
 - BlockingObject, 43
- UnitTest, 156
 - Complete, 157
 - GetFailed, 157
 - GetName, 157
 - GetPassed, 157
 - GetResult, 158
 - GetTotal, 158
 - SetName, 158
- UpdateTimer
 - Quantum, 122
- WakeMe
 - Mutex, 114
 - Semaphore, 129
- Write
 - DevNull, 57
 - Driver, 67
- WriteBuffer16, 158
 - SetBuffers, 159
 - SetCallback, 159
 - WriteData, 160
 - WriteVector, 160
- WriteData
 - Slip, 133
 - WriteBuffer16, 160
- WriteVector
 - Slip, 133
 - WriteBuffer16, 160

Yield

Thread, [146](#)