

Mark3 Realtime Kernel

Generated by Doxygen 1.8.6

Thu Mar 5 2015 21:04:26

Contents

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

6.6	Event Flags	21
6.7	Messages	22
6.7.1	Message Objects	22
6.7.2	Global Message Pool	23
6.7.3	Message Queues	23
6.7.4	Messaging Example	23
6.8	Sleep	24
6.9	Round-Robin Quantum	24
7	Build System	25
7.1	Source Layout	25
7.2	Building the kernel	25
7.3	Building on Windows	27
8	License	29
8.1	License	29
9	Profiling Results	31
9.1	Date Performed	31
9.2	Compiler Information	31
9.3	Profiling Results	31
10	Code Size Profiling	33
10.1	Information	33
10.2	Compiler Version	33
10.3	Profiling Results	33
11	Hierarchical Index	35
11.1	Class Hierarchy	35
12	Class Index	39
12.1	Class List	39
13	File Index	43
13.1	File List	43
14	Class Documentation	49
14.1	BitStreamer Class Reference	49
14.1.1	Detailed Description	49
14.1.2	Member Function Documentation	49
14.1.2.1	AdvanceByte	49
14.1.2.2	Init	50
14.1.2.3	ReadBits	50

14.2	BlockHeap Class Reference	50
14.2.1	Detailed Description	51
14.2.2	Member Function Documentation	51
14.2.2.1	Alloc	51
14.2.2.2	Create	51
14.2.2.3	Free	51
14.2.2.4	IsFree	51
14.3	BlockingObject Class Reference	52
14.3.1	Detailed Description	52
14.3.2	Member Function Documentation	52
14.3.2.1	Block	52
14.3.2.2	UnBlock	53
14.4	ButtonControl Class Reference	53
14.4.1	Detailed Description	54
14.4.2	Member Function Documentation	54
14.4.2.1	Activate	54
14.4.2.2	Draw	54
14.4.2.3	Init	54
14.4.2.4	ProcessEvent	54
14.5	CheckBoxControl Class Reference	55
14.5.1	Detailed Description	56
14.5.2	Member Function Documentation	56
14.5.2.1	Activate	56
14.5.2.2	Draw	56
14.5.2.3	Init	56
14.5.2.4	ProcessEvent	56
14.6	CircularLinkedList Class Reference	57
14.6.1	Detailed Description	57
14.6.2	Member Function Documentation	57
14.6.2.1	Add	57
14.6.2.2	Remove	57
14.7	CommandLine_t Struct Reference	58
14.7.1	Detailed Description	58
14.8	DCPU Class Reference	58
14.8.1	Detailed Description	60
14.8.2	Member Function Documentation	60
14.8.2.1	AddPlugin	60
14.8.2.2	GetOperand	60
14.8.2.3	GetRegisters	61
14.8.2.4	HWN	61

14.8.2.5	IAQ	61
14.8.2.6	Init	61
14.8.2.7	RFI	61
14.8.2.8	SendInterrupt	61
14.8.3	Member Data Documentation	62
14.8.3.1	m_clPluginList	62
14.9	DCPU_Registers Struct Reference	62
14.9.1	Detailed Description	62
14.10	DCPUPlugin Class Reference	62
14.10.1	Detailed Description	63
14.10.2	Member Function Documentation	63
14.10.2.1	Enumerate	63
14.10.2.2	GetDeviceNumber	64
14.10.2.3	Init	64
14.10.2.4	Interrupt	64
14.11	DevNull Class Reference	64
14.11.1	Detailed Description	65
14.11.2	Member Function Documentation	65
14.11.2.1	Close	65
14.11.2.2	Control	65
14.11.2.3	Open	66
14.11.2.4	Read	66
14.11.2.5	Write	66
14.12	DoubleLinkedList Class Reference	67
14.12.1	Detailed Description	67
14.12.2	Member Function Documentation	67
14.12.2.1	Add	67
14.12.2.2	Remove	68
14.13	DrawBitmap_t Struct Reference	68
14.13.1	Detailed Description	68
14.14	DrawCircle_t Struct Reference	69
14.14.1	Detailed Description	69
14.15	DrawEllipse_t Struct Reference	69
14.15.1	Detailed Description	70
14.16	DrawLine_t Struct Reference	70
14.16.1	Detailed Description	70
14.17	DrawMove_t Struct Reference	70
14.17.1	Detailed Description	71
14.18	DrawPoint_t Struct Reference	71
14.18.1	Detailed Description	71

14.19DrawPoly_t Struct Reference	71
14.19.1 Detailed Description	72
14.20DrawRectangle_t Struct Reference	72
14.20.1 Detailed Description	72
14.21DrawStamp_t Struct Reference	73
14.21.1 Detailed Description	73
14.22DrawText_t Struct Reference	73
14.22.1 Detailed Description	74
14.23DrawVector_t Struct Reference	74
14.23.1 Detailed Description	74
14.24DrawWindow_t Struct Reference	74
14.24.1 Detailed Description	75
14.25Driver Class Reference	75
14.25.1 Detailed Description	76
14.25.2 Member Function Documentation	76
14.25.2.1 Close	76
14.25.2.2 Control	76
14.25.2.3 GetPath	76
14.25.2.4 Open	77
14.25.2.5 Read	77
14.25.2.6 SetName	77
14.25.2.7 Write	77
14.26DriverList Class Reference	78
14.26.1 Detailed Description	78
14.26.2 Member Function Documentation	78
14.26.2.1 Add	78
14.26.2.2 FindByPath	78
14.26.2.3 Init	78
14.26.2.4 Remove	79
14.27EventFlag Class Reference	79
14.27.1 Detailed Description	80
14.27.2 Member Function Documentation	80
14.27.2.1 Clear	80
14.27.2.2 GetMask	80
14.27.2.3 Set	80
14.27.2.4 Wait	80
14.27.2.5 Wait	81
14.27.2.6 Wait_i	81
14.28FixedHeap Class Reference	81
14.28.1 Detailed Description	82

14.28.2 Member Function Documentation	82
14.28.2.1 Alloc	82
14.28.2.2 Create	82
14.28.2.3 Free	82
14.29Font_t Struct Reference	83
14.29.1 Detailed Description	83
14.30GamePanelControl Class Reference	83
14.30.1 Detailed Description	84
14.30.2 Member Function Documentation	84
14.30.2.1 Activate	84
14.30.2.2 Draw	84
14.30.2.3 Init	84
14.30.2.4 ProcessEvent	84
14.31GlobalMessagePool Class Reference	85
14.31.1 Detailed Description	85
14.31.2 Member Function Documentation	85
14.31.2.1 Pop	85
14.31.2.2 Push	85
14.32Glyph_t Struct Reference	86
14.32.1 Detailed Description	86
14.33GraphicsDriver Class Reference	86
14.33.1 Detailed Description	87
14.33.2 Member Function Documentation	88
14.33.2.1 Bitmap	88
14.33.2.2 Circle	88
14.33.2.3 DrawPixel	88
14.33.2.4 Ellipse	88
14.33.2.5 Line	88
14.33.2.6 Move	88
14.33.2.7 Point	89
14.33.2.8 ReadPixel	89
14.33.2.9 Rectangle	89
14.33.2.10SetWindow	89
14.33.2.11Stamp	89
14.33.2.12Text	89
14.33.2.13TextFX	90
14.33.2.14TriangleFill	90
14.33.2.15TriangleWire	90
14.34GroupBoxControl Class Reference	90
14.34.1 Detailed Description	91

14.34.2 Member Function Documentation	91
14.34.2.1 Activate	91
14.34.2.2 Draw	91
14.34.2.3 Init	92
14.34.2.4 ProcessEvent	92
14.35 GuiControl Class Reference	92
14.35.1 Detailed Description	95
14.35.2 Member Function Documentation	95
14.35.2.1 Activate	95
14.35.2.2 ClearStale	95
14.35.2.3 Draw	95
14.35.2.4 GetControllIndex	96
14.35.2.5 GetControlOffset	96
14.35.2.6 GetHeight	96
14.35.2.7 GetLeft	96
14.35.2.8 GetParentControl	96
14.35.2.9 GetParentWindow	97
14.35.2.10 GetTop	97
14.35.2.11 GetWidth	97
14.35.2.12 GetZOrder	97
14.35.2.13 Init	97
14.35.2.14 IsInFocus	97
14.35.2.15 IsStale	98
14.35.2.16 ProcessEvent	98
14.35.2.17 SetControllIndex	98
14.35.2.18 SetHeight	98
14.35.2.19 SetLeft	98
14.35.2.20 SetParentControl	98
14.35.2.21 SetParentWindow	99
14.35.2.22 SetTop	99
14.35.2.23 SetWidth	99
14.35.2.24 SetZOrder	99
14.35.3 Member Data Documentation	99
14.35.3.1 m_ucControllIndex	99
14.35.3.2 m_ucZOrder	100
14.36 GuiEvent_t Struct Reference	100
14.36.1 Detailed Description	100
14.37 GuiEventSurface Class Reference	100
14.37.1 Detailed Description	101
14.37.2 Member Function Documentation	101

14.37.2.1 AddWindow	101
14.37.2.2 CopyEvent	101
14.37.2.3 Init	102
14.37.2.4 InvalidateRegion	102
14.37.2.5 ProcessEvent	102
14.37.2.6 RemoveWindow	102
14.37.2.7 SendEvent	102
14.38 GuiWindow Class Reference	103
14.38.1 Detailed Description	104
14.38.2 Member Function Documentation	105
14.38.2.1 AddControl	105
14.38.2.2 CycleFocus	105
14.38.2.3 GetDriver	105
14.38.2.4 GetHeight	105
14.38.2.5 GetLeft	105
14.38.2.6 GetMaxZOrder	106
14.38.2.7 GetTop	106
14.38.2.8 GetWidth	106
14.38.2.9 Init	106
14.38.2.10 InvalidateRegion	106
14.38.2.11 IsInFocus	106
14.38.2.12 ProcessEvent	107
14.38.2.13 Redraw	107
14.38.2.14 RemoveControl	107
14.38.2.15 SetDriver	107
14.38.2.16 SetFocus	107
14.38.2.17 SetHeight	108
14.38.2.18 SetLeft	109
14.38.2.19 SetTop	109
14.38.2.20 SetWidth	109
14.38.3 Member Data Documentation	109
14.38.3.1 m_pclDriver	109
14.39 HeapConfig Class Reference	109
14.39.1 Detailed Description	110
14.40 JoystickEvent_t Struct Reference	110
14.40.1 Detailed Description	110
14.41 Kernel Class Reference	110
14.41.1 Detailed Description	111
14.41.2 Member Function Documentation	111
14.41.2.1 Init	111

14.41.2.2 IsPanic	111
14.41.2.3 IsStarted	111
14.41.2.4 Panic	112
14.41.2.5 SetPanic	112
14.41.2.6 Start	112
14.42Kernel_Aware Class Reference	112
14.42.1 Detailed Description	113
14.42.2 Member Function Documentation	113
14.42.2.1 Exit_Simulator	113
14.42.2.2 Print	113
14.42.2.3 Profile_Init	113
14.42.2.4 Profile_Report	114
14.42.2.5 Profile_Start	114
14.42.2.6 Profile_Stop	114
14.42.2.7 Trace	114
14.42.2.8 Trace	114
14.42.2.9 Trace	115
14.42.2.10Trace_i	115
14.43KernelAwareData_t Union Reference	115
14.43.1 Detailed Description	116
14.44KernelSWI Class Reference	116
14.44.1 Detailed Description	116
14.44.2 Member Function Documentation	116
14.44.2.1 DI	116
14.44.2.2 RI	116
14.45KernelTimer Class Reference	117
14.45.1 Detailed Description	117
14.45.2 Member Function Documentation	118
14.45.2.1 GetOvertime	118
14.45.2.2 Read	118
14.45.2.3 RI	118
14.45.2.4 SetExpiry	118
14.45.2.5 SubtractExpiry	118
14.45.2.6 TimeToExpiry	119
14.46KeyEvent_t Struct Reference	119
14.46.1 Detailed Description	119
14.47LabelControl Class Reference	119
14.47.1 Detailed Description	120
14.47.2 Member Function Documentation	120
14.47.2.1 Activate	120

14.47.2.2 Draw	121
14.47.2.3 Init	121
14.47.2.4 ProcessEvent	121
14.48LinkedList Class Reference	121
14.48.1 Detailed Description	122
14.48.2 Member Function Documentation	122
14.48.2.1 Add	122
14.48.2.2 GetHead	122
14.48.2.3 GetTail	122
14.48.2.4 Remove	122
14.49LinkedListNode Class Reference	123
14.49.1 Detailed Description	124
14.49.2 Member Function Documentation	124
14.49.2.1 GetNext	124
14.49.2.2 GetPrev	124
14.50MemUtil Class Reference	124
14.50.1 Detailed Description	125
14.50.2 Member Function Documentation	125
14.50.2.1 Checksum16	125
14.50.2.2 Checksum8	126
14.50.2.3 CompareMemory	126
14.50.2.4 CompareStrings	126
14.50.2.5 CopyMemory	126
14.50.2.6 CopyString	127
14.50.2.7 DecimalToHex	127
14.50.2.8 DecimalToString	127
14.50.2.9 SetMemory	127
14.50.2.10StringLength	127
14.50.2.11StringSearch	128
14.50.2.12Tokenize	128
14.51Message Class Reference	128
14.51.1 Detailed Description	129
14.51.2 Member Function Documentation	129
14.51.2.1 GetCode	129
14.51.2.2 GetData	129
14.51.2.3 SetCode	130
14.51.2.4 SetData	130
14.52MessageQueue Class Reference	130
14.52.1 Detailed Description	131
14.52.2 Member Function Documentation	131

14.52.2.1	GetCount	131
14.52.2.2	Receive	131
14.52.2.3	Receive	131
14.52.2.4	Send	131
14.53	MouseEvent_t Struct Reference	132
14.53.1	Detailed Description	132
14.54	Mutex Class Reference	132
14.54.1	Detailed Description	133
14.54.2	Member Function Documentation	133
14.54.2.1	Claim	133
14.54.2.2	Claim	133
14.54.2.3	Release	134
14.54.2.4	WakeMe	134
14.55	NLFS Class Reference	134
14.55.1	Detailed Description	136
14.55.2	Member Function Documentation	136
14.55.2.1	Append_Block_To_Node	136
14.55.2.2	Cleanup_Node_Links	137
14.55.2.3	Create_Dir	137
14.55.2.4	Create_File	137
14.55.2.5	Create_File_i	137
14.55.2.6	Delete_File	138
14.55.2.7	Delete_Folder	138
14.55.2.8	File_Names_Match	138
14.55.2.9	Find_File	138
14.55.2.10	Find_Last_Slash	139
14.55.2.11	Find_Parent_Dir	139
14.55.2.12	Format	139
14.55.2.13	GetBlockSize	140
14.55.2.14	GetFirstChild	140
14.55.2.15	GetNextPeer	140
14.55.2.16	GetNumBlocks	140
14.55.2.17	GetNumBlocksFree	141
14.55.2.18	GetNumFiles	141
14.55.2.19	GetNumFilesFree	141
14.55.2.20	GetStat	141
14.55.2.21	Mount	141
14.55.2.22	Pop_Free_Block	142
14.55.2.23	Pop_Free_Node	142
14.55.2.24	Print_Dir_Details	142

14.55.2.25	Print_File_Details	142
14.55.2.26	Print_Free_Details	142
14.55.2.27	Print_Node_Details	142
14.55.2.28	Push_Free_Block	143
14.55.2.29	Push_Free_Node	143
14.55.2.30	Read_Block	143
14.55.2.31	Read_Block_Header	143
14.55.2.32	Read_Node	143
14.55.2.33	RootSync	144
14.55.2.34	Set_Node_Name	144
14.55.2.35	Write_Block	144
14.55.2.36	Write_Block_Header	144
14.55.2.37	Write_Node	144
14.56	NLFS_Block_t Struct Reference	145
14.56.1	Detailed Description	145
14.57	NLFS_File Class Reference	145
14.57.1	Detailed Description	146
14.57.2	Member Function Documentation	146
14.57.2.1	Close	146
14.57.2.2	Open	146
14.57.2.3	Read	146
14.57.2.4	Seek	147
14.57.2.5	Write	147
14.58	NLFS_File_Node_t Struct Reference	147
14.58.1	Detailed Description	148
14.59	NLFS_File_Stat_t Struct Reference	148
14.59.1	Detailed Description	149
14.60	NLFS_Host_t Union Reference	149
14.60.1	Detailed Description	149
14.61	NLFS_Node_t Struct Reference	149
14.61.1	Detailed Description	149
14.62	NLFS_RAM Class Reference	150
14.62.1	Detailed Description	150
14.62.2	Member Function Documentation	151
14.62.2.1	Read_Block	151
14.62.2.2	Read_Block_Header	152
14.62.2.3	Read_Node	152
14.62.2.4	Write_Block	152
14.62.2.5	Write_Block_Header	153
14.62.2.6	Write_Node	154

14.63NLFS_Root_Node_t Struct Reference	154
14.63.1 Detailed Description	155
14.64NotificationControl Class Reference	155
14.64.1 Detailed Description	155
14.64.2 Member Function Documentation	156
14.64.2.1 Activate	156
14.64.2.2 Draw	157
14.64.2.3 Init	157
14.64.2.4 ProcessEvent	157
14.65Option_t Struct Reference	157
14.65.1 Detailed Description	158
14.66PanelControl Class Reference	158
14.66.1 Detailed Description	159
14.66.2 Member Function Documentation	159
14.66.2.1 Activate	159
14.66.2.2 Draw	159
14.66.2.3 Init	159
14.66.2.4 ProcessEvent	159
14.67Profiler Class Reference	160
14.67.1 Detailed Description	160
14.67.2 Member Function Documentation	160
14.67.2.1 Init	160
14.68ProfileTimer Class Reference	161
14.68.1 Detailed Description	161
14.68.2 Member Function Documentation	162
14.68.2.1 ComputeCurrentTicks	162
14.68.2.2 GetAverage	163
14.68.2.3 GetCurrent	163
14.68.2.4 Init	163
14.68.2.5 Start	163
14.69ProgressControl Class Reference	163
14.69.1 Detailed Description	164
14.69.2 Member Function Documentation	164
14.69.2.1 Activate	164
14.69.2.2 Draw	165
14.69.2.3 Init	165
14.69.2.4 ProcessEvent	165
14.70PseudoRandom Class Reference	165
14.70.1 Detailed Description	166
14.70.2 Constructor & Destructor Documentation	166

14.70.2.1 PseudoRandom	166
14.70.3 Member Function Documentation	166
14.70.3.1 GetRandom	166
14.70.3.2 Seed	166
14.70.3.3 Shift	166
14.71 Quantum Class Reference	167
14.71.1 Detailed Description	167
14.71.2 Member Function Documentation	168
14.71.2.1 AddThread	168
14.71.2.2 ClearInTimer	168
14.71.2.3 RemoveThread	168
14.71.2.4 SetInTimer	168
14.71.2.5 SetTimer	168
14.71.2.6 UpdateTimer	168
14.72 Scheduler Class Reference	169
14.72.1 Detailed Description	169
14.72.2 Member Function Documentation	170
14.72.2.1 Add	170
14.72.2.2 GetCurrentThread	171
14.72.2.3 GetNextThread	171
14.72.2.4 GetStopList	171
14.72.2.5 GetThreadList	171
14.72.2.6 IsEnabled	171
14.72.2.7 Remove	172
14.72.2.8 Schedule	173
14.72.2.9 SetScheduler	173
14.73 Screen Class Reference	173
14.73.1 Detailed Description	174
14.73.2 Member Function Documentation	174
14.73.2.1 Activate	174
14.73.2.2 Deactivate	174
14.74 ScreenList Class Reference	174
14.74.1 Detailed Description	175
14.75 ScreenManager Class Reference	175
14.75.1 Detailed Description	175
14.76 Semaphore Class Reference	176
14.76.1 Detailed Description	176
14.76.2 Member Function Documentation	177
14.76.2.1 GetCount	177
14.76.2.2 Init	177

14.76.2.3 Pend	177
14.76.2.4 Pend	177
14.76.2.5 Post	177
14.76.2.6 WakeMe	178
14.77ShellCommand_t Struct Reference	178
14.77.1 Detailed Description	178
14.78ShellSupport Class Reference	178
14.78.1 Detailed Description	179
14.78.2 Member Function Documentation	179
14.78.2.1 CheckForOption	179
14.78.2.2 RunCommand	179
14.78.2.3 TokensToCommandLine	179
14.78.2.4 UnescapeToken	180
14.79SlickButtonControl Class Reference	180
14.79.1 Detailed Description	181
14.79.2 Member Function Documentation	181
14.79.2.1 Activate	181
14.79.2.2 Draw	181
14.79.2.3 Init	181
14.79.2.4 ProcessEvent	181
14.80SlickGroupBoxControl Class Reference	182
14.80.1 Detailed Description	182
14.80.2 Member Function Documentation	183
14.80.2.1 Activate	183
14.80.2.2 Draw	184
14.80.2.3 Init	184
14.80.2.4 ProcessEvent	184
14.81SlickProgressControl Class Reference	184
14.81.1 Detailed Description	185
14.81.2 Member Function Documentation	185
14.81.2.1 Activate	185
14.81.2.2 Draw	185
14.81.2.3 Init	185
14.81.2.4 ProcessEvent	186
14.82Slip Class Reference	186
14.82.1 Detailed Description	187
14.82.2 Member Function Documentation	187
14.82.2.1 DecodeByte	187
14.82.2.2 EncodeByte	187
14.82.2.3 GetDriver	187

14.82.2.4 ReadData	187
14.82.2.5 SetDriver	188
14.82.2.6 WriteData	188
14.82.2.7 WriteVector	188
14.83SlipDataVector Struct Reference	188
14.83.1 Detailed Description	189
14.84SlipMux Class Reference	189
14.84.1 Detailed Description	190
14.84.2 Member Function Documentation	190
14.84.2.1 GetDriver	190
14.84.2.2 GetQueue	190
14.84.2.3 GetSlip	190
14.84.2.4 Init	190
14.84.2.5 InstallHandler	190
14.84.2.6 MessageReceive	191
14.84.2.7 SetQueue	191
14.85SlipTerm Class Reference	191
14.85.1 Detailed Description	192
14.85.2 Member Function Documentation	192
14.85.2.1 Init	192
14.85.2.2 PrintLn	192
14.85.2.3 PrintLn	192
14.85.2.4 SetVerbosity	192
14.85.2.5 StrLen	192
14.85.3 Member Data Documentation	193
14.85.3.1 m_ucVerbosity	193
14.86StubControl Class Reference	193
14.86.1 Detailed Description	193
14.86.2 Member Function Documentation	194
14.86.2.1 Activate	194
14.86.2.2 Draw	194
14.86.2.3 Init	194
14.86.2.4 ProcessEvent	194
14.87SystemHeap Class Reference	194
14.87.1 Detailed Description	195
14.87.2 Member Function Documentation	195
14.87.2.1 Alloc	195
14.87.2.2 Free	195
14.88TextFX_t Struct Reference	196
14.88.1 Detailed Description	196

14.89Thread Class Reference	196
14.89.1 Detailed Description	198
14.89.2 Member Function Documentation	199
14.89.2.1 ContextSwitchSWI	199
14.89.2.2 Exit	199
14.89.2.3 GetCurPriority	199
14.89.2.4 GetCurrent	199
14.89.2.5 GetEventFlagMask	199
14.89.2.6 GetEventFlagMode	199
14.89.2.7 GetID	200
14.89.2.8 GetOwner	200
14.89.2.9 GetPriority	200
14.89.2.10GetQuantum	200
14.89.2.11GetStackSlack	200
14.89.2.12InheritPriority	201
14.89.2.13Init	201
14.89.2.14SetCurrent	201
14.89.2.15SetEventFlagMask	201
14.89.2.16SetEventFlagMode	201
14.89.2.17SetID	202
14.89.2.18SetOwner	203
14.89.2.19SetPriority	203
14.89.2.20SetPriorityBase	203
14.89.2.21SetQuantum	203
14.89.2.22Sleep	203
14.89.2.23Stop	204
14.89.2.24USleep	204
14.89.2.25Yield	204
14.90ThreadList Class Reference	204
14.90.1 Detailed Description	205
14.90.2 Member Function Documentation	205
14.90.2.1 Add	205
14.90.2.2 Add	205
14.90.2.3 HighestWaiter	205
14.90.2.4 Remove	206
14.90.2.5 SetFlagPointer	206
14.90.2.6 SetPriority	206
14.91ThreadPort Class Reference	206
14.91.1 Detailed Description	207
14.91.2 Member Function Documentation	207

14.91.2.1 InitStack	207
14.92Tile_8x8 Class Reference	207
14.92.1 Detailed Description	208
14.92.2 Member Function Documentation	208
14.92.2.1 LoadTile	208
14.92.2.2 Render	208
14.93TileDef_t Struct Reference	208
14.93.1 Detailed Description	209
14.94Timer Class Reference	209
14.94.1 Detailed Description	210
14.94.2 Member Function Documentation	210
14.94.2.1 SetCallback	210
14.94.2.2 SetData	210
14.94.2.3 SetFlags	211
14.94.2.4 SetIntervalMSeconds	211
14.94.2.5 SetIntervalSeconds	211
14.94.2.6 SetIntervalTicks	211
14.94.2.7 SetIntervalUSeconds	211
14.94.2.8 SetOwner	212
14.94.2.9 SetTolerance	213
14.94.2.10Start	213
14.94.2.11Start	213
14.94.2.12Stop	213
14.95TimerEvent_t Struct Reference	214
14.95.1 Detailed Description	214
14.96TimerList Class Reference	214
14.96.1 Detailed Description	215
14.96.2 Member Function Documentation	215
14.96.2.1 Add	215
14.96.2.2 Init	215
14.96.2.3 Process	215
14.96.2.4 Remove	215
14.97TimerScheduler Class Reference	215
14.97.1 Detailed Description	216
14.97.2 Member Function Documentation	216
14.97.2.1 Add	216
14.97.2.2 Init	216
14.97.2.3 Process	216
14.97.2.4 Remove	216
14.98Token_t Struct Reference	217

14.98.1 Detailed Description	217
14.99TouchEvent_t Struct Reference	217
14.99.1 Detailed Description	217
14.100UnitTest Class Reference	218
14.100.1Detailed Description	219
14.100.2Member Function Documentation	219
14.100.2.1Complete	219
14.100.2.2GetFailed	219
14.100.2.3GetName	220
14.100.2.4GetPassed	220
14.100.2.5GetResult	220
14.100.2.6GetTotal	220
14.100.2.7SetName	220
15 File Documentation	221
15.1 /home/moslevin/Project/R1/stage/src/atomic.cpp File Reference	221
15.1.1 Detailed Description	221
15.2 atomic.cpp	221
15.3 /home/moslevin/Project/R1/stage/src/atomic.h File Reference	223
15.3.1 Detailed Description	223
15.4 atomic.h	223
15.5 /home/moslevin/Project/R1/stage/src/blocking.cpp File Reference	224
15.5.1 Detailed Description	224
15.6 blocking.cpp	224
15.7 /home/moslevin/Project/R1/stage/src/blocking.h File Reference	225
15.7.1 Detailed Description	225
15.8 blocking.h	225
15.9 /home/moslevin/Project/R1/stage/src/control_button.cpp File Reference	226
15.9.1 Detailed Description	226
15.10control_button.cpp	226
15.11/home/moslevin/Project/R1/stage/src/control_button.h File Reference	229
15.11.1 Detailed Description	229
15.12control_button.h	229
15.13/home/moslevin/Project/R1/stage/src/control_checkbox.cpp File Reference	230
15.13.1 Detailed Description	231
15.13.2 Variable Documentation	231
15.13.2.1 aucBox	231
15.13.2.2 aucCheck	231
15.14control_checkbox.cpp	231
15.15/home/moslevin/Project/R1/stage/src/control_checkbox.h File Reference	234

15.15.1 Detailed Description	234
15.16control_checkbox.h	234
15.17/home/moslevin/Project/R1/stage/src/control_gamepanel.cpp File Reference	235
15.17.1 Detailed Description	235
15.18control_gamepanel.cpp	235
15.19/home/moslevin/Project/R1/stage/src/control_gamepanel.h File Reference	236
15.19.1 Detailed Description	236
15.20control_gamepanel.h	236
15.21/home/moslevin/Project/R1/stage/src/control_groupbox.cpp File Reference	236
15.21.1 Detailed Description	237
15.22control_groupbox.cpp	237
15.23/home/moslevin/Project/R1/stage/src/control_groupbox.h File Reference	238
15.23.1 Detailed Description	238
15.24control_groupbox.h	238
15.25/home/moslevin/Project/R1/stage/src/control_label.h File Reference	239
15.25.1 Detailed Description	239
15.26control_label.h	239
15.27/home/moslevin/Project/R1/stage/src/control_notification.cpp File Reference	240
15.27.1 Detailed Description	240
15.28control_notification.cpp	240
15.29/home/moslevin/Project/R1/stage/src/control_notification.h File Reference	242
15.29.1 Detailed Description	242
15.30control_notification.h	242
15.31/home/moslevin/Project/R1/stage/src/control_panel.cpp File Reference	243
15.31.1 Detailed Description	243
15.32control_panel.cpp	243
15.33/home/moslevin/Project/R1/stage/src/control_panel.h File Reference	244
15.33.1 Detailed Description	244
15.34control_panel.h	244
15.35/home/moslevin/Project/R1/stage/src/control_progress.cpp File Reference	245
15.35.1 Detailed Description	245
15.36control_progress.cpp	245
15.37/home/moslevin/Project/R1/stage/src/control_progress.h File Reference	246
15.37.1 Detailed Description	247
15.38control_progress.h	247
15.39/home/moslevin/Project/R1/stage/src/control_slickbutton.h File Reference	247
15.39.1 Detailed Description	248
15.40control_slickbutton.h	248
15.41/home/moslevin/Project/R1/stage/src/control_slickprogress.cpp File Reference	248
15.41.1 Detailed Description	249

15.42control_slickprogress.cpp	249
15.43/home/moslevin/Project/R1/stage/src/control_slickprogress.h File Reference	250
15.43.1 Detailed Description	250
15.44control_slickprogress.h	250
15.45/home/moslevin/Project/R1/stage/src/dcpu.cpp File Reference	251
15.45.1 Detailed Description	251
15.46dcpu.cpp	252
15.47/home/moslevin/Project/R1/stage/src/dcpu.h File Reference	262
15.47.1 Detailed Description	264
15.47.2 Macro Definition Documentation	264
15.47.2.1 DCPU_NORMAL_OPCODE_MASK	264
15.47.3 Enumeration Type Documentation	264
15.47.3.1 DCPU_OpBasic	264
15.47.3.2 DCPU_OpExtended	265
15.48dcpu.h	266
15.49/home/moslevin/Project/R1/stage/src/debug_tokens.h File Reference	270
15.49.1 Detailed Description	272
15.50debug_tokens.h	272
15.51/home/moslevin/Project/R1/stage/src/draw.h File Reference	273
15.51.1 Detailed Description	274
15.52draw.h	274
15.53/home/moslevin/Project/R1/stage/src/driver.cpp File Reference	276
15.53.1 Detailed Description	277
15.54driver.cpp	277
15.55/home/moslevin/Project/R1/stage/src/driver.h File Reference	278
15.55.1 Detailed Description	279
15.55.2 Intro	279
15.55.3 Driver Design	279
15.55.4 Driver API	279
15.56driver.h	280
15.57/home/moslevin/Project/R1/stage/src/eventflag.cpp File Reference	281
15.57.1 Detailed Description	281
15.58eventflag.cpp	281
15.59/home/moslevin/Project/R1/stage/src/eventflag.h File Reference	285
15.59.1 Detailed Description	285
15.60eventflag.h	285
15.61/home/moslevin/Project/R1/stage/src/fixed_heap.cpp File Reference	286
15.61.1 Detailed Description	286
15.62fixed_heap.cpp	286
15.63/home/moslevin/Project/R1/stage/src/fixed_heap.h File Reference	288

15.63.1 Detailed Description	288
15.64fixed_heap.h	288
15.65/home/moslevin/Project/R1/stage/src/font.h File Reference	289
15.65.1 Detailed Description	290
15.66font.h	290
15.67/home/moslevin/Project/R1/stage/src/graphics.cpp File Reference	290
15.67.1 Detailed Description	291
15.68graphics.cpp	291
15.69/home/moslevin/Project/R1/stage/src/graphics.h File Reference	303
15.69.1 Detailed Description	304
15.70graphics.h	304
15.71/home/moslevin/Project/R1/stage/src/gui.cpp File Reference	305
15.71.1 Detailed Description	305
15.72gui.cpp	305
15.73/home/moslevin/Project/R1/stage/src/gui.h File Reference	313
15.73.1 Detailed Description	315
15.73.2 Enumeration Type Documentation	315
15.73.2.1 GuiEventType_t	315
15.73.2.2 GuiReturn_t	315
15.74gui.h	315
15.75/home/moslevin/Project/R1/stage/src/kernel.cpp File Reference	320
15.75.1 Detailed Description	321
15.76kernel.cpp	321
15.77/home/moslevin/Project/R1/stage/src/kernel.h File Reference	322
15.77.1 Detailed Description	322
15.78kernel.h	322
15.79/home/moslevin/Project/R1/stage/src/kernel_debug.h File Reference	323
15.79.1 Detailed Description	323
15.80kernel_debug.h	323
15.81/home/moslevin/Project/R1/stage/src/kernelswi.cpp File Reference	325
15.81.1 Detailed Description	325
15.82kernelswi.cpp	325
15.83/home/moslevin/Project/R1/stage/src/kernelswi.h File Reference	326
15.83.1 Detailed Description	326
15.84kernelswi.h	327
15.85/home/moslevin/Project/R1/stage/src/kerneltimer.cpp File Reference	327
15.85.1 Detailed Description	327
15.86kerneltimer.cpp	327
15.87/home/moslevin/Project/R1/stage/src/kerneltimer.h File Reference	330
15.87.1 Detailed Description	330

15.88	kerneltimer.h	330
15.89	/home/moslevin/Project/R1/stage/src/kerneltypes.h File Reference	331
15.89.1	Detailed Description	331
15.90	kerneltypes.h	331
15.91	/home/moslevin/Project/R1/stage/src/keycodes.h File Reference	332
15.91.1	Detailed Description	333
15.92	keycodes.h	333
15.93	/home/moslevin/Project/R1/stage/src/kprofile.cpp File Reference	335
15.93.1	Detailed Description	335
15.94	kprofile.cpp	335
15.95	/home/moslevin/Project/R1/stage/src/kprofile.h File Reference	336
15.95.1	Detailed Description	337
15.96	kprofile.h	337
15.97	/home/moslevin/Project/R1/stage/src/ksemaphore.cpp File Reference	337
15.97.1	Detailed Description	338
15.98	ksemaphore.cpp	338
15.99	/home/moslevin/Project/R1/stage/src/ksemaphore.h File Reference	341
15.99.1	Detailed Description	341
15.100	ksemaphore.h	341
15.101	/home/moslevin/Project/R1/stage/src/ll.cpp File Reference	342
15.101.1	Detailed Description	342
15.102	ll.cpp	342
15.103	/home/moslevin/Project/R1/stage/src/ll.h File Reference	344
15.103.1	Detailed Description	345
15.104	ll.h	345
15.105	/home/moslevin/Project/R1/stage/src/manual.h File Reference	346
15.105.1	Detailed Description	346
15.106	manual.h	346
15.107	/home/moslevin/Project/R1/stage/src/mark3cfg.h File Reference	347
15.107.1	Detailed Description	348
15.107.2	Macro Definition Documentation	348
15.107.2.1	GLOBAL_MESSAGE_POOL_SIZE	348
15.107.2.2	KERNEL_AWARE_SIMULATION	348
15.107.2.3	KERNEL_TIMERS_TICKLESS	348
15.107.2.4	KERNEL_USE_ATOMIC	348
15.107.2.5	KERNEL_USE_DYNAMIC_THREADS	349
15.107.2.6	KERNEL_USE_EVENTFLAG	349
15.107.2.7	KERNEL_USE_MESSAGE	349
15.107.2.8	KERNEL_USE_PROFILER	349
15.107.2.9	KERNEL_USE_QUANTUM	349

15.107.2.10	KERNEL_USE_SEMAPHORE	349
15.107.2.11	KERNEL_USE_THREADNAME	350
15.107.2.12	KERNEL_USE_TIMEOUTS	350
15.107.2.13	KERNEL_USE_TIMERS	350
15.107.2.14	SAFE_UNLINK	350
15.107.2.15	THREAD_QUANTUM_DEFAULT	350
15.108	mark3cfg.h	350
15.109	home/moslevin/Project/R1/stage/src/memutil.cpp File Reference	351
15.109.1	Detailed Description	351
15.110	memutil.cpp	352
15.111	home/moslevin/Project/R1/stage/src/memutil.h File Reference	357
15.111.1	Detailed Description	357
15.112	memutil.h	357
15.113	home/moslevin/Project/R1/stage/src/message.cpp File Reference	358
15.113.1	Detailed Description	359
15.114	message.cpp	359
15.115	home/moslevin/Project/R1/stage/src/message.h File Reference	361
15.115.1	Detailed Description	361
15.115.2	Using Messages, Queues, and the Global Message Pool	361
15.116	message.h	362
15.117	home/moslevin/Project/R1/stage/src/mutex.cpp File Reference	363
15.117.1	Detailed Description	363
15.118	mutex.cpp	363
15.119	home/moslevin/Project/R1/stage/src/mutex.h File Reference	367
15.119.1	Detailed Description	367
15.119.2	Initializing	367
15.119.3	Resource protection example	367
15.120	mutex.h	367
15.121	home/moslevin/Project/R1/stage/src/nlfs.cpp File Reference	368
15.121.1	Detailed Description	368
15.122	nlfs.cpp	369
15.123	home/moslevin/Project/R1/stage/src/nlfs.h File Reference	380
15.123.1	Detailed Description	381
15.123.2	Enumeration Type Documentation	382
15.123.2.1	NLFS_Type_t	382
15.124	nlfs.h	382
15.125	home/moslevin/Project/R1/stage/src/nlfs_config.h File Reference	385
15.125.1	Detailed Description	385
15.126	nlfs_config.h	386
15.127	home/moslevin/Project/R1/stage/src/nlfs_file.cpp File Reference	386

15.127. Detailed Description	386
15.128. nlfs_file.cpp	386
15.129. home/moslevin/Project/R1/stage/src/nlfs_file.h File Reference	390
15.129. Detailed Description	390
15.129. Enumeration Type Documentation	390
15.129.2. NLFS_File_Mode	390
15.130. nlfs_file.h	391
15.131. home/moslevin/Project/R1/stage/src/nlfs_ram.cpp File Reference	391
15.131. Detailed Description	391
15.132. nlfs_ram.cpp	392
15.133. home/moslevin/Project/R1/stage/src/nlfs_ram.h File Reference	393
15.133. Detailed Description	393
15.134. nlfs_ram.h	393
15.135. home/moslevin/Project/R1/stage/src/panic_codes.h File Reference	394
15.135. Detailed Description	394
15.136. panic_codes.h	394
15.137. home/moslevin/Project/R1/stage/src/profile.cpp File Reference	394
15.137. Detailed Description	395
15.138. profile.cpp	395
15.139. home/moslevin/Project/R1/stage/src/profile.h File Reference	396
15.139. Detailed Description	397
15.140. profile.h	397
15.141. home/moslevin/Project/R1/stage/src/quantum.cpp File Reference	398
15.141. Detailed Description	398
15.142. quantum.cpp	398
15.143. home/moslevin/Project/R1/stage/src/quantum.h File Reference	400
15.143. Detailed Description	400
15.144. quantum.h	400
15.145. home/moslevin/Project/R1/stage/src/rand_lfsr.h File Reference	401
15.145. Detailed Description	401
15.146. rand_lfsr.h	401
15.147. home/moslevin/Project/R1/stage/src/scheduler.cpp File Reference	402
15.147. Detailed Description	402
15.148. scheduler.cpp	402
15.149. home/moslevin/Project/R1/stage/src/scheduler.h File Reference	403
15.149. Detailed Description	404
15.150. scheduler.h	404
15.151. home/moslevin/Project/R1/stage/src/screen.cpp File Reference	405
15.151. Detailed Description	405
15.152. screen.cpp	405

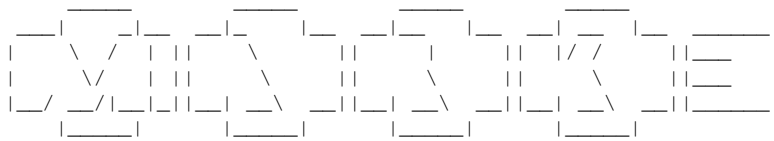
15.153	home/moslevin/Project/R1/stage/src/screen.h File Reference	406
15.153.1	Detailed Description	406
15.154	screen.h	407
15.155	home/moslevin/Project/R1/stage/src/shell_support.cpp File Reference	408
15.155.1	Detailed Description	408
15.156	shell_support.cpp	408
15.157	home/moslevin/Project/R1/stage/src/shell_support.h File Reference	410
15.157.1	Detailed Description	411
15.157.2	Typedef Documentation	411
15.157.2.1	fp_internal_command	411
15.158	shell_support.h	411
15.159	home/moslevin/Project/R1/stage/src/slip.cpp File Reference	412
15.159.1	Detailed Description	412
15.160	slip.cpp	412
15.161	home/moslevin/Project/R1/stage/src/slip.h File Reference	416
15.161.1	Detailed Description	416
15.161.2	Enumeration Type Documentation	416
15.161.2.1	SlipChannel	416
15.162	slip.h	417
15.163	home/moslevin/Project/R1/stage/src/slip_mux.cpp File Reference	417
15.163.1	Detailed Description	418
15.163.2	Function Documentation	418
15.163.2.1	SlipMux_CallBack	418
15.164	slip_mux.cpp	418
15.165	home/moslevin/Project/R1/stage/src/slip_mux.h File Reference	419
15.165.1	Detailed Description	420
15.166	slip_mux.h	420
15.167	home/moslevin/Project/R1/stage/src/slipterm.cpp File Reference	421
15.167.1	Detailed Description	421
15.168	slipterm.cpp	421
15.169	home/moslevin/Project/R1/stage/src/slipterm.h File Reference	422
15.169.1	Detailed Description	422
15.170	slipterm.h	422
15.171	home/moslevin/Project/R1/stage/src/system_heap.cpp File Reference	423
15.171.1	Detailed Description	423
15.172	system_heap.cpp	423
15.173	home/moslevin/Project/R1/stage/src/system_heap.h File Reference	425
15.173.1	Detailed Description	426
15.173.2	Macro Definition Documentation	426
15.173.2.1	HEAP_RAW_SIZE	426

15.173.2.2HEAP_RAW_SIZE_1	426
15.174system_heap.h	426
15.175home/moslevin/Project/R1/stage/src/system_heap_config.h File Reference	429
15.175.1Detailed Description	429
15.175.2Macro Definition Documentation	430
15.175.2.1HEAP_BLOCK_SIZE_1	430
15.176system_heap_config.h	430
15.177home/moslevin/Project/R1/stage/src/thread.cpp File Reference	430
15.177.1Detailed Description	431
15.178hread.cpp	431
15.179home/moslevin/Project/R1/stage/src/thread.h File Reference	435
15.179.1Detailed Description	436
15.180hread.h	436
15.181home/moslevin/Project/R1/stage/src/threadlist.cpp File Reference	438
15.181.1Detailed Description	438
15.182hreadlist.cpp	439
15.183home/moslevin/Project/R1/stage/src/threadlist.h File Reference	440
15.183.1Detailed Description	440
15.184hreadlist.h	440
15.185home/moslevin/Project/R1/stage/src/threadport.cpp File Reference	441
15.185.1Detailed Description	441
15.186hreadport.cpp	441
15.187home/moslevin/Project/R1/stage/src/threadport.h File Reference	443
15.187.1Detailed Description	444
15.187.2Macro Definition Documentation	444
15.187.2.1CS_ENTER	444
15.187.2.2CS_EXIT	444
15.188hreadport.h	444
15.189home/moslevin/Project/R1/stage/src/timerlist.cpp File Reference	446
15.189.1Detailed Description	447
15.190merlist.cpp	447
15.191home/moslevin/Project/R1/stage/src/timerlist.h File Reference	451
15.191.1Detailed Description	451
15.191.2Macro Definition Documentation	452
15.191.2.1TIMERLIST_FLAG_EXPIRED	452
15.192merlist.h	452
15.193home/moslevin/Project/R1/stage/src/tracebuffer.cpp File Reference	454
15.193.1Detailed Description	454
15.194racebuffer.cpp	454
15.195home/moslevin/Project/R1/stage/src/tracebuffer.h File Reference	455

15.195. Detailed Description	455
15.196.acebuffer.h	455
15.197.home/moslevin/Project/R1/stage/src/unit_test.cpp File Reference	456
15.197. Detailed Description	456
15.198.nit_test.cpp	456
15.199.home/moslevin/Project/R1/stage/src/unit_test.h File Reference	457
15.199. Detailed Description	457
15.200.nit_test.h	457
15.201.home/moslevin/Project/R1/stage/src/writebuf16.cpp File Reference	459
15.201. Detailed Description	459
15.202.writebuf16.cpp	459
15.203.home/moslevin/Project/R1/stage/src/writebuf16.h File Reference	461
15.203. Detailed Description	461
15.204.writebuf16.h	461
Index	462

Chapter 1

The Mark3 Realtime Kernel



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

Copyright (c) 2012-2015 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, documentation, and user-projects 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
- Support for kernel-aware simulators, specifically, Funkenstein Software's own fIAVR AVR simulator

Chapter 2

Preface

2.1 Who should read this

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

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

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

2.2 Why Mark3?

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

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

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

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

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

Regardless, I learned a lot about software development.

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

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

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

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

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

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

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

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

Chapter 3

Can you Afford an RTOS?

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

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

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

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

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

3.1 Intro

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

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

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

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

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

RTOS overhead can be broken into three distinct areas:

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

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

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

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

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

31600 Bytes Code Space 2014 Bytes RAM

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

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

29904 Bytes Code Space 1648 Bytes RAM

3.2 Memory overhead:

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

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

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

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

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

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

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

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

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

3.3 Code Space Overhead:

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

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

3466 Bytes

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

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

1850 Bytes

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

3.4 Runtime Overhead

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

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

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

Preemptive mode:

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

Cooperative mode:

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

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

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

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

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

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

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

Chapter 4

Superloops

4.1 Intro to Superloops

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

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

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

4.2 The simplest loop

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

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

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

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

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

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

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

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

4.3 Interrupt-Driven Super-loop

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

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

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

The following concepts are shown in the example below:

```
volatile K_BOOL something_to_do = false;

__interrupt__ My_Interrupt_Source(void)
{
    something_to_do = true;
}

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

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

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

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

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

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

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

4.4 Cooperative multi-tasking

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

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

volatile K_UCHAR event_flags = 0;

// Interrupt sources used to trigger event execution

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

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

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

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

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

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

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

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

4.5 Hybrid cooperative/preemptive multi-tasking

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

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

volatile K_UCHAR hp_tasks = 0;

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

volatile K_UCHAR lp_tasks = 0;

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

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

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

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

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

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

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

4.6 Problems with superloops

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

Hidden Costs

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

Tightly-coupled code

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

No blocking calls

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

Difficult to guarantee responsiveness

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

Limited preemption capability

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

Chapter 5

Mark3 Overview

5.1 Intro

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

5.2 Features

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

The key features of this RTOS are:

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

- Robust Interprocess Communications
 - Threadsafe global [Message](#) pool and configurable message queues
- Support for kernel-aware simulation
 - Provides advanced test and verification functionality, allowing for easy integration into continuous-integration systems
 - Provide accurate engineering data on key metrics like stack usage and realtime performance, with easy-to-use APIs and little overhead

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();           // MUST be before other kernel ops  
  
    //2) Initialize all of the threads we've defined
```

```

AppThread.Init( aucAppStack,      // Pointer to the stack
                STACK_SIZE_APP,   // Size of the stack
                1,                // Thread priority
                (void*)AppEntry,   // Entry function
                NULL );           // Entry function argument

IdleThread.Init( aucIdleStack,    // Pointer to the stack
                 STACK_SIZE_IDLE, // Size of the stack
                 0,              // Thread priority
                 (void*)IdleEntry, // Entry function
                 NULL );         // Entry function argument

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

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

```

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

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

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

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

6.2 Threads

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

6.2.1 Thread Setup

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

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

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

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

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

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

An example thread initialization is shown below:

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

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

...
{
    clMyThread.Init(aucStack,    // Pointer to the stack to use by this thread
                   192,        // Size of the stack in bytes
                   1,          // Thread priority (0 = idle, 7 = max)
                   (void*)AppEntry, // Function where the thread starts executing
                   NULL );      // Argument passed into the entry function
}
```

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

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

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

6.2.2 Entry Functions

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

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

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

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

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

6.3 Timers

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

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

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

```

{
    Timer clTimer;
    clTimer.Init();

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

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

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

```

6.4 Semaphores

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

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

```

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

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

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

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

        // Consume the data!
    }
}

```

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

```

Semaphore clSemaphore;

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

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

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

```

6.5 Mutexes

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

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

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

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

```

Mutex clMutex; // Create a mutex globally.

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

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

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

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

    clMutex.Release();
}

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

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

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

    clMutex.Release();
}

```

6.6 Event Flags

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

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

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

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

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

Examples demonstrating the use of event flags are shown below.

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

EventFlag clEventFlag;

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

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

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

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

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

        // <do something>
    }
}
```

6.7 Messages

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

Sending a message consists of the following operations:

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

While receiving a message consists of the following steps:

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

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

6.7.1 Message Objects

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

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

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

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

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

6.7.2 Global Message Pool

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

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

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

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

6.7.3 Message Queues

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

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

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

6.7.4 Messaging Example

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

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

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

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

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

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

```

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

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

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

```

6.8 Sleep

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

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

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

```

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

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

```

6.9 Round-Robin Quantum

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

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

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

Chapter 7

Build System

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

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

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

7.1 Source Layout

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

Root	Base folder, contains recursive makefiles for build system
arduino	Arduino-specific headers and API documentation files
bootloader	Mark3 Bootloader code for AVR microcontrollers
build	Makefiles and device-configuraton data for various platforms
docs	Documentation (including this)
drivers	Device driver code for various supported devices
example	Example applications
fonts	Bitmap fonts converted from TTF, used by Mark3 graphics library
kernel	Basic Mark3 Components (the focus of this manual)
cpu	CPU-specific porting code
scripts	Scripts used to simplify build, documentation, and profiling
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
util	.net-based utils: font conversion, terminal, programmer, and configuration

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.

These steps are chained together automatically as part of the build.sh script found under the /scripts subdirectory. Running ./scripts/build.sh from the root of the embedded source directory will result in all headers being exported, libraries built, and applications built. This script will also default to building for atmega328p using GCC if none of the required environment variables have previously been configured.

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.

7.3 Building on Windows

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

Step 1 - Install Latest Atmel Studio IDE

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

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

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

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

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

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

Step 2 - Install MinGW and MinSys

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

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

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

Step 3 - Setup Include Paths in Platform Makefile

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

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

Step 4 - Build Mark3 using Bash

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

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

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

Chapter 8

License

8.1 License

Copyright (c) 2012-2015, Funkenstein Software Consulting All rights reserved.

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

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

Chapter 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

Thu Mar 5 21:04:16 EST 2015

9.2 Compiler Information

The kernel and test code used in these results were built using the following compiler: Using built-in specs. COLLECT_GCC=avr-gcc COLLECT_LTO_WRAPPER=/usr/lib/gcc/avr/4.8.2/lto-wrapper Target: avr Configured with: ../src/configure -v --enable-languages=c,c++ --prefix=/usr/lib --infodir=/usr/share/info --mandir=/usr/share/man --bindir=/usr/bin --libexecdir=/usr/lib --libdir=/usr/lib --enable-shared --with-system-zlib --enable-long-long --enable-nls --without-included-gettext --disable-libssp --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=avr [Thread](#) model: single gcc version 4.8.2 (GCC)

9.3 Profiling Results

- [Semaphore](#) Initialization: 40 cycles (averaged over 169 iterations)
- [Semaphore](#) Post (uncontested): 111 cycles (averaged over 169 iterations)
- [Semaphore](#) Pend (uncontested): 78 cycles (averaged over 169 iterations)
- [Semaphore](#) Flyback Time (Contested Pend): 1575 cycles (averaged over 169 iterations)
- [Mutex](#) Init: 223 cycles (averaged over 169 iterations)
- [Mutex](#) Claim: 223 cycles (averaged over 169 iterations)
- [Mutex](#) Release: 119 cycles (averaged over 169 iterations)
- [Thread](#) Initialize: 8280 cycles (averaged over 169 iterations)
- [Thread](#) Start: 775 cycles (averaged over 169 iterations)
- Context Switch: 191 cycles (averaged over 168 iterations)
- [Thread](#) Schedule: 95 cycles (averaged over 168 iterations)

Chapter 10

Code Size Profiling

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

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

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

10.1 Information

Subversion Repository Information:

- Repository Root: `svn+ssh://m0slevin.code.sf.net/p/mark3/source`
- Revision: 188
- URL: `svn+ssh://m0slevin.code.sf.net/p/mark3/source/branch/release/R1/embedded` Relative URL: `^/branch/release/R1/embedded`

Date Profiled: Thu Mar 5 21:04:20 EST 2015

10.2 Compiler Version

avr-gcc (GCC) 4.8.2 Copyright (C) 2013 Free Software Foundation, Inc. This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

10.3 Profiling Results

Mark3 Module Size Report:

- Synchronization Objects - Base Class..... : 84 Bytes
- Device [Driver](#) Framework (including /dev/null)... : 226 Bytes
- Synchronization Object - Event Flag..... : 770 Bytes
- Fundamental [Kernel](#) Linked-List Classes..... : 496 Bytes

- Message-based IPC..... : 426 Bytes
- [Mutex](#) (Synchronization Object)..... : 658 Bytes
- Performance-profiling timers..... : 546 Bytes
- Round-Robin Scheduling Support..... : 252 Bytes
- [Thread](#) Scheduling..... : 475 Bytes
- [Semaphore](#) (Synchronization Object)..... : 544 Bytes
- [Thread](#) Implementation..... : 1433 Bytes
- Fundamental [Kernel](#) Thread-list Data Structures.. : 212 Bytes
- Mark3 [Kernel](#) Base Class..... : 80 Bytes
- Software [Timer](#) Implementation..... : 1015 Bytes
- Runtime [Kernel](#) Trace Implementation..... : 0 Bytes
- Circular Logging Buffer Base Class..... : 0 Bytes
- Atmel AVR - [Kernel](#) Aware Simulation Support..... : 287 Bytes
- Atmel AVR - Basic Threading Support..... : 528 Bytes
- Atmel AVR - [Kernel](#) Interrupt Implementation..... : 56 Bytes
- Atmel AVR - [Kernel Timer](#) Implementation..... : 322 Bytes
- Atmel AVR - Profiling [Timer](#) Implementation..... : 256 Bytes

Mark3 [Kernel](#) Size Summary:

- [Kernel](#) : 2780 Bytes
- Synchronization Objects : 2398 Bytes
- Port : 1449 Bytes
- Features : 2039 Bytes
- Total Size : 8666 Bytes

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

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

BitStreamer	49
BlockHeap	50
BlockingObject	52
EventFlag	79
Mutex	132
Semaphore	176
CommandLine_t	58
DCPU	58
DCPU_Registers	62
DrawBitmap_t	68
DrawCircle_t	69
DrawEllipse_t	69
DrawLine_t	70
DrawMove_t	70
DrawPoint_t	71
DrawPoly_t	71
DrawRectangle_t	72
DrawStamp_t	73
DrawText_t	73
DrawVector_t	74
DrawWindow_t	74
DriverList	78
FixedHeap	81
Font_t	83
GlobalMessagePool	85
Glyph_t	86
GuiEvent_t	100
GuiEventSurface	100
HeapConfig	109
JoystickEvent_t	110
Kernel	110
Kernel_Aware	112
KernelAwareData_t	115
KernelSWI	116
KernelTimer	117
KeyEvent_t	119
LinkList	121

CircularLinkedList	57
ThreadList	204
DoubleLinkedList	67
TimerList	214
LinkedListNode	123
DCPUPlugin	62
Driver	75
DevNull	64
GraphicsDriver	86
GuiControl	92
ButtonControl	53
CheckBoxControl	55
GamePanelControl	83
GroupBoxControl	90
LabelControl	119
NotificationControl	155
PanelControl	158
ProgressControl	163
SlickButtonControl	180
SlickGroupBoxControl	182
SlickProgressControl	184
StubControl	193
GuiWindow	103
Message	128
Screen	173
Thread	196
Timer	209
MemUtil	124
MessageQueue	130
MouseEvent_t	132
NLFS	134
NLFS_RAM	150
NLFS_Block_t	145
NLFS_File	145
NLFS_File_Node_t	147
NLFS_File_Stat_t	148
NLFS_Host_t	149
NLFS_Node_t	149
NLFS_Root_Node_t	154
Option_t	157
Profiler	160
ProfileTimer	161
PseudoRandom	165
Quantum	167
Scheduler	169
ScreenList	174
ScreenManager	175
ShellCommand_t	178
ShellSupport	178
Slip	186
SlipDataVector	188
SlipMux	189
SlipTerm	191
SystemHeap	194
TextFX_t	196
ThreadPort	206
Tile_8x8	207

TileDef_t	208
TimerEvent_t	214
TimerScheduler	215
Token_t	217
TouchEvent_t	217
UnitTest	218

Chapter 12

Class Index

12.1 Class List

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

BitStreamer	49
BlockHeap	
Single-block-size heap	50
BlockingObject	
Class implementing thread-blocking primitives	52
ButtonControl	53
CheckBoxControl	55
CircularLinkedList	
Circular-linked-list data type, inherited from the base LinkedList type	57
CommandLine_t	
Structure containing multiple representations for command-line data	58
DCPU	
DCPU emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH	58
DCPU_Registers	
Structure defining the DCPU hardware registers	62
DCPUPlugin	
Class used to provide the hardware device abstraction between the DCPU-16 emulator/VM and the host system	62
DevNull	
This class implements the "default" driver (/dev/null)	64
DoubleLinkedList	
Doubly-linked-list data type, inherited from the base LinkedList type	67
DrawBitmap_t	
Defines a bitmap	68
DrawCircle_t	
Defines a circle	69
DrawEllipse_t	
Defines a ellipse	69
DrawLine_t	
Defines a simple line	70
DrawMove_t	
Simple 2D copy/paste	70
DrawPoint_t	
Defines a pixel	71
DrawPoly_t	
Defines the structure of an arbitrary polygon	71

DrawRectangle_t	Defines a rectangle	72
DrawStamp_t	Defines a 1-bit 2D bitmap of arbitrary resolution	73
DrawText_t	Defines a bitmap-rendered string	73
DrawVector_t	Specifies a single 2D point	74
DrawWindow_t	Defines the active window - establishes boundaries for drawing on the current display	74
Driver	Base device-driver class used in hardware abstraction	75
DriverList	List of Driver objects used to keep track of all device drivers in the system	78
EventFlag	Blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system	79
FixedHeap	Fixed-size-block heap allocator with multiple block sizes	81
Font_t	83
GamePanelControl	83
GlobalMessagePool	Implements a list of message objects shared between all threads	85
Glyph_t	86
GraphicsDriver	Defines the base graphics driver class, which is inherited by all other graphics drivers	86
GroupBoxControl	90
GuiControl	GUI Control Base Class	92
GuiEvent_t	Composite UI event structure	100
GuiEventSurface	GUI Event Surface Object	100
GuiWindow	Basic Window Class	103
HeapConfig	Heap configuration object	109
JoystickEvent_t	Joystick UI event structure	110
Kernel	Class that encapsulates all of the kernel startup functions	110
Kernel_Aware	The Kernel_Aware class	112
KernelAwareData_t	115
KernelSWI	Class providing the software-interrupt required for context-switching in the kernel	116
KernelTimer	Hardware timer interface, used by all scheduling/timer subsystems	117
KeyEvent_t	Keyboard UI event structure definition	119
LabelControl	119
LinkList	Abstract-data-type from which all other linked-lists are derived	121
LinkListNode	Basic linked-list node data structure	123
MemUtil	String and Memory manipulation class	124

Message	Class to provide message-based IPC services in the kernel	128
MessageQueue	List of messages, used as the channel for sending and receiving messages between threads	130
MouseEvent_t	Mouse UI event structure	132
Mutex	Mutual-exclusion locks, based on BlockingObject	132
NLFS	Nice Little File System class	134
NLFS_Block_t	Block data structure	145
NLFS_File	The NLFS_File class	145
NLFS_File_Node_t	Data structure for the "file" FS-node type	147
NLFS_File_Stat_t	Structure used to report the status of a given file	148
NLFS_Host_t	Union used for managing host-specific pointers/data-types	149
NLFS_Node_t	Filesystem node data structure	149
NLFS_RAM	The NLFS_RAM class	150
NLFS_Root_Node_t	Data structure for the Root-configuration FS-node type	154
NotificationControl	155
Option_t	Structure used to represent a command-line option with its arguments	157
PanelControl	158
Profiler	System profiling timer interface	160
ProfileTimer	Profiling timer	161
ProgressControl	163
PseudoRandom	The PseudoRandom class	165
Quantum	Static-class used to implement Thread quantum functionality, which is a key part of round-robin scheduling	167
Scheduler	Priority-based round-robin Thread scheduling, using ThreadLists for housekeeping	169
Screen	173
ScreenList	174
ScreenManager	175
Semaphore	Counting semaphore, based on BlockingObject base class	176
ShellCommand_t	Data structure defining a lookup table correlating a command name to its handler function	178
ShellSupport	Features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell	178
SlickButtonControl	180
SlickGroupBoxControl	182
SlickProgressControl	184
Slip	Object used to frame communications over an abstract device using the serial-line internet protocol (SLIP)	186

SlipDataVector	Data structure used for vector-based SLIP data transmission	188
SlipMux	Static-class which implements a multiplexed stream of SLIP data over a single interface	189
SlipTerm	Class implementing a simple debug terminal interface	191
StubControl	Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented	193
SystemHeap	Implements a heap which is accessible from all components in the system	194
TextFX_t	196
Thread	Object providing fundamental multitasking support in the kernel	196
ThreadList	This class is used for building thread-management facilities, such as schedulers, and blocking objects	204
ThreadPort	Class defining the architecture specific functions required by the kernel	206
Tile_8x8	207
TileDef_t	TileDef_t Structure defining parameters for a color-indexed tile	208
Timer	Timer - an event-driven execution context based on a specified time interval	209
TimerEvent_t	Timer UI event structure	214
TimerList	TimerList class - a doubly-linked-list of timer objects	214
TimerScheduler	"Static" Class used to interface a global TimerList with the rest of the kernel	215
Token_t	Token descriptor struct format	217
TouchEvent_t	Touch UI event structure	217
UnitTest	Class used to implement a simple unit-testing framework	218

Chapter 13

File Index

13.1 File List

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

/home/moslevin/Project/R1/stage/src/ atomic.cpp	
Basic Atomic Operations	221
/home/moslevin/Project/R1/stage/src/ atomic.h	
Basic Atomic Operations	223
/home/moslevin/Project/R1/stage/src/ bitstream.cpp	??
/home/moslevin/Project/R1/stage/src/ bitstream.h	??
/home/moslevin/Project/R1/stage/src/ blocking.cpp	
Implementation of base class for blocking objects	224
/home/moslevin/Project/R1/stage/src/ blocking.h	
Blocking object base class declarations	225
/home/moslevin/Project/R1/stage/src/ colordepth.h	??
/home/moslevin/Project/R1/stage/src/ colorspace.h	??
/home/moslevin/Project/R1/stage/src/ control_button.cpp	
GUI Button Control Implementation	226
/home/moslevin/Project/R1/stage/src/ control_button.h	
GUI Button Control	229
/home/moslevin/Project/R1/stage/src/ control_checkbox.cpp	
Checkbox Control	231
/home/moslevin/Project/R1/stage/src/ control_checkbox.h	
Checkbox Control	234
/home/moslevin/Project/R1/stage/src/ control_gamepanel.cpp	
GUI Panel Control Implementation with joystick control and tick-based state machine updates	235
/home/moslevin/Project/R1/stage/src/ control_gamepanel.h	
GUI Game Panel Control	236
/home/moslevin/Project/R1/stage/src/ control_groupbox.cpp	
GUI GroupBox Control Implementation	237
/home/moslevin/Project/R1/stage/src/ control_groupbox.h	
GUI Group Box Control	238
/home/moslevin/Project/R1/stage/src/ control_label.cpp	??
/home/moslevin/Project/R1/stage/src/ control_label.h	
GUI Label Control	239
/home/moslevin/Project/R1/stage/src/ control_notification.cpp	
Notification pop-up control	240
/home/moslevin/Project/R1/stage/src/ control_notification.h	
Notification pop-up control	242
/home/moslevin/Project/R1/stage/src/ control_panel.cpp	
GUI Panel Control Implementation	243

/home/moslevin/Project/R1/stage/src/control_panel.h	
GUI Panel Control	244
/home/moslevin/Project/R1/stage/src/control_progress.cpp	
GUI Progress Bar Control	245
/home/moslevin/Project/R1/stage/src/control_progress.h	
GUI Progress Bar Control	247
/home/moslevin/Project/R1/stage/src/control_slickbutton.cpp	??
/home/moslevin/Project/R1/stage/src/control_slickbutton.h	
GUI Button Control, with a flare	248
/home/moslevin/Project/R1/stage/src/control_slickgroupbox.cpp	??
/home/moslevin/Project/R1/stage/src/control_slickgroupbox.h	??
/home/moslevin/Project/R1/stage/src/control_slickprogress.cpp	
GUI Progress Bar Control, with flare	249
/home/moslevin/Project/R1/stage/src/control_slickprogress.h	
GUI Progress Bar Control, with flare	250
/home/moslevin/Project/R1/stage/src/dcpu.cpp	
Portable DCPU-16 CPU emulator	252
/home/moslevin/Project/R1/stage/src/dcpu.h	
DCPU-16 emulator	266
/home/moslevin/Project/R1/stage/src/debug_tokens.h	
Hex codes/translation tables used for efficient string tokenization	272
/home/moslevin/Project/R1/stage/src/draw.h	
Raster graphics APIs Description: Implements basic drawing functionality	274
/home/moslevin/Project/R1/stage/src/driver.cpp	
Device driver/hardware abstraction layer	277
/home/moslevin/Project/R1/stage/src/driver.h	
Driver abstraction framework	280
/home/moslevin/Project/R1/stage/src/eventflag.cpp	
Event Flag Blocking Object/IPC-Object implementation	281
/home/moslevin/Project/R1/stage/src/eventflag.h	
Event Flag Blocking Object/IPC-Object definition	285
/home/moslevin/Project/R1/stage/src/fixed_heap.cpp	
Fixed-block-size memory management	286
/home/moslevin/Project/R1/stage/src/fixed_heap.h	
Fixed-block-size heaps	288
/home/moslevin/Project/R1/stage/src/font.h	
Font structure definitions	290
/home/moslevin/Project/R1/stage/src/fontport.h	??
/home/moslevin/Project/R1/stage/src/graphics.cpp	
Generic graphics driver implementation	291
/home/moslevin/Project/R1/stage/src/graphics.h	
Graphics driver class declaration	304
/home/moslevin/Project/R1/stage/src/gui.cpp	
Graphical User Interface classes and data structure definitions	305
/home/moslevin/Project/R1/stage/src/gui.h	
Graphical User Interface classes and data structure declarations	315
/home/moslevin/Project/R1/stage/src/kernel.cpp	
Kernel initialization and startup code	321
/home/moslevin/Project/R1/stage/src/kernel.h	
Kernel initialization and startup class	322
/home/moslevin/Project/R1/stage/src/kernel_aware.cpp	??
/home/moslevin/Project/R1/stage/src/kernel_aware.h	??
/home/moslevin/Project/R1/stage/src/kernel_debug.h	
Macros and functions used for assertions, kernel traces, etc	323
/home/moslevin/Project/R1/stage/src/kernelswi.cpp	
Kernel Software interrupt implementation for ATMega328p	325
/home/moslevin/Project/R1/stage/src/kernelswi.h	
Kernel Software interrupt declarations	327

/home/moslevin/Project/R1/stage/src/ kerneltimer.cpp	
Kernel Timer Implementation for ATmega328p	327
/home/moslevin/Project/R1/stage/src/ kerneltimer.h	
Kernel Timer Class declaration	330
/home/moslevin/Project/R1/stage/src/ kerneltypes.h	
Basic data type primitives used throughout the OS	331
/home/moslevin/Project/R1/stage/src/ keycodes.h	
Standard ASCII keyboard codes	333
/home/moslevin/Project/R1/stage/src/ kprofile.cpp	
ATmega328p Profiling timer implementation	335
/home/moslevin/Project/R1/stage/src/ kprofile.h	
Profiling timer hardware interface	337
/home/moslevin/Project/R1/stage/src/ ksemaphore.cpp	
Semaphore Blocking-Object Implementation	338
/home/moslevin/Project/R1/stage/src/ ksemaphore.h	
Semaphore Blocking Object class declarations	341
/home/moslevin/Project/R1/stage/src/ ll.cpp	
Core Linked-List implementation, from which all kernel objects are derived	342
/home/moslevin/Project/R1/stage/src/ ll.h	
Core linked-list declarations, used by all kernel list types	345
/home/moslevin/Project/R1/stage/src/ manual.h	
Ascii-format documentation, used by doxygen to create various printable and viewable forms	346
/home/moslevin/Project/R1/stage/src/ mark3cfg.h	
Mark3 Kernel Configuration	350
/home/moslevin/Project/R1/stage/src/ memutil.cpp	
Implementation of memory, string, and conversion routines	352
/home/moslevin/Project/R1/stage/src/ memutil.h	
Utility class containing memory, string, and conversion routines	357
/home/moslevin/Project/R1/stage/src/ message.cpp	
Inter-thread communications via message passing	359
/home/moslevin/Project/R1/stage/src/ message.h	
Inter-thread communication via message-passing	362
/home/moslevin/Project/R1/stage/src/ mutex.cpp	
Mutual-exclusion object	363
/home/moslevin/Project/R1/stage/src/ mutex.h	
Mutual exclusion class declaration	367
/home/moslevin/Project/R1/stage/src/ nlfs.cpp	
Nice Little Filesystem (NLFS) implementation for Mark3	369
/home/moslevin/Project/R1/stage/src/ nlfs.h	
Nice Little Filesystem (NLFS) - a simple, embeddable filesystem	382
/home/moslevin/Project/R1/stage/src/ nlfs_config.h	
NLFS configuration parameters	386
/home/moslevin/Project/R1/stage/src/ nlfs_file.cpp	
Nice Little Filesystem - File Access Class	386
/home/moslevin/Project/R1/stage/src/ nlfs_file.h	
NLFS file access class	391
/home/moslevin/Project/R1/stage/src/ nlfs_ram.cpp	
RAM-based Nice Little Filesystem (NLFS) driver	392
/home/moslevin/Project/R1/stage/src/ nlfs_ram.h	
RAM-based Nice Little Filesystem (NLFS) driver	393
/home/moslevin/Project/R1/stage/src/ panic_codes.h	
Defines the reason codes thrown when a kernel panic occurs	394
/home/moslevin/Project/R1/stage/src/ profile.cpp	
Code profiling utilities	395
/home/moslevin/Project/R1/stage/src/ profile.h	
High-precision profiling timers	397
/home/moslevin/Project/R1/stage/src/ profiling_results.h	??

/home/moslevin/Project/R1/stage/src/quantum.cpp	
Thread Quantum Implementation for Round-Robin Scheduling	398
/home/moslevin/Project/R1/stage/src/quantum.h	
Thread Quantum declarations for Round-Robin Scheduling	400
/home/moslevin/Project/R1/stage/src/rand_lfsr.cpp	??
/home/moslevin/Project/R1/stage/src/rand_lfsr.h	
Linear Feedback Shift Register based Pseudo-Random Number Generator	401
/home/moslevin/Project/R1/stage/src/scheduler.cpp	
Strict-Priority + Round-Robin thread scheduler implementation	402
/home/moslevin/Project/R1/stage/src/scheduler.h	
Thread scheduler function declarations	404
/home/moslevin/Project/R1/stage/src/screen.cpp	
Higher level window management framework	405
/home/moslevin/Project/R1/stage/src/screen.h	
Higher level window management framework	407
/home/moslevin/Project/R1/stage/src/shell_support.cpp	
Support functions & data structures useful in implementing a shell	408
/home/moslevin/Project/R1/stage/src/shell_support.h	
Support functions & data structures useful in implementing a shell	411
/home/moslevin/Project/R1/stage/src/sizeprofile.h	??
/home/moslevin/Project/R1/stage/src/slip.cpp	
Serial Line IP framing code	412
/home/moslevin/Project/R1/stage/src/slip.h	
Serial Line IP framing code	417
/home/moslevin/Project/R1/stage/src/slip_mux.cpp	
FunkenSlip Channel Multiplexer	418
/home/moslevin/Project/R1/stage/src/slip_mux.h	
FunkenSlip Channel Multiplexer	420
/home/moslevin/Project/R1/stage/src/slipterm.cpp	
Serial debug interface using SLIP protocol, and FunkenSlip multiplexing	421
/home/moslevin/Project/R1/stage/src/slipterm.h	
Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing	422
/home/moslevin/Project/R1/stage/src/system_heap.cpp	
Global system-heap implementation	423
/home/moslevin/Project/R1/stage/src/system_heap.h	
Global system-heap implementation	426
/home/moslevin/Project/R1/stage/src/system_heap_config.h	
System heap configuration - defines the block sizes and counts used to fulfill system/service allocations	430
/home/moslevin/Project/R1/stage/src/thread.cpp	
Platform-Independent thread class Definition	431
/home/moslevin/Project/R1/stage/src/thread.h	
Platform independent thread class declarations	436
/home/moslevin/Project/R1/stage/src/threadlist.cpp	
Thread linked-list definitions	439
/home/moslevin/Project/R1/stage/src/threadlist.h	
Thread linked-list declarations	440
/home/moslevin/Project/R1/stage/src/threadport.cpp	
ATMega328p Multithreading	441
/home/moslevin/Project/R1/stage/src/threadport.h	
ATMega328p Multithreading support	444
/home/moslevin/Project/R1/stage/src/tiles.cpp	??
/home/moslevin/Project/R1/stage/src/tiles.h	??
/home/moslevin/Project/R1/stage/src/timerlist.cpp	
Timer data structure + scheduler implementations	447
/home/moslevin/Project/R1/stage/src/timerlist.h	
Timer list and timer-scheduling declarations	452

/home/moslevin/Project/R1/stage/src/ tracebuffer.cpp	
Kernel trace buffer class definition	454
/home/moslevin/Project/R1/stage/src/ tracebuffer.h	
Kernel trace buffer class declaration	455
/home/moslevin/Project/R1/stage/src/ unit_test.cpp	
Unit test class definition	456
/home/moslevin/Project/R1/stage/src/ unit_test.h	
Unit test class declarations	457
/home/moslevin/Project/R1/stage/src/ writebuf16.cpp	
16 bit circular buffer implementation with callbacks	459
/home/moslevin/Project/R1/stage/src/ writebuf16.h	
Thread-safe circular buffer implementation with 16-bit elements	461

Chapter 14

Class Documentation

14.1 BitStreamer Class Reference

Public Member Functions

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

Private Attributes

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

14.1.1 Detailed Description

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

14.1.2 Member Function Documentation

14.1.2.1 void BitStreamer::AdvanceByte (void)

AdvanceByte.

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

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

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

Init.

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

Parameters

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

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

14.1.2.3 K_UCHAR BitStreamer::ReadBits (K_UCHAR ucNumBits_)

ReadBits.

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

Parameters

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

Returns

Bits read as an 8-bit unsigned integer.

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

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

- /home/moslevin/Project/R1/stage/src/bitstream.h
- /home/moslevin/Project/R1/stage/src/bitstream.cpp

14.2 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_cList](#)

Linked list used to manage the blocks.

14.2.1 Detailed Description

Single-block-size heap.

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

14.2.2 Member Function Documentation

14.2.2.1 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](#).

14.2.2.2 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](#).

14.2.2.3 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](#).

14.2.2.4 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:

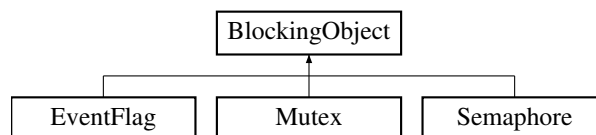
- [/home/moslevin/Project/R1/stage/src/fixed_heap.h](#)
- [/home/moslevin/Project/R1/stage/src/fixed_heap.cpp](#)

14.3 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.

14.3.1 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](#).

14.3.2 Member Function Documentation

14.3.2.1 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](#).

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

Parameters

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

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

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

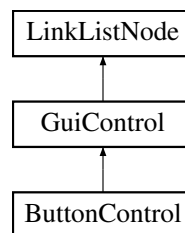
Definition at line 52 of file [blocking.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/blocking.h](#)
- [/home/moslevin/Project/R1/stage/src/blocking.cpp](#)

14.4 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

14.4.1 Detailed Description

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

14.4.2 Member Function Documentation

14.4.2.1 `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](#).

14.4.2.2 `void ButtonControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.4.2.3 `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](#).

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

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

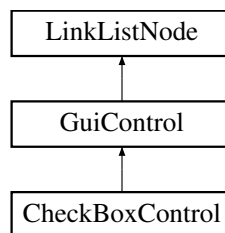
Definition at line 117 of file [control_button.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/control_button.h](#)
- [/home/moslevin/Project/R1/stage/src/control_button.cpp](#)

14.5 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

14.5.1 Detailed Description

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

14.5.2 Member Function Documentation

14.5.2.1 `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](#).

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

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

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

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

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

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

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

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

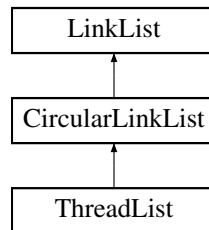
- [/home/moslevin/Project/R1/stage/src/control_checkbox.h](#)
- [/home/moslevin/Project/R1/stage/src/control_checkbox.cpp](#)

14.6 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

14.6.1 Detailed Description

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

Definition at line 196 of file [ll.h](#).

14.6.2 Member Function Documentation

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

Add the linked list node to this linked list.

Parameters

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

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

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

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

Add the linked list node to this linked list.

Parameters

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

Implements [LinkedList](#).

Reimplemented in [ThreadList](#).

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

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

- [/home/moslevin/Project/R1/stage/src/ll.h](#)
- [/home/moslevin/Project/R1/stage/src/ll.cpp](#)

14.7 CommandLine_t Struct Reference

Structure containing multiple representations for command-line data.

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

Public Attributes

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

14.7.1 Detailed Description

Structure containing multiple representations for command-line data.

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

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

- [/home/moslevin/Project/R1/stage/src/shell_support.h](#)

14.8 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](#) *pciPlugin_)
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.

14.8.1 Detailed Description

[DCPU](#) emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Definition at line 360 of file [dcpu.h](#).

14.8.2 Member Function Documentation

14.8.2.1 void [DCPU::AddPlugin](#) ([DCPUPlugin](#) * *pclPlugin_*)

Add a plugin to the CPU.

Parameters

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

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

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

Parameters

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

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

14.8.2.3 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 392 of file [dcpu.h](#).

14.8.2.4 void DCPU::HWN () [private]

Returns the number of connected hardware devices to "a"

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

14.8.2.5 void DCPU::IAQ () [private]

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

Interrupts queued

Interrupts triggered

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

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

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

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

Parameters

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

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

14.8.2.7 void DCPU::RFI () [private]

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

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

14.8.2.8 void DCPU::SendInterrupt (K_USHORT *usMessage_*)

Send an interrupt to the CPU with a given message.

Parameters

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

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

14.8.3 Member Data Documentation

14.8.3.1 DoubleLinkedList DCPU::m_clPluginList [private]

Linked-list of plug-ins.

Definition at line 490 of file [dcpu.h](#).

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

- [/home/moslevin/Project/R1/stage/src/dcpu.h](#)
- [/home/moslevin/Project/R1/stage/src/dcpu.cpp](#)

14.9 DCPU_Registers Struct Reference

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

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

Public Attributes

14.9.1 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:

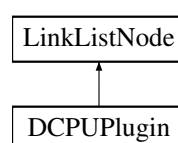
- [/home/moslevin/Project/R1/stage/src/dcpu.h](#)

14.10 DCPUPugin Class Reference

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

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

Inheritance diagram for DCPUPugin:



Public Member Functions

- void [Init](#) (K_USHORT usDeviceNumber_, K_ULONG ulHWID_, K_ULONG ulVID_, K_USHORT usVersion_, [DCPU_Callback](#) pfCallback_)
Initialize the [DCPU](#) plugin extension.
- void [Enumerate](#) ([DCPU_Registers](#) *pstRegisters_)
Perform hardware enumeration to the target VM specified by the register set.
- void [Interrupt](#) ([DCPU](#) *pciCPU_)
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 [DCPUPuginList](#)

Additional Inherited Members

14.10.1 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](#).

14.10.2 Member Function Documentation

14.10.2.1 void [DCPUPugin::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 312 of file [dcpu.h](#).

14.10.2.2 K_USHORT DCPUPugin::GetDeviceNumber () [inline]

Return the device number associated with this plugin.

Returns

Device number associated with this plugin

Definition at line 340 of file [dcpu.h](#).

14.10.2.3 void DCPUPugin::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](#).

14.10.2.4 void DCPUPugin::Interrupt (DCPU * *pcICPU_*) [inline]

Execute the hardware callback.

Parameters

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

Definition at line 328 of file [dcpu.h](#).

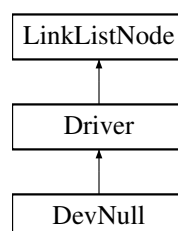
The documentation for this class was generated from the following file:

- [/home/moslevin/Project/R1/stage/src/dcpu.h](#)

14.11 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

14.11.1 Detailed Description

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

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

14.11.2 Member Function Documentation

14.11.2.1 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](#).

14.11.2.2 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 analogous 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](#).

14.11.2.3 `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](#).

14.11.2.4 `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](#).

14.11.2.5 `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)
-----------------	--

<code>pucData_</code>	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:

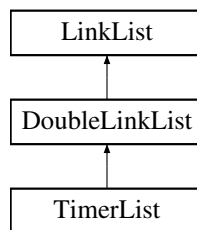
- [/home/moslevin/Project/R1/stage/src/driver.cpp](#)

14.12 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

14.12.1 Detailed Description

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

Definition at line 165 of file [ll.h](#).

14.12.2 Member Function Documentation

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

Add the linked list node to this linked list.

Parameters

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

Implements [LinkedList](#).

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

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

Add the linked list node to this linked list.

Parameters

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

Implements [LinkedList](#).

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

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

- [/home/moslevin/Project/R1/stage/src/ll.h](#)
- [/home/moslevin/Project/R1/stage/src/ll.cpp](#)

14.13 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.

14.13.1 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/Project/R1/stage/src/draw.h](#)

14.14 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.

14.14.1 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/Project/R1/stage/src/draw.h](#)

14.15 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.

14.15.1 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/Project/R1/stage/src/draw.h](#)

14.16 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.

14.16.1 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/Project/R1/stage/src/draw.h](#)

14.17 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.

14.17.1 Detailed Description

Simple 2D copy/paste.

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

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

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

- [/home/moslevin/Project/R1/stage/src/draw.h](#)

14.18 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.

14.18.1 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/Project/R1/stage/src/draw.h](#)

14.19 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.

14.19.1 Detailed Description

Defines the structure of an arbitrary polygon.

Can be used to specify the

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

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

- [/home/moslevin/Project/R1/stage/src/draw.h](#)

14.20 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.

14.20.1 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/Project/R1/stage/src/draw.h](#)

14.21 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.

14.21.1 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/Project/R1/stage/src/draw.h](#)

14.22 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.

14.22.1 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/Project/R1/stage/src/draw.h](#)

14.23 DrawVector_t Struct Reference

Specifies a single 2D point.

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

Public Attributes

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

14.23.1 Detailed Description

Specifies a single 2D point.

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

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

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

- [/home/moslevin/Project/R1/stage/src/draw.h](#)

14.24 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.

14.24.1 Detailed Description

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

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

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

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

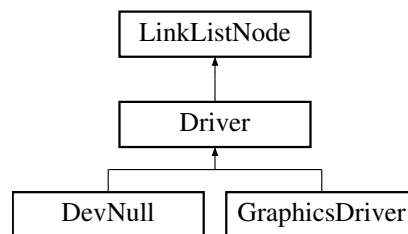
- [/home/moslevin/Project/R1/stage/src/draw.h](#)

14.25 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

14.25.1 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](#).

14.25.2 Member Function Documentation

14.25.2.1 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](#).

14.25.2.2 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](#).

14.25.2.3 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](#).

14.25.2.4 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](#).

14.25.2.5 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](#).

14.25.2.6 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](#).

14.25.2.7 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/Project/R1/stage/src/driver.h](#)

14.26 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.

14.26.1 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](#).

14.26.2 Member Function Documentation

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

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

Parameters

pclDriver_	pointer to the driver object to add to the global driver list.
----------------------------	--

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

14.26.2.2 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](#).

14.26.2.3 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](#).

14.26.2.4 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:

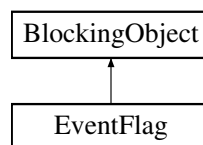
- /home/moslevin/Project/R1/stage/src/[driver.h](#)
- /home/moslevin/Project/R1/stage/src/[driver.cpp](#)

14.27 EventFlag Class Reference

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

```
#include <eventflag.h>
```

Inheritance diagram for EventFlag:



Public Member Functions

- void [Init](#) ()
Init Initializes the [EventFlag](#) object prior to use.
- K_USHORT [Wait](#) (K_USHORT usMask_, EventFlagOperation_t eMode_)
Wait - Block a thread on the specific flags in this event flag group.
- K_USHORT [Wait](#) (K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_)
Wait - Block a thread on the specific flags in this event flag group.
- void [WakeMe](#) ([Thread](#) *pclOwner_)
- void [Set](#) (K_USHORT usMask_)
Set - Set additional flags in this object (logical OR).
- void [Clear](#) (K_USHORT usMask_)
ClearFlags - Clear a specific set of flags within this object, specific by bitmask.
- K_USHORT [GetMask](#) ()
GetMask Returns the state of the 16-bit bitmask within this object.

Private Member Functions

- K_USHORT [Wait_i](#) (K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_)

Private Attributes

- K_USHORT **m_usSetMask**

Additional Inherited Members

14.27.1 Detailed Description

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

Each [EventFlag](#) object contains a 16-bit bitmask, which is used to trigger events on associated threads. Threads wishing to block, waiting for a specific event to occur can wait on any pattern within this 16-bit bitmask to be set. Here, we provide the ability for a thread to block, waiting for ANY bits in a specified mask to be set, or for ALL bits within a specific mask to be set. Depending on how the object is configured, the bits that triggered the wakeup can be automatically cleared once a match has occurred.

Definition at line 46 of file [eventflag.h](#).

14.27.2 Member Function Documentation

14.27.2.1 void EventFlag::Clear (K_USHORT *usMask_*)

ClearFlags - Clear a specific set of flags within this object, specific by bitmask.

Parameters

<i>usMask_</i>	- Bitmask of flags to clear
----------------	-----------------------------

Definition at line 283 of file [eventflag.cpp](#).

14.27.2.2 K_USHORT EventFlag::GetMask ()

GetMask Returns the state of the 16-bit bitmask within this object.

Returns

The state of the 16-bit bitmask

Definition at line 292 of file [eventflag.cpp](#).

14.27.2.3 void EventFlag::Set (K_USHORT *usMask_*)

Set - Set additional flags in this object (logical OR).

This API can potentially result in threads blocked on [Wait\(\)](#) to be unblocked.

Parameters

<i>usMask_</i>	- Bitmask of flags to set.
----------------	----------------------------

Definition at line 164 of file [eventflag.cpp](#).

14.27.2.4 K_USHORT EventFlag::Wait (K_USHORT *usMask_*, EventFlagOperation_t *eMode_*)

Wait - Block a thread on the specific flags in this event flag group.

Parameters

<i>usMask_</i>	- 16-bit bitmask to block on
<i>eMode_</i>	- EVENT_FLAG_ANY: Thread will block on any of the bits in the mask • EVENT_FLAG_ALL: Thread will block on all of the bits in the mask

Returns

Bitmask condition that caused the thread to unblock, or 0 on error or timeout

Definition at line 146 of file [eventflag.cpp](#).

14.27.2.5 K_USHORT EventFlag::Wait (K_USHORT *usMask_*, EventFlagOperation_t *eMode_*, K_ULONG *ulTimeMS_*)

Wait - Block a thread on the specific flags in this event flag group.

Parameters

<i>usMask_</i>	- 16-bit bitmask to block on
<i>eMode_</i>	- EVENT_FLAG_ANY: Thread will block on any of the bits in the mask • EVENT_FLAG_ALL: Thread will block on all of the bits in the mask
<i>ulTimeMS_</i>	- Time to block (in ms)

Returns

Bitmask condition that caused the thread to unblock, or 0 on error or timeout

Definition at line 157 of file [eventflag.cpp](#).

14.27.2.6 K_USHORT EventFlag::Wait_i (K_USHORT *usMask_*, EventFlagOperation_t *eMode_*, K_ULONG *ulTimeMS_*)
[private]

! If the Yield operation causes a new thread to be chosen, there will ! Be a context switch at the above [CS_EXIT\(\)](#).
The original calling ! thread will not return back until a matching SetFlags call is made ! or a timeout occurs.

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

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

- [/home/moslevin/Project/R1/stage/src/eventflag.h](#)
- [/home/moslevin/Project/R1/stage/src/eventflag.cpp](#)

14.28 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.

14.28.1 Detailed Description

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

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

14.28.2 Member Function Documentation

14.28.2.1 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](#).

14.28.2.2 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](#).

14.28.2.3 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/Project/R1/stage/src/fixed_heap.h](#)
- [/home/moslevin/Project/R1/stage/src/fixed_heap.cpp](#)

14.29 Font_t Struct Reference

Public Attributes

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

14.29.1 Detailed Description

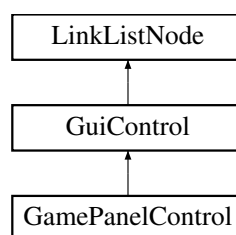
Definition at line 43 of file [font.h](#).

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

- [/home/moslevin/Project/R1/stage/src/font.h](#)

14.30 GamePanelControl Class Reference

Inheritance diagram for GamePanelControl:



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.

Private Attributes

- [JoystickEvent_t m_stLastJoy](#)
- [JoystickEvent_t m_stCurrentJoy](#)

Additional Inherited Members

14.30.1 Detailed Description

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

14.30.2 Member Function Documentation

14.30.2.1 `virtual void GamePanelControl::Activate (bool bActivate_) [inline],[virtual]`

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

Parameters

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

Implements [GuiControl](#).

Definition at line 38 of file [control_gamepanel.h](#).

14.30.2.2 `void GamePanelControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.30.2.3 `virtual void GamePanelControl::Init () [inline],[virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.30.2.4 `GuiReturn_t GamePanelControl::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 33 of file [control_gamepanel.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/control_gamepanel.h](#)

- [/home/moslevin/Project/R1/stage/src/control_gamepanel.cpp](#)

14.31 GlobalMessagePool Class Reference

Implements a list of message objects shared between all threads.

```
#include <message.h>
```

Static Public Member Functions

- static void [Init](#) ()
Initialize the message queue prior to use.
- static void [Push](#) ([Message](#) *pclMessage_)
Return a previously-claimed message object back to the global queue.
- static [Message](#) * [Pop](#) ()
Pop a message from the global queue, returning it to the user to be populated before sending by a transmitter.

Static Private Attributes

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

14.31.1 Detailed Description

Implements a list of message objects shared between all threads.

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

14.31.2 Member Function Documentation

14.31.2.1 [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 [70](#) of file [message.cpp](#).

14.31.2.2 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

<code>pciMessage_</code>	Pointer to the Message object to return back to the global queue
--------------------------	--

Definition at line 58 of file [message.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/message.h](#)
- [/home/moslevin/Project/R1/stage/src/message.cpp](#)

14.32 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.

14.32.1 Detailed Description

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

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

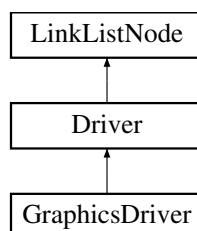
- [/home/moslevin/Project/R1/stage/src/font.h](#)

14.33 GraphicsDriver Class Reference

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

```
#include <graphics.h>
```

Inheritance diagram for GraphicsDriver:



Public Member Functions

- virtual void [DrawPixel](#) ([DrawPoint_t](#) *pstPoint_)
Draw a single pixel to the display.
- virtual void [ReadPixel](#) ([DrawPoint_t](#) *pstPoint_)
Read a single pixel from the display.

- virtual void [ClearScreen](#) ()
Clear the screen (initializes to all black pixels)
- virtual void [Point](#) ([DrawPoint_t](#) *pstPoint_)
Draw a pixel to the display.
- virtual void [Line](#) ([DrawLine_t](#) *pstLine_)
Draw a line to the display using Bresenham's line drawing algorithm.
- virtual void [Rectangle](#) ([DrawRectangle_t](#) *pstRectangle_)
Draws a rectangle on the display.
- virtual void [Circle](#) ([DrawCircle_t](#) *pstCircle_)
Draw a circle to the display.
- virtual void [Ellipse](#) ([DrawEllipse_t](#) *pstEllipse_)
Draw an ellipse to the display.
- virtual void [Bitmap](#) ([DrawBitmap_t](#) *pstBitmap_)
Draw an RGB image on the display.
- virtual void [Stamp](#) ([DrawStamp_t](#) *pstStamp_)
Draws a stamp (a 1-bit bitmap) on the display.
- virtual void [Move](#) ([DrawMove_t](#) *pstMove_)
Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.
- virtual void [TriangleWire](#) ([DrawPoly_t](#) *pstPoly_)
Draw a wireframe triangle to the display.
- virtual void [TriangleFill](#) ([DrawPoly_t](#) *pstPoly_)
Draw a filled triangle to the display.
- virtual void [Polygon](#) ([DrawPoly_t](#) *pstPoly_)
- virtual void [Text](#) ([DrawText_t](#) *pstText_)
Draw a string of text to the display using a bitmap font.
- void [TextFX](#) ([DrawText_t](#) *pstText_, [TextFX_t](#) *pstFX_)
Render a string of text to the display with effects.
- virtual K_USHORT [TextWidth](#) ([DrawText_t](#) *pstText_)
- void [SetWindow](#) ([DrawWindow_t](#) *pstWindow_)
Set the drawable window of the screen.
- void [ClearWindow](#) ()
Clear the window - resetting the boundaries to the entire drawable area of the screen.

Protected Attributes

- K_USHORT **m_usResX**
- K_USHORT **m_usResY**
- K_USHORT **m_usLeft**
- K_USHORT **m_usTop**
- K_USHORT **m_usRight**
- K_USHORT **m_usBottom**
- K_UCHAR **m_ucBPP**

Additional Inherited Members

14.33.1 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](#).

14.33.2 Member Function Documentation

14.33.2.1 void GraphicsDriver::Bitmap (DrawBitmap_t * *pstBitmap_*) [virtual]

Draw an RGB image on the display.

Parameters

<i>pstBitmap_</i>	- pointer to the bitmap object to display
-------------------	---

Definition at line 302 of file [graphics.cpp](#).

14.33.2.2 void GraphicsDriver::Circle (DrawCircle_t * *pstCircle_*) [virtual]

Draw a circle to the display.

Parameters

<i>pstCircle_</i>	- pointer to the circle to draw
-------------------	---------------------------------

Definition at line 178 of file [graphics.cpp](#).

14.33.2.3 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](#).

14.33.2.4 void GraphicsDriver::Ellipse (DrawEllipse_t * *pstEllipse_*) [virtual]

Draw an ellipse to the display.

Parameters

<i>pstEllipse_</i>	- pointer to the ellipse to draw on the display
--------------------	---

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

14.33.2.5 void GraphicsDriver::Line (DrawLine_t * *pstLine_*) [virtual]

Draw a line to the display using Bresenham's line drawing algorithm.

Parameters

<i>pstLine_</i>	- pointer to the line structure
-----------------	---------------------------------

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

14.33.2.6 void GraphicsDriver::Move (DrawMove_t * *pstMove_*) [virtual]

Move a the contents from one rectangle on screen to another rectangle, specified by the values of the input structure.

Parameters

<i>pstMove_</i>	- object describing the graphics movement operation (framebuffer operations only).
-----------------	--

Definition at line 440 of file [graphics.cpp](#).

14.33.2.7 void GraphicsDriver::Point (DrawPoint_t * *pstPoint_*) [virtual]

Draw a pixel to the display.

Parameters

<i>pstPoint_</i>	- pointer to the struct containing the pixel to draw
------------------	--

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

14.33.2.8 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](#).

14.33.2.9 void GraphicsDriver::Rectangle (DrawRectangle_t * *pstRectangle_*) [virtual]

Draws a rectangle on the display.

Parameters

<i>pstRectangle_</i>	- pointer to the rectangle struct
----------------------	-----------------------------------

Definition at line 133 of file [graphics.cpp](#).

14.33.2.10 void GraphicsDriver::SetWindow (DrawWindow_t * *pstWindow_*)

Set the drawable window of the screen.

Parameters

<i>pstWindow_</i>	- pointer to the window struct defining the drawable area
-------------------	---

Definition at line 1050 of file [graphics.cpp](#).

14.33.2.11 void GraphicsDriver::Stamp (DrawStamp_t * *pstStamp_*) [virtual]

Draws a stamp (a 1-bit bitmap) on the display.

Parameters

<i>pstStamp_</i>	- pointer to the stamp object to draw
------------------	---------------------------------------

Definition at line 401 of file [graphics.cpp](#).

14.33.2.12 void GraphicsDriver::Text (DrawText_t * *pstText_*) [virtual]

Draw a string of text to the display using a bitmap font.

Parameters

<i>pstText_</i>	- pointer to the text object to render
-----------------	--

Definition at line 501 of file [graphics.cpp](#).

14.33.2.13 GraphicsDriver::TextFX (DrawText_t * *pstText_*, TextFX_t * *pstFX_*)

Render a string of text to the display with effects.

Parameters

<i>pstText_</i>	- pointer to the text object to render
<i>pstFX_</i>	- struct defining special text formatting to apply

ToDo - Add rotation

Definition at line 589 of file [graphics.cpp](#).

14.33.2.14 void GraphicsDriver::TriangleFill (DrawPoly_t * *pstPoly_*) [virtual]

Draw a filled triangle to the display.

Parameters

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

Definition at line 823 of file [graphics.cpp](#).

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

Draw a wireframe triangle to the display.

Parameters

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

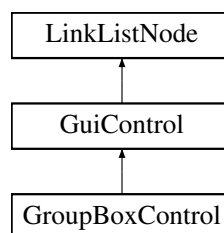
Definition at line 798 of file [graphics.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/graphics.h](#)
- [/home/moslevin/Project/R1/stage/src/graphics.cpp](#)

14.34 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

14.34.1 Detailed Description

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

14.34.2 Member Function Documentation

14.34.2.1 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](#).

14.34.2.2 void [GroupBoxControl::Draw](#) () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.34.2.3 `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](#).

14.34.2.4 `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:

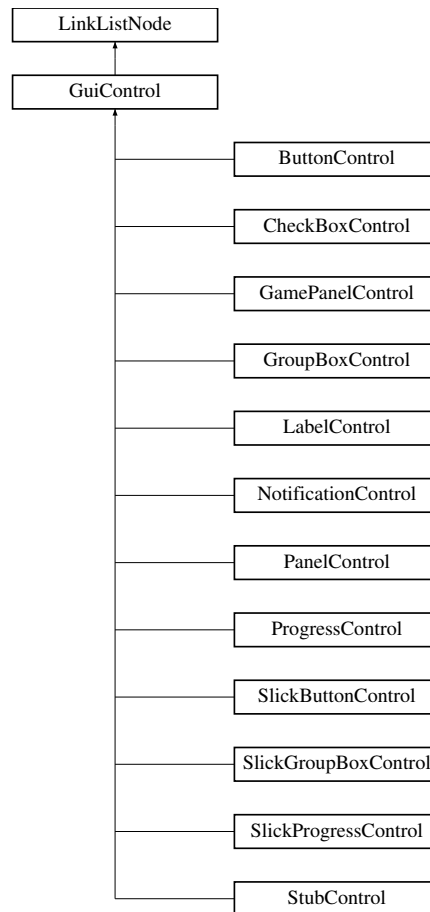
- [/home/moslevin/Project/R1/stage/src/control_groupbox.h](#)
- [/home/moslevin/Project/R1/stage/src/control_groupbox.cpp](#)

14.35 GuiControl Class Reference

GUI Control Base Class.

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

Inheritance diagram for GuiControl:



Public Member Functions

- virtual void **Init** ()=0
Initialize the control - must be called before use.
- virtual void **Draw** ()=0
Redraw the control "cleanly".
- virtual **GuiReturn_t** **ProcessEvent** (**GuiEvent_t** *pstEvent_)=0
Process an event sent to the control.
- void **SetTop** (K_USHORT usTop_)
Set the location of the topmost pixel of the control.
- void **SetLeft** (K_USHORT usLeft_)
Set the location of the leftmost pixel of the control.
- void **SetHeight** (K_USHORT usHeight_)
Set the height of the control (in pixels)
- void **SetWidth** (K_USHORT usWidth_)
Set the width of the control (in pixels)
- void **SetZOrder** (K_UCHAR ucZ_)
Set the Z-order (depth) of the control.
- void **SetControlIndex** (K_UCHAR ucIdx_)
Set the index of the control, used for cycling through focus (ala tab order in VB).
- K_USHORT **GetTop** ()
Return the topmost pixel of the control.
- K_USHORT **GetLeft** ()
Return the leftmost pixel of the control.

- K_USHORT [GetHeight](#) ()
Get the height of the control in pixels.
- K_USHORT [GetWidth](#) ()
Get the width of the control in pixels.
- K_UCHAR [GetZOrder](#) ()
Return the Z-order of the control.
- K_UCHAR [GetControllIndex](#) ()
Return the Control Index of the control.
- K_BOOL [IsStale](#) ()
Return whether or not the control needs to be redrawn or not.
- void [GetControlOffset](#) (K_USHORT *pusX_, K_USHORT *pusY_)
Return the absolute offset of the control within an event surface.
- K_BOOL [IsInFocus](#) ()
Return whether or not the current control has the focus in the window.
- virtual void [Activate](#) (bool bActivate_)=0
Activate or deactivate the current control - used when switching from one active control to another.

Protected Member Functions

- void [SetParentControl](#) (GuiControl *pclParent_)
Set the parent control of this control.
- void [SetParentWindow](#) (GuiWindow *pclWindow_)
Set the parent window of this control.
- GuiControl * [GetParentControl](#) ()
Return the pointer to the control's currently-assigned parent control.
- GuiWindow * [GetParentWindow](#) ()
Get the parent window of this control.
- void [ClearStale](#) ()
Clear the stale flag for this control.
- void [SetStale](#) ()
Signal that the object needs to be redrawn.
- void [SetAcceptFocus](#) (bool bFocus_)
Tell the control whether or not to accept focus.
- bool [AcceptsFocus](#) ()
Returns whether or not this control accepts focus.

Private Attributes

- K_BOOL [m_bStale](#)
true if the control is stale and needs to be redrawn, false otherwise
- K_BOOL [m_bAcceptsFocus](#)
Whether or not the control accepts focus or not.
- K_UCHAR [m_ucZOrder](#)
The Z-Order (depth) of the control.
- K_UCHAR [m_ucControllIndex](#)
Index of the control in the window.
- K_USHORT [m_usTop](#)
Topmost location of the control on the window.
- K_USHORT [m_usLeft](#)
Leftmost location of the control on the window.

- K_USHORT [m_usWidth](#)
Width of the control in pixels.
- K_USHORT [m_usHeight](#)
Height of the control in pixels.
- [GuiControl](#) * [m_pclParentControl](#)
Pointer to the parent control.
- [GuiWindow](#) * [m_pclParentWindow](#)
Pointer to the parent window associated with this control.

Friends

- class **GuiWindow**
- class **GuiEventSurface**

Additional Inherited Members

14.35.1 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 539 of file [gui.h](#).

14.35.2 Member Function Documentation

14.35.2.1 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](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

14.35.2.2 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 742 of file [gui.h](#).

14.35.2.3 void [GuiControl::Draw](#) () [pure virtual]

Redraw the control "cleanly".

Subclass specific.

Implemented in [StubControl](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

14.35.2.4 K_UCHAR GuiControl::GetControlIndex () [inline]

Return the Control Index of the control.

Returns

The control index of the control

Definition at line 649 of file [gui.h](#).

14.35.2.5 void GuiControl::GetControlOffset (K_USHORT * *pusX_*, K_USHORT * *pusY_*)

Return the absolute offset of the control within an event surface.

This function will traverse through all of the object's parents, and their parents, until the root control and root window are identified. The absolute pixel locations of the Topmost (Y) and Leftmost (X) pixels are populated in the

Parameters

<i>pusX_</i>	Pointer to the K_USHORT containing the leftmost pixel
<i>pusY_</i>	Pointer to the K_USHORT containing the topmost pixel

Definition at line 669 of file [gui.cpp](#).

14.35.2.6 K_USHORT GuiControl::GetHeight () [inline]

Get the height of the control in pixels.

Returns

Height of the control in pixels

Definition at line 628 of file [gui.h](#).

14.35.2.7 K_USHORT GuiControl::GetLeft () [inline]

Return the leftmost pixel of the control.

Returns

Leftmost pixel of the control

Definition at line 621 of file [gui.h](#).

14.35.2.8 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 726 of file [gui.h](#).

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

Get the parent window of this control.

Returns

Pointer to the control's window

Definition at line 734 of file [gui.h](#).

14.35.2.10 `K_USHORT GuiControl::GetTop () [inline]`

Return the topmost pixel of the control.

Returns

Topmost pixel of the control

Definition at line 614 of file [gui.h](#).

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

Get the width of the control in pixels.

Returns

Width of the control in pixels

Definition at line 635 of file [gui.h](#).

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

Return the Z-order of the control.

Returns

Z-order of the control

Definition at line 642 of file [gui.h](#).

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

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implemented in [StubControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [LabelControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), [GroupBoxControl](#), [NotificationControl](#), and [SlickGroupBoxControl](#).

14.35.2.14 `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 678 of file [gui.h](#).

14.35.2.15 `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 656 of file [gui.h](#).

14.35.2.16 `GuiReturn_t GuiControl::ProcessEvent (GuiEvent_t * pstEvent_) [pure virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implemented in [StubControl](#), [NotificationControl](#), [LabelControl](#), [ButtonControl](#), [PanelControl](#), [SlickButtonControl](#), [GamePanelControl](#), [GroupBoxControl](#), [ProgressControl](#), [SlickProgressControl](#), [CheckBoxControl](#), and [SlickGroupBoxControl](#).

14.35.2.17 `void GuiControl::SetControllIndex (K_UCHAR ucIdx_) [inline]`

Set the index of the control, used for cycling through focus (ala tab order in VB).

Parameters

<i>ucIdx_</i>	Focus index of the control
---------------	----------------------------

Definition at line 607 of file [gui.h](#).

14.35.2.18 `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 585 of file [gui.h](#).

14.35.2.19 `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 578 of file [gui.h](#).

14.35.2.20 `void GuiControl::SetParentControl (GuiControl * pclParent_) [inline], [protected]`

Set the parent control of this control.

When a control has its parent set, it is considered "nested" within that control. Moving the control will thus result in all of its child controls to become invalidated, thus requiring redraws. The control's object offsets (Top, Bottom, Height, and Width) also become relative to the origin of the parent control.

Parameters

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

Definition at line 707 of file [gui.h](#).

14.35.2.21 void GuiControl::SetParentWindow (GuiWindow * *pciWindow_*) [inline], [protected]

Set the parent window of this control.

All controls within the same window are all associated together, and share events targetted towards a specific window. Event tabbing, focus, and Z-ordering is also shared between controls within a window.

Parameters

<i>pciWindow_</i>	Pointer to the control's parent window.
-------------------	---

Definition at line 718 of file [gui.h](#).

14.35.2.22 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 571 of file [gui.h](#).

14.35.2.23 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 592 of file [gui.h](#).

14.35.2.24 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 599 of file [gui.h](#).

14.35.3 Member Data Documentation

14.35.3.1 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 771 of file [gui.h](#).

14.35.3.2 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 767 of file [gui.h](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)
- [/home/moslevin/Project/R1/stage/src/gui.cpp](#)

14.36 GuiEvent_t Struct Reference

Composite UI event structure.

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

Public Attributes

- K_UCHAR [ucEventType](#)
GuiEventType_t event type.
- K_UCHAR [ucTargetID](#)
Control index that this event is targeted towards.

14.36.1 Detailed Description

Composite UI event structure.

Depending on the event type, can contain either a keyboard, mouse, touch, joystick, timer event, etc.

Definition at line 187 of file [gui.h](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)

14.37 GuiEventSurface Class Reference

GUI Event Surface Object.

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

Public Member Functions

- void [Init](#) ()
Initialize an event surface before use.
- void [AddWindow](#) ([GuiWindow](#) *pclWindow_)
Add a window to the event surface.
- void [RemoveWindow](#) ([GuiWindow](#) *pclWindow_)
Remove a window from the event surface.

- K_BOOL [SendEvent](#) ([GuiEvent_t](#) *pstEvent_)
Send an event to this window surface.
- K_BOOL [ProcessEvent](#) ()
Process an event in the event queue.
- K_UCHAR [GetEventCount](#) ()
Get the count of pending events in the event surface's queue.
- [GuiWindow](#) * [FindWindowByName](#) (const K_CHAR *szName_)
Return a pointer to a window by name, or NULL on failure.
- void [InvalidateRegion](#) (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)
Invalidate a region of the window, specified by the bounding box.

Private Member Functions

- void [CopyEvent](#) ([GuiEvent_t](#) *pstDst_, [GuiEvent_t](#) *pstSrc_)
Copy the contents of one message structure to another.

Private Attributes

- [DoubleLinkedList](#) [m_clWindowList](#)
List of windows managed on this event surface.
- [MessageQueue](#) [m_clMessageQueue](#)
Message queue used to manage window events.

14.37.1 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 453 of file [gui.h](#).

14.37.2 Member Function Documentation

14.37.2.1 void [GuiEventSurface::AddWindow](#) ([GuiWindow](#) * [pclWindow_](#))

Add a window to the event surface.

Parameters

pclWindow_	Pointer to the window object to add to the surface
----------------------------	--

Definition at line 525 of file [gui.cpp](#).

14.37.2.2 void [GuiEventSurface::CopyEvent](#) ([GuiEvent_t](#) * [pstDst_](#), [GuiEvent_t](#) * [pstSrc_](#)) [private]

Copy the contents of one message structure to another.

Parameters

<i>pstDst_</i>	Destination event pointer
<i>pstSrc_</i>	Source event pointer

Definition at line 645 of file [gui.cpp](#).

14.37.2.3 void GuiEventSurface::Init () [inline]

Initialize an event surface before use.

Must be called prior to any other object methods.

Definition at line 460 of file [gui.h](#).

14.37.2.4 void GuiEventSurface::InvalidateRegion (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)

Invalidate a region of the window, specified by the bounding box.

The coordinates specified in the parameters (top and left) refer to absolute display coordinates, and are not relative to coordinates within a window.

Definition at line 658 of file [gui.cpp](#).

14.37.2.5 K_BOOL GuiEventSurface::ProcessEvent ()

Process an event in the event queue.

If no events are pending, the call will block until an event is available.

Definition at line 577 of file [gui.cpp](#).

14.37.2.6 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 533 of file [gui.cpp](#).

14.37.2.7 K_BOOL GuiEventSurface::SendEvent (GuiEvent_t * pstEvent_)

Send an event to this window surface.

The event will be forwarded to all windows managed by this service.

Parameters

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

Returns

true on success, false on failure

Definition at line 541 of file [gui.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)

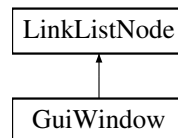
- [/home/moslevin/Project/R1/stage/src/gui.cpp](#)

14.38 GuiWindow Class Reference

Basic Window Class.

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

Inheritance diagram for GuiWindow:



Public Member Functions

- void [Init](#) ()
Initialize the GUI Window object prior to use.
- void [SetDriver](#) ([GraphicsDriver](#) *pclDriver_)
Set the graphics driver to use for rendering controls on the window.
- [GraphicsDriver](#) * [GetDriver](#) ()
Set the graphics driver to use for rendering controls on the window.
- void [AddControl](#) ([GuiControl](#) *pclControl_, [GuiControl](#) *pclParent_)
Assign a GUI Control to this window object.
- void [RemoveControl](#) ([GuiControl](#) *pclControl_)
Removes a previously-added control from the Window.
- K_UCHAR [GetMaxZOrder](#) ()
Returns the highest Z-Order of all controls attached to this window.
- void [Redraw](#) (K_BOOL bRedrawAll_)
Redraw objects in the window.
- void [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to this window.
- void [SetFocus](#) ([GuiControl](#) *pclControl_)
Set the control used to accept "focus" events.
- K_BOOL [IsInFocus](#) ([GuiControl](#) *pclControl_)
Return whether or not the selected control is in focus or not.
- void [SetTop](#) (K_USHORT usTop_)
Set the location of the topmost pixel of the window.
- void [SetLeft](#) (K_USHORT usLeft_)
Set the location of the leftmost pixel of the window.
- void [SetHeight](#) (K_USHORT usHeight_)
Set the height of the window (in pixels)
- void [SetWidth](#) (K_USHORT usWidth_)
Set the width of the window (in pixels)
- K_USHORT [GetTop](#) ()
Return the topmost pixel of the window.
- K_USHORT [GetLeft](#) ()
Return the leftmost pixel of the window.
- K_USHORT [GetHeight](#) ()

- Get the height of the window in pixels.*

 - K_USHORT [GetWidth](#) ()

Get the width of the window in pixels.
- K_UCHAR [GetZOrder](#) ()

Get the Z-order of the window on the event surface.
- void [SetZOrder](#) (K_UCHAR ucZ_)

Set the Z-order of the window on the event surface.
- void [CycleFocus](#) (bool bForward_)

Cycle the focus to the next active control in the window.
- void [SetName](#) (const K_CHAR *szName_)

Set the name for this window.
- const K_CHAR * [GetName](#) ()

Return the name of this window.
- void [InvalidateRegion](#) (K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT usHeight_)

Invalidate a region of the window, specified by the bounding box.

Private Attributes

- K_USHORT [m_usTop](#)

Topmost pixel of the window on the event surface.
- K_USHORT [m_usLeft](#)

Leftmost pixel of the window on the event surface.
- K_USHORT [m_usHeight](#)

Height of the window in pixels.
- K_USHORT [m_usWidth](#)

Width of the window in pixels.
- K_UCHAR [m_ucZ](#)

Z-order of the window on the event surface.
- const K_CHAR * [m_szName](#)

Name applied to this window.
- [DoubleLinkedList](#) [m_clControlList](#)

List of controls managed by this window.
- [GuiControl](#) * [m_pclInFocus](#)

Pointer to the control in event focus.
- K_UCHAR [m_ucControlCount](#)

Number of controls in this window.
- [GraphicsDriver](#) * [m_pclDriver](#)

Graphics driver for this window.

Additional Inherited Members

14.38.1 Detailed Description

Basic Window Class.

A Window is loosely defined as a container of controls, all sharing a coordinate reference coordinate frame. Events are managed on a per-window basis, and each window is isolated from eachother.

Definition at line 223 of file [gui.h](#).

14.38.2 Member Function Documentation

14.38.2.1 GuiWindow::AddControl (**GuiControl** * *pclControl_*, **GuiControl** * *pclParent_*)

Assign a GUI Control to this window object.

Adding an object to a window ensures that the object will be drawn on the specific window surface, and ensures that events directed to this window will be forwarded to the controls appropriately.

Parameters

<i>pclControl_</i>	Pointer to the control object to add
<i>pclParent_</i>	Pointer to the control's "parent" object (or NULL)

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

14.38.2.2 void GuiWindow::CycleFocus (**bool** *bForward_*)

Cycle the focus to the next active control in the window.

Parameters

<i>bForward_</i>	- Cycle to the next control when true, previous control when false
------------------	--

Definition at line 395 of file [gui.cpp](#).

14.38.2.3 **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 253 of file [gui.h](#).

14.38.2.4 **K_USHORT** GuiWindow::GetHeight () [inline]

Get the height of the window in pixels.

Returns

Height of the window in pixels

Definition at line 380 of file [gui.h](#).

14.38.2.5 **K_USHORT** GuiWindow::GetLeft () [inline]

Return the leftmost pixel of the window.

Returns

Leftmost pixel of the window

Definition at line 373 of file [gui.h](#).

14.38.2.6 K_UCHAR GuiWindow::GetMaxZOrder ()

Returns the highest Z-Order of all controls attached to this window.

Returns

The highest Z-Order used by controls in this window

Definition at line 61 of file [gui.cpp](#).

14.38.2.7 K_USHORT GuiWindow::GetTop () [inline]

Return the topmost pixel of the window.

Returns

Topmost pixel of the window

Definition at line 366 of file [gui.h](#).

14.38.2.8 K_USHORT GuiWindow::GetWidth () [inline]

Get the width of the window in pixels.

Returns

Width of the window in pixels

Definition at line 387 of file [gui.h](#).

14.38.2.9 void GuiWindow::Init () [inline]

Initialize the GUI Window object prior to use.

Must be called before calling other methods on this object

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

14.38.2.10 void GuiWindow::InvalidateRegion (K_USHORT *usLeft_*, K_USHORT *usTop_*, K_USHORT *usWidth_*, K_USHORT *usHeight_*)

Invalidate a region of the window, specified by the bounding box.

The coordinates specified in the parameters (top and left) refer to absolute display coordinates, and are not relative to coordinates within a window.

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

14.38.2.11 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 324 of file [gui.h](#).

14.38.2.12 void GuiWindow::ProcessEvent (GuiEvent_t * pstEvent_)

Process an event sent to this window.

This method handles all of the plumbing required to target the event towards specific controls, or all controls in the window depending on the event payload.

Definition at line 245 of file [gui.cpp](#).

14.38.2.13 void GuiWindow::Redraw (K_BOOL bRedrawAll_)

Redraw objects in the window.

Typically, only the affected controls will need to be redrawn, but in some cases (such as window initialization), the entire window will need to be redrawn cleanly. This behavior is defined by the value of the bRedrawAll_ parameter.

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

14.38.2.14 GuiWindow::RemoveControl (GuiControl * pclControl_)

Removes a previously-added control from the Window.

Parameters

<i>pclControl_</i>	Pointer to the control object to remove
--------------------	---

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

14.38.2.15 void GuiWindow::SetDriver (GraphicsDriver * pclDriver_) [inline]

Set the graphics driver to use for rendering controls on the window.

Parameters

<i>pclDriver_</i>	Pointer to the graphics driver
-------------------	--------------------------------

Definition at line 245 of file [gui.h](#).

14.38.2.16 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 387 of file [gui.cpp](#).

14.38.2.17 void GuiWindow::SetHeight (K_USHORT *usHeight_*) [inline]

Set the height of the window (in pixels)

Parameters

<i>usHeight_</i>	Height of the window in pixels
------------------	--------------------------------

Definition at line 352 of file [gui.h](#).

14.38.2.18 void GuiWindow::SetLeft (K_USHORT *usLeft_*) [inline]

Set the location of the leftmost pixel of the window.

Parameters

<i>usLeft_</i>	Leftmost pixel of the window
----------------	------------------------------

Definition at line 345 of file [gui.h](#).

14.38.2.19 void GuiWindow::SetTop (K_USHORT *usTop_*) [inline]

Set the location of the topmost pixel of the window.

Parameters

<i>usTop_</i>	Topmost pixel of the window
---------------	-----------------------------

Definition at line 338 of file [gui.h](#).

14.38.2.20 void GuiWindow::SetWidth (K_USHORT *usWidth_*) [inline]

Set the width of the window (in pixels)

Parameters

<i>usWidth_</i>	Width of the window in pixels
-----------------	-------------------------------

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

14.38.3 Member Data Documentation

14.38.3.1 GraphicsDriver* GuiWindow::m_pclDriver [private]

Graphics driver for this window.

Definition at line 437 of file [gui.h](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)
- [/home/moslevin/Project/R1/stage/src/gui.cpp](#)

14.39 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**

14.39.1 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/Project/R1/stage/src/fixed_heap.h](#)

14.40 JoystickEvent_t Struct Reference

Joystick UI event structure.

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

Public Attributes

14.40.1 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/Project/R1/stage/src/gui.h](#)

14.41 Kernel Class Reference

Class that encapsulates all of the kernel startup functions.

```
#include <kernel.h>
```

Static Public Member Functions

- static void [Init](#) (void)
Kernel Initialization Function, call before any other OS function.

- static void [Start](#) (void)
Start the kernel; function never returns.
- static bool [IsStarted](#) ()
IsStarted.
- static void [SetPanic](#) (panic_func_t pfPanic_)
SetPanic Set a function to be called when a kernel panic occurs, giving the user to determine the behavior when a catastrophic failure is observed.
- static bool [IsPanic](#) ()
IsPanic Returns whether or not the kernel is in a panic state.
- static void [Panic](#) (K_USHORT usCause_)
Panic Cause the kernel to enter its panic state.

Static Private Attributes

- static bool [m_bIsStarted](#)
true if kernel is running, false otherwise
- static bool [m_bIsPanic](#)
true if kernel is in panic state, false otherwise
- static panic_func_t [m_pfPanic](#)
user-set panic function

14.41.1 Detailed Description

Class that encapsulates all of the kernel startup functions.

Definition at line 42 of file [kernel.h](#).

14.41.2 Member Function Documentation

14.41.2.1 `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 47 of file [kernel.cpp](#).

14.41.2.2 `static bool Kernel::IsPanic () [inline],[static]`

[IsPanic](#) Returns whether or not the kernel is in a panic state.

Returns

Whether or not the kernel is in a panic state

Definition at line 89 of file [kernel.h](#).

14.41.2.3 `static bool Kernel::IsStarted () [inline],[static]`

[IsStarted](#).

Returns

Whether or not the kernel has started - true = running, false = not started

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

14.41.2.4 void Kernel::Panic (K_USHORT *usCause_*) [static]

Panic Cause the kernel to enter its panic state.

Parameters

<i>usCause_</i>	Reason for the kernel panic
-----------------	-----------------------------

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

14.41.2.5 static void Kernel::SetPanic (panic_func_t *pfPanic_*) [inline],[static]

SetPanic Set a function to be called when a kernel panic occurs, giving the user to determine the behavior when a catastrophic failure is observed.

Parameters

<i>pfPanic_</i>	Panic function pointer
-----------------	------------------------

Definition at line 83 of file [kernel.h](#).

14.41.2.6 Kernel::Start (void) [static]

Start the kernel; function never returns.

Start the operating system kernel - the current execution context is cancelled, all kernel services are started, and the processor resumes execution at the entrypoint for the highest-priority thread.

You must have at least one thread added to the kernel before calling this function, otherwise the behavior is undefined.

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

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

- [/home/moslevin/Project/R1/stage/src/kernel.h](#)
- [/home/moslevin/Project/R1/stage/src/kernel.cpp](#)

14.42 Kernel_Aware Class Reference

The [Kernel_Aware](#) class.

```
#include <kernel_aware.h>
```

Static Public Member Functions

- static void [Profile_Init](#) (const K_CHAR *szStr_)
Profile_Init.
- static void [Profile_Start](#) (void)
Profile_Start.
- static void [Profile_Stop](#) (void)
Profile_Stop.
- static void [Profile_Report](#) (void)
Profile_Report.
- static void [Exit_Simulator](#) (void)
Exit_Simulator.
- static void [Print](#) (const K_CHAR *szStr_)

Print.

- static void [Trace](#) (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_)

Trace.

- static void [Trace](#) (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_)

Trace.

- static void [Trace](#) (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_, K_USHORT usArg2_)

Trace.

Static Private Member Functions

- static void [Trace_i](#) (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_, K_USHORT usArg2_, KernelAwareCommand_t eCmd_)

Trace_i.

14.42.1 Detailed Description

The [Kernel_Aware](#) class.

This class contains functions that are used to trigger kernel-aware functionality within a supported simulation environment (i.e. fIAVR).

These static methods operate on a singleton set of global variables, which are monitored for changes from within the simulator. The simulator hooks into these variables by looking for the correctly-named symbols in an elf-formatted binary being run and registering callbacks that are called whenever the variables are changed. On each change of the command variable, the kernel-aware data is analyzed and interpreted appropriately.

If these methods are run in an unsupported simulator or on actual hardware the commands generally have no effect (except for the exit-on-reset command, which will result in a jump-to-0 reset).

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

14.42.2 Member Function Documentation

14.42.2.1 void Kernel_Aware::Exit_Simulator (void) [static]

Exit_Simulator.

Instruct the kernel-aware simulator to terminate (destroying the virtual CPU).

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

14.42.2.2 void Kernel_Aware::Print (const K_CHAR * szStr) [static]

Print.

Instruct the kernel-aware simulator to print a char string

Parameters

<i>szStr_</i>	
---------------	--

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

14.42.2.3 void Kernel_Aware::Profile_Init (const K_CHAR * szStr) [static]

Profile_Init.

Initializes the kernel-aware profiler. This function instructs the kernel-aware simulator to reset its accounting variables, and prepare to start counting profiling data tagged to the given string. How this is handled is the responsibility of the simulator.

Parameters

<i>szStr_</i>	String to use as a tag for the profiling session.
---------------	---

Definition at line 58 of file [kernel_aware.cpp](#).

14.42.2.4 void Kernel_Aware::Profile_Report (void) [static]

Profile_Report.

Instruct the kernel-aware simulator to print a report for its current profiling data.

Definition at line 79 of file [kernel_aware.cpp](#).

14.42.2.5 void Kernel_Aware::Profile_Start (void) [static]

Profile_Start.

Instruct the kernel-aware simulator to begin counting cycles towards the current profiling counter.

Definition at line 67 of file [kernel_aware.cpp](#).

14.42.2.6 void Kernel_Aware::Profile_Stop (void) [static]

Profile_Stop.

Instruct the kernel-aware simulator to end counting cycles relative to the current profiling counter's iteration.

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

14.42.2.7 void Kernel_Aware::Trace (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_) [static]

Trace.

Insert a kernel trace statement into the kernel-aware simulator's debug data stream.

Parameters

<i>usFile_</i>	16-bit code representing the file
<i>usLine_</i>	16-bit code representing the line in the file
<i>usCode_</i>	16-bit data code, which indicates the line's format.

Definition at line 92 of file [kernel_aware.cpp](#).

14.42.2.8 void Kernel_Aware::Trace (K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_) [static]

Trace.

Insert a kernel trace statement into the kernel-aware simulator's debug data stream.

Parameters

<i>usFile_</i>	16-bit code representing the file
----------------	-----------------------------------

<i>usLine_</i>	16-bit code representing the line in the file
<i>usCode_</i>	16-bit data code, which indicates the line's format
<i>usArg1_</i>	16-bit argument to the format string.

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

```
14.42.2.9 void Kernel_Aware::Trace ( K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_,
K_USHORT usArg2_ ) [static]
```

Trace.

Insert a kernel trace statement into the kernel-aware simulator's debug data stream.

Parameters

<i>usFile_</i>	16-bit code representing the file
<i>usLine_</i>	16-bit code representing the line in the file
<i>usCode_</i>	16-bit data code, which indicates the line's format
<i>usArg1_</i>	16-bit argument to the format string.
<i>usArg2_</i>	16-bit argument to the format string.

Definition at line 109 of file [kernel_aware.cpp](#).

```
14.42.2.10 void Kernel_Aware::Trace_i ( K_USHORT usFile_, K_USHORT usLine_, K_USHORT usCode_, K_USHORT usArg1_,
K_USHORT usArg2_, KernelAwareCommand_t eCmd_ ) [static],[private]
```

Trace_i.

Private function by which the class's [Trace\(\)](#) methods are reflected, which allows us to realize a modest code saving.

Parameters

<i>usFile_</i>	16-bit code representing the file
<i>usLine_</i>	16-bit code representing the line in the file
<i>usCode_</i>	16-bit data code, which indicates the line's format
<i>usArg1_</i>	16-bit argument to the format string.
<i>usArg2_</i>	16-bit argument to the format string.
<i>eCmd_</i>	Code indicating the number of arguments to emit.

Definition at line 119 of file [kernel_aware.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/kernel_aware.h](#)
- [/home/moslevin/Project/R1/stage/src/kernel_aware.cpp](#)

14.43 KernelAwareData_t Union Reference

Public Attributes

- volatile K_USHORT **ausBuffer** [5]

14.43.1 Detailed Description

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

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

- [/home/moslevin/Project/R1/stage/src/kernel_aware.cpp](#)

14.44 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](#) (bool bEnable_)
Restore the state of the SWI to the value specified.

14.44.1 Detailed Description

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

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

14.44.2 Member Function Documentation

14.44.2.1 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](#).

14.44.2.2 void KernelSWI::RI (bool bEnable_) [static]

Restore the state of the SWI to the value specified.

Parameters

<code>bEnable_</code>	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/Project/R1/stage/src/kernelswi.h](#)
- [/home/moslevin/Project/R1/stage/src/kernelswi.cpp](#)

14.45 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](#) (bool bEnable_)
Restore the state of the kernel timer's expiry interrupt.
- static void [EI](#) (void)
Enable the kernel timer's expiry interrupt.
- static K_ULONG [SubtractExpiry](#) (K_ULONG ulInterval_)
Subtract the specified number of ticks from the timer's expiry count register.
- static K_ULONG [TimeToExpiry](#) (void)
Returns the number of ticks remaining before the next timer expiry.
- static K_ULONG [SetExpiry](#) (K_ULONG ulInterval_)
Resets the kernel timer's expiry interval to the specified value.
- static K_ULONG [GetOvertime](#) (void)
Return the number of ticks that have elapsed since the last expiry.
- static void [ClearExpiry](#) (void)
Clear the hardware timer expiry register.

Static Private Member Functions

- static K_USHORT [Read](#) (void)
Safely read the current value in the timer register.

14.45.1 Detailed Description

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

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

14.45.2 Member Function Documentation

14.45.2.1 K_ULONG KernelTimer::GetOvertime (void) [static]

Return the number of ticks that have elapsed since the last expiry.

Returns

Number of ticks that have elapsed after last timer expiration

Definition at line 115 of file [kerneltimer.cpp](#).

14.45.2.2 K_USHORT KernelTimer::Read (void) [static],[private]

Safely read the current value in the timer register.

Returns

Value held in the timer register

Definition at line 66 of file [kerneltimer.cpp](#).

14.45.2.3 void KernelTimer::RI (bool bEnable_) [static]

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

Parameters

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

Definition at line 168 of file [kerneltimer.cpp](#).

14.45.2.4 K_ULONG KernelTimer::SetExpiry (K_ULONG ulInterval_) [static]

Resets the kernel timer's expiry interval to the specified value.

Parameters

<i>ulInterval_</i>	Desired interval in ticks to set the timer for
--------------------	--

Returns

Actual number of ticks set (may be less than desired)

Definition at line 121 of file [kerneltimer.cpp](#).

14.45.2.5 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

<code>ulInterval_</code>	Time (in HW-specific) ticks to subtract
--------------------------	---

Returns

Value in ticks stored in the timer's expiry register

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

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

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

Returns

Time before next expiry in platform-specific ticks

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

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

- [/home/moslevin/Project/R1/stage/src/kerneltimer.h](#)
- [/home/moslevin/Project/R1/stage/src/kerneltimer.cpp](#)

14.46 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

14.46.1 Detailed Description

Keyboard UI event structure definition.

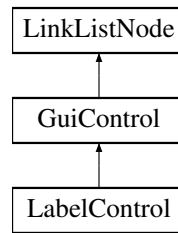
Definition at line 80 of file [gui.h](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)

14.47 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

14.47.1 Detailed Description

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

14.47.2 Member Function Documentation

14.47.2.1 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](#).

14.47.2.2 `void LabelControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.47.2.3 `virtual void LabelControl::Init () [inline],[virtual]`

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.47.2.4 `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:

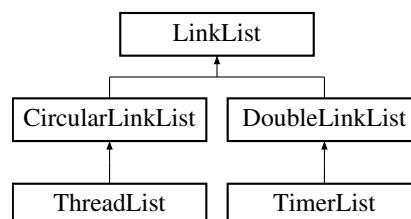
- [/home/moslevin/Project/R1/stage/src/control_label.h](#)
- [/home/moslevin/Project/R1/stage/src/control_label.cpp](#)

14.48 LinkedList Class Reference

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

`#include <ll.h>`

Inheritance diagram for LinkedList:



Public Member Functions

- `void Init ()`
Clear the linked list.
- `virtual void Add (LinkedListNode *node_)=0`

- Add the linked list node to this linked list.*
 • virtual void [Remove](#) ([LinkedListNode](#) **node_*)=0
Add the linked list node to this linked list.
- [LinkedListNode](#) * [GetHead](#) ()
Get the head node in the linked list.
- [LinkedListNode](#) * [GetTail](#) ()
Get the tail node of the linked list.

Protected Attributes

- [LinkedListNode](#) * [m_pstHead](#)
Pointer to the head node in the list.
- [LinkedListNode](#) * [m_pstTail](#)
Pointer to the tail node in the list.

14.48.1 Detailed Description

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

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

14.48.2 Member Function Documentation

14.48.2.1 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](#).

14.48.2.2 [LinkedListNode](#) * [LinkedList::GetHead](#) () [inline]

Get the head node in the linked list.

Returns

Pointer to the head node in the list

Definition at line 149 of file [ll.h](#).

14.48.2.3 [LinkedListNode](#) * [LinkedList::GetTail](#) () [inline]

Get the tail node of the linked list.

Returns

Pointer to the tail node in the list

Definition at line 158 of file [ll.h](#).

14.48.2.4 void [LinkedList::Remove](#) ([LinkedListNode](#) * *node_*) [pure virtual]

Add the linked list node to this linked list.

Parameters

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

Implemented in [CircularLinkedList](#), [DoubleLinkedList](#), and [ThreadList](#).

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

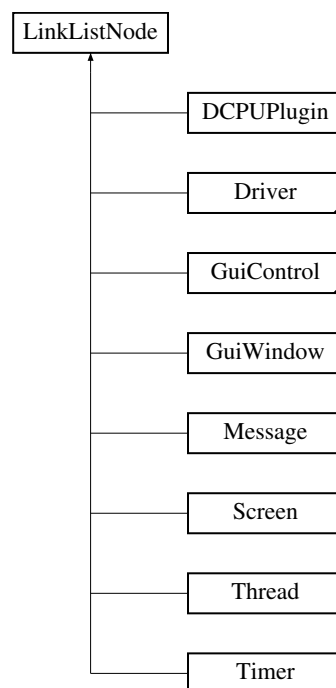
- [/home/moslevin/Project/R1/stage/src/ll.h](#)

14.49 LinkedListNode Class Reference

Basic linked-list node data structure.

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

Inheritance diagram for LinkedListNode:



Public Member Functions

- [LinkedListNode](#) * [GetNext](#) (void)
Returns a pointer to the next node in the list.
- [LinkedListNode](#) * [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

- [LinkedListNode](#) * [next](#)

Pointer to the next node in the list.

- [LinkedListNode](#) * prev

Pointer to the previous node in the list.

Friends

- class **LinkedList**
- class **DoubleLinkedList**
- class **CircularLinkedList**

14.49.1 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 68 of file [ll.h](#).

14.49.2 Member Function Documentation

14.49.2.1 [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 92 of file [ll.h](#).

14.49.2.2 [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 101 of file [ll.h](#).

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

- [/home/moslevin/Project/R1/stage/src/ll.h](#)
- [/home/moslevin/Project/R1/stage/src/ll.cpp](#)

14.50 MemUtil Class Reference

String and Memory manipulation class.

```
#include <memutil.h>
```

Static Public Member Functions

- static void [DecimalToHex](#) (K_UCHAR ucData_, char *szText_)
Convert an 8-bit unsigned binary value as a hexadecimal string.
- static void **DecimalToHex** (K_USHORT usData_, char *szText_)
- static void **DecimalToHex** (K_ULONG ulData_, char *szText_)
- static void [DecimalToString](#) (K_UCHAR ucData_, char *szText_)
Convert an 8-bit unsigned binary value as a decimal string.
- static void **DecimalToString** (K_USHORT usData_, char *szText_)
- static void **DecimalToString** (K_ULONG ulData_, char *szText_)
- static K_UCHAR [Checksum8](#) (const void *pvSrc_, K_USHORT usLen_)
Compute the 8-bit additive checksum of a memory buffer.
- static K_USHORT [Checksum16](#) (const void *pvSrc_, K_USHORT usLen_)
Compute the 16-bit additive checksum of a memory buffer.
- static K_USHORT [StringLength](#) (const char *szStr_)
Compute the length of a string in bytes.
- static bool [CompareStrings](#) (const char *szStr1_, const char *szStr2_)
Compare the contents of two zero-terminated string buffers to eachother.
- static void [CopyMemory](#) (void *pvDst_, const void *pvSrc_, K_USHORT usLen_)
Copy one buffer in memory into another.
- static void [CopyString](#) (char *szDst_, const char *szSrc_)
Copy a string from one buffer into another.
- static K_SHORT [StringSearch](#) (const char *szBuffer_, const char *szPattern_)
Search for the presence of one string as a substring within another.
- static bool [CompareMemory](#) (const void *pvMem1_, const void *pvMem2_, K_USHORT usLen_)
Compare the contents of two memory buffers to eachother.
- static void [SetMemory](#) (void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_)
Initialize a buffer of memory to a specified 8-bit pattern.
- static K_UCHAR [Tokenize](#) (const char *szBuffer_, [Token_t](#) *pastTokens_, K_UCHAR ucMaxTokens_)
Tokenize Function to tokenize a string based on a space delimiter.

14.50.1 Detailed Description

String and Memory manipulation class.

Utility method class implementing common memory and string manipulation functions, without relying on an external standard library implementation which might not be available on some toolchains, may be closed source, or may not be thread-safe.

Definition at line 47 of file [memutil.h](#).

14.50.2 Member Function Documentation

14.50.2.1 static K_USHORT MemUtil::Checksum16 (const void * *pvSrc_*, K_USHORT *usLen_*) [static]

Compute the 16-bit additive checksum of a memory buffer.

Parameters

<i>pvSrc_</i>	Memory buffer to compute a 16-bit checksum of.
---------------	--

<i>usLen_</i>	Length of the buffer in bytes.
---------------	--------------------------------

Returns

16-bit checksum of the memory block.

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

14.50.2.2 `static K_USHORT MemUtil::Checksum8 (const void * pvSrc_, K_USHORT usLen_) [static]`

Compute the 8-bit additive checksum of a memory buffer.

Parameters

<i>pvSrc_</i>	Memory buffer to compute a 8-bit checksum of.
<i>usLen_</i>	Length of the buffer in bytes.

Returns

8-bit checksum of the memory block.

Definition at line 199 of file [memutil.cpp](#).

14.50.2.3 `static bool MemUtil::CompareMemory (const void * pvMem1_, const void * pvMem2_, K_USHORT usLen_) [static]`

Compare the contents of two memory buffers to eachother.

Parameters

<i>pvMem1_</i>	First buffer to compare
<i>pvMem2_</i>	Second buffer to compare
<i>usLen_</i>	Length of buffer (in bytes) to compare

Returns

true if the buffers match, false if they do not.

Definition at line 342 of file [memutil.cpp](#).

14.50.2.4 `static bool MemUtil::CompareStrings (const char * szStr1_, const char * szStr2_) [static]`

Compare the contents of two zero-terminated string buffers to eachother.

Parameters

<i>szStr1_</i>	First string to compare
<i>szStr2_</i>	Second string to compare

Returns

true if strings match, false otherwise.

Definition at line 247 of file [memutil.cpp](#).

14.50.2.5 `static void MemUtil::CopyMemory (void * pvDst_, const void * pvSrc_, K_USHORT usLen_) [static]`

Copy one buffer in memory into another.

Parameters

<i>pvDst_</i>	Pointer to the destination buffer
<i>pvSrc_</i>	Pointer to the source buffer
<i>usLen_</i>	Number of bytes to copy from source to destination

Definition at line 273 of file [memutil.cpp](#).

14.50.2.6 static void MemUtil::CopyString (char * *szDst_*, const char * *szSrc_*) [static]

Copy a string from one buffer into another.

Parameters

<i>szDst_</i>	Pointer to the buffer to copy into
<i>szSrc_</i>	Pointer to the buffer to copy data from

Definition at line 290 of file [memutil.cpp](#).

14.50.2.7 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](#).

14.50.2.8 static void MemUtil::DecimalToString (K_UCHAR *ucData_*, char * *szText_*) [static]

Convert an 8-bit unsigned binary value as a decimal string.

Parameters

<i>ucData_</i>	Value to convert into a string
<i>szText_</i>	Destination string buffer (4 bytes minimum)

Definition at line 122 of file [memutil.cpp](#).

14.50.2.9 static void MemUtil::SetMemory (void * *pvDst_*, K_UCHAR *ucVal_*, K_USHORT *usLen_*) [static]

Initialize a buffer of memory to a specified 8-bit pattern.

Parameters

<i>pvDst_</i>	Destination buffer to set
<i>ucVal_</i>	8-bit pattern to initialize each byte of destination with
<i>usLen_</i>	Length of the buffer (in bytes) to initialize

Definition at line 363 of file [memutil.cpp](#).

14.50.2.10 static K_USHORT MemUtil::StringLength (const char * *szStr_*) [static]

Compute the length of a string in bytes.

Parameters

<i>szStr_</i>	Pointer to the zero-terminated string to calculate the length of
---------------	--

Returns

length of the string (in bytes), not including the 0-terminator.

Definition at line 232 of file [memutil.cpp](#).

14.50.2.11 static K_SHORT MemUtil::StringSearch (const char * *szBuffer_*, const char * *szPattern_*) [static]

Search for the presence of one string as a substring within another.

Parameters

<i>szBuffer_</i>	Buffer to search for pattern within
<i>szPattern_</i>	Pattern to search for in the buffer

Returns

Index of the first instance of the pattern in the buffer, or -1 on no match.

Definition at line 307 of file [memutil.cpp](#).

14.50.2.12 K_UCHAR MemUtil::Tokenize (const char * *szBuffer_*, Token_t * *pastTokens_*, K_UCHAR *ucMaxTokens_*)
[static]

Tokenize Function to tokenize a string based on a space delimiter.

This is a non-destructive function, which populates a [Token_t](#) descriptor array.

Parameters

<i>szBuffer_</i>	String to tokenize
<i>pastTokens_</i>	Pointer to the array of token descriptors
<i>ucMaxTokens_</i>	Maximum number of tokens to parse (i.e. size of <i>pastTokens_</i>)

Returns

Count of tokens parsed

Definition at line 376 of file [memutil.cpp](#).

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

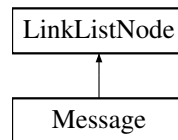
- [/home/moslevin/Project/R1/stage/src/memutil.h](#)
- [/home/moslevin/Project/R1/stage/src/memutil.cpp](#)

14.51 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

14.51.1 Detailed Description

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

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

14.51.2 Member Function Documentation

14.51.2.1 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](#).

14.51.2.2 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](#).

14.51.2.3 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](#).

14.51.2.4 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/Project/R1/stage/src/message.h](#)

14.52 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 Member Functions

- [Message](#) * [Receive_i](#) (K_ULONG ulTimeWaitMS_)

Private Attributes

- [Semaphore](#) [m_clSemaphore](#)
Counting semaphore used to manage thread blocking.
- [DoubleLinkedList](#) [m_clLinkList](#)
List object used to store messages.

14.52.1 Detailed Description

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

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

14.52.2 Member Function Documentation

14.52.2.1 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 156 of file [message.cpp](#).

14.52.2.2 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 92 of file [message.cpp](#).

14.52.2.3 Message * MessageQueue::Receive (K_ULONG ulWaitTimeMS_)

Receive a message from the message queue.

If the message queue is empty, the thread will block until a message is available for the duration specified. If no message arrives within that duration, the call will return with NULL.

Parameters

<i>ulWaitTimeMS_</i>	The amount of time in ms to wait for a message before timing out and unblocking the waiting thread.
----------------------	---

Returns

Pointer to a message object at the head of the queue or NULL on timeout.

Definition at line 103 of file [message.cpp](#).

14.52.2.4 void MessageQueue::Send (Message * pclSrc_)

Send a message object into this message queue.

Will un-block the first waiting thread blocked on this queue if that occurs.

Parameters

<i>pcSrc_</i>	Pointer to the message object to add to the queue
---------------	---

Definition at line 140 of file [message.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/message.h](#)
- [/home/moslevin/Project/R1/stage/src/message.cpp](#)

14.53 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)

14.53.1 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:

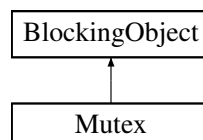
- [/home/moslevin/Project/R1/stage/src/gui.h](#)

14.54 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 [Release](#) ()
Release the mutex.

Private Member Functions

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

Private Attributes

- K_UCHAR [m_ucRecurse](#)
The recursive lock-count when a mutex is claimed multiple times by the same owner.
- K_UCHAR [m_bReady](#)
State of the mutex - true = ready, false = claimed.
- K_UCHAR [m_ucMaxPri](#)
Maximum priority of thread in queue, used for priority inheritance.
- [Thread](#) * [m_pclOwner](#)
Pointer to the thread that owns the mutex (when claimed)

Additional Inherited Members

14.54.1 Detailed Description

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

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

14.54.2 Member Function Documentation

14.54.2.1 void Mutex::Claim (void)

Claim the mutex.

When the mutex is claimed, no other thread can claim a region protected by the object.

Definition at line 199 of file [mutex.cpp](#).

14.54.2.2 bool Mutex::Claim (K_ULONG ulWaitTimeMS_)

Parameters

ulWaitTimeMS_

Returns

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

Definition at line 210 of file [mutex.cpp](#).

14.54.2.3 void Mutex::Release ()

Release the mutex.

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

Definition at line 217 of file [mutex.cpp](#).

14.54.2.4 void Mutex::WakeMe (Thread * *pciOwner_*)

Wake a thread blocked on the mutex.

This is an internal function used for implementing timed mutexes relying on timer callbacks. Since these do not have access to the private data of the mutex and its base classes, we have to wrap this as a public method - do not use this for any other purposes.

Parameters

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

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

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

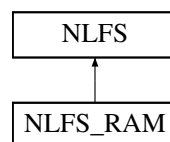
- [/home/moslevin/Project/R1/stage/src/mutex.h](#)
- [/home/moslevin/Project/R1/stage/src/mutex.cpp](#)

14.55 NLFS Class Reference

Nice Little File System class.

```
#include <nlfs.h>
```

Inheritance diagram for NLFS:



Public Member Functions

- void [Format](#) ([NLFS_Host_t](#) *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_, K_USHORT usDataBlockSize_)
Format/Create a new filesystem with the configuration specified in the parameters.
- void [Mount](#) ([NLFS_Host_t](#) *puHost_)
Re-mount a previously-created filesystem using this FS object.
- K_USHORT [Create_File](#) (const K_CHAR *szPath_)
Create_File creates a new file object at the specified path.
- K_USHORT [Create_Dir](#) (const K_CHAR *szPath_)
Create_Dir creates a new directory at the specified path.
- K_USHORT [Delete_File](#) (const K_CHAR *szPath_)
Delete_File Removes a file from disk.
- K_USHORT [Delete_Folder](#) (const K_CHAR *szPath_)
Delete_Folder Remove a folder from disk.

- void [Cleanup_Node_Links](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Cleanup_Node_Links Remove the links between the given node and its parent/peer nodes.
- K_USHORT [Find_Parent_Dir](#) (const K_CHAR *szPath_)
Find_Parent_Dir returns the directory under which the specified file object lives.
- K_USHORT [Find_File](#) (const K_CHAR *szPath_)
Find_File returns the file node ID of the object at a given path.
- void [Print](#) (void)
Print displays a summary of files in the filesystem.
- K_ULONG [GetBlockSize](#) (void)
GetBlockSize retrieves the data block size for the filesystem.
- K_ULONG [GetNumBlocks](#) (void)
GetNumBlocks retrieves the number of data blocks in the filesystem.
- K_ULONG [GetNumBlocksFree](#) (void)
GetNumBlocksFree retrieves the number of free data blocks in the filesystem.
- K_ULONG [GetNumFiles](#) (void)
GetNumFiles retrieves the maximum number of files in the filesystem.
- K_USHORT [GetNumFilesFree](#) (void)
GetNumFilesFree retrieves the number of free blocks in the filesystem.
- K_USHORT [GetFirstChild](#) (K_USHORT usNode_)
GetFirstChild Return the first child node for a node representing a directory.
- K_USHORT [GetNextPeer](#) (K_USHORT usNode_)
GetNextPeer Return the Node ID of a File/Directory's next peer.
- K_BOOL [GetStat](#) (K_USHORT usNode_, [NLFS_File_Stat_t](#) *pstStat_)
GetStat Get the status of a file on-disk.

Protected Member Functions

- K_CHAR [Find_Last_Slash](#) (const K_CHAR *szPath_)
Find_Last_Slash Finds the location of the last '/' character in a path.
- K_BOOL [File_Names_Match](#) (const K_CHAR *szPath_, [NLFS_Node_t](#) *pstNode_)
File_Names_Match Determines if a given path matches the name in a file node.
- virtual void [Read_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)=0
Read_Node is an implementation-specific method used to read a file node from physical storage into a local data struture.
- virtual void [Write_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)=0
Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.
- virtual void [Read_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstBlock_)=0
Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.
- virtual void [Write_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstFileBlock_)=0
Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.
- virtual void [Read_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)=0
Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.
- virtual void [Write_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)=0
Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.
- void [RootSync](#) ()
RootSync Synchronize the filesystem config in the object back to the underlying storage mechanism.
- void [Repair](#) ()

Repair Checks a filesystem for inconsistencies and makes repairs in order to avoid losing storage blocks.

- void [Print_Free_Details](#) (K_USHORT usNode_)
Print_Free_Details Print details about a free node.
- void [Print_File_Details](#) (K_USHORT usNode_)
Print_File_Details displays information about a given file node.
- void [Print_Dir_Details](#) (K_USHORT usNode_)
Print_Dir_Details displays information about a given directory node.
- void [Print_Node_Details](#) (K_USHORT usNode_)
Print_Node_Details prints details about a node, the details differ based on whether it's a file/directory/root node.
- void [Push_Free_Node](#) (K_USHORT usNode_)
Push_Free_Node returns a file node back to the free node list.
- K_USHORT [Pop_Free_Node](#) (void)
Pop_Free_Node returns the first free file node in the free list.
- void [Push_Free_Block](#) (K_ULONG ulBlock_)
Push_Free_Block returns a file block back to the head of the free block list.
- K_ULONG [Pop_Free_Block](#) (void)
Pop_Free_Block pops a file data block from the head of the free list.
- K_ULONG [Append_Block_To_Node](#) (NLFS_Node_t *pstFile_)
Append_Block_To_Node adds a file data block to the end of a file.
- K_USHORT [Create_File_i](#) (const K_CHAR *szPath_, NLFS_Type_t eType_)
Create_File_i is the private method used to create a file or directory.
- void [Set_Node_Name](#) (NLFS_Node_t *pstFileNode_, const K_CHAR *szPath_)
Set_Node_Name sets the name of a file or directory node.

Protected Attributes

- [NLFS_Host_t](#) * [m_puHost](#)
Local, cached copy of host FS pointer.
- [NLFS_Root_Node_t](#) [m_stLocalRoot](#)
Local, cached copy of root.

Friends

- class [NLFS_File](#)

14.55.1 Detailed Description

Nice Little File System class.

Definition at line 280 of file [nlfs.h](#).

14.55.2 Member Function Documentation

14.55.2.1 K_ULONG NLFS::Append_Block_To_Node (NLFS_Node_t * *pstFile_*) [protected]

Append_Block_To_Node adds a file data block to the end of a file.

Parameters

<i>in</i>	<i>pstFile_</i>	- Pointer to the file node to add a block to
-----------	-----------------	--

Returns

Data block ID of the allocated block, or INVALID_BLOCK on failure.

Definition at line 245 of file [nlfs.cpp](#).

14.55.2.2 void NLFS::Cleanup_Node_Links (K_USHORT *usNode_*, NLFS_Node_t * *pstNode_*)

Cleanup_Node_Links Remove the links between the given node and its parent/peer nodes.

Parameters

<i>usNode_</i>	Index of the node
<i>pstNode_</i>	Pointer to a local copy of the node data

Definition at line 598 of file [nlfs.cpp](#).

14.55.2.3 K_USHORT NLFS::Create_Dir (const K_CHAR * *szPath_*)

Create_Dir creates a new directory at the specified path.

Parameters

<i>in</i>	<i>szPath_</i>	- Path to the directory to create
-----------	----------------	-----------------------------------

Returns

ID of the created dir, or INVALID_NODE if the path cannot be resolved, or the file already exists.

Definition at line 586 of file [nlfs.cpp](#).

14.55.2.4 K_USHORT NLFS::Create_File (const K_CHAR * *szPath_*)

Create_File creates a new file object at the specified path.

Parameters

<i>in</i>	<i>szPath_</i>	- Path to the file to create
-----------	----------------	------------------------------

Returns

ID of the created file, or INVALID_NODE if the path cannot be resolved, or the file already exists.

Definition at line 573 of file [nlfs.cpp](#).

14.55.2.5 K_USHORT NLFS::Create_File_i (const K_CHAR * *szPath_*, NLFS_Type_t *eType_*) [protected]

Create_File_i is the private method used to create a file or directory.

Parameters

in	<i>szPath_</i>	- Path of the file or directory to create
in	<i>eType_</i>	- Type of file to create

Returns

File node ID of the newly created file, or INVALID_NODE on failure.

! ToDo - set real user/group IDs

Definition at line 490 of file [nlfs.cpp](#).

14.55.2.6 K_USHORT NLFS::Delete_File (const K_CHAR * *szPath_*)

Delete_File Removes a file from disk.

Parameters

<i>szPath_</i>	Path of the file to remove
----------------	----------------------------

Returns

Index of the node deleted or INVALID_NODE on error

Definition at line 705 of file [nlfs.cpp](#).

14.55.2.7 K_USHORT NLFS::Delete_Folder (const K_CHAR * *szPath_*)

Delete_Folder Remove a folder from disk.

Parameters

<i>szPath_</i>	Path of the folder to remove
----------------	------------------------------

Returns

Index of the node deleted or INVALID_NODE on error

Definition at line 662 of file [nlfs.cpp](#).

14.55.2.8 K_BOOL NLFS::File_Names_Match (const K_CHAR * *szPath_*, NLFS_Node_t * *pstNode_*) [protected]

File_Names_Match Determines if a given path matches the name in a file node.

Parameters

in	<i>szPath_</i>	- file path to search for
in	<i>pstNode_</i>	- pointer to a fs node

Returns

true if the filename in the path matches the filename in the node.

Definition at line 42 of file [nlfs.cpp](#).

14.55.2.9 K_USHORT NLFS::Find_File (const K_CHAR * *szPath_*)

Find_File returns the file node ID of the object at a given path.

Parameters

in	<i>szPath_</i>	- Path of the file to search for
----	----------------	----------------------------------

Returns

file node ID, or INVALID_NODE if the path is invalid.

Definition at line 405 of file [nlfs.cpp](#).

14.55.2.10 K_CHAR NLFS::Find_Last_Slash (const K_CHAR * *szPath_*) [protected]

Find_Last_Slash Finds the location of the last '/' character in a path.

Parameters

in	<i>szPath_</i>	- String representing a '/' delimited path.
----	----------------	---

Returns

the byte offset of the last slash char in the path.

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

14.55.2.11 K_USHORT NLFS::Find_Parent_Dir (const K_CHAR * *szPath_*)

Find_Parent_Dir returns the directory under which the specified file object lives.

Parameters

in	<i>szPath_</i>	- Path of the file to find parent directory node for
----	----------------	--

Returns

directory node ID, or INVALID_NODE if the path is invalid.

Definition at line 289 of file [nlfs.cpp](#).

14.55.2.12 void NLFS::Format (NLFS_Host_t * *puHost_*, K_ULONG *ulTotalSize_*, K_USHORT *usNumFiles_*, K_USHORT *usDataBlockSize_*)

Format/Create a new filesystem with the configuration specified in the parameters.

Parameters

in	<i>puHost_</i>	- Pointer to the FS storage object, interpreted by the physical medium driver.
in	<i>ulTotalSize_</i>	- Total size of the object to format (in bytes)
in	<i>usNumFiles_</i>	- Number of file nodes to create in the FS. This parameter determines the maximum number of files and directories that can exist simultaneously in the filesystem. All filesystem storage not allocated towards file nodes is automatically used as data-blocks.

	<i>usDataBlockSize_</i>	- Size of each data block (in bytes). Setting a lower block size is a good way to avoid wasting space in small-files due to over-allocation of storage (size on-disk vs. actual file size). However, each block requires a metadata object, which can also add to overhead. Also, file read/write speed can vary significantly based on the block size - in many scenarios, larger blocks can lead to higher throughput.
--	-------------------------	--

Definition at line 756 of file [nlfs.cpp](#).

14.55.2.13 K_ULONG NLFS::GetBlockSize (void) [inline]

GetBlockSize retrieves the data block size for the filesystem.

Returns

The size of a data block in the filesystem, as configured at format.

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

14.55.2.14 K_USHORT NLFS::GetFirstChild (K_USHORT *usNode_*)

GetFirstChild Return the first child node for a node representing a directory.

Parameters

<i>usNode_</i>	Index of a directory node
----------------	---------------------------

Returns

Node ID of the first child node or INVALID_NODE on failure

Definition at line 890 of file [nlfs.cpp](#).

14.55.2.15 K_USHORT NLFS::GetNextPeer (K_USHORT *usNode_*)

GetNextPeer Return the Node ID of a File/Directory's next peer.

Parameters

<i>usNode_</i>	Node index of the current object
----------------	----------------------------------

Returns

Node ID of the next peer object

Definition at line 908 of file [nlfs.cpp](#).

14.55.2.16 K_ULONG NLFS::GetNumBlocks (void) [inline]

GetNumBlocks retrieves the number of data blocks in the filesystem.

Returns

The total number of blocks in the filesystem

Definition at line 388 of file [nlfs.h](#).

14.55.2.17 K_ULONG NLFS::GetNumBlocksFree (void) [inline]

GetNumBlocksFree retrieves the number of free data blocks in the filesystem.

Returns

The number of available blocks in the filesystem

Definition at line 395 of file [nlfs.h](#).

14.55.2.18 K_ULONG NLFS::GetNumFiles (void) [inline]

GetNumFiles retrieves the maximum number of files in the filesystem.

Returns

The maximum number of files that can be allocated in the system

Definition at line 401 of file [nlfs.h](#).

14.55.2.19 K_USHORT NLFS::GetNumFilesFree (void) [inline]

GetNumFilesFree retrieves the number of free blocks in the filesystem.

Returns

The number of free file nodes in the filesystem

Definition at line 407 of file [nlfs.h](#).

14.55.2.20 K_BOOL NLFS::GetStat (K_USHORT *usNode_*, NLFS_File_Stat_t * *pstStat_*)

GetStat Get the status of a file on-disk.

Parameters

<i>usNode_</i>	Node representing the file
<i>pstStat_</i>	Pointer to the object containing the status

Returns

true on success, false on failure

Definition at line 920 of file [nlfs.cpp](#).

14.55.2.21 void NLFS::Mount (NLFS_Host_t * *puHost_*)

Re-mount a previously-created filesystem using this FS object.

Parameters

<i>in</i>	<i>puHost_</i>	- Pointer to the filesystem object
-----------	----------------	------------------------------------

! Must set the host pointer first.

Definition at line 859 of file [nlfs.cpp](#).

14.55.2.22 `K_ULONG NLFS::Pop_Free_Block (void)` [protected]

`Pop_Free_Block` pops a file data block from the head of the free list.

Returns

the block index of the file node popped from the head of the free block list

Definition at line 192 of file [nlfs.cpp](#).

14.55.2.23 `K_USHORT NLFS::Pop_Free_Node (void)` [protected]

`Pop_Free_Node` returns the first free file node in the free list.

Returns

the index of the file node popped off the free list

Definition at line 145 of file [nlfs.cpp](#).

14.55.2.24 `void NLFS::Print_Dir_Details (K_USHORT usNode_)` [protected]

`Print_Dir_Details` displays information about a given directory node.

Parameters

<code>in</code>	<code><i>usNode_</i></code>	- directory index to display details for
-----------------	-----------------------------	--

Definition at line 90 of file [nlfs.cpp](#).

14.55.2.25 `void NLFS::Print_File_Details (K_USHORT usNode_)` [protected]

`Print_File_Details` displays information about a given file node.

Parameters

<code>in</code>	<code><i>usNode_</i></code>	- file index to display details for
-----------------	-----------------------------	-------------------------------------

Definition at line 68 of file [nlfs.cpp](#).

14.55.2.26 `void NLFS::Print_Free_Details (K_USHORT usNode_)` [protected]

`Print_Free_Details` Print details about a free node.

Parameters

<code><i>usNode_</i></code>	Node to print details for
-----------------------------	---------------------------

Definition at line 106 of file [nlfs.cpp](#).

14.55.2.27 `void NLFS::Print_Node_Details (K_USHORT usNode_)` [protected]

`Print_Node_Details` prints details about a node, the details differ based on whether it's a file/directory/root node.

Parameters

in	<i>usNode_</i>	- node to show details for
----	----------------	----------------------------

Definition at line 115 of file [nlfs.cpp](#).

14.55.2.28 void NLFS::Push_Free_Block (K_ULONG *ulBlock_*) [protected]

Push_Free_Block returns a file block back to the head of the free block list.

Parameters

in	<i>ulBlock_</i>	- index of the data block to free
----	-----------------	-----------------------------------

Definition at line 224 of file [nlfs.cpp](#).

14.55.2.29 void NLFS::Push_Free_Node (K_USHORT *usNode_*) [protected]

Push_Free_Node returns a file node back to the free node list.

Parameters

in	<i>usNode_</i>	- index of the file node to push back to the free list.
----	----------------	---

Definition at line 172 of file [nlfs.cpp](#).

14.55.2.30 virtual void NLFS::Read_Block (K_ULONG *ulBlock_*, K_ULONG *ulOffset_*, void * *pvData_*, K_ULONG *ulLen_*) [protected], [pure virtual]

Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
out	<i>pvData_</i>	- output buffer to read into
in	<i>ulLen_</i>	- length of data to read (in bytes)

Implemented in [NLFS_RAM](#).

14.55.2.31 virtual void NLFS::Read_Block_Header (K_ULONG *ulBlock_*, NLFS_Block_t * *pstBlock_*) [protected], [pure virtual]

Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
out	<i>pstBlock_</i>	- block header structure to read into

Implemented in [NLFS_RAM](#).

14.55.2.32 virtual void NLFS::Read_Node (K_USHORT *usNode_*, NLFS_Node_t * *pstNode_*) [protected], [pure virtual]

Read_Node is an implementation-specific method used to read a file node from physical storage into a local data structure.

Parameters

in	<i>usNode_</i>	- File node index
out	<i>pstNode_</i>	- Pointer to the file node object to read into

Implemented in [NLFS_RAM](#).

14.55.2.33 `void NLFS::RootSync ()` [protected]

RootSync Synchronize the filesystem config in the object back to the underlying storage mechanism.

This needs to be called to ensure that underlying storage is kept consistent when creating or deleting files.

Definition at line 879 of file [nlfs.cpp](#).

14.55.2.34 `void NLFS::Set_Node_Name (NLFS_Node_t * pstFileNode_, const K_CHAR * szPath_)` [protected]

Set_Node_Name sets the name of a file or directory node.

Parameters

in	<i>pstFileNode_</i>	- Pointer to a file node structure to name
in	<i>szPath_</i>	- Name for the file

Definition at line 458 of file [nlfs.cpp](#).

14.55.2.35 `virtual void NLFS::Write_Block (K_ULONG ulBlock_, K_ULONG ulOffset_, void * pvData_, K_ULONG ulLen_)`
[protected],[pure virtual]

Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
in	<i>pvData_</i>	- data buffer to write to disk
in	<i>ulLen_</i>	- length of data to write (in bytes)

Implemented in [NLFS_RAM](#).

14.55.2.36 `virtual void NLFS::Write_Block_Header (K_ULONG ulBlock_, NLFS_Block_t * pstFileBlock_)`
[protected],[pure virtual]

Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
in	<i>pstFileBlock_</i>	- pointer to the local data structure to write from

Implemented in [NLFS_RAM](#).

14.55.2.37 `virtual void NLFS::Write_Node (K_USHORT usNode_, NLFS_Node_t * pstNode_)` [protected],[pure virtual]

Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.

Parameters

in	<i>usNode_</i>	- File node index
in	<i>pstNode_</i>	- Pointer to the file node object to write from

Implemented in [NLFS_RAM](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)
- [/home/moslevin/Project/R1/stage/src/nlfs.cpp](#)

14.56 NLFS_Block_t Struct Reference

Block data structure.

```
#include <nlfs.h>
```

Public Attributes

- K_ULONG [ulNextBlock](#)
Index of the next block.

14.56.1 Detailed Description

Block data structure.

Contains the block index of the next data block (either in the file, or in the free-data pool), as well as any special flags.

Definition at line [232](#) of file [nlfs.h](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.57 NLFS_File Class Reference

The [NLFS_File](#) class.

```
#include <nlfs_file.h>
```

Public Member Functions

- int [Open](#) (NLFS *pclFS_, const K_CHAR *szPath_, NLFS_File_Mode_t eMode_)
Open Opens a file from a given filesystem.
- int [Read](#) (void *pvBuf_, K_ULONG ulLen_)
Read Read bytes from a file into a specified data buffer.
- int [Write](#) (void *pvBuf_, K_ULONG ulLen_)
Write Write a specified blob of data to the file.
- int [Seek](#) (K_ULONG ulOffset_)
Seek Seek to the specified byte offset within the file.
- int [Close](#) (void)
Close Is used to close an open file buffer.

Private Attributes

- [NLFS * m_pclFileSystem](#)
Pointer to the host filesystem.
- [K_ULONG m_ulOffset](#)
Current byte offset within the file.
- [K_ULONG m_ulCurrentBlock](#)
Index of the current filesystem block.
- [K_USHORT m_usFile](#)
File index of the current file.
- [NLFS_File_Mode_t m_ucFlags](#)
File mode flags.
- [NLFS_Node_t m_stNode](#)
Local copy of the file node.

14.57.1 Detailed Description

The [NLFS_File](#) class.

This class contains an implementation of file-level access built on-top of the [NLFS](#) filesystem architecture. An instance of this class represents an active/open file from inside the NLFSfilesystem.

Definition at line 45 of file [nlfs_file.h](#).

14.57.2 Member Function Documentation

14.57.2.1 int NLFS_File::Close (void)

Close Is used to close an open file buffer.

Returns

0 on success, -1 on failure.

Definition at line 272 of file [nlfs_file.cpp](#).

14.57.2.2 int NLFS_File::Open (NLFS * pclFS_, const K_CHAR * szPath_, NLFS_File_Mode_t eMode_)

Open Opens a file from a given filesystem.

Parameters

<i>pcIFS_</i>	- Pointer to the NLFS filesystem containing the file
<i>szPath_</i>	- Path to the file within the NLFS filesystem
<i>eMode_</i>	- File open mode

Returns

0 on success, -1 on failure

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

14.57.2.3 int NLFS_File::Read (void * pvBuf_, K_ULONG ulLen_)

Read Read bytes from a file into a specified data buffer.

Parameters

in	<i>uLen_</i>	- Length (in bytes) of data to read
out	<i>pvBuf_</i>	- Pointer to the buffer to read into

Returns

Number of bytes read from the file

Definition at line 151 of file [nlfs_file.cpp](#).

14.57.2.4 int NLFS_File::Seek (K_ULONG uOffset_)

Seek Seek to the specified byte offset within the file.

Parameters

in	<i>uOffset_</i>	Offset in bytes from the beginning of the file
----	-----------------	--

Returns

0 on success, -1 on failure

Definition at line 112 of file [nlfs_file.cpp](#).

14.57.2.5 int NLFS_File::Write (void * pvBuf_, K_ULONG uLen_)

Write Write a specified blob of data to the file.

Parameters

in	<i>uLen_</i>	- Length (in bytes) of the source buffer
in	<i>pvBuf_</i>	- Pointer to the data buffer containing the data to be written

Returns

Number of bytes written to the file

Definition at line 217 of file [nlfs_file.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs_file.h](#)
- [/home/moslevin/Project/R1/stage/src/nlfs_file.cpp](#)

14.58 NLFS_File_Node_t Struct Reference

Data structure for the "file" FS-node type.

```
#include <nlfs.h>
```

Public Attributes

- K_CHAR [acFileName](#) [16]
Arbitrary, 16-char filename.
- K_USHORT [usNextPeer](#)

- *Index of the next peer file node.*
- K_USHORT [usPrevPeer](#)
- *Index of the previous peer node.*
- K_UCHAR [ucGroup](#)
- *Group ID of the owner.*
- K_UCHAR [ucUser](#)
- *User ID of the owner.*
- K_USHORT [usPerms](#)
- *File permissions (POSIX-style)*
- K_USHORT [usParent](#)
- *Index of the parent file node.*
- K_USHORT [usChild](#)
- *Index of the first child node.*
- K_ULONG [ulAllocSize](#)
- *Size of the file (allocated)*
- K_ULONG [ulFileSize](#)
- *Size of the file (in-bytes)*
- K_ULONG [ulFirstBlock](#)
- *Index of the first file block.*
- K_ULONG [ulLastBlock](#)
- *Index of the last file block.*

14.58.1 Detailed Description

Data structure for the "file" FS-node type.

Note that this is the same as for a directory node (although fewer fields are used for that case, as documented).

Definition at line 168 of file [nlfs.h](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.59 NLFS_File_Stat_t Struct Reference

Structure used to report the status of a given file.

```
#include <nlfs.h>
```

Public Attributes

- K_ULONG [ulAllocSize](#)
- *Size of the file including partial blocks.*
- K_ULONG [ulFileSize](#)
- *Actual size of the file.*
- K_USHORT [usPerms](#)
- *Permissions attached to the file.*
- K_UCHAR [ucUser](#)
- *User associated with this file.*
- K_UCHAR [ucGroup](#)
- *Group associated with this file.*
- K_CHAR [acFileName](#) [16]
- *Copy of the file name.*

14.59.1 Detailed Description

Structure used to report the status of a given file.

Definition at line 266 of file [nlfs.h](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.60 NLFS_Host_t Union Reference

Union used for managing host-specific pointers/data-types.

```
#include <nlfs.h>
```

Public Attributes

- void * **pvData**
- uint32_t **u32Data**
- uint64_t **u64Data**
- K_ADDR **kaData**

14.60.1 Detailed Description

Union used for managing host-specific pointers/data-types.

This is all pretty abstract, as the data represented here is only accessed by the underlying physical media drive.

Definition at line 253 of file [nlfs.h](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.61 NLFS_Node_t Struct Reference

Filesystem node data structure.

```
#include <nlfs.h>
```

Public Attributes

- [NLFS_Type_t](#) **eBlockType**
Block type ID.

14.61.1 Detailed Description

Filesystem node data structure.

Contains the block type, as well as the union between the various FS-node data structures. This is also the same data format as how data is stored "on-disk"

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

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

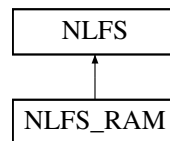
- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.62 NLFS_RAM Class Reference

The [NLFS_RAM](#) class.

```
#include <nlfs_ram.h>
```

Inheritance diagram for NLFS_RAM:



Private Member Functions

- virtual void [Read_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Read_Node is an implementation-specific method used to read a file node from physical storage into a local data structure.
- virtual void [Write_Node](#) (K_USHORT usNode_, [NLFS_Node_t](#) *pstNode_)
Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.
- virtual void [Read_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstBlock_)
Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.
- virtual void [Write_Block_Header](#) (K_ULONG ulBlock_, [NLFS_Block_t](#) *pstFileBlock_)
Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.
- virtual void [Read_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)
Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.
- void [Write_Block](#) (K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)
Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Additional Inherited Members

14.62.1 Detailed Description

The [NLFS_RAM](#) class.

This class implements an [NLFS](#) filesystem in a RAM buffer. In this case, the host pointer passed into the "format" call is a pointer to the locally- allocated buffer in which the filesystem lives.

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

14.62.2 Member Function Documentation

14.62.2.1 void NLFS_RAM::Read_Block (K_ULONG *ulBlock_*, K_ULONG *ulOffset_*, void * *pvData_*, K_ULONG *ulLen_*)
[private], [virtual]

Read_Block is an implementation-specific method used to read raw file data from physical storage into a local buffer.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
out	<i>pvData_</i>	- output buffer to read into
in	<i>ulLen_</i>	- length of data to read (in bytes)

Implements [NLFS](#).

Definition at line 63 of file [nlfs_ram.cpp](#).

```
14.62.2.2 void NLFS_RAM::Read_Block_Header ( K_ULONG ulBlock_, NLFS_Block_t* pstBlock_ ) [private],
          [virtual]
```

Read_Block_Header is an implementation-specific method used to read a file block header from physical storage into a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
out	<i>pstBlock_</i>	- block header structure to read into

Implements [NLFS](#).

Definition at line 43 of file [nlfs_ram.cpp](#).

```
14.62.2.3 void NLFS_RAM::Read_Node ( K_USHORT usNode_, NLFS_Node_t* pstNode_ ) [private],
          [virtual]
```

Read_Node is an implementation-specific method used to read a file node from physical storage into a local data structure.

Parameters

in	<i>usNode_</i>	- File node index
out	<i>pstNode_</i>	- Pointer to the file node object to read into

Implements [NLFS](#).

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

```
14.62.2.4 void NLFS_RAM::Write_Block ( K_ULONG ulBlock_, K_ULONG ulOffset_, void* pvData_, K_ULONG ulLen_ )
          [private], [virtual]
```

Write_Block is an implementation-specific method used to write a piece of file data to its data block in the underlying physical storage.

Parameters

in	<i>ulBlock_</i>	- filesystem block ID corresponding to the file
in	<i>ulOffset_</i>	- offset (in bytes) from the beginning of the block
in	<i>pvData_</i>	- data buffer to write to disk
in	<i>ulLen_</i>	- length of data to write (in bytes)

Implements [NLFS](#).

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

14.62.2.5 void NLFS_RAM::Write_Block_Header (K_ULONG *ulBlock_*, NLFS_Block_t* *pstFileBlock_*) [private],
[virtual]

Write_Block_Header is an implementation-specific method used to write a file block header back to physical storage from a local struct.

Parameters

in	<i>ulBlock_</i>	- data block index
in	<i>pstFileBlock_</i>	- pointer to the local data structure to write from

Implements [NLFS](#).

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

```
14.62.2.6 void NLFS_RAM::Write_Node ( K_USHORT usNode_, NLFS_Node_t * pstNode_ ) [private],
          [virtual]
```

Write_Node is an implementation-specific method used to write a file node from a local structure back to the physical storage.

Parameters

in	<i>usNode_</i>	- File node index
in	<i>pstNode_</i>	- Pointer to the file node object to write from

Implements [NLFS](#).

Definition at line 34 of file [nlfs_ram.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs_ram.h](#)
- [/home/moslevin/Project/R1/stage/src/nlfs_ram.cpp](#)

14.63 NLFS_Root_Node_t Struct Reference

Data structure for the Root-configuration FS-node type.

```
#include <nlfs.h>
```

Public Attributes

- K_USHORT [usNumFiles](#)
Number of file nodes in the FS.
- K_USHORT [usNumFilesFree](#)
Number of free file nodes.
- K_USHORT [usNextFreeNode](#)
Index of the next free file.
- K_ULONG [ulNumBlocks](#)
Number of blocks in the FS.
- K_ULONG [ulNumBlocksFree](#)
Number of free blocks.
- K_ULONG [ulNextFreeBlock](#)
Index of the next free block.
- K_ULONG [ulBlockSize](#)
Size of each block on disk.
- K_ULONG [ulBlockOffset](#)
Byte-offset to the first block struct.
- K_ULONG [ulDataOffset](#)
Byte-offset to the first data block.

14.63.1 Detailed Description

Data structure for the Root-configuration FS-node type.

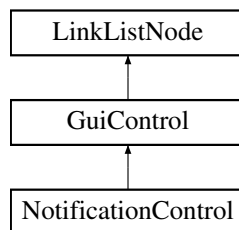
Definition at line 194 of file [nlfs.h](#).

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

- [/home/moslevin/Project/R1/stage/src/nlfs.h](#)

14.64 NotificationControl Class Reference

Inheritance diagram for NotificationControl:



Public Member Functions

- virtual void [Init](#) ()
Initialize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *szCaption_)
- void [Trigger](#) (K_USHORT usTimeout_)

Private Attributes

- const K_CHAR * [m_szCaption](#)
- [Font_t](#) * [m_pstFont](#)
- K_USHORT [m_usTimeout](#)
- bool [m_bTrigger](#)
- bool [m_bVisible](#)

Additional Inherited Members

14.64.1 Detailed Description

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

14.64.2 Member Function Documentation

14.64.2.1 `virtual void NotificationControl::Activate (bool bActivate_)` `[inline], [virtual]`

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

Parameters

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

Implements [GuiControl](#).

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

14.64.2.2 void NotificationControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.64.2.3 virtual void NotificationControl::Init () [inline],[virtual]

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.64.2.4 GuiReturn_t NotificationControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

Definition at line 92 of file [control_notification.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/control_notification.h](#)
- [/home/moslevin/Project/R1/stage/src/control_notification.cpp](#)

14.65 Option_t Struct Reference

Structure used to represent a command-line option with its arguments.

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

Public Attributes

- [Token_t](#) * [pstStart](#)
Pointer to the beginning of a token array contain the option and its arguments.
- [K_UCHAR](#) [ucCount](#)
Number of tokens in the token array.

14.65.1 Detailed Description

Structure used to represent a command-line option with its arguments.

An option is defined as any token beginning with a "-" value. The tokens arguments are subsequent tokens that do not begin with "-".

Where no "-" values are specified, each token becomes its own option.

i.e. given the following command-line

```
mycmd -opt1 a b c -opt2 d e f -opt 3
```

The possible `Option_t` structures would be:

```
pstStart => Array containing tokens for -opt1, a, b, c
ucCount  => 4 (4 tokens, including the option token, "-opt1")

pstStart => Array containing tokens for -opt2, d, e, f
ucCount  => 4 (4 tokens, including the option token, "-opt2")

pstStart => Array containing tokens for -opt, 3
ucCount  => 2 (2 tokens, including the option token, "-opt3")
```

in the case of:

```
mycmd a b c
```

Possible token values would be:

```
pstStart => Array containing tokens for a
ucCount  => 1

pstStart => Array containing tokens for b
ucCount  => 1

pstStart => Array containing tokens for c
ucCount  => 1
```

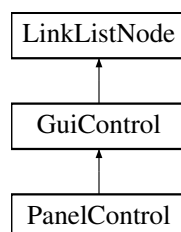
Definition at line 83 of file [shell_support.h](#).

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

- [/home/moslevin/Project/R1/stage/src/shell_support.h](#)

14.66 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

14.66.1 Detailed Description

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

14.66.2 Member Function Documentation

14.66.2.1 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](#).

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

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.66.2.3 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](#).

14.66.2.4 virtual [GuiReturn_t](#) [PanelControl::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 38 of file [control_panel.h](#).

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

- [/home/moslevin/Project/R1/stage/src/control_panel.h](#)
- [/home/moslevin/Project/R1/stage/src/control_panel.cpp](#)

14.67 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**

14.67.1 Detailed Description

System profiling timer interface.

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

14.67.2 Member Function Documentation

14.67.2.1 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/Project/R1/stage/src/kprofile.h](#)
- [/home/moslevin/Project/R1/stage/src/kprofile.cpp](#)

14.68 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.

14.68.1 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-timer behavior.

Definition at line 69 of file [profile.h](#).

14.68.2 Member Function Documentation

14.68.2.1 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](#).

14.68.2.2 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](#).

14.68.2.3 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](#).

14.68.2.4 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](#).

14.68.2.5 void ProfileTimer::Start (void)

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

Has no effect if the timer is already active.

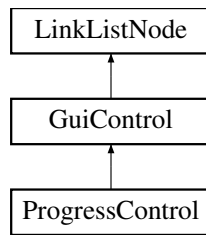
Definition at line 46 of file [profile.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/profile.h](#)
- [/home/moslevin/Project/R1/stage/src/profile.cpp](#)

14.69 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

14.69.1 Detailed Description

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

14.69.2 Member Function Documentation

14.69.2.1 virtual void [ProgressControl::Activate](#) (bool *bActivate_*) [inline], [virtual]

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

Parameters

bActivate_	- true to activate, false to deactivate
----------------------------	---

Implements [GuiControl](#).

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

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

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

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

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

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

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

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

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

- [/home/moslevin/Project/R1/stage/src/control_progress.h](#)
- [/home/moslevin/Project/R1/stage/src/control_progress.cpp](#)

14.70 PseudoRandom Class Reference

The [PseudoRandom](#) class.

```
#include <rand_lfsr.h>
```

Public Member Functions

- [PseudoRandom \(\)](#)
PseudoRandom.
- void [Seed](#) (K_ULONG ulSeed1_, K_ULONG ulSeed2_)
Seed.
- K_USHORT [GetRandom \(\)](#)
GetRandom.

Private Member Functions

- K_ULONG [Shift](#) (K_ULONG *pulFSR_, K_ULONG ulMask_)
Shift.

Private Attributes

- K_ULONG m_ulFSR1
- K_ULONG m_ulFSR2

14.70.1 Detailed Description

The [PseudoRandom](#) class.

Pseudro-Random Number Generator based on a 32-bit linear-feedback shift register. For reference, the implementation is based on the Maxim IC application note "AN4000"

<http://www.maximintegrated.com/app-notes/index.mvp/id/4400>

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

14.70.2 Constructor & Destructor Documentation

14.70.2.1 PseudoRandom::PseudoRandom ()

[PseudoRandom](#).

Default constructor

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

14.70.3 Member Function Documentation

14.70.3.1 K_USHORT PseudoRandom::GetRandom ()

GetRandom.

Return a 16-bit random number.

Returns

16-bit random number

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

14.70.3.2 void PseudoRandom::Seed (K_ULONG ulSeed1_, K_ULONG ulSeed2_)

Seed.

Initialize the LFSR registers in the object, seeding the random number generator.

Parameters

<i>ulSeed1_</i>	Seed value for LFSR register 1
<i>ulSeed2_</i>	Seed value for LFSR register 2

Definition at line 33 of file [rand_lfsr.cpp](#).

14.70.3.3 K_ULONG PseudoRandom::Shift (K_ULONG * pulFSR_, K_ULONG ulMask_) [private]

Shift.

Deep magic to drive the the LFSR state machine

Parameters

<i>pulLFSR_</i>	Pointer to the register to drive
<i>ulMask_</i>	Bitmask to apply as an XOR

Returns

new value in the LFSR register

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

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

- [/home/moslevin/Project/R1/stage/src/rand_lfsr.h](#)
- [/home/moslevin/Project/R1/stage/src/rand_lfsr.cpp](#)

14.71 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 void [SetInTimer](#) (void)
SetInTimer.
- static void [ClearInTimer](#) (void)
ClearInTimer.

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**
- static K_UCHAR **m_bInTimer**

14.71.1 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](#).

14.71.2 Member Function Documentation

14.71.2.1 void Quantum::AddThread (Thread * *pciThread_*) [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 71 of file [quantum.cpp](#).

14.71.2.2 static void Quantum::ClearInTimer (void) [inline],[static]

ClearInTimer.

Clear the flag once the timer callback function has been completed.

Definition at line 82 of file [quantum.h](#).

14.71.2.3 void Quantum::RemoveThread (void) [static]

Remove the thread from the quantum timer.

This will cancel the timer.

Definition at line 97 of file [quantum.cpp](#).

14.71.2.4 static void Quantum::SetInTimer (void) [inline],[static]

SetInTimer.

Set a flag to indicate that the CPU is currently running within the timer-callback routine. This prevents the [Quantum](#) timer from being updated in the middle of a callback cycle, potentially resulting in the kernel timer becoming disabled.

Definition at line 75 of file [quantum.h](#).

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

Definition at line 61 of file [quantum.cpp](#).

14.71.2.6 void Quantum::UpdateTimer (void) [static]

This function is called to update the thread quantum timer whenever something in the scheduler has changed.

This can result in the timer being re-loaded or started. The timer is never stopped, but it may be ignored on expiry.

Definition at line 110 of file [quantum.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/quantum.h](#)
- [/home/moslevin/Project/R1/stage/src/quantum.cpp](#)

14.72 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](#) *pciThread_)
Add a thread to the scheduler at its current priority level.
- static void [Remove](#) ([Thread](#) *pciThread_)
Remove a thread from the scheduler at its current priority level.
- static K_BOOL [SetScheduler](#) (K_BOOL bEnable_)
Set the active state of the scheduler.
- static [Thread](#) * [GetCurrentThread](#) ()
Return the pointer to the currently-running thread.
- static [Thread](#) * [GetNextThread](#) ()
Return the pointer to the thread that should run next, according to the last run of the scheduler.
- static [ThreadList](#) * [GetThreadList](#) (K_UCHAR ucPriority_)
Return the pointer to the active list of threads that are at the given priority level in the scheduler.
- static [ThreadList](#) * [GetStopList](#) ()
Return the pointer to the list of threads that are in the scheduler's stopped state.
- static K_UCHAR [IsEnabled](#) ()
Return the current state of the scheduler - whether or not scheudling is enabled or disabled.
- static void [QueueScheduler](#) ()

Static Private Attributes

- static K_BOOL [m_bEnabled](#)
Scheduler's state - enabled or disabled.
- static K_BOOL [m_bQueuedSchedule](#)
Variable representing whether or not there's a queued scheduler operation.
- static [ThreadList](#) [m_clStopList](#)
ThreadList for all stopped threads.
- static [ThreadList](#) [m_aclPriorities](#) [NUM_PRIORITIES]
ThreadLists for all threads at all priorities.
- static K_UCHAR [m_ucPriFlag](#)
Bitmap flag for each.

14.72.1 Detailed Description

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

Definition at line 62 of file [scheduler.h](#).

14.72.2 Member Function Documentation

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

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

Parameters

<i>pcThread_</i>	Pointer to the thread to add to the scheduler
------------------	---

Definition at line 81 of file [scheduler.cpp](#).

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

Return the pointer to the currently-running thread.

Returns

Pointer to the currently-running thread

Definition at line 119 of file [scheduler.h](#).

14.72.2.3 static Thread* Scheduler::GetNextThread () [inline],[static]

Return the pointer to the thread that should run next, according to the last run of the scheduler.

Returns

Pointer to the next-running thread

Definition at line 127 of file [scheduler.h](#).

14.72.2.4 static ThreadList* Scheduler::GetStopList () [inline],[static]

Return the pointer to the list of threads that are in the scheduler's stopped state.

Returns

Pointer to the [ThreadList](#) containing the stopped threads

Definition at line 145 of file [scheduler.h](#).

14.72.2.5 static ThreadList* Scheduler::GetThreadList (K_UCHAR ucPriority_) [inline],[static]

Return the pointer to the active list of threads that are at the given priority level in the scheduler.

Parameters

<i>ucPriority_</i>	Priority level of
--------------------	-------------------

Returns

Pointer to the [ThreadList](#) for the given priority level

Definition at line 137 of file [scheduler.h](#).

14.72.2.6 K_UCHAR Scheduler::IsEnabled () [inline],[static]

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

Returns

true - scheduler enabled, false - disabled

Definition at line 155 of file [scheduler.h](#).

14.72.2.7 `void Scheduler::Remove (Thread * pcThread_) [static]`

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

Parameters

<code>pcThread_</code>	Pointer to the thread to be removed from the scheduler
------------------------	--

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

14.72.2.8 Scheduler::Schedule () [static]

Run the scheduler, determines the next thread to run based on the current state of the threads.

Note that the next-thread chosen from this function is only valid while in a critical section.

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

14.72.2.9 void Scheduler::SetScheduler (K_BOOL bEnable_) [static]

Set the active state of the scheduler.

When the scheduler is disabled, the *next thread* is never set; the currently running thread will run forever until the scheduler is enabled again. Care must be taken to ensure that we don't end up trying to block while the scheduler is disabled, otherwise the system ends up in an unusable state.

Parameters

<code>bEnable_</code>	true to enable, false to disable the scheduler
-----------------------	--

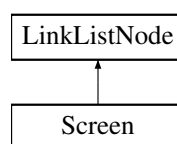
Definition at line 95 of file [scheduler.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/scheduler.h](#)
- [/home/moslevin/Project/R1/stage/src/scheduler.cpp](#)

14.73 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**

14.73.1 Detailed Description

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

14.73.2 Member Function Documentation

14.73.2.1 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 42 of file [screen.h](#).

14.73.2.2 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 49 of file [screen.h](#).

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

- /home/moslevin/Project/R1/stage/src/[screen.h](#)
- /home/moslevin/Project/R1/stage/src/[screen.cpp](#)

14.74 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.*

14.74.1 Detailed Description

Definition at line 86 of file [screen.h](#).

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

- [/home/moslevin/Project/R1/stage/src/screen.h](#)

14.75 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.*

14.75.1 Detailed Description

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

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

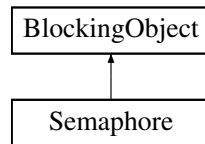
- [/home/moslevin/Project/R1/stage/src/screen.h](#)
- [/home/moslevin/Project/R1/stage/src/screen.cpp](#)

14.76 Semaphore Class Reference

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

```
#include <ksemaphore.h>
```

Inheritance diagram for Semaphore:



Public Member Functions

- void [Init](#) (K_USHORT usInitVal_, K_USHORT usMaxVal_)
Initialize a semaphore before use.
- bool [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.

Private Member Functions

- K_UCHAR [WakeNext](#) ()
Wake the next thread waiting on the semaphore.
- bool [Pend_i](#) (K_ULONG ulWaitTimeMS_)

Private Attributes

- K_USHORT **m_usValue**
- K_USHORT **m_usMaxValue**

Additional Inherited Members

14.76.1 Detailed Description

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

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

14.76.2 Member Function Documentation

14.76.2.1 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 223 of file [ksemaphore.cpp](#).

14.76.2.2 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 [ksemaphore.cpp](#).

14.76.2.3 void Semaphore::Pend ()

Decrement the semaphore count.

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

Definition at line 205 of file [ksemaphore.cpp](#).

14.76.2.4 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 216 of file [ksemaphore.cpp](#).

14.76.2.5 void Semaphore::Post ()

Increment the semaphore count.

Returns

true if the semaphore was posted, false if the count is already maxed out.

Definition at line 96 of file [ksemaphore.cpp](#).

14.76.2.6 void Semaphore::WakeMe (Thread * *pclChosenOne_*)

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 [ksemaphore.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/ksemaphore.h](#)
- [/home/moslevin/Project/R1/stage/src/ksemaphore.cpp](#)

14.77 ShellCommand_t Struct Reference

Data structure defining a lookup table correlating a command name to its handler function.

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

Public Attributes

- const K_CHAR * [szCommand](#)
Command name.
- [fp_internal_command](#) pfHandler
Command handler function.

14.77.1 Detailed Description

Data structure defining a lookup table correlating a command name to its handler function.

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

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

- [/home/moslevin/Project/R1/stage/src/shell_support.h](#)

14.78 ShellSupport Class Reference

The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

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

Static Public Member Functions

- static K_CHAR [RunCommand](#) (CommandLine_t *pstCommand_, const [ShellCommand_t](#) *pstShellCommands_)
RunCommand Given a command-line, attempts to run the corresponding shell command based where it exists within the supplied [ShellCommand_t](#) array.
- static void [UnescapeToken](#) (Token_t *pstToken_, K_CHAR *szDest_)
UnescapeToken Convert a token which has special parsing characters in it to a "flattened" string, where all unescaped double quotes and escaped tab, newline, space, etc.
- static [Option_t](#) * [CheckForOption](#) (CommandLine_t *pstCommand_, const K_CHAR *szOption_)

CheckForOption Check to see whether or not a specific option has been set within the commandline arguments.

- static K_CHAR [TokensToCommandLine](#) ([Token_t](#) *pastTokens_, K_UCHAR ucTokens_, [CommandLine_t](#) *pstCommand_)

TokensToCommandLine Convert an array of tokens to a commandline object.

14.78.1 Detailed Description

The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

Definition at line 129 of file [shell_support.h](#).

14.78.2 Member Function Documentation

14.78.2.1 [Option_t](#) * [ShellSupport::CheckForOption](#) ([CommandLine_t](#) * *pstCommand_*, const K_CHAR * *szOption_*)
[static]

CheckForOption Check to see whether or not a specific option has been set within the commandline arguments.

Parameters

<i>pstCommand_</i>	Pointer to the commandline object containing the options
<i>szOption_</i>	0-terminated string corresponding to the command-line option.

Returns

Pointer to the command line option on match, or 0 on failure.

Definition at line 104 of file [shell_support.cpp](#).

14.78.2.2 K_CHAR [ShellSupport::RunCommand](#) ([CommandLine_t](#) * *pstCommand_*, const [ShellCommand_t](#) * *pastShellCommands_*) [static]

RunCommand Given a command-line, attempts to run the corresponding shell command based where it exists within the supplied [ShellCommand_t](#) array.

Parameters

<i>pstCommand_</i>	Pointer to the command-line to execute
<i>pstCommands_</i>	Pointer to an array of shell commands to execute against

Returns

1 on success, 0 on error (command not found)

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

14.78.2.3 K_CHAR [ShellSupport::TokensToCommandLine](#) ([Token_t](#) * *pastTokens_*, K_UCHAR *ucTokens_*, [CommandLine_t](#) * *pstCommand_*) [static]

TokensToCommandLine Convert an array of tokens to a commandline object.

This operation is non-destructive to the source token array.

Parameters

<i>pstTokens_</i>	Pointer to the token array to process
<i>ucTokens_</i>	Number of tokens in the token array
<i>pstCommand_</i>	Pointer to the CommandLine_t object which will represent the shell command and its arguments.

Returns

Number of options processed

Definition at line 123 of file [shell_support.cpp](#).

14.78.2.4 void ShellSupport::UnescapeToken (Token_t * *pstToken_*, K_CHAR * *szDest_*) [static]

UnescapeToken Convert a token which has special parsing characters in it to a "flattened" string, where all un-escaped double quotes and escaped tab, newline, space, etc.

characters are converted into their ascii-code equivalents.

Parameters

<i>pstToken_</i>	Pointer to the source token to convert
<i>szDest_</i>	Pointer to a destination string which will contain the parsed result string

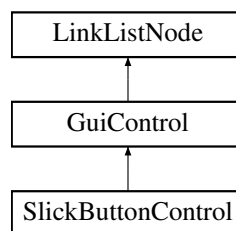
Definition at line 49 of file [shell_support.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/shell_support.h](#)
- [/home/moslevin/Project/R1/stage/src/shell_support.cpp](#)

14.79 SlickButtonControl Class Reference

Inheritance diagram for SlickButtonControl:



Public Member Functions

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

Private Attributes

- `const K_CHAR * m_szCaption`
- `Font_t * m_pstFont`
- `bool m_bState`
- `K_UCHAR m_ucTimeout`
- `void * m_pvCallbackData`
- `ButtonCallback m_pfCallback`

Additional Inherited Members

14.79.1 Detailed Description

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

14.79.2 Member Function Documentation

14.79.2.1 `void SlickButtonControl::Activate (bool bActivate_) [virtual]`

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

Parameters

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

Implements [GuiControl](#).

Definition at line 286 of file [control_slickbutton.cpp](#).

14.79.2.2 `void SlickButtonControl::Draw () [virtual]`

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 51 of file [control_slickbutton.cpp](#).

14.79.2.3 `void SlickButtonControl::Init () [virtual]`

Initialize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.79.2.4 `GuiReturn_t SlickButtonControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

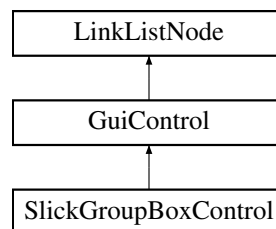
Definition at line 164 of file [control_slickbutton.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/control_slickbutton.h](#)
- [/home/moslevin/Project/R1/stage/src/control_slickbutton.cpp](#)

14.80 SlickGroupBoxControl Class Reference

Inheritance diagram for SlickGroupBoxControl:



Public Member Functions

- virtual void [Init](#) ()
Initialiize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetFont](#) ([Font_t](#) *pstFont_)
- void [SetCaption](#) (const K_CHAR *pcCaption_)
- void [SetBGColor](#) (COLOR uColor_)

Private Attributes

- [Font_t](#) * [m_pstFont](#)
- const K_CHAR * [m_pcCaption](#)
- COLOR [m_uBGColor](#)

Additional Inherited Members

14.80.1 Detailed Description

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

14.80.2 Member Function Documentation

14.80.2.1 `virtual void SlickGroupBoxControl::Activate (bool bActivate_) [inline], [virtual]`

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

Parameters

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

Implements [GuiControl](#).

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

14.80.2.2 void SlickGroupBoxControl::Draw () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

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

14.80.2.3 virtual void SlickGroupBoxControl::Init () [inline],[virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.80.2.4 virtual GuiReturn_t SlickGroupBoxControl::ProcessEvent (GuiEvent_t * *pstEvent_*) [inline],[virtual]

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

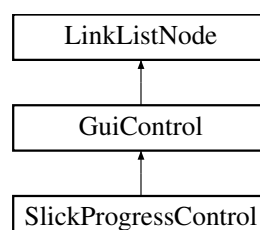
Definition at line 34 of file [control_slickgroupbox.h](#).

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

- [/home/moslevin/Project/R1/stage/src/control_slickgroupbox.h](#)
- [/home/moslevin/Project/R1/stage/src/control_slickgroupbox.cpp](#)

14.81 SlickProgressControl Class Reference

Inheritance diagram for SlickProgressControl:



Public Member Functions

- virtual void [Init](#) ()
Initialiize the control - must be called before use.
- virtual void [Draw](#) ()
Redraw the control "cleanly".
- virtual [GuiReturn_t](#) [ProcessEvent](#) ([GuiEvent_t](#) *pstEvent_)
Process an event sent to the control.
- virtual void [Activate](#) (bool bActivate_)
Activate or deactivate the current control - used when switching from one active control to another.
- void [SetProgress](#) (K_UCHAR ucProgress_)

Private Attributes

- K_UCHAR [m_ucProgress](#)

Additional Inherited Members

14.81.1 Detailed Description

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

14.81.2 Member Function Documentation

14.81.2.1 virtual void [SlickProgressControl::Activate](#) (bool *bActivate_*) [inline],[virtual]

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

Parameters

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

Implements [GuiControl](#).

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

14.81.2.2 void [SlickProgressControl::Draw](#) () [virtual]

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 33 of file [control_slickprogress.cpp](#).

14.81.2.3 void [SlickProgressControl::Init](#) () [virtual]

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

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

14.81.2.4 `GuiReturn_t SlickProgressControl::ProcessEvent (GuiEvent_t * pstEvent_) [virtual]`

Process an event sent to the control.

Subclass specific implementation.

Parameters

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

Implements [GuiControl](#).

Definition at line 107 of file [control_slickprogress.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/control_slickprogress.h](#)
- [/home/moslevin/Project/R1/stage/src/control_slickprogress.cpp](#)

14.82 Slip Class Reference

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

```
#include <slip.h>
```

Public Member Functions

- void [SetDriver](#) ([Driver](#) *pclDriver_)
Set the driver to attach to this object.
- [Driver](#) * [GetDriver](#) ()
Return the pointer to the driver attached to this object.
- void [WriteData](#) (K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_)
Write a packet of data in the FunkenSlip format.
- K_USHORT [ReadData](#) (K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_)
Read a packet from a specified device, parse, and copy to a specified output buffer.
- void [WriteVector](#) (K_UCHAR ucChannel_, [SlipDataVector](#) *astData_, K_USHORT usLen_)
Write a single message composed of multiple data-vector fragments.
- void [SendAck](#) ()
Send an acknowledgement character to the host.
- void [SendNack](#) ()
Send a negative-acknowledgement character to the host.

Static Public Member Functions

- static K_USHORT [EncodeByte](#) (K_UCHAR ucChar_, K_UCHAR *aucBuf_)
Encode a single byte into a stream, returning the size of the encoded value (either 1 or 2 bytes).
- static K_USHORT [DecodeByte](#) (K_UCHAR *ucChar_, const K_UCHAR *aucBuf_)
Decode a byte from a stream into a specified value.

Private Member Functions

- void [WriteByte](#) (K_UCHAR ucData_)

Private Attributes

- [Driver](#) * `m_pclDriver`

14.82.1 Detailed Description

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

Definition at line 70 of file [slip.h](#).

14.82.2 Member Function Documentation

14.82.2.1 `K_USHORT Slip::DecodeByte (K_UCHAR * ucChar_, const K_UCHAR * aucBuf_) [static]`

Decode a byte from a stream into a specified value.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

<i>ucChar_</i>	Destination K_CHAR
<i>aucBuf_</i>	Source buffer

Returns

bytes read, or 0 on terminating character (192)

Definition at line 56 of file [slip.cpp](#).

14.82.2.2 `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](#).

14.82.2.3 `Driver* Slip::GetDriver () [inline]`

Return the pointer to the driver attached to this object.

Returns

Pointer to the driver attached

Definition at line 85 of file [slip.h](#).

14.82.2.4 `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](#).

14.82.2.5 void Slip::SetDriver (Driver * *pclDriver_*) [inline]

Set the driver to attach to this object.

Parameters

<i>pclDriver_</i>	Pointer to the driver to attach
-------------------	---------------------------------

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

14.82.2.6 void Slip::WriteData (K_UCHAR *ucChannel_*, const K_CHAR * *aucBuf_*, K_USHORT *usLen_*)

Write a packet of data in the FunkenSlip format.

Returns the number of bytes from the source array that were processed, (1 or 2), or 0 if an end-of-packet (192) was encountered.

Parameters

<i>ucChannel_</i>	Channel to encode the packet to
<i>aucBuf_</i>	Payload to encode
<i>usLen_</i>	Length of payload data

Definition at line 164 of file [slip.cpp](#).

14.82.2.7 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/Project/R1/stage/src/slip.h](#)
- [/home/moslevin/Project/R1/stage/src/slip.cpp](#)

14.83 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.

14.83.1 Detailed Description

Data structure used for vector-based SLIP data transmission.

Allows for building and transmitting complex data structures without having to copy data into intermediate buffers.

Definition at line 59 of file [slip.h](#).

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

- [/home/moslevin/Project/R1/stage/src/slip.h](#)

14.84 SlipMux Class Reference

Static-class which implements a multiplexed stream of SLIP data over a single interface.

```
#include <slip_mux.h>
```

Static Public Member Functions

- static void [Init](#) (const K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT usTxSize_, K_UCHAR *aucTx_)
Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.
- static void [InstallHandler](#) (K_UCHAR ucChannel_, Slip_Channel pfHandler_)
Install a slip handler function for the given communication channel.
- static void [MessageReceive](#) ()
Wait for a valid packet to arrive, and call the appropriate handler function for the channel the message was attached to.
- static Driver * [GetDriver](#) ()
Return the pointer of the current driver used by the [SlipMux](#) module.
- static MessageQueue * [GetQueue](#) ()
Return the pointer to the message queue attached to the slip mux channel.
- static void [SetQueue](#) (MessageQueue *pclMessageQueue_)
Set the message queue that will receive the notification when the slip mux channel has received data.
- static Slip * [GetSlip](#) ()
Return the pointer to the [SlipMux](#)' Slip object.

Static Private Attributes

- static MessageQueue * [m_pclMessageQueue](#)
- static Driver * [m_pclDriver](#)
- static Slip_Channel [m_apfChannelHandlers](#) [SLIP_CHANNEL_COUNT] = {0}
- static K_UCHAR [m_aucData](#) [SLIP_BUFFER_SIZE]
- static Semaphore [m_clSlipSem](#)
- static Slip [m_clSlip](#)

14.84.1 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](#).

14.84.2 Member Function Documentation

14.84.2.1 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](#).

14.84.2.2 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](#).

14.84.2.3 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](#).

14.84.2.4 void SlipMux::Init (const K_CHAR * pcDriverPath_, K_USHORT usRxSize_, K_UCHAR * aucRx_, K_USHORT usTxSize_, K_UCHAR * aucTx_) [static]

Attach a driver to the Slip-stream multiplexer and initialize the internal data associated with the module.

Must be called before any of the other functions in this module are called.

Parameters

<i>pcDriverPath_</i>	Filesystem path to the driver to attach to
<i>usRxSize_</i>	Size of the RX Buffer to attach to the driver
<i>aucRx_</i>	Pointer to the RX Buffer to attach to the driver
<i>usTxSize_</i>	Size of the TX Buffer to attach to the driver
<i>aucTx_</i>	Pointer to the TX Buffer to attach to the driver

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

14.84.2.5 void SlipMux::InstallHandler (K_UCHAR ucChannel_, Slip_Channel pfHandler_) [static]

Install a slip handler function for the given communication channel.

Parameters

<i>ucChannel_</i>	Channel to attach the handler to
<i>pfHandler_</i>	Pointer to the handler function to attach

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

14.84.2.6 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](#).

14.84.2.7 static void SlipMux::SetQueue (MessageQueue * *pcMessageQueue_*) [inline],[static]

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

Parameters

<i>pcMessage-Queue_</i>	Pointer to the message queue to use for notification.
-------------------------	---

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

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

- [/home/moslevin/Project/R1/stage/src/slip_mux.h](#)
- [/home/moslevin/Project/R1/stage/src/slip_mux.cpp](#)

14.85 SlipTerm Class Reference

Class implementing a simple debug terminal interface.

```
#include <slipterm.h>
```

Public Member Functions

- void [Init](#) ()
Initialize the terminal by opening a handle to the serial interface attached at /dev/tty.
- void [PrintLn](#) (const char *szLine_)
Print a string of text to the SLIP interface, multiplexed using the FunkenSlip terminal channel.
- void [PrintLn](#) (K_UCHAR ucSeverity_, const char *szLine_)
Print a string of text to the SLIP interface, but only if the current logging verbosity level is greater than or equal to the specified message severity.
- void [SetVerbosity](#) (K_UCHAR ucLevel_)
Set the logging verbosity level - the minimum severity level that will be printed to the terminal.

Private Member Functions

- K_USHORT [StrLen](#) (const char *szString_)
Quick 'n' dirty StrLen functionality used for printing the string.

Private Attributes

- K_UCHAR [m_ucVerbosity](#)
level greater than this Are not displayed.
- Slip [m_clSlip](#)
Slip object that this module interfaces with.

14.85.1 Detailed Description

Class implementing a simple debug terminal interface.

This is useful for printf style debugging.

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

14.85.2 Member Function Documentation

14.85.2.1 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](#).

14.85.2.2 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](#).

14.85.2.3 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](#).

14.85.2.4 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](#).

14.85.2.5 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](#).

14.85.3 Member Data Documentation**14.85.3.1 K_UCHAR SlipTerm::m_ucVerbosity [private]**

level greater than this Are not displayed.

Verbosity level. Messages with a severity

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

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

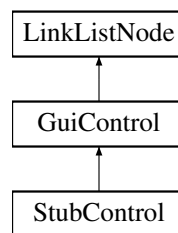
- [/home/moslevin/Project/R1/stage/src/slipterm.h](#)
- [/home/moslevin/Project/R1/stage/src/slipterm.cpp](#)

14.86 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**14.86.1 Detailed Description**

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

Definition at line 797 of file [gui.h](#).

14.86.2 Member Function Documentation

14.86.2.1 `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 803 of file [gui.h](#).

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

Redraw the control "cleanly".

Subclass specific.

Implements [GuiControl](#).

Definition at line 801 of file [gui.h](#).

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

Initialiize the control - must be called before use.

Implementation is subclass specific.

Implements [GuiControl](#).

Definition at line 800 of file [gui.h](#).

14.86.2.4 `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 802 of file [gui.h](#).

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

- [/home/moslevin/Project/R1/stage/src/gui.h](#)

14.87 SystemHeap Class Reference

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

```
#include <system_heap.h>
```

Static Public Member Functions

- static void [Init](#) (void)

Init Initialize the system heap prior to usage.

- static void * [Alloc](#) (K_USHORT usSize_)

Alloc allocate a block of data from the heap.

- static void [Free](#) (void *pvData_)

Free free a block of data previously allocated from the heap.

Static Private Attributes

- static K_UCHAR [m_pucRawHeap](#) [HEAP_RAW_SIZE]

Raw heap buffer.

- static [HeapConfig](#) [m_pclSystemHeapConfig](#) [HEAP_NUM_SIZES+1]

Heap configuration metadata.

- static [FixedHeap](#) [m_clSystemHeap](#)

Heap management object.

- static bool [m_bInit](#)

True if initialized, false if uninitialized.

14.87.1 Detailed Description

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

Definition at line 189 of file [system_heap.h](#).

14.87.2 Member Function Documentation

14.87.2.1 void * SystemHeap::Alloc (K_USHORT usSize_) [static]

[Alloc](#) allocate a block of data from the heap.

Parameters

usSize_	size of the block (in bytes) to allocate
-------------------------	--

Returns

pointer to a block of data allocated from the heap, or NULL on failure.

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

14.87.2.2 void SystemHeap::Free (void * pvData_) [static]

[Free](#) free a block of data previously allocated from the heap.

Parameters

pvData_	Pointer to a block of data allocated from the system heap
-------------------------	---

Definition at line 140 of file [system_heap.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/system_heap.h](#)
- [/home/moslevin/Project/R1/stage/src/system_heap.cpp](#)

14.88 TextFX_t Struct Reference

Public Attributes

- K_UCHAR [ucFlags](#)
Text effects applied.
- COLOR [uBGColor](#)
Background color for opaque backgrounds.
- K_USHORT [usRotateDeg](#)
Rotation in degrees.
- K_USHORT [usScaleX100](#)
Scaling factor, fixed point modulo 100.
- K_USHORT [usScaleY100](#)
Scaling factor, fixed point modulo 100.

14.88.1 Detailed Description

Definition at line 160 of file [draw.h](#).

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

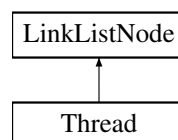
- [/home/moslevin/Project/R1/stage/src/draw.h](#)

14.89 Thread Class Reference

Object providing fundamental multitasking support in the kernel.

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

Inheritance diagram for Thread:



Public Member Functions

- void [Init](#) (K_WORD *paucStack_, K_USHORT usStackSize_, K_UCHAR ucPriority_, [ThreadEntry_t](#) pfEntryPoint_, void *pvArg_)
Initialize a thread prior to its use.
- void [Start](#) ()
Start the thread - remove it from the stopped list, add it to the scheduler's list of threads (at the thread's set priority), and continue along.
- void [Stop](#) ()
Stop a thread that's actively scheduled without destroying its stacks.
- [ThreadList](#) * [GetOwner](#) (void)
Return the [ThreadList](#) where the thread belongs when it's in the active/ready state in the scheduler.
- [ThreadList](#) * [GetCurrent](#) (void)
Return the [ThreadList](#) where the thread is currently located.
- K_UCHAR [GetPriority](#) (void)

- Return the priority of the current thread.*

 - K_UCHAR [GetCurPriority](#) (void)
- Return the priority of the current thread.*

 - void [SetQuantum](#) (K_USHORT usQuantum_)
- Set the thread's round-robin execution quantum.*

 - K_USHORT [GetQuantum](#) (void)
- Get the thread's round-robin execution quantum.*

 - void [SetCurrent](#) (ThreadList *pclNewList_)
- Set the thread's current to the specified thread list.*

 - void [SetOwner](#) (ThreadList *pclNewList_)
- Set the thread's owner to the specified thread list.*

 - void [SetPriority](#) (K_UCHAR ucPriority_)
- Set the priority of the [Thread](#) (running or otherwise) to a different level.*

 - void [InheritPriority](#) (K_UCHAR ucPriority_)
- Allow the thread to run at a different priority level (temporarily) for the purpose of avoiding priority inversions.*

 - void [Exit](#) ()
- Remove the thread from being scheduled again.*

 - void [SetID](#) (K_UCHAR ucID_)
- Set an 8-bit ID to uniquely identify this thread.*

 - K_UCHAR [GetID](#) ()
- Return the 8-bit ID corresponding to this thread.*

 - K_USHORT [GetStackSlack](#) ()
- Performs a (somewhat lengthy) check on the thread stack to check the amount of stack margin (or "slack") remaining on the stack.*

 - K_USHORT [GetEventFlagMask](#) ()
- GetEventFlagMask returns the thread's current event-flag mask, which is used in conjunction with the [EventFlag](#) blocking object type.*

 - void [SetEventFlagMask](#) (K_USHORT usMask_)
- SetEventFlagMask Sets the active event flag bitfield mask.*

 - void [SetEventFlagMode](#) (EventFlagOperation_t eMode_)
- SetEventFlagMode Sets the active event flag operation mode.*

 - EventFlagOperation_t [GetEventFlagMode](#) ()
- GetEventFlagMode Returns the thread's event flag's operating mode.*

 - [Timer](#) * [GetTimer](#) ()
- Return a pointer to the thread's timer object.*

 - void [SetExpired](#) (K_BOOL bExpired_)
- Return a pointer to the thread's timer object.*

 - K_BOOL [GetExpired](#) ()

Static Public Member Functions

- static void [Sleep](#) (K_ULONG ulTimeMs_)

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

Put the thread to sleep for the specified time (in microseconds).
- static void [Yield](#) (void)

Yield the thread - this forces the system to call the scheduler and determine what thread should run next.

Private Member Functions

- void [SetPriorityBase](#) (K_UCHAR ucPriority_)

Static Private Member Functions

- static void [ContextSwitchSWI](#) (void)
This code is used to trigger the context switch interrupt.

Private Attributes

- K_WORD * [m_pwStackTop](#)
Pointer to the top of the thread's stack.
- K_WORD * [m_pwStack](#)
Pointer to the thread's stack.
- K_USHORT [m_usStackSize](#)
Size of the stack (in bytes)
- K_USHORT [m_usQuantum](#)
Thread quantum (in milliseconds)
- K_UCHAR [m_ucThreadID](#)
Thread ID.
- K_UCHAR [m_ucPriority](#)
Default priority of the thread.
- K_UCHAR [m_ucCurPriority](#)
Current priority of the thread (priority inheritance)
- [ThreadEntry_t](#) [m_pfEntryPoint](#)
The entry-point function called when the thread starts.
- void * [m_pvArg](#)
Pointer to the argument passed into the thread's entrypoint.
- K_USHORT [m_usFlagMask](#)
Event-flag mask.
- [EventFlagOperation_t](#) [m_eFlagMode](#)
Event-flag mode.
- [Timer](#) [m_clTimer](#)
Timer used for blocking-object timeouts.
- K_BOOL [m_bExpired](#)
- [ThreadList](#) * [m_pclCurrent](#)
Pointer to the thread-list where the thread currently resides.
- [ThreadList](#) * [m_pclOwner](#)
Pointer to the thread-list where the thread resides when active.

Friends

- class [ThreadPort](#)

Additional Inherited Members

14.89.1 Detailed Description

Object providing fundamental multitasking support in the kernel.

Definition at line 57 of file [thread.h](#).

14.89.2 Member Function Documentation

14.89.2.1 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 353 of file [thread.cpp](#).

14.89.2.2 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 151 of file [thread.cpp](#).

14.89.2.3 K_UCHAR Thread::GetCurPriority(void) [inline]

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 160 of file [thread.h](#).

14.89.2.4 ThreadList * Thread::GetCurrent(void) [inline]

Return the [ThreadList](#) where the thread is currently located.

Returns

Pointer to the thread's current list

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

14.89.2.5 K_USHORT Thread::GetEventFlagMask() [inline]

GetEventFlagMask returns the thread's current event-flag mask, which is used in conjunction with the [EventFlag](#) blocking object type.

Returns

A copy of the thread's event flag mask

Definition at line 313 of file [thread.h](#).

14.89.2.6 EventFlagOperation_t Thread::GetEventFlagMode() [inline]

GetEventFlagMode Returns the thread's event flag's operating mode.

Returns

The thread's event flag mode.

Definition at line 332 of file [thread.h](#).

14.89.2.7 K_UCHAR Thread::GetID () [inline]

Return the 8-bit ID corresponding to this thread.

Returns

[Thread](#)'s 8-bit ID, set by the user

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

14.89.2.8 ThreadList * Thread::GetOwner (void) [inline]

Return the [ThreadList](#) where the thread belongs when it's in the active/ready state in the scheduler.

Returns

Pointer to the [Thread](#)'s owner list

Definition at line 132 of file [thread.h](#).

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

Return the priority of the current thread.

Returns

Priority of the current thread

Definition at line 151 of file [thread.h](#).

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

Get the thread's round-robin execution quantum.

Returns

The thread's quantum

Definition at line 179 of file [thread.h](#).

14.89.2.11 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 242 of file [thread.cpp](#).

14.89.2.12 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 346 of file [thread.cpp](#).

14.89.2.13 void Thread::Init (K_WORD * *paucStack_*, K_USHORT *usStackSize_*, K_UCHAR *ucPriority_*, ThreadEntry_t *pfEntryPoint_*, void * *pvArg_*)

Initialize a thread prior to its use.

Initialized threads are placed in the stopped state, and are not scheduled until the thread's start method has been invoked first.

Parameters

<i>paucStack_</i>	Pointer to the stack to use for the thread
<i>usStackSize_</i>	Size of the stack (in bytes)
<i>ucPriority_</i>	Priority of the thread (0 = idle, 7 = max)
<i>pfEntryPoint_</i>	This is the function that gets called when the thread is started
<i>pvArg_</i>	Pointer to the argument passed into the thread's entypoint function.

< Default round-robin thread quantum of 4ms

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

14.89.2.14 void Thread::SetCurrent (ThreadList * *pciNewList_*) [inline]

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

Parameters

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

Definition at line 189 of file [thread.h](#).

14.89.2.15 void Thread::SetEventFlagMask (K_USHORT *usMask_*) [inline]

SetEventFlagMask Sets the active event flag bitfield mask.

Parameters

<i>usMask_</i>	
----------------	--

Definition at line 319 of file [thread.h](#).

14.89.2.16 void Thread::SetEventFlagMode (EventFlagOperation_t *eMode_*) [inline]

SetEventFlagMode Sets the active event flag operation mode.

Parameters

<i>eMode_</i>	Event flag operation mode, defines the logical operator to apply to the event flag.
---------------	---

Definition at line 326 of file [thread.h](#).

14.89.2.17 void Thread::SetID (K_UCHAR *ucID_*) [inline]

Set an 8-bit ID to uniquely identify this thread.

Parameters

<i>ucID_</i>	8-bit Thread ID, set by the user
--------------	--

Definition at line 279 of file [thread.h](#).

14.89.2.18 void Thread::SetOwner (ThreadList * *pciNewList_*) [inline]

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

Parameters

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

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

14.89.2.19 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 303 of file [thread.cpp](#).

14.89.2.20 void Thread::SetPriorityBase (K_UCHAR *ucPriority_*) [private]

Parameters

<i>ucPriority_</i>	
--------------------	--

Definition at line 293 of file [thread.cpp](#).

14.89.2.21 void Thread::SetQuantum (K_USHORT *usQuantum_*) [inline]

Set the thread's round-robin execution quantum.

Parameters

<i>usQuantum_</i>	Thread 's execution quantum (in milliseconds)
-------------------	---

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

14.89.2.22 void Thread::Sleep (K_ULONG *ulTimeMs_*) [static]

Put the thread to sleep for the specified time (in milliseconds).

Actual time slept may be longer (but not less than) the interval specified.

Parameters

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

Definition at line 197 of file [thread.cpp](#).

14.89.2.23 void Thread::Stop (void)

Stop a thread that's actively scheduled without destroying its stacks.

Stopped threads can be restarted using the [Start\(\)](#) API.

Definition at line 123 of file [thread.cpp](#).

14.89.2.24 void Thread::USleep (K_ULONG ulTimeUs_) [static]

Put the thread to sleep for the specified time (in microseconds).

Actual time slept may be longer (but not less than) the interval specified.

Parameters

<i>ulTimeUs_</i>	Time to sleep (in microseconds)
------------------	---------------------------------

Definition at line 219 of file [thread.cpp](#).

14.89.2.25 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 263 of file [thread.cpp](#).

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

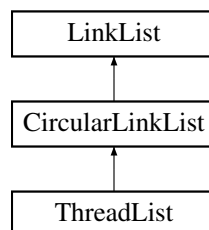
- [/home/moslevin/Project/R1/stage/src/thread.h](#)
- [/home/moslevin/Project/R1/stage/src/thread.cpp](#)

14.90 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

14.90.1 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](#).

14.90.2 Member Function Documentation

14.90.2.1 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](#).

14.90.2.2 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](#).

14.90.2.3 [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](#).

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

Remove the specified thread from the threadlist.

Parameters

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

Reimplemented from [CircularLinkList](#).

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

14.90.2.5 void ThreadList::SetFlagPointer (K_UCHAR * pucFlag_)

Set the pointer to a bitmap to use for this threadlist.

Once again, only needed when the threadlist is being used for scheduling purposes.

Parameters

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

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

14.90.2.6 void ThreadList::SetPriority (K_UCHAR ucPriority_)

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

Parameters

<i>ucPriority_</i>	Priority level of the thread list
--------------------	-----------------------------------

Definition at line 34 of file [threadlist.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/threadlist.h](#)
- [/home/moslevin/Project/R1/stage/src/threadlist.cpp](#)

14.91 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](#)

14.91.1 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](#).

14.91.2 Member Function Documentation

14.91.2.1 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/Project/R1/stage/src/threadport.h](#)
- [/home/moslevin/Project/R1/stage/src/threadport.cpp](#)

14.92 Tile_8x8 Class Reference

Public Member Functions

- void [LoadTile](#) ([TileDef_t](#) *pstTileDef_)
LoadTile.
- void [Render](#) ([GraphicsDriver](#) *pclDriver_, K_USHORT usX_, K_USHORT usY_)
Render.

Private Attributes

- COLOR [m_auTileBuffer](#) [TILE_8x8_BUFFER_SIZE]
m_auTileBuffer Object's local storage for tile data
- K_UCHAR [m_ucWidth](#)
m_ucWidth Width of the tile (may be smaller than width of buffer)
- K_UCHAR [m_ucHeight](#)
m_ucHeight Height of the tile (may be smaler than the height of buffer)

14.92.1 Detailed Description

Definition at line 63 of file [tiles.h](#).

14.92.2 Member Function Documentation

14.92.2.1 void Tile_8x8::LoadTile (TileDef_t * pstTileDef_)

LoadTile.

Load the tile specified by pstTileDef_ into memory. This takes some time as it parses the indexed colors, does a lookup, and then writes to the local tile buffer. Once a tile has been loaded, it can be rendered any number of times.

Parameters

<i>pstTileDef_</i>	Pointer to a struct containing configuration data for the tile to be loaded.
--------------------	--

Definition at line 24 of file [tiles.cpp](#).

14.92.2.2 void Tile_8x8::Render (GraphicsDriver * pcdDriver_, K_USHORT usX_, K_USHORT usY_)

Render.

Render loaded tile data to a specific location on a specified display.

Parameters

<i>pcdDriver_</i>	Pointer to the graphics driver to render with
<i>usX_</i>	Leftmost pixel index
<i>usY_</i>	Topmost pixel index

Definition at line 51 of file [tiles.cpp](#).

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

- /home/moslevin/Project/R1/stage/src/tiles.h
- /home/moslevin/Project/R1/stage/src/tiles.cpp

14.93 TileDef_t Struct Reference

[TileDef_t](#) Structure defining parameters for a color-indexed tile.

```
#include <tiles.h>
```

Public Attributes

- [TileFormat_t m_eFormat](#)
Color-indexing of the tile (bits-per-pixel)
- [K_UCHAR * m_pucData](#)
Pointer to color-indexed tile data.
- [COLOR * m_puPalette](#)
Pointer to a palette assigned to this tile.
- [K_UCHAR m_ucHeight](#)
Height of the tile (in pixels)
- [K_UCHAR m_ucWidth](#)
Width of the tile (in pixels)

14.93.1 Detailed Description

[TileDef_t](#) Structure defining parameters for a color-indexed tile.

Definition at line 48 of file [tiles.h](#).

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

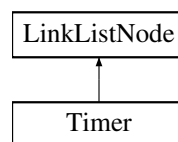
- /home/moslevin/Project/R1/stage/src/tiles.h

14.94 Timer Class Reference

[Timer](#) - an event-driven execution context based on a specified time interval.

```
#include <timerlist.h>
```

Inheritance diagram for Timer:



Public Member Functions

- [Timer](#) ()
Default Constructor - zero-initializes all internal data.
- void [Init](#) ()
Re-initialize the [Timer](#) to default values.
- void [Start](#) (bool bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *pvData_)
Start a timer using default ownership, using repeats as an option, and millisecond resolution.
- void [Start](#) (bool bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_, TimerCallback_t pfCallback_, void *pvData_)
Start a timer using default ownership, using repeats as an option, and millisecond resolution.
- void [Stop](#) ()
Stop a timer already in progress.
- void [SetFlags](#) (K_UCHAR ucFlags_)
Set the timer's flags based on the bits in the ucFlags_ argument.
- void [SetCallback](#) (TimerCallback_t pfCallback_)
Define the callback function to be executed on expiry of the timer.
- void [SetData](#) (void *pvData_)
Define a pointer to be sent to the timer callback on timer expiry.
- void [SetOwner](#) (Thread *pOwner_)
Set the owner-thread of this timer object (all timers must be owned by a thread).
- void [SetIntervalTicks](#) (K_ULONG ulTicks_)
Set the timer expiry in system-ticks (platform specific!)
- void [SetIntervalSeconds](#) (K_ULONG ulSeconds_)
! The next three cost us 330 bytes of flash on AVR...
- K_ULONG [GetInterval](#) ()
- void [SetIntervalMSeconds](#) (K_ULONG ulMSeconds_)
Set the timer expiry interval in milliseconds (platform agnostic)
- void [SetIntervalUSeconds](#) (K_ULONG ulUSeconds_)

Set the timer expiry interval in microseconds (platform agnostic)

- void [SetTolerance](#) (K_ULONG ulTicks_)

Set the timer's maximum tolerance in order to synchronize timer processing with other timers in the system.

Private Attributes

- K_UCHAR [m_ucFlags](#)
Flags for the timer, defining if the timer is one-shot or repeated.
- TimerCallback_t [m_pfCallback](#)
Pointer to the callback function.
- K_ULONG [m_ulInterval](#)
Interval of the timer in timer ticks.
- K_ULONG [m_ulTimeLeft](#)
Time remaining on the timer.
- K_ULONG [m_ulTimerTolerance](#)
Maximum tolerance (used for timer harmonization)
- Thread * [m_pclOwner](#)
Pointer to the owner thread.
- void * [m_pvData](#)
Pointer to the callback data.

Friends

- class **TimerList**

Additional Inherited Members

14.94.1 Detailed Description

[Timer](#) - an event-driven execution context based on a specified time interval.

This inherits from a [LinkedListNode](#) for ease of management by a global [TimerList](#) object.

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

14.94.2 Member Function Documentation

14.94.2.1 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 160 of file [timerlist.h](#).

14.94.2.2 void [Timer::SetData](#) (void * *pvData_*) [[inline](#)]

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

Parameters

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

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

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

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

Parameters

<i>ucFlags_</i>	Flags to assign to the timer object. TIMERLIST_FLAG_ONE_SHOT for a one-shot timer, 0 for a continuous timer.
-----------------	--

Definition at line 151 of file [timerlist.h](#).

14.94.2.4 void Timer::SetIntervalMSeconds (K_ULONG *uIMSeconds_*)

Set the timer expiry interval in milliseconds (platform agnostic)

Parameters

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

Definition at line 304 of file [timerlist.cpp](#).

14.94.2.5 void Timer::SetIntervalSeconds (K_ULONG *uISeconds_*)

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

Set the timer expiry interval in seconds (platform agnostic)

Parameters

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

Definition at line 298 of file [timerlist.cpp](#).

14.94.2.6 void Timer::SetIntervalTicks (K_ULONG *uITicks_*)

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

Parameters

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

Definition at line 290 of file [timerlist.cpp](#).

14.94.2.7 void Timer::SetIntervalUSeconds (K_ULONG *uIUSeconds_*)

Set the timer expiry interval in microseconds (platform agnostic)

Parameters

<i>uIUSeconds_</i>	Time in microseconds
--------------------	----------------------

Definition at line 310 of file [timerlist.cpp](#).

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

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

Parameters

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

Definition at line 179 of file [timerlist.h](#).

14.94.2.9 void Timer::SetTolerance (K_ULONG ulTicks_)

Set the timer's maximum tolerance in order to synchronize timer processing with other timers in the system.

Parameters

<i>ulTicks_</i>	Maximum tolerance in ticks
-----------------	----------------------------

Definition at line 316 of file [timerlist.cpp](#).

14.94.2.10 void Timer::Start (bool bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void * pvData_)

Start a timer using default ownership, using repeats as an option, and millisecond resolution.

Parameters

<i>bRepeat_</i>	0 - timer is one-shot. 1 - timer is repeating.
<i>ulIntervalMs_</i>	- Interval of the timer in milliseconds
<i>pfCallback_</i>	- Function to call on timer expiry
<i>pvData_</i>	- Data to pass into the callback function

Definition at line 259 of file [timerlist.cpp](#).

14.94.2.11 void Timer::Start (bool bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_, TimerCallback_t pfCallback_, void * pvData_)

Start a timer using default ownership, using repeats as an option, and millisecond resolution.

Parameters

<i>bRepeat_</i>	0 - timer is one-shot. 1 - timer is repeating.
<i>ulIntervalMs_</i>	- Interval of the timer in milliseconds
<i>ulToleranceMs</i>	- Allow the timer expiry to be delayed by an additional maximum time, in order to have as many timers expire at the same time as possible.
<i>pfCallback_</i>	- Function to call on timer expiry
<i>pvData_</i>	- Data to pass into the callback function

Definition at line 277 of file [timerlist.cpp](#).

14.94.2.12 void Timer::Stop (void)

Stop a timer already in progress.

Has no effect on timers that have already been stopped.

Definition at line 284 of file [timerlist.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/timerlist.h](#)
- [/home/moslevin/Project/R1/stage/src/timerlist.cpp](#)

14.95 TimerEvent_t Struct Reference

[Timer](#) UI event structure.

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

Public Attributes

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

14.95.1 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:

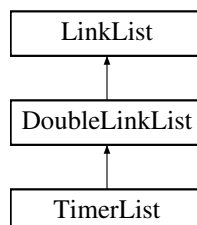
- [/home/moslevin/Project/R1/stage/src/gui.h](#)

14.96 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

14.96.1 Detailed Description

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

Definition at line 261 of file [timerlist.h](#).

14.96.2 Member Function Documentation

14.96.2.1 void TimerList::Add (Timer * *pclListNode_*)

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

Parameters

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

Definition at line 49 of file [timerlist.cpp](#).

14.96.2.2 void TimerList::Init (void)

Initialize the [TimerList](#) object.

Must be called before using the object.

Definition at line 42 of file [timerlist.cpp](#).

14.96.2.3 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 114 of file [timerlist.cpp](#).

14.96.2.4 void TimerList::Remove (Timer * *pclListNode_*)

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

Parameters

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

Definition at line 97 of file [timerlist.cpp](#).

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

- [/home/moslevin/Project/R1/stage/src/timerlist.h](#)
- [/home/moslevin/Project/R1/stage/src/timerlist.cpp](#)

14.97 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](#).

14.97.1 Detailed Description

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

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

14.97.2 Member Function Documentation

14.97.2.1 void [TimerScheduler::Add](#) ([Timer](#) * [pclListNode_](#)) [inline],[static]

Add a timer to the timer scheduler.

Adding a timer implicitly starts the timer as well.

Parameters

pclListNode_	Pointer to the timer list node to add
------------------------------	---------------------------------------

Definition at line [330](#) of file [timerlist.h](#).

14.97.2.2 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 [320](#) of file [timerlist.h](#).

14.97.2.3 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 [352](#) of file [timerlist.h](#).

14.97.2.4 void [TimerScheduler::Remove](#) ([Timer](#) * [pclListNode_](#)) [inline],[static]

Remove a timer from the timer scheduler.

May implicitly stop the timer if this is the only active timer scheduled.

Parameters

<code>pcListNode_</code>	Pointer to the timer list node to remove
--------------------------	--

Definition at line 341 of file [timerlist.h](#).

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

- [/home/moslevin/Project/R1/stage/src/timerlist.h](#)
- [/home/moslevin/Project/R1/stage/src/timerlist.cpp](#)

14.98 Token_t Struct Reference

Token descriptor struct format.

```
#include <memutil.h>
```

Public Attributes

- `const K_CHAR * pcToken`
Pointer to the beginning of the token string.
- `K_UCHAR ucLen`
Length of the token (in bytes)

14.98.1 Detailed Description

Token descriptor struct format.

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

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

- [/home/moslevin/Project/R1/stage/src/memutil.h](#)

14.99 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)

14.99.1 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/Project/R1/stage/src/gui.h](#)

14.100 UnitTest Class Reference

Class used to implement a simple unit-testing framework.

```
#include <unit_test.h>
```

Public Member Functions

- void [SetName](#) (const K_CHAR *szName_)
Set the name of the test object.
- void [Start](#) ()
Start a new test iteration.
- void [Pass](#) ()
Stop the current iteration (if started), and register that the test was successful.
- void [Fail](#) ()
Stop the current iterations (if started), and register that the current test failed.
- void [ExpectTrue](#) (bool bExpression_)
- void [ExpectFalse](#) (bool bExpression_)
- void [ExpectEquals](#) (bool bVal_, bool bExpression_)
- void [ExpectEquals](#) (K_UCHAR ucVal_, K_UCHAR ucExpression_)
- void [ExpectEquals](#) (K_USHORT usVal_, K_USHORT usExpression_)
- void [ExpectEquals](#) (K_ULONG ulVal_, K_ULONG ulExpression_)
- void [ExpectEquals](#) (K_CHAR cVal_, K_CHAR cExpression_)
- void [ExpectEquals](#) (K_SHORT sVal_, K_SHORT sExpression_)
- void [ExpectEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectEquals](#) (void *pvVal_, void *pvExpression_)
- void [ExpectFailTrue](#) (bool bExpression_)
- void [ExpectFailFalse](#) (bool bExpression_)
- void [ExpectFailEquals](#) (bool bVal_, bool bExpression_)
- void [ExpectFailEquals](#) (K_UCHAR ucVal_, K_UCHAR ucExpression_)
- void [ExpectFailEquals](#) (K_USHORT usVal_, K_USHORT usExpression_)
- void [ExpectFailEquals](#) (K_ULONG ulVal_, K_ULONG ulExpression_)
- void [ExpectFailEquals](#) (K_CHAR cVal_, K_CHAR cExpression_)
- void [ExpectFailEquals](#) (K_SHORT sVal_, K_SHORT sExpression_)
- void [ExpectFailEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectFailEquals](#) (void *pvVal_, void *pvExpression_)
- void [ExpectGreaterThan](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectLessThan](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectGreaterThanEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectLessThanEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectFailGreaterThan](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectFailLessThan](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectFailGreaterThanEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- void [ExpectFailLessThanEquals](#) (K_LONG lVal_, K_LONG lExpression_)
- 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.

14.100.1 Detailed Description

Class used to implement a simple unit-testing framework.

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

14.100.2 Member Function Documentation

14.100.2.1 void UnitTest::Complete () [\[inline\]](#)

Complete the test.

Once a test has been completed, no new iterations can be started (i.e [Start\(\)](#)/[Pass\(\)](#)/[Fail\(\)](#) will have no effect).

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

14.100.2.2 K_USHORT UnitTest::GetFailed () [\[inline\]](#)

Return the number of failed test points/iterations.

Returns

Failed test point/iteration count

Definition at line 193 of file [unit_test.h](#).

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

Get the name of the tests associated with this object.

Returns

Name of the test

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

14.100.2.4 `K_USHORT UnitTest::GetPassed () [inline]`

Return the total number of test points/iterations passed.

Returns

Count of all successful test points/iterations

Definition at line 184 of file [unit_test.h](#).

14.100.2.5 `K_BOOL UnitTest::GetResult () [inline]`

Return the result of the last test.

Returns

Status of the last run test (false = fail, true = pass)

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

14.100.2.6 `K_USHORT UnitTest::GetTotal () [inline]`

Return the total number of iterations/test-points executed.

Returns

Total number of iterations/test-points executed

Definition at line 202 of file [unit_test.h](#).

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

Set the name of the test object.

Parameters

<i>szName_</i>	Name of the tests associated with this object
----------------	---

Definition at line 41 of file [unit_test.h](#).

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

- [/home/moslevin/Project/R1/stage/src/unit_test.h](#)
- [/home/moslevin/Project/R1/stage/src/unit_test.cpp](#)

File Documentation

```

00001  /*-----
00002
00003
00004
00005
00006
00007
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00021  #include "kerneltypes.h"
00022  #include "mark3cfg.h"
00023  #include "atomic.h"
00024  #include "threadport.h"
00025
00026  #if KERNEL_USE_ATOMIC
00027
00028  //-----
00029  K_UCHAR Atomic::Set( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00030  {
00031      K_UCHAR ucRet;
00032      CS_ENTER();
00033      ucRet = *pucSource_;
00034      *pucSource_ = ucVal_;
00035      CS_EXIT();
00036      return ucRet;
00037  }
00038  //-----
00039  K_USHORT Atomic::Set( K_USHORT *pusSource_, K_USHORT usVal_ )
00040  {

```

```

00041     K_USHORT usRet;
00042     CS_ENTER();
00043     usRet = *pusSource_;
00044     *pusSource_ = usVal_;
00045     CS_EXIT();
00046     return usRet;
00047 }
00048 //-----
00049 K_ULONG Atomic::Set( K_ULONG *pulSource_, K_ULONG ulVal_ )
00050 {
00051     K_ULONG ulRet;
00052     CS_ENTER();
00053     ulRet = *pulSource_;
00054     *pulSource_ = ulVal_;
00055     CS_EXIT();
00056     return ulRet;
00057 }
00058
00059 //-----
00060 K_UCHAR Atomic::Add( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00061 {
00062     K_UCHAR ucRet;
00063     CS_ENTER();
00064     ucRet = *pucSource_;
00065     *pucSource_ += ucVal_;
00066     CS_EXIT();
00067     return ucRet;
00068 }
00069
00070 //-----
00071 K_USHORT Atomic::Add( K_USHORT *pusSource_, K_USHORT usVal_ )
00072 {
00073     K_USHORT usRet;
00074     CS_ENTER();
00075     usRet = *pusSource_;
00076     *pusSource_ += usVal_;
00077     CS_EXIT();
00078     return usRet;
00079 }
00080
00081 //-----
00082 K_ULONG Atomic::Add( K_ULONG *pulSource_, K_ULONG ulVal_ )
00083 {
00084     K_ULONG ulRet;
00085     CS_ENTER();
00086     ulRet = *pulSource_;
00087     *pulSource_ += ulVal_;
00088     CS_EXIT();
00089     return ulRet;
00090 }
00091
00092 //-----
00093 K_UCHAR Atomic::Sub( K_UCHAR *pucSource_, K_UCHAR ucVal_ )
00094 {
00095     K_UCHAR ucRet;
00096     CS_ENTER();
00097     ucRet = *pucSource_;
00098     *pucSource_ -= ucVal_;
00099     CS_EXIT();
00100     return ucRet;
00101 }
00102
00103 //-----
00104 K_USHORT Atomic::Sub( K_USHORT *pusSource_, K_USHORT usVal_ )
00105 {
00106     K_USHORT usRet;
00107     CS_ENTER();
00108     usRet = *pusSource_;
00109     *pusSource_ -= usVal_;
00110     CS_EXIT();
00111     return usRet;
00112 }
00113
00114 //-----
00115 K_ULONG Atomic::Sub( K_ULONG *pulSource_, K_ULONG ulVal_ )
00116 {
00117     K_ULONG ulRet;
00118     CS_ENTER();
00119     ulRet = *pulSource_;
00120     *pulSource_ -= ulVal_;
00121     CS_EXIT();
00122     return ulRet;
00123 }
00124
00125 //-----
00126 K_BOOL Atomic::TestAndSet( K_BOOL *pbLock_ )
00127 {

```

```

00128     K_UCHAR ucRet;
00129     CS_ENTER();
00130     ucRet = *pbLock_;
00131     if (!ucRet)
00132     {
00133         *pbLock_ = 1;
00134     }
00135     CS_EXIT();
00136     return ucRet;
00137 }
00138
00139 #endif // KERNEL_USE_ATOMIC

```

15.3 /home/moslevin/Project/R1/stage/src/atomic.h File Reference

Basic Atomic Operations.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "threadport.h"

```

15.3.1 Detailed Description

Basic Atomic Operations.

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

15.4 atomic.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #ifndef __ATOMIC_H__
00022 #define __ATOMIC_H__
00023
00024 #include "kerneltypes.h"
00025 #include "mark3cfg.h"
00026 #include "threadport.h"
00027
00028 #if KERNEL_USE_ATOMIC
00029
00039 class Atomic
00040 {
00041 public:
00048     static K_UCHAR Set( K_UCHAR *pucSource_, K_UCHAR ucVal_ );
00049     static K_USHORT Set( K_USHORT *pusSource_, K_USHORT usVal_ );
00050     static K_ULONG Set( K_ULONG *pulSource_, K_ULONG ulVal_ );
00051
00058     static K_UCHAR Add( K_UCHAR *pucSource_, K_UCHAR ucVal_ );
00059     static K_USHORT Add( K_USHORT *pusSource_, K_USHORT usVal_ );
00060     static K_ULONG Add( K_ULONG *pulSource_, K_ULONG ulVal_ );
00061
00068     static K_UCHAR Sub( K_UCHAR *pucSource_, K_UCHAR ucVal_ );
00069     static K_USHORT Sub( K_USHORT *pusSource_, K_USHORT usVal_ );
00070     static K_ULONG Sub( K_ULONG *pulSource_, K_ULONG ulVal_ );
00071
00086     static K_BOOL TestAndSet( K_BOOL *pbLock );
00087 };
00088
00089 #endif // KERNEL_USE_ATOMIC
00090
00091 #endif //__ATOMIC_H__

```

15.5 /home/moslevin/Project/R1/stage/src/blocking.cpp File Reference

Implementation of base class for blocking objects.

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

Macros

- `#define __FILE_ID__ BLOCKING_CPP`

15.5.1 Detailed Description

Implementation of base class for blocking objects.

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

15.6 blocking.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "mark3cfg.h"
00023 #include "kernel_debug.h"
00024
00025 #include "blocking.h"
00026 #include "thread.h"
00027
00028 //-----
00029 #if defined __FILE_ID__
00030 #undef __FILE_ID__
00031 #endif
00032 #define __FILE_ID__ BLOCKING_CPP
00033
00034 #if KERNEL_USE_SEMAPHORE || KERNEL_USE_MUTEX
00035 //-----
00036 void BlockingObject::Block(Thread *pclThread_)
00037 {
00038     KERNEL_ASSERT( pclThread_ );
00039     KERNEL_TRACE_1( STR_THREAD_BLOCK_1, (K_USHORT)pclThread_>GetID() );
00040
00041     // Remove the thread from its current thread list (the "owner" list)
00042     // ... And add the thread to this object's block list
00043     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

```

15.7 /home/moslevin/Project/R1/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.

15.7.1 Detailed Description

Blocking object base class declarations. A Blocking object in Mark3 is essentially a thread list. Any blocking object implementation (being a semaphore, mutex, event flag, etc.) can be built on top of this class, utilizing the provided functions to manipulate thread location within the [Kernel](#).

Blocking a thread results in that thread becoming de-scheduled, placed in the blocking object's own private list of threads which are waiting on the object.

Unblocking a thread results in the reverse: The thread is moved back to its original location from the blocking list.

The only difference between a blocking object based on this class is the logic used to determine what constitutes a Block or Unblock condition.

For instance, a semaphore Pend operation may result in a call to the Block() method with the currently-executing thread in order to make that thread wait for a semaphore Post. That operation would then invoke the Unblock() method, removing the blocking thread from the semaphore's list, and back into the the appropriate thread inside the scheduler.

Care must be taken when implementing blocking objects to ensure that critical sections are used judiciously, otherwise asynchronous events like timers and interrupts could result in non-deterministic and often catastrophic behavior.

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

15.8 blocking.h

```

00001 /*****
00002
00003
00004
00005
00006
00007

```



```

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 click.
00130             if ((KEYCODE_SPACE == pstEvent_>stKey.ucKeyCode) ||
00131                 (KEYCODE_RETURN == pstEvent_>stKey.ucKeyCode))
00132             {
00133                 if (pstEvent_>stKey.bKeyState)
00134                 {
00135                     m_bState = true;
00136                 }
00137                 else
00138                 {
00139                     m_bState = false;
00140                     if (m_pfCallback)
00141                     {
00142                         m_pfCallback(m_pvCallbackData);
00143                     }
00144                 }
00145                 SetStale();
00146             }
00147         }
00148         break;
00149         case EVENT_TYPE_MOUSE:
00150         {
00151             // Is this control currently in the "active"/pressed state?
00152             if (m_bState)
00153             {
00154                 // Check to see if the movement is out-of-bounds based on the coordinates.
00155                 // If so, de-activate the control
00156                 if (pstEvent_>stMouse.bLeftState)
00157                 {
00158                     if ((pstEvent_>stMouse.usX < GetLeft() + usXOffset) ||
00159                         (pstEvent_>stMouse.usX >= GetLeft() + usXOffset +
00160                          GetWidth()-1) ||
00161                         (pstEvent_>stMouse.usY < GetTop() + usYOffset) ||
00162                         (pstEvent_>stMouse.usY >= GetTop() + usYOffset +
00163                          GetHeight() - 1))
00164                     {
00165                         m_bState = false;
00166                         SetStale();
00167                     }
00168                     // left button state is now up, and the control was previously active.
00169                     // Run the event callback for the mouse, and go from there.
00170                     else
00171                     {
00172                         if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00173                             (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00174                              GetWidth()-1) &&
00175                             (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00176                             (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00177                              GetHeight() - 1))
00178                         {
00179                             m_bState = false;
00180                             SetStale();
00181                             if (m_pfCallback)
00182                             {
00183                                 m_pfCallback(m_pvCallbackData);
00184                             }
00185                         }
00186                     }
00187                 }
00188             }
00189             else if (!m_bState)
00190             {
00191                 // If we registered a down-click in the bounding box, set the state of the
00192                 // control to activated.
00193                 if (pstEvent_>stMouse.bLeftState)
00194                 {
00195                     if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00196                         (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00197                          GetWidth()-1) &&
00198                         (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00199                         (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00200                          GetHeight() - 1))
00201                     {
00202                         m_bState = true;
00203                         SetStale();
00204                     }
00205                 }
00206             }
00207         }
00208     }

```

```
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
00218     // a redraw
00219     if (!bActivate_)
00220     {
00221         m_bState = false;
00222     }
00223     SetStale();
00224 }
```

15.11 /home/moslevin/Project/R1/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_)

15.11.1 Detailed Description

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

Definition in file [control](#) [button.h](#).

15.12 control_button.h

```
00001
00002 /*=====
00003
00004      _   _   _   _   _   _   _   _
00005     / \ / \ / \ / \ / \ / \ / \ / \
00006    |   |   |   |   |   |   |   |   |
00007    |___|___|___|___|___|___|___|___|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 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 *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

```

15.13 /home/moslevin/Project/R1/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 []

15.13.1 Detailed Description

Checkbox Control. A binary On/Off switch control

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

15.13.2 Variable Documentation

15.13.2.1 `const K_UCHAR aucBox[]` [static]

Initial value:

```
=
{ 0x7E,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x81,
  0x7E }
```

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

15.13.2.2 `const K_UCHAR aucCheck[]` [static]

Initial value:

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

Definition at line 42 of file [control_checkbox.cpp](#).

15.14 control_checkbox.cpp

```

00001  /*-----
00002
00003  |_____|_____|_____|_____|_____|_____|_____|_____|
00004  |  /  \  /  \  /  \  /  \  /  \  /  \  /  \  /  \  |
00005  | /    \ /    \ /    \ /    \ /    \ /    \ /    \ |
00006  |/_    _/_    _/_    _/_    _/_    _/_    _/_    _|
00007  |_____|_____|_____|_____|_____|_____|_____|_____|
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00021  #include "gui.h"
00022  #include "kerneltypes.h"
00023  #include "draw.h"
00024  #include "font.h"
00025  #include "control_checkbox.h"
00026
00027  //-----
00028  #define TEXT_X_OFFSET (13)
00029
00030  //-----
00031  static const K_UCHAR aucBox[] =
00032  { 0x7E,
00033    0x81,
00034    0x81,
00035    0x81,

```

```

00036     0x81,
00037     0x81,
00038     0x81,
00039     0x7E };
00040
00041 //-----
00042 static const K_UCHAR aucCheck[] =
00043 { 0,
00044   0,
00045   0x3C,
00046   0x3C,
00047   0x3C,
00048   0x3C,
00049   0,
00050   0 };
00051
00052 //-----
00053 void CheckBoxControl::Init()
00054 {
00055     SetAcceptFocus(true);
00056 }
00057
00058 //-----
00059 void CheckBoxControl::Draw()
00060 {
00061     GraphicsDriver *pclDriver = GetParentWindow()->
    GetDriver();
00062     K_USHORT usX, usY;
00063     K_USHORT usTextWidth;
00064
00065     GetControlOffset(&usX, &usY);
00066
00067     // Draw the box, (and check, if necessary)
00068     {
00069         DrawRectangle_t stRect;
00070
00071         if (GetParentWindow()->IsInFocus(this))
00072         {
00073             stRect.uLineColor = m_uActiveColor;
00074         }
00075         else
00076         {
00077             stRect.uLineColor = m_uBackColor;
00078         }
00079
00080         stRect.uFillColor = m_uBackColor;
00081         stRect.usTop = usY + GetTop();
00082         stRect.usLeft = usX + GetLeft();
00083         stRect.usRight = stRect.usLeft + GetWidth() - 1;
00084         stRect.usBottom = stRect.usTop + GetHeight() - 1;
00085         stRect.bFill = true;
00086         pclDriver->Rectangle(&stRect);
00087
00088         stRect.uLineColor = m_uBoxBGColor;
00089         stRect.uFillColor = m_uBoxBGColor;
00090         stRect.usTop = usY + GetTop() + ((GetHeight() - 5) >> 1) - 1;
00091         stRect.usLeft = usX + GetLeft() + 2;
00092         stRect.usRight = stRect.usLeft + 7;
00093         stRect.usBottom = stRect.usTop + 7;
00094         stRect.bFill = true;
00095         pclDriver->Rectangle(&stRect);
00096     }
00097
00098     {
00099         DrawStamp_t stStamp;
00100         stStamp.uColor = m_uBoxColor;
00101         stStamp.usY = usY + GetTop() + ((GetHeight() - 5) >> 1) - 1;
00102         stStamp.usX = usX + GetLeft() + 2;
00103         stStamp.usWidth = 8;
00104         stStamp.usHeight = 8;
00105         stStamp.pucData = (K_UCHAR*)aucBox;
00106         pclDriver->Stamp(&stStamp);
00107
00108         if (m_bChecked)
00109         {
00110             stStamp.pucData = (K_UCHAR*)aucCheck;
00111             pclDriver->Stamp(&stStamp);
00112         }
00113     }
00114
00115     // Draw the caption
00116     {
00117         DrawText_t stText;
00118         stText.usLeft = usX + GetLeft() + TEXT_X_OFFSET;
00119         stText.usTop = usY + GetTop();
00120         stText.uColor = m_uFontColor;
00121         stText.pstFont = m_pstFont;

```

```

00122         stText.pcString = m_szCaption;
00123
00124         usTextWidth = pclDriver->TextWidth(&stText);
00125         pclDriver->Text(&stText);
00126     }
00127 }
00128
00129 //-----
00130 GuiReturn_t CheckBoxControl::ProcessEvent(
00131     GuiEvent_t *pstEvent_ )
00132 {
00133     K_USHORT usXOffset, usYOffset;
00134     GetControlOffset(&usXOffset, &usYOffset);
00135
00136     GUI_DEBUG_PRINT("ButtonControl::ProcessEvent\n");
00137
00138     switch (pstEvent_>ucEventType)
00139     {
00140         case EVENT_TYPE_KEYBOARD:
00141         {
00142             // If this is a space bar or an enter key, behave like a mouse click.
00143             if ((KEYCODE_SPACE == pstEvent_>stKey.ucKeyCode) ||
00144                 (KEYCODE_RETURN == pstEvent_>stKey.ucKeyCode))
00145             {
00146                 if (pstEvent_>stKey.bKeyState)
00147                 {
00148                     m_bChecked = true;
00149                 }
00150                 else
00151                 {
00152                     m_bChecked = false;
00153                 }
00154                 SetStale();
00155             }
00156         }
00157         break;
00158         case EVENT_TYPE_MOUSE:
00159         {
00160             // Is this control currently in the "active"/pressed state?
00161             if (m_bChecked)
00162             {
00163                 // Check to see if the movement is out-of-bounds based on the coordinates.
00164                 // If so, de-activate the control
00165                 if (pstEvent_>stMouse.bLeftState)
00166                 {
00167                     if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00168                         (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00169                          GetWidth()-1) &&
00170                         (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00171                         (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00172                          GetHeight() - 1))
00173                     {
00174                         m_bChecked = false;
00175                         SetStale();
00176                     }
00177                 }
00178                 else if (!m_bChecked)
00179                 {
00180                     // If we registered a down-click in the bounding box, set the state of the
00181                     // control to activated.
00182                     if (pstEvent_>stMouse.bLeftState)
00183                     {
00184                         if ((pstEvent_>stMouse.usX >= GetLeft() + usXOffset) &&
00185                             (pstEvent_>stMouse.usX < GetLeft() + usXOffset +
00186                              GetWidth()-1) &&
00187                             (pstEvent_>stMouse.usY >= GetTop() + usYOffset) &&
00188                             (pstEvent_>stMouse.usY < GetTop() + usYOffset +
00189                              GetHeight() - 1))
00190                         {
00191                             m_bChecked = true;
00192                             SetStale();
00193                         }
00194                     }
00195                 }
00196                 if (!IsInFocus())
00197                 {
00198                     GetParentWindow()->SetFocus(this);
00199                     SetStale();
00200                 }
00201             }
00202             break;
00203         }
00204     }
00205 }

```

15.15 /home/moslevin/Project/R1/stage/src/control_checkbox.h File Reference

Checkbox Control.

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

Classes

- class [CheckBoxControl](#)

15.15.1 Detailed Description

Checkbox Control. A binary On/Off switch control

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

15.16 control_checkbox.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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
```


15.17 /home/moslevin/Project/R1/stage/src/control_gamepanel.cpp File Reference

GUI Panel Control Implementation with joystick control and tick-based state machine updates.

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

15.17.1 Detailed Description

GUI Panel Control Implementation with joystick control and tick-based state machine updates.

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

15.18 control_gamepanel.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "gui.h"
00021 #include "kerneltypes.h"
00022 #include "draw.h"
00023 #include "graphics.h"
00024 #include "control_gamepanel.h"
00025
00026 //-----
00027 void GamePanelControl::Draw()
00028 {
00029     // Game state machine goes here.
00030 }
00031
00032 //-----
00033 GuiReturn_t GamePanelControl::ProcessEvent (
00034     GuiEvent_t *pstEvent_ )
00035 {
00036     K_USHORT usXOffset, usYOffset;
00037
00038     switch (pstEvent_>ucEventType)
00039     {
00040         case EVENT_TYPE_TIMER:
00041             // Every tick, call Draw(). This is used to kick the state
00042             // machine
00043             SetStale();
00044             break;
00045         case EVENT_TYPE_KEYBOARD:
00046             break;
00047         case EVENT_TYPE_MOUSE:
00048             break;
00049         case EVENT_TYPE_JOYSTICK:
00050             m_stLastJoy.usRawData = m_stCurrentJoy.usRawData;
00051             m_stCurrentJoy.usRawData = pstEvent_>stJoystick.
00052             usRawData;
00053             break;
00054     }
00055     return GUI_EVENT_OK;
00056 }
```

15.19 /home/moslevin/Project/R1/stage/src/control_gamepanel.h File Reference

GUI Game Panel Control.

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

Classes

- class [GamePanelControl](#)

15.19.1 Detailed Description

GUI Game Panel Control. A game panel is a blank UI element whose dimensions define the dimensions of a gameplay surface. The element triggers a draw() call on every tick event (which can be used to kick a game's state machine). The control also responds to joystick events, which can then be used to control the game.

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

15.20 control_gamepanel.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00025 #ifndef __CONTROL_GAMEPANEL_H__
00026 #define __CONTROL_GAMEPANEL_H__
00027
00028 #include "gui.h"
00029 #include "kerneltypes.h"
00030 #include "draw.h"
00031
00032 class GamePanelControl : public GuiControl
00033 {
00034 public:
00035     virtual void Init() { SetAcceptFocus(false); m_stCurrentJoy.
00036         usRawData = 0; m_stLastJoy.usRawData = 0;}
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00039     virtual void Activate( bool bActivate_ ) {}
00040 private:
00041     JoystickEvent_t m_stLastJoy;
00042     JoystickEvent_t m_stCurrentJoy;
00043 };
00044
00045
00046 #endif
00047
```

15.21 /home/moslevin/Project/R1/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)

15.21.1 Detailed Description

GUI GroupBox Control Implementation.

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

15.22 control_groupbox.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "gui.h"
00020 #include "kerneltypes.h"
00021 #include "draw.h"
00022 #include "graphics.h"
00023 #include "control_groupbox.h"
00024
00025 #define BORDER_OFFSET          (4)
00026 #define TEXT_X_OFFSET          (8)
00027 #define TEXT_Y_OFFSET          (0)
00028
00029 //-----
00030 void GroupBoxControl::Draw()
00031 {
00032     GUI_DEBUG_PRINT( "GroupBoxControl::Draw()\n");
00033     GraphicsDriver *pclDriver = GetParentWindow()->
        GetDriver();
00034     K_USHORT usX, usY;
00035     K_USHORT usTextWidth;
00036
00037     GetControlOffset(&usX, &usY);
00038
00039     // Draw the background panel
00040     {
00041         DrawRectangle_t stRectangle;
00042         stRectangle.usTop = GetTop() + usY;
00043         stRectangle.usBottom = stRectangle.usTop + GetHeight() - 1;
00044         stRectangle.usLeft = GetLeft() + usX;
00045         stRectangle.usRight = stRectangle.usLeft + GetWidth() - 1;
00046         stRectangle.bFill = true;
00047         stRectangle.uLineColor = m_uPanelColor;
00048         stRectangle.uFillColor = m_uPanelColor;
00049
00050         pclDriver->Rectangle(&stRectangle);
00051     }
00052
00053     // Draw the caption
00054     {
00055         DrawText_t stText;
00056         stText.usLeft = usX + TEXT_X_OFFSET;
```

```

00057         stText.usTop = usY + TEXT_Y_OFFSET;
00058         stText.uColor = m_uFontColor;
00059         stText.pstFont = m_pstFont;
00060         stText.pcString = m_pcCaption;
00061
00062         usTextWidth = pclDriver->TextWidth(&stText);
00063         pclDriver->Text(&stText);
00064     }
00065
00066     // Draw the lines surrounding the panel
00067     {
00068         DrawLine_t stLine;
00069
00070         stLine.uColor = m_uLineColor;
00071         stLine.usY1 = GetTop() + usY + BORDER_OFFSET;
00072         stLine.usY2 = GetTop() + usY + GetHeight() - BORDER_OFFSET - 1;
00073         stLine.usX1 = usX + BORDER_OFFSET;
00074         stLine.usX2 = usX + BORDER_OFFSET;
00075         pclDriver->Line(&stLine);
00076
00077         stLine.usY1 = GetTop() + usY + BORDER_OFFSET;
00078         stLine.usY2 = GetTop() + usY + GetHeight() - BORDER_OFFSET - 1;
00079         stLine.usX1 = usX + GetWidth() - BORDER_OFFSET - 1;
00080         stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
00081         pclDriver->Line(&stLine);
00082
00083         stLine.usY1 = GetTop() + usY + GetHeight() - BORDER_OFFSET - 1;
00084         stLine.usY2 = GetTop() + usY + GetHeight() - BORDER_OFFSET - 1;
00085         stLine.usX1 = usX + BORDER_OFFSET;
00086         stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
00087         pclDriver->Line(&stLine);
00088
00089         stLine.usY1 = GetTop() + BORDER_OFFSET - 1;
00090         stLine.usY2 = GetTop() + BORDER_OFFSET - 1;
00091         stLine.usX1 = usX + BORDER_OFFSET;
00092         stLine.usX2 = usX + TEXT_X_OFFSET - 2;
00093         pclDriver->Line(&stLine);
00094
00095         stLine.usX1 = usX + TEXT_X_OFFSET + usTextWidth;
00096         stLine.usX2 = usX + GetWidth() - BORDER_OFFSET - 1;
00097         pclDriver->Line(&stLine);
00098     }
00099
00100
00101 }

```

15.23 /home/moslevin/Project/R1/stage/src/control_groupbox.h File Reference

GUI Group Box Control.

```

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

```

Classes

- class [GroupBoxControl](#)

15.23.1 Detailed Description

GUI Group Box Control. A groupbox control is essentially a panel with a text caption, and a lined border.

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

15.24 control_groupbox.h

```

00001 /*=====
00002
00003  _ _ _ _ _

```

```

00004 | \ / | | | \ / | | | | / | | | |
00005 | | \ / | | | \ / | | | | / | | | |
00006 | | \ / | | | \ / | | | | / | | | |
00007 | | \ / | | | \ / | | | | / | | | |
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #ifndef __CONTROL_GROUPBOX_H__
00023 #define __CONTROL_GROUPBOX_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028
00029 class GroupBoxControl : public GuiControl
00030 {
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

```

15.25 /home/moslevin/Project/R1/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](#)

15.25.1 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](#).

15.26 control_label.h

```

00001 /*=====

```

```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 class LabelControl : public GuiControl
00031 {
00032 public:
00033     virtual void Init() { m_uBackColor = COLOR_BLACK;
00034                         m_uFontColor = COLOR_WHITE;
00035                         m_pstFont = NULL;
00036                         m_pcCaption = "";
00037                         SetAcceptFocus(false); }
00038     virtual void Draw();
00039     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) {}
00040     virtual void Activate( bool bActivate_ ) {}
00041
00042     void SetBackColor( COLOR eColor_ ) { m_uBackColor = eColor_; }
00043     void SetFontColor( COLOR eColor_ ) { m_uFontColor = eColor_; }
00044     void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00045     void SetCaption( const K_CHAR *pcData_ ) { m_pcCaption = pcData_; }
00046
00047 private:
00048     Font_t *m_pstFont;
00049     const K_CHAR *m_pcCaption;
00050     COLOR m_uBackColor;
00051     COLOR m_uFontColor;
00052
00053 };
00054
00055 #endif
00056

```

15.27 /home/moslevin/Project/R1/stage/src/control_notification.cpp File Reference

Notification pop-up control.

```

#include "control_notification.h"
#include "kerneltypes.h"

```

15.27.1 Detailed Description

Notification pop-up control. A pop-up control that can be used to present the user with information about system state changes, events, etc.

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

15.28 control_notification.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008

```

```

00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "control_notification.h"
00023 #include "kerneltypes.h"
00024
00025 //-----
00026 void NotificationControl::Draw()
00027 {
00028     if (!m_bVisible)
00029     {
00030         return;
00031     }
00032
00033     DrawRectangle_t stRect;
00034     DrawLine_t stLine;
00035     DrawText_t stText;
00036
00037     GraphicsDriver *pclDriver = GetParentWindow()->
        GetDriver();
00038
00039     K_USHORT usXOffset = 0;
00040     K_USHORT usHalfWidth = 0;
00041     K_USHORT usYOffset = 0;
00042
00043     // Get the location of the control relative to elements higher in the heirarchy
00044     GetControlOffset(&usXOffset, &usYOffset);
00045
00046     // Draw the rounded-off rectangle
00047     stLine.usX1 = GetLeft() + usXOffset + 1;
00048     stLine.usX2 = stLine.usX1 + GetWidth() - 3;
00049     stLine.usY1 = GetTop() + usYOffset;
00050     stLine.usY2 = stLine.usY1;
00051     stLine.uColor = COLOR_WHITE;
00052     pclDriver->Line(&stLine);
00053
00054     stLine.usY1 = GetTop() + usYOffset + GetHeight() - 1;
00055     stLine.usY2 = stLine.usY1;
00056     pclDriver->Line(&stLine);
00057
00058     // Draw the rounded-off rectangle
00059     stLine.usX1 = GetLeft() + usXOffset;
00060     stLine.usX2 = stLine.usX1;
00061
00062     stLine.usY1 = GetTop() + usYOffset + 1;
00063     stLine.usY2 = stLine.usY1 + GetHeight() - 3;
00064     pclDriver->Line(&stLine);
00065
00066     // Draw the rounded-off rectangle
00067     stLine.usX1 = GetLeft() + usXOffset + GetWidth() - 1;
00068     stLine.usX2 = stLine.usX1;
00069     pclDriver->Line(&stLine);
00070
00071     stRect.usTop = GetTop() + usYOffset + 1;
00072     stRect.usBottom = stRect.usTop + GetHeight() - 3;
00073     stRect.usLeft = GetLeft() + usXOffset + 1;
00074     stRect.usRight = stRect.usLeft + GetWidth() - 3;
00075     stRect.bFill = true;
00076     stRect.uFillColor = COLOR_BLACK;
00077     stRect.uLineColor = COLOR_BLACK;
00078     pclDriver->Rectangle(&stRect);
00079
00080     // Draw the Text
00081     stText.pstFont = m_pstFont;
00082     stText.pcString = m_szCaption;
00083     stText.uColor = COLOR_WHITE;
00084     usHalfWidth = pclDriver->TextWidth(&stText);
00085     usHalfWidth >= 1;
00086     stText.usLeft = GetLeft() + (GetWidth()>>1) - usHalfWidth + usXOffset;
00087     stText.usTop = GetTop() + usYOffset;
00088     pclDriver->Text(&stText);
00089 }
00090
00091 //-----
00092 GuiReturn_t NotificationControl::ProcessEvent(
    GuiEvent_t *pstEvent_)
00093 {
00094
00095     switch (pstEvent_->ucEventType)
00096     {
00097     case EVENT_TYPE_TIMER:
00098     {
00099         if (m_bTrigger && m_usTimeout)
00100         {
00101             m_usTimeout--;

```



```

00034         SetAcceptFocus(false);
00035         m_szCaption = "";
00036         m_pstFont = NULL;
00037         m_bVisible = true;
00038         m_bTrigger = false;
00039     }
00040
00041     virtual void Draw();
00042     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00043     virtual void Activate( bool bActivate_ ) {}
00044
00045     void SetFont( Font_t *pstFont_ ) { m_pstFont = pstFont_; }
00046     void SetCaption( const K_CHAR *szCaption_ ) { m_szCaption = szCaption_; }
00047
00048     void Trigger( K_USHORT usTimeout_ )
00049     {
00050         m_usTimeout = usTimeout_;
00051         m_bTrigger = true;
00052         m_bVisible = true;
00053         SetStale();
00054     }
00055
00056 private:
00057     const K_CHAR * m_szCaption;
00058     Font_t *m_pstFont;
00059     K_USHORT m_usTimeout;
00060     bool m_bTrigger;
00061     bool m_bVisible;
00062 };
00063
00064 #endif
00065

```

15.31 /home/moslevin/Project/R1/stage/src/control_panel.cpp File Reference

GUI Panel Control Implementation.

```

#include "gui.h"
#include "kerneltypes.h"
#include "draw.h"
#include "graphics.h"
#include "control_panel.h"

```

15.31.1 Detailed Description

GUI Panel Control Implementation.

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

15.32 control_panel.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "gui.h"
00020 #include "kerneltypes.h"
00021 #include "draw.h"
00022 #include "graphics.h"
00023 #include "control_panel.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 }

```

15.33 /home/moslevin/Project/R1/stage/src/control_panel.h File Reference

GUI Panel Control.

```

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

```

Classes

- class [PanelControl](#)

15.33.1 Detailed Description

GUI Panel Control. The "panel" is probably the simplest control that can be implemented in a GUI. It serves as a dock for other controls, and also as an example for implementing more complex controls.

A panel is essentially a flat rectangle, specified by a control's typical top/left/height/width parameters, and a color value.

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

15.34 control_panel.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00026 #ifndef __CONTROL_PANEL_H__
00027 #define __CONTROL_PANEL_H__
00028
00029 #include "gui.h"
00030 #include "kerneltypes.h"
00031 #include "draw.h"
00032
00033 class PanelControl : public GuiControl
00034 {
00035 public:

```

```

00036     virtual void Init() { m_uColor = COLOR_BLACK; SetAcceptFocus(false); }
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) {};
00039     virtual void Activate( bool bActivate_ ) {}
00040
00041     void SetColor( COLOR eColor_ ) { m_uColor = eColor_; }
00042
00043 private:
00044     COLOR m_uColor;
00045
00046 };
00047
00048 #endif
00049

```

15.35 /home/moslevin/Project/R1/stage/src/control_progress.cpp File Reference

GUI Progress Bar Control.

```

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

```

15.35.1 Detailed Description

GUI Progress Bar Control. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

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

15.36 control_progress.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "gui.h"
00022 #include "kerneltypes.h"
00023 #include "draw.h"
00024 #include "control_progress.h"
00025
00026 //-----
00027 void ProgressControl::Init()
00028 {
00029     m_uBackColor = COLOR_BLACK;
00030     m_uBorderColor = COLOR_GREY75;
00031     m_uProgressColor = COLOR_GREEN;
00032     SetAcceptFocus(false);
00033 }
00034
00035 //-----
00036 void ProgressControl::Draw()
00037 {
00038     GraphicsDriver *pclDriver = GetParentWindow()->
    GetDriver();
00039     DrawRectangle_t stRect;
00040     DrawLine_t stLine;
00041
00042     K_USHORT usX, usY;
00043     K_USHORT usProgressWidth;
00044

```

```

00045     GetControlOffset(&usX, &usY);
00046
00047     // Draw the outside of the progress bar region
00048     stLine.uColor = m_uBorderColor;
00049     stLine.usX1 = usX + GetLeft() + 1;
00050     stLine.usX2 = usX + GetLeft() + GetWidth() - 2;
00051     stLine.usY1 = usY + GetTop();
00052     stLine.usY2 = usY + GetTop();
00053     pclDriver->Line(&stLine);
00054
00055     stLine.usY1 = usY + GetTop() + GetHeight() - 1;
00056     stLine.usY2 = usY + GetTop() + GetHeight() - 1;
00057     pclDriver->Line(&stLine);
00058
00059     stLine.usY1 = usY + GetTop() + 1;
00060     stLine.usY2 = usY + GetTop() + GetHeight() - 2;
00061     stLine.usX1 = usX + GetLeft();
00062     stLine.usX2 = usX + GetLeft();
00063     pclDriver->Line(&stLine);
00064
00065     stLine.usX1 = usX + GetLeft() + GetWidth() - 1;
00066     stLine.usX2 = usX + GetLeft() + GetWidth() - 1;
00067     pclDriver->Line(&stLine);
00068
00069     // Draw the "completed" portion
00070     usProgressWidth = (K_USHORT)( ( ( (K_ULONG)m_ucProgress) * (GetWidth()-2) ) + 50 ) / 100);
00071     stRect.usTop = usY + GetTop() + 1;
00072     stRect.usBottom = usY + GetTop() + GetHeight() - 2;
00073     stRect.usLeft = usX + GetLeft() + 1;
00074     stRect.usRight = stRect.usLeft + usProgressWidth - 1;
00075     stRect.bFill = true;
00076     stRect.uLineColor = m_uProgressColor;
00077     stRect.uFillColor = m_uProgressColor;
00078     pclDriver->Rectangle(&stRect);
00079
00080     // Draw the "incomplete" portion
00081     stRect.usLeft = stRect.usRight + 1;
00082     stRect.usRight = usX + GetLeft() + GetWidth() - 2;
00083     stRect.bFill = true;
00084     stRect.uLineColor = m_uBackColor;
00085     stRect.uFillColor = m_uBackColor;
00086     pclDriver->Rectangle(&stRect);
00087
00088 }
00089
00090 //-----
00091 void ProgressControl::SetProgress( K_UCHAR ucProgress_ )
00092 {
00093     m_ucProgress = ucProgress_;
00094     if (m_ucProgress > 100)
00095     {
00096         m_ucProgress;
00097     }
00098     SetStale();
00099 }
00100
00101 //-----
00102 GuiReturn_t ProgressControl::ProcessEvent(
00103     GuiEvent_t *pstEvent_)
00104 {
00105     return GUI_EVENT_OK;
00106 }

```

15.37 /home/moslevin/Project/R1/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](#)

Typedefs

- typedef void(* **ButtonCallback**)(void *pvData_)

15.39.1 Detailed Description

GUI Button Control, with a flare. Basic pushbutton control with an up/down state, and Mark3 visual style

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

15.40 control_slickbutton.h

```

00001
00002 /*=====
00003
00004
00005
00006
00007
00008
00009
00010 --[Mark3 Realtime Platform]-----
00011
00012 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00013 See license.txt for more information
00014 =====*/
00022 #ifndef __CONTROL_SLICKBUTTON_H__
00023 #define __CONTROL_SLICKBUTTON_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 typedef void (*ButtonCallback)( void *pvData_ );
00031
00032 class SlickButtonControl : public GuiControl
00033 {
00034 public:
00035
00036     virtual void Init();
00037     virtual void Draw();
00038     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00039     virtual void Activate( bool bActivate_ );
00040
00041     void SetFont( Font_t *pstFont_ )      { m_pstFont = pstFont_; }
00042
00043     void SetCaption( const K_CHAR *szCaption_ )      { m_szCaption = szCaption_;}
00044
00045     void SetCallback( ButtonCallback pfCallback_, void *pvData_ )
00046     { m_pfCallback = pfCallback_; m_pvCallbackData = pvData_; }
00047 private:
00048
00049     const K_CHAR *m_szCaption;
00050     Font_t *m_pstFont;
00051     bool m_bState;
00052     K_UCHAR m_ucTimeout;
00053
00054     void *m_pvCallbackData;
00055     ButtonCallback m_pfCallback;
00056 };
00057
00058
00059 #endif
00060

```

15.41 /home/moslevin/Project/R1/stage/src/control_slickprogress.cpp File Reference

GUI Progress Bar Control, with flare.

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

15.41.1 Detailed Description

GUI Progress Bar Control, with flare. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

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

15.42 control_slickprogress.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "gui.h"
00022 #include "kerneltypes.h"
00023 #include "draw.h"
00024 #include "control_slickprogress.h"
00025
00026 //-----
00027 void SlickProgressControl::Init()
00028 {
00029     SetAcceptFocus(false);
00030 }
00031
00032 //-----
00033 void SlickProgressControl::Draw()
00034 {
00035     GraphicsDriver *pclDriver = GetParentWindow()->
        GetDriver();
00036     DrawRectangle_t stRect;
00037     DrawLine_t stLine;
00038
00039     K_USHORT usX, usY;
00040     K_USHORT usProgressWidth;
00041
00042     GetControlOffset(&usX, &usY);
00043
00044     // Draw the outside of the progress bar region
00045     stLine.uColor = COLOR_GREY50;
00046     stLine.usX1 = usX + GetLeft() + 1;
00047     stLine.usX2 = usX + GetLeft() + GetWidth() - 2;
00048     stLine.usY1 = usY + GetTop();
00049     stLine.usY2 = usY + GetTop();
00050     pclDriver->Line(&stLine);
00051
00052     stLine.usY1 = usY + GetTop() + GetHeight() - 1;
00053     stLine.usY2 = usY + GetTop() + GetHeight() - 1;
00054     pclDriver->Line(&stLine);
00055
00056     stLine.usY1 = usY + GetTop() + 1;
00057     stLine.usY2 = usY + GetTop() + GetHeight() - 2;
00058     stLine.usX1 = usX + GetLeft();
00059     stLine.usX2 = usX + GetLeft();
00060     pclDriver->Line(&stLine);
00061
00062     stLine.usX1 = usX + GetLeft() + GetWidth() - 1;
00063     stLine.usX2 = usX + GetLeft() + GetWidth() - 1;
00064     pclDriver->Line(&stLine);
00065
00066     // Draw the "completed" portion
00067     usProgressWidth = (K_USHORT)( ( ( (K_ULONG)m_ucProgress) * (GetWidth()-2) ) + 50 ) / 100);
00068     stRect.usTop = usY + GetTop() + 1;
```

```

00069     stRect.usBottom = usY + GetTop() + ((GetHeight() - 1) / 2);
00070     stRect.usLeft = usX + GetLeft() + 1;
00071     stRect.usRight = stRect.usLeft + usProgressWidth - 1;
00072     stRect.bFill = true;
00073     stRect.uLineColor = RGB_COLOR( 0, (K_UCHAR)(MAX_GREEN * 0.85), (K_UCHAR)(MAX_BLUE * 0.25));
00074     stRect.uFillColor = stRect.uLineColor;
00075     pclDriver->Rectangle(&stRect);
00076
00077     stRect.usTop = stRect.usBottom + 1;
00078     stRect.usBottom = usY + GetTop() + GetHeight() - 2;
00079     stRect.uLineColor = RGB_COLOR( 0, (K_ULONG)(MAX_GREEN * 0.75), (K_ULONG)(MAX_BLUE * 0.20));
00080     stRect.uFillColor = stRect.uLineColor;
00081     pclDriver->Rectangle(&stRect);
00082
00083     // Draw the "incomplete" portion
00084     stRect.usTop = usY + GetTop() + 1;
00085     stRect.usBottom = usY + GetTop() + GetHeight() - 2;
00086     stRect.usLeft = stRect.usRight + 1;
00087     stRect.usRight = usX + GetLeft() + GetWidth() - 2;
00088     stRect.bFill = true;
00089     stRect.uLineColor = RGB_COLOR( (K_ULONG)(MAX_RED * 0.10), (K_ULONG)(MAX_GREEN * 0.10), (
K_ULONG)(MAX_BLUE * 0.10));
00090     stRect.uFillColor = stRect.uLineColor;
00091     pclDriver->Rectangle(&stRect);
00092
00093 }
00094
00095 //-----
00096 void SlickProgressControl::SetProgress( K_UCHAR ucProgress_ )
00097 {
00098     m_ucProgress = ucProgress_;
00099     if (m_ucProgress > 100)
00100     {
00101         m_ucProgress;
00102     }
00103     SetStale();
00104 }
00105
00106 //-----
00107 GuiReturn_t SlickProgressControl::ProcessEvent(
GuiEvent_t *pstEvent_)
00108 {
00109     return GUI_EVENT_OK;
00110 }

```

15.43 /home/moslevin/Project/R1/stage/src/control_slickprogress.h File Reference

GUI Progress Bar Control, with flare.

```

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

```

Classes

- class [SlickProgressControl](#)

15.43.1 Detailed Description

GUI Progress Bar Control, with flare. A simple progress bar control using lines and rectangles to display the status of an operation from initialization to completion

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

15.44 control_slickprogress.h

```

00001 /*=====

```



```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #ifndef __CONTROL_SLICKPROGRESS_H__
00023 #define __CONTROL_SLICKPROGRESS_H__
00024
00025 #include "gui.h"
00026 #include "kerneltypes.h"
00027 #include "draw.h"
00028 #include "font.h"
00029
00030 class SlickProgressControl : public GuiControl
00031 {
00032 public:
00033     virtual void Init();
00034     virtual void Draw();
00035     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ );
00036     virtual void Activate( bool bActivate_ ) {}
00037
00038     void SetProgress( K_UCHAR ucProgress_ );
00039
00040 private:
00041     K_UCHAR m_ucProgress;
00042 };
00043
00044 #endif
00045

```

15.45 /home/moslevin/Project/R1/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.

15.45.1 Detailed Description

Portable DCPU-16 CPU emulator. The DCPU-16 is the in-game CPU used in the upcoming game 0x10[^]c, from the creators of the wildly successful Minecraft. While the [DCPU](#) is supposed to be part of the game, it has serious potential for use in all sorts of embedded applications.

The fact that [DCPU](#) is a very lightweight VM to implement and contains built-in instructions for accessing hardware peripherals and handling external interrupts lends itself to being used on microcontrollers.

Unlike a lot of embedded CPUs, DCPU-16 assembly is extremely simple to learn, since it has a very limited number of opcodes (37), each of which provide the same register/memory addressing modes for all operands. There are also only 2 opcode formats which make interpreting opcodes very efficient.

The DCPU-16 is extended using a variable number of "external hardware devices" which communicate with the CPU core using interrupts. These devices are enumerated on startup, and since there is no defined format for how these devices work, we can hijack this interface to provide a way for the [DCPU](#) to access resources supplied by the OS (i.e Timers, Drivers), or the hardware directly. This also lends itself to inter-VM communications (multiple DCPUs communicating with eachother in different OS threads). There's an immense amount of flexibility here - applications from debugging to scripting to runtime-configuration are all easily supported by this machine.

But what is a platform without tools support? Fortunately, the hype around 0x10c is building - and a development community for this platform has grown immensely. There are a number of compilers, assemblers, and IDEs, many of which support virtualized hardware extensions. One of the compilers is a CLANG/LLVM backend, which should allow for very good C language support.

I had attempted to do something similar by creating a VM based on the 8051 (see the Funk51 project on sourceforge), but that project was at least four times as large - and the tools support was very spotty. There were C compilers, but there was a lot of shimming required to produce output that was suitable for the VM. Also, the lack of a native host interface (interrupts, hardware bus enumerations, etc.) forced a non-standard approach to triggering native methods by writing commands to a reserved chunk of memory and writing to a special "trigger" address to invoke the native system. Using a DCPU-16 based simulator addresses this in a nice, clean way by providing modern tools, and a VM infrastructure tailored to be interfaced with a host.

Regarding this version of the [DCPU](#) emulator - it's very simple to use. Program binaries are loaded into buffers in the host CPU's RAM, with the host also providing a separate buffer for [DCPU](#) RAM. The size of the [DCPU](#) RAM buffer will contain both the RAM area, as well as the program stack, so care must be taken to ensure that the stack doesn't overflow. The [DCPU](#) specification allows for 64K words (128KB) of RAM and ROM each, but this implementation allows us to tailor the CPU for more efficient or minimal environments.

In the future, this emulator will be extended to provide a mechanism to allow programs to be run out of flash, EEPROM, or other interfaces via the Mark3 Drivers API.

Once the program has been loaded into the host's address space, the [DCPU](#) class can be initialized.

```
// Use 16-bit words for 16-bit emulator.
K_USHORT ausRAM[ RAM_SIZE ];
K_USHORT ausROM[ ROM_SIZE ];
{
    class DCPU 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();
```

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

15.46 dcpu.cpp

```
00001 /*=====
```

```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00014 #include "dcpu.h"
00015 #include "kerneltypes.h"
00016 #include "ll.h"
00017
00018 #define CORE_DEBUG 0
00019
00020 //-----
00021 #if CORE_DEBUG
00022
00023     #include <stdio.h>
00024     #include <string.h>
00025     #include <stdlib.h>
00026
00027     #define DBG_PRINT(...)      printf(__VA_ARGS__)
00028 #else
00029     #define DBG_PRINT(...)
00030 #endif
00031
00032 //-----
00033 static const K_UCHAR aucBasicOpcodeCycles[] =
00034 {
00035     0,      // OP_NON_BASIC = 0
00036     1,      // OP_SET
00037     2,      // OP_ADD
00038     2,      // OP_SUB
00039     2,      // OP_MUL
00040     2,      // OP_MLI
00041     3,      // OP_DIV
00042     3,      // OP_DVI,
00043     3,      // OP_MOD,
00044     3,      // OP_MDI,
00045     1,      // OP_AND,
00046     1,      // OP_BOR,
00047     1,      // OP_XOR,
00048     1,      // OP_SHR,
00049     1,      // OP_ASR,
00050     1,      // OP_SHL,
00051     2,      // OP_IFB,
00052     2,      // OP_IFC,
00053     2,      // OP_IFE,
00054     2,      // OP_IFN,
00055     2,      // OP_IFG,
00056     2,      // OP_IFA,
00057     2,      // OP_IFL,
00058     2,      // OP_IFU,
00059     0,      // OP_18,
00060     0,      // OP_19,
00061     3,      // OP_ADX,
00062     3,      // OP_SBX,
00063     0,      // OP_1C,
00064     0,      // OP_1D,
00065     2,      // OP_STI,
00066     2,      // OP_STD
00067 };
00068
00069 //-----
00070 static const K_UCHAR aucExtendedOpcodeCycles[] =
00071 {
00072     0,      // "RESERVED",
00073     3,      // "JSR",
00074     0,      // "UNDEFINED"
00075     0,      // "UNDEFINED"
00076     0,      // "UNDEFINED"
00077     0,      // "UNDEFINED"
00078     0,      // "UNDEFINED"
00079     4,      // "INT",
00080     1,      // "IAG",
00081     1,      // "IAS",
00082     3,      // "RFI",
00083     2,      // "IAQ",
00084     0,      // "UNDEFINED"
00085     0,      // "UNDEFINED"
00086     0,      // "UNDEFINED"
00087     2,      // "HWN",

```

```

00195     4,    // "HWQ",
00196     4,    // "HWI",
00197     0,    // "UNDEFINED"
00198     0,    // "UNDEFINED"
00199     0,    // "UNDEFINED"
00200     0,    // "UNDEFINED"
00201     0,    // "UNDEFINED"
00202     0,    // "UNDEFINED"
00203     0,    // "UNDEFINED"
00204     0,    // "UNDEFINED"
00205     0,    // "UNDEFINED"
00206     0,    // "UNDEFINED"
00207     0,    // "UNDEFINED"
00208     0,    // "UNDEFINED"
00209     0,    // "UNDEFINED"
00210 };
00211
00212 //-----
00213 void DCPU::SET()
00214 {
00215     DBG_PRINT("SET\n");
00216     *b = *a;
00217 }
00218
00219 //-----
00220 void DCPU::ADD()
00221 {
00222     K_ULONG ulTemp;
00223     DBG_PRINT("ADD\n");
00224
00225     ulTemp = (K_ULONG)*a + (K_ULONG)*b;
00226     if (ulTemp >= 65536)
00227     {
00228         m_stRegisters.EX = 0x0001;
00229     }
00230     else
00231     {
00232         m_stRegisters.EX = 0;
00233     }
00234
00235     *b = *b + *a;
00236 }
00237
00238 //-----
00239 void DCPU::SUB()
00240 {
00241     K_LONG lTemp;
00242     DBG_PRINT("SUB\n");
00243
00244     lTemp = (K_LONG)*b - (K_LONG)*a;
00245     if (lTemp < 0)
00246     {
00247         m_stRegisters.EX = 0xFFFF;
00248     }
00249     else
00250     {
00251         m_stRegisters.EX = 0;
00252     }
00253
00254     *b = *b - *a;
00255 }
00256
00257 //-----
00258 void DCPU::MUL()
00259 {
00260     K_ULONG ulTemp;
00261
00262     DBG_PRINT("MUL\n");
00263     ulTemp = ((K_ULONG)*a * (K_ULONG)*b);
00264     m_stRegisters.EX = (K_USHORT)(ulTemp >> 16);
00265     *b = (K_USHORT)(ulTemp & 0x0000FFFF);
00266 }
00267
00268 //-----
00269 void DCPU::MLI()
00270 {
00271     K_LONG lTemp;
00272
00273     DBG_PRINT("MLI\n");
00274     lTemp = ((K_LONG)(*a * (K_SHORT)*b)) * (K_LONG)(*a * (K_SHORT)*b));
00275     m_stRegisters.EX = (K_USHORT)(lTemp >> 16);
00276     *b = (K_USHORT)(lTemp & 0x0000FFFF);
00277 }
00278
00279 //-----
00280 void DCPU::DIV()
00281 {

```

```

00282     K_USHORT usTemp;
00283
00284     DBG_PRINT("DIV\n");
00285     if (*a == 0)
00286     {
00287         *b = 0;
00288         m_stRegisters.EX = 0;
00289     }
00290     else
00291     {
00292         usTemp = (K_USHORT) (((K_ULONGLONG)*b) << 16) / (K_ULONGLONG)*a;
00293         *b = *b / *a;
00294         m_stRegisters.EX = usTemp;
00295     }
00296 }
00297
00298 //-----
00299 void DCPU::DVI()
00300 {
00301     K_USHORT usTemp;
00302
00303     DBG_PRINT("DVI\n");
00304     if (*a == 0)
00305     {
00306         *b = 0;
00307         m_stRegisters.EX = 0;
00308     }
00309     else
00310     {
00311         usTemp = (K_USHORT) (((K_LONG)*(K_SHORT*)b) << 16) / (K_LONG) (*(K_SHORT*)
00312 a));
00313         *b = (K_USHORT) (*(K_SHORT*)b / *(K_SHORT*)a);
00314         m_stRegisters.EX = usTemp;
00315     }
00316 }
00317
00318 //-----
00319 void DCPU::MOD()
00320 {
00321     DBG_PRINT("MOD\n");
00322     if (*a == 0)
00323     {
00324         *b = 0;
00325     }
00326     else
00327     {
00328         *b = *b % *a;
00329     }
00330 }
00331
00332 //-----
00333 void DCPU::MDI()
00334 {
00335     DBG_PRINT("MDI\n");
00336     if (*b == 0)
00337     {
00338         *a = 0;
00339     }
00340     else
00341     {
00342         *b = (K_USHORT) (*(K_SHORT*)b % *(K_SHORT*)a);
00343     }
00344 }
00345
00346 //-----
00347 void DCPU::AND()
00348 {
00349     DBG_PRINT("AND\n");
00350     *b = *b & *a;
00351 }
00352
00353 //-----
00354 void DCPU::BOR()
00355 {
00356     DBG_PRINT("BOR\n");
00357     *b = *b | *a;
00358 }
00359
00360 //-----
00361 void DCPU::XOR()
00362 {
00363     DBG_PRINT("XOR\n");
00364     *b = *b ^ *a;
00365 }
00366
00367 //-----

```

```

00368 void DCPU::SHR()
00369 {
00370     K_USHORT usTemp = (K_USHORT) (((K_ULONGLONG)*b) << 16) >> (K_ULONGLONG)*a);
00371
00372     DBG_PRINT("SHR\n");
00373     *b = *b >> *a;
00374     m_stRegisters.EX = usTemp;
00375 }
00376
00377 //-----
00378 void DCPU::ASR()
00379 {
00380     K_USHORT usTemp = (K_USHORT) (((K_ULONGLONG)*b) << 16) >> (K_ULONGLONG)*a);
00381
00382     DBG_PRINT("ASR\n");
00383     *b = (K_USHORT) (* (K_SHORT*)b >> * (K_SHORT*)a);
00384     m_stRegisters.EX = usTemp;
00385 }
00386 //-----
00387 void DCPU::SHL()
00388 {
00389     K_USHORT usTemp = (K_USHORT) (((K_ULONGLONG)*b) << (K_ULONGLONG)*a) >> 16);
00390
00391     DBG_PRINT("SHL\n");
00392     *b = *b << *a;
00393     m_stRegisters.EX = usTemp;
00394 }
00395
00396 //-----
00397 bool DCPU::IFB()
00398 {
00399     DBG_PRINT("IFB\n");
00400     if ((*b & *a) != 0)
00401     {
00402         return true;
00403     }
00404     return false;
00405 }
00406
00407 //-----
00408 bool DCPU::IFC()
00409 {
00410     DBG_PRINT("IFC\n");
00411     if ((*b & *a) == 0)
00412     {
00413         return true;
00414     }
00415     return false;
00416 }
00417
00418 //-----
00419 bool DCPU::IFE()
00420 {
00421     DBG_PRINT("IFE\n");
00422     if (*b == *a)
00423     {
00424         return true;
00425     }
00426     return false;
00427 }
00428
00429 //-----
00430 bool DCPU::IFN()
00431 {
00432     DBG_PRINT("IFN\n");
00433     if (*b != *a)
00434     {
00435         return true;
00436     }
00437     return false;
00438 }
00439
00440 //-----
00441 bool DCPU::IFG()
00442 {
00443     DBG_PRINT("IFG\n");
00444     if (*b > *a)
00445     {
00446         return true;
00447     }
00448     return false;
00449 }
00450
00451 //-----
00452 bool DCPU::IFA()
00453 {
00454     DBG_PRINT("IFA\n");

```

```

00455     if (*(K_SHORT*)b) > (*(K_SHORT*)a)
00456     {
00457         return true;
00458     }
00459     return false;
00460 }
00461
00462 //-----
00463 bool DCPU::IFL()
00464 {
00465     DBG_PRINT("IFL\n");
00466     if (*b < *a)
00467     {
00468         return true;
00469     }
00470     return false;
00471 }
00472
00473 //-----
00474 bool DCPU::IFU()
00475 {
00476     DBG_PRINT("IFU\n");
00477     if (*(K_SHORT*)b < *(K_SHORT*)a)
00478     {
00479         return true;
00480     }
00481     return false;
00482 }
00483
00484 //-----
00485 void DCPU::ADX()
00486 {
00487     K_ULONG ulTemp;
00488     DBG_PRINT("ADX\n");
00489     ulTemp = (K_ULONG)*b + (K_ULONG)*a + (K_ULONG)m_stRegisters.EX;
00490     if (ulTemp >= 0x10000)
00491     {
00492         m_stRegisters.EX = 1;
00493     }
00494     else
00495     {
00496         m_stRegisters.EX = 0;
00497     }
00498     *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     *b = ((K_USHORT)(lTemp & 0x0000FFFF));
00516 }
00517
00518 //-----
00519 void DCPU::STI()
00520 {
00521     DBG_PRINT("STI\n");
00522     *b = *a;
00523     m_stRegisters.I++;
00524     m_stRegisters.J++;
00525 }
00526
00527 //-----
00528 void DCPU::STD()
00529 {
00530     DBG_PRINT("STD\n");
00531     *b = *a;
00532     m_stRegisters.I--;
00533     m_stRegisters.J--;
00534 }
00535
00536 //-----
00537 void DCPU::JSR()
00538 {
00539     DBG_PRINT("JSR 0x%04X\n", *a);

```

```

00542     m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00543     m_stRegisters.PC = *a;
00544 }
00545
00546 //-----
00547 void DCPU::INT()
00548 {
00549     DBG_PRINT("INT\n");
00550
00551     if (m_stRegisters.IA == 0)
00552     {
00553         // If IA is not set, return out.
00554         return;
00555     }
00556
00557     // Either acknowledge the interrupt immediately, or queue it.
00558     if (m_bInterruptQueueing == false)
00559     {
00560         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00561         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00562
00563         m_stRegisters.A = *a;
00564         m_stRegisters.PC = m_stRegisters.IA;
00565         m_bInterruptQueueing = true;
00566     }
00567     else
00568     {
00569         // Add interrupt message to the queue
00570         m_ausInterruptQueue[ ++m_ucQueueLevel ] = *
a;
00571     }
00572 }
00573
00574 //-----
00575 void DCPU::ProcessInterruptQueue()
00576 {
00577     // If there's an interrupt address specified, queueing is disabled, and
00578     // the queue isn't empty
00579     if (m_stRegisters.IA && !m_bInterruptQueueing &&
m_ucQueueLevel)
00580     {
00581         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00582         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00583
00584         m_stRegisters.A = m_ausInterruptQueue[
m_ucQueueLevel-- ];
00585         m_stRegisters.PC = m_stRegisters.IA;
00586
00587         m_bInterruptQueueing = true;
00588     }
00589 }
00590
00591 //-----
00592 void DCPU::IAG()
00593 {
00594     DBG_PRINT("IAG\n");
00595
00596     *a = m_stRegisters.IA;
00597 }
00598
00599 //-----
00600 void DCPU::IAS()
00601 {
00602     DBG_PRINT("IAS\n");
00603
00604     m_stRegisters.IA = *a;
00605 }
00606
00607 //-----
00608 void DCPU::RFI()
00609 {
00610     DBG_PRINT("RFI\n");
00611
00612     m_bInterruptQueueing = false;
00613
00614     m_stRegisters.A = m_pusRAM[ ++m_stRegisters.SP ];
00615     m_stRegisters.PC = m_pusRAM[ ++m_stRegisters.SP ];
00616 }
00617
00618 //-----
00619 void DCPU::IAQ()
00620 {
00621     DBG_PRINT("IAQ\n");
00622
00623     if (*a)

```



```

00632     {
00633         m_bInterruptQueueing = true;
00634     }
00635     else
00636     {
00637         m_bInterruptQueueing = false;
00638     }
00639 }
00640
00641 //-----
00642 void DCPU::HWN()
00643 {
00644     LinkListNode *pclNode;
00645
00646     DBG_PRINT("HWN\n");
00647     m_usTempA = 0;
00648     pclNode = m_clPluginList.GetHead();
00649     while (pclNode)
00650     {
00651         {
00652             m_usTempA++;
00653             pclNode = pclNode->GetNext();
00654         }
00655
00656         *a = m_usTempA;
00657     }
00658
00659 //-----
00660 void DCPU::HWQ()
00661 {
00662     DBG_PRINT("HWQ\n");
00663     DCPUPugin *pclPlugin;
00664     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead();
00665
00666     while (pclPlugin)
00667     {
00668         if (pclPlugin->GetDeviceNumber() == *a)
00669         {
00670             pclPlugin->Enumerate(&m_stRegisters);
00671             break;
00672         }
00673         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00674     }
00675 }
00676
00677 //-----
00678 void DCPU::HWI()
00679 {
00680     DBG_PRINT("HWI\n");
00681
00682     DCPUPugin *pclPlugin;
00683     pclPlugin = (DCPUPugin*)m_clPluginList.GetHead();
00684
00685     while (pclPlugin)
00686     {
00687         if (pclPlugin->GetDeviceNumber() == *a)
00688         {
00689             pclPlugin->Interrupt(this);
00690             break;
00691         }
00692         pclPlugin = (DCPUPugin*)pclPlugin->GetNext();
00693     }
00694 }
00695
00696 //-----
00697 void DCPU::Init(      K_USHORT *pusRAM_, K_USHORT usRAMSize_,
00698                  const K_USHORT *pusROM_, K_USHORT usROMSize_)
00699 {
00700     m_stRegisters.PC = 0;
00701     m_stRegisters.SP = usRAMSize_ ;
00702     m_stRegisters.A = 0;
00703     m_stRegisters.B = 0;
00704     m_stRegisters.C = 0;
00705     m_stRegisters.X = 0;
00706     m_stRegisters.Y = 0;
00707     m_stRegisters.Z = 0;
00708     m_stRegisters.I = 0;
00709     m_stRegisters.J = 0;
00710     m_stRegisters.EX = 0;
00711     m_stRegisters.IA = 0;
00712     m_ulCycleCount = 0;
00713
00714     m_pusROM = (K_USHORT*)pusROM_;
00715     m_usROMSize = usROMSize_;
00716
00717     m_pusRAM = pusRAM_;
00718     m_usRAMSize = usRAMSize_;
00719 }

```

```

00720
00721 //-----
00722 K_UCHAR DCPU::GetOperand( K_UCHAR ucOpType_, K_USHORT **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_ - ARG_A ];
00730             break;
00731
00732         case ARG_BRACKET_A:  case ARG_BRACKET_B:  case ARG_BRACKET_C:  case ARG_BRACKET_X:
00733         case ARG_BRACKET_Y:  case ARG_BRACKET_Z:  case ARG_BRACKET_I:  case ARG_BRACKET_J:
00734             *pusResult_ = &m_pusRAM[ m_stRegisters.ausRegisters[ ucOpType_ -
ARG_BRACKET_A ] ];
00735             break;
00736
00737         case ARG_WORD_A: case ARG_WORD_B: case ARG_WORD_C: case ARG_WORD_X:
00738         case ARG_WORD_Y: case ARG_WORD_Z: case ARG_WORD_I: case ARG_WORD_J:
00739             {
00740                 K_USHORT usTemp = m_pusROM[ m_stRegisters.PC++ ];
00741                 usTemp += m_stRegisters.ausRegisters[ ucOpType_ - ARG_WORD_A ];
00742                 *pusResult_ = &m_pusRAM[ usTemp ];
00743                 ucRetVal = 1;
00744             }
00745             break;
00746         case ARG_PUSH_POP_SP:
00747             if (*pusResult_ == a)
00748             {
00749                 a = &m_pusRAM[ ++m_stRegisters.SP ];
00750             }
00751             else
00752             {
00753                 b = &m_pusRAM[ m_stRegisters.SP-- ];
00754             }
00755             break;
00756         case ARG_PEEK_SP:
00757             *pusResult_ = &m_pusRAM[ m_stRegisters.SP ];
00758             break;
00759         case ARG_WORD_SP:
00760             {
00761                 K_USHORT usTemp = m_pusROM[ ++m_stRegisters.PC ];
00762                 usTemp += m_stRegisters.SP;
00763                 *pusResult_ = &m_pusRAM[ usTemp ];
00764                 ucRetVal++;
00765             }
00766             break;
00767         case ARG_SP:
00768             *pusResult_ = &(m_stRegisters.SP);
00769             break;
00770         case ARG_PC:
00771             *pusResult_ = &(m_stRegisters.PC);
00772             break;
00773         case ARG_EX:
00774             *pusResult_ = &(m_stRegisters.EX);
00775             break;
00776         case ARG_NEXT_WORD:
00777             *pusResult_ = &m_pusRAM[ m_pusROM[ m_stRegisters.PC++ ] ];
00778             ucRetVal++;
00779             break;
00780         case ARG_NEXT_LITERAL:
00781             *pusResult_ = &m_pusROM[ m_stRegisters.PC++ ];
00782             ucRetVal++;
00783             break;
00784
00785         case ARG_LITERAL_0:
00786             *pusResult_ = &m_usTempA;
00787             m_usTempA = (K_USHORT) (-1);
00788             break;
00789         case ARG_LITERAL_1:  case ARG_LITERAL_2:  case ARG_LITERAL_3:  case ARG_LITERAL_4:
00790         case ARG_LITERAL_5:  case ARG_LITERAL_6:  case ARG_LITERAL_7:  case ARG_LITERAL_8:
00791         case ARG_LITERAL_9:  case ARG_LITERAL_A:  case ARG_LITERAL_B:  case ARG_LITERAL_C:
00792         case ARG_LITERAL_D:  case ARG_LITERAL_E:  case ARG_LITERAL_F:  case ARG_LITERAL_10:
00793         case ARG_LITERAL_11: case ARG_LITERAL_12: case ARG_LITERAL_13: case ARG_LITERAL_14:
00794         case ARG_LITERAL_15: case ARG_LITERAL_16: case ARG_LITERAL_17: case ARG_LITERAL_18:
00795         case ARG_LITERAL_19: case ARG_LITERAL_1A: case ARG_LITERAL_1B: case ARG_LITERAL_1C:
00796         case ARG_LITERAL_1D: case ARG_LITERAL_1E: case ARG_LITERAL_1F:
00797             *pusResult_ = &m_usTempA;
00798             m_usTempA = (K_USHORT) (ucOpType_ - ARG_LITERAL_1);
00799             break;
00800         default:
00801             break;
00802     }
00803     return ucRetVal;
00804 }

```

```

00805
00806 //-----
00807 void DCPU::RunOpcode()
00808 {
00809     // Fetch the opcode @ the current program counter
00810     K_USHORT usWord = m_pusROM[ m_stRegisters.PC++ ];
00811     K_UCHAR ucOp = (K_UCHAR)DCPU_NORMAL_OPCODE_MASK(usWord);
00812     K_UCHAR ucA = (K_UCHAR)DCPU_A_MASK(usWord);
00813     K_UCHAR ucB = (K_UCHAR)DCPU_B_MASK(usWord);
00814     K_UCHAR ucSize = 1;
00815
00816     DBG_PRINT("0x%04X: %04X\n", m_stRegisters.PC - 1, usWord );
00817
00818     // Decode the opcode
00819     if (ucOp)
00820     {
00821         bool bRunNext = true;
00822
00823         a = &m_usTempA;
00824         b = 0;
00825
00826         // If this is a "basic" opcode, decode "a" and "b"
00827         ucSize += GetOperand( ucA , &a );
00828         ucSize += GetOperand( ucB, &b );
00829
00830         // Add the cycles to the runtime clock
00831         m_ulCycleCount += (K_ULONG)aucBasicOpcodeCycles[ ucOp ];
00832         m_ulCycleCount += (ucSize - 1);
00833
00834         // Execute the instruction once we've decoded the opcode and
00835         // processed the arguments.
00836         switch (DCPU_NORMAL_OPCODE_MASK(usWord))
00837         {
00838             case OP_SET:      SET();          break;
00839             case OP_ADD:      ADD();          break;
00840             case OP_SUB:      SUB();          break;
00841
00842             case OP_MUL:      MUL();          break;
00843             case OP_MLI:      MLI();          break;
00844             case OP_DIV:      DIV();          break;
00845             case OP_DVI:      DVI();          break;
00846             case OP_MOD:      MOD();          break;
00847             case OP_MDI:      MDI();          break;
00848             case OP_AND:      AND();          break;
00849             case OP_BOR:      BOR();          break;
00850             case OP_XOR:      XOR();          break;
00851             case OP_SHR:      SHR();          break;
00852             case OP_ASR:      ASR();          break;
00853             case OP_SHL:      SHL();          break;
00854             case OP_IFB:      bRunNext = IFB(); break;
00855             case OP_IFC:      bRunNext = IFC(); break;
00856             case OP_IFE:      bRunNext = IFE(); break;
00857             case OP_IFN:      bRunNext = IFN(); break;
00858             case OP_IFG:      bRunNext = IFG(); break;
00859             case OP_IFA:      bRunNext = IFA(); break;
00860             case OP_IFL:      bRunNext = IFL(); break;
00861             case OP_IFU:      bRunNext = IFU(); break;
00862             case OP_ADX:      ADX();          break;
00863             case OP_SBX:      SBX();          break;
00864             case OP_STI:      STI();          break;
00865             case OP_STD:      STD();          break;
00866             default:          break;
00867         }
00868
00869         // If we're not supposed to run the next instruction (i.e. skip it
00870         // due to failed condition), adjust the PC.
00871         if (!bRunNext)
00872         {
00873             // Skipped branches take an extra cycle
00874             m_ulCycleCount++;
00875
00876             // Skip the next opcode
00877             usWord = m_pusROM[ m_stRegisters.PC++ ];
00878             if (DCPU_NORMAL_OPCODE_MASK(usWord))
00879             {
00880                 DBG_PRINT( "Skipping Basic Opcode: %X\n",
DCPU_NORMAL_OPCODE_MASK(usWord));
00881                 // If this is a "basic" opcode, decode "a" and "b" - we do this to make sure our
00882                 // PC gets adjusted properly.
00883                 GetOperand( DCPU_A_MASK(usWord), &a );
00884                 GetOperand( DCPU_B_MASK(usWord), &b );
00885             }
00886             else
00887             {
00888                 DBG_PRINT( "Skipping Extended Opcode: %X\n", DCPU_EXTENDED_OPCODE_MASK(usWord));
00889                 GetOperand( DCPU_A_MASK(usWord), &a );
00890             }

```

```

00891     }
00892   }
00893   else
00894   {
00895       // Extended opcode. These only have a single argument, stored in the
00896       // "a" field.
00897       GetOperand( ucA, &a );
00898       m_ulCycleCount++;
00899
00900       // Execute the "extended" instruction now that the opcode has been
00901       // decoded, and the arguments processed.
00902       switch (ucB)
00903       {
00904           case OP_EX_JSR:      JSR();      break;
00905           case OP_EX_INT:     INT();      break;
00906           case OP_EX_IAG:     IAG();      break;
00907           case OP_EX_IAS:     IAS();      break;
00908           case OP_EX_RFI:     RFI();      break;
00909           case OP_EX_IAQ:     IAQ();      break;
00910           case OP_EX_HWN:     HWN();      break;
00911           case OP_EX_HWQ:     HWQ();      break;
00912           case OP_EX_HWI:     HWI();      break;
00913           default:           break;
00914       }
00915   }
00916
00917   // Process an interrupt from the queue (if there is one)
00918   ProcessInterruptQueue();
00919 }
00920
00921 //-----
00922 void DCPU::SendInterrupt( K_USHORT usMessage_ )
00923 {
00924     if (m_stRegisters.IA == 0)
00925     {
00926         // If IA is not set, return out.
00927         return;
00928     }
00929
00930     // Either acknowledge the interrupt immediately, or queue it.
00931     if (m_bInterruptQueueing == false)
00932     {
00933         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.PC;
00934         m_pusRAM[ m_stRegisters.SP-- ] = m_stRegisters.A;
00935
00936         m_stRegisters.A = usMessage_;
00937         m_stRegisters.PC = m_stRegisters.IA;
00938         m_bInterruptQueueing = true;
00939     }
00940     else
00941     {
00942         // Add interrupt message to the queue
00943         m_ausInterruptQueue[ ++m_ucQueueLevel ] = usMessage_;
00944     }
00945 }
00946
00947 //-----
00948 void DCPU::AddPlugin( DCPUPlugin *pclPlugin_ )
00949 {
00950     m_clPluginList.Add( (LinkListNode*)pclPlugin_ );
00951 }

```

15.47 /home/moslevin/Project/R1/stage/src/dcpu.h File Reference

DCPU-16 emulator.

```

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

```

Classes

- struct [DCPU_Registers](#)
Structure defining the [DCPU](#) hardware registers.
- class [DCPUPlugin](#)
Class used to provide the hardware device abstraction between the [DCPU-16](#) emulator/VM and the host system.

- class [DCPU](#)

[DCPU](#) emulator, used for running code out of EEPROM, RAM, or other memory interfaces than FLASH.

Macros

- #define [DCPU_NORMAL_OPCODE_MASK](#)(x) ((K_USHORT)(x & 0x001F))
[DCPU](#) v1.7 CPU emulator.
- #define [DCPU_EXTENDED_OPCODE_MASK](#)(x) ((K_USHORT)((x >> 5) & 0x001F))
- #define [DCPU_A_MASK](#)(x) ((K_USHORT)((x >> 10) & 0x003F))
- #define [DCPU_B_MASK](#)(x) ((K_USHORT)((x >> 5) & 0x001F))
- #define [DCPU_BUILD_NORMAL](#)(x, y, z) (((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 | ((K_USHORT)(z) & 0x003F) << 10)
- #define [DCPU_BUILD_EXTENDED](#)(x, y) (((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10))

Typedefs

- typedef void(* [DCPU_Callback](#))(DCPU *pclVM_)
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.

15.47.1 Detailed Description

DCPU-16 emulator.

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

15.47.2 Macro Definition Documentation

15.47.2.1 `#define DCPU_NORMAL_OPCODE_MASK(x) ((K_USHORT)(x & 0x001F))`

[DCPU](#) v1.7 CPU emulator.

Basic opcode format: [aaaaaabbbbbbooooo]

Where: - aaaaaa 6-bit source argument

- bbbbb 5-bit destination argument
- o is the opcode itself in a

If oooo = 0, then it's an "extended" opcode

Extended opcode format: [aaaaaaoooooxxxxx]

Where:

- xxxxx = all 0's - (basic opcode)
- ooooo = an extended opcode
- aaaaaa = the argument

Definition at line 48 of file [dcpu.h](#).

15.47.3 Enumeration Type Documentation

15.47.3.1 `enum DCPU_OpBasic`

[DCPU](#) Basic Opcodes.

Enumerator

OP_NON_BASIC special instruction - see below

OP_SET b, a | sets b to a

OP_ADD b, a | sets b to b+a, sets EX to 0x0001 if there's an overflow, 0x0 otherwise

OP_SUB b, a | sets b to b-a, sets EX to 0xffff if there's an underflow, 0x0 otherwise

OP_MUL b, a | sets b to b*a, sets EX to ((b*a)>>16)&0xffff (treats b, a as unsigned)

OP_MLI b, a | like MUL, but treat b, a as signed

OP_DIV b, a | sets b to b/a, sets EX to ((b<<16)/a)&0xffff. if a==0, sets b and EX to 0 instead. (treats b, a as unsigned)

OP_DVI b, a | like DIV, but treat b, a as signed. Rounds towards 0

OP_MOD b, a | sets b to ba. if a==0, sets b to 0 instead.

OP_MDI b, a | like MOD, but treat b, a as signed. (MDI -7, 16 == -7)

OP_AND b, a | sets b to b&a

OP_BOR b, a | sets b to b|a

OP_XOR b, a | sets b to b^a

OP_SHR b, a | sets b to b>>>a, sets EX to ((b<<16)>>>a)&0xffff (logical shift)

OP_ASR b, a | sets b to b>>>a, sets EX to ((b<<16)>>>a)&0xffff (arithmetic shift) (treats b as signed)

OP_SHL b, a | sets b to b<<<a, sets EX to ((b<<<a)>>>16)&0xffff

OP_IFB b, a | performs next instruction only if (b&a)!=0

OP_IFC b, a | performs next instruction only if (b&a)==0

OP_IFE b, a | performs next instruction only if b==a

OP_IFN b, a | performs next instruction only if b!=a

OP_IFG b, a | performs next instruction only if b>a

OP_IFA b, a | performs next instruction only if b>a (signed)

OP_IFL b, a | performs next instruction only if b<a

OP_IFU b, a | performs next instruction only if b<a (signed)

OP_18 UNDEFINED

OP_19 UNDEFINED

OP_ADX b, a | sets b to b+a+EX, sets EX to 0x0001 if there is an over-flow, 0x0 otherwise

OP_SBX b, a | sets b to b-a+EX, sets EX to 0xFFFF if there is an under-flow, 0x0 otherwise

OP_1C UNDEFINED

OP_1D UNDEFINED

OP_STI b, a | sets b to a, then increases I and J by 1

OP_STD b, a | sets b to a, then decreases I and J by 1

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

15.47.3.2 enum DCPU_OpExtended

[DCPU](#) Extended opcodes.

Enumerator

OP_EX_JSR a - pushes the address of the next instruction to the stack, then sets PC to a

OP_EX_2 UNDEFINED

OP_EX_3 UNDEFINED

OP_EX_4 UNDEFINED

OP_EX_5 UNDEFINED
OP_EX_6 UNDEFINED
OP_EX_7 UNDEFINED
OP_EX_INT Invoke software interrupt "a".
OP_EX_IAG Get interrupt address in "a".
OP_EX_IAS Set interrupt address from "a".
OP_EX_RFI Disables interrupt queueing, pops A from the stack, then pops PC from the stack.
OP_EX_IAQ if a is nonzero, interrupts will be added to the queue instead of triggered. if a is zero, interrupts will be triggered as normal again
OP_EX_D UNDEFINED
OP_EX_E UNDEFINED
OP_EX_F UNDEFINED
OP_EX_HWN Sets "a" to number of connected HW devices.
OP_EX_HWQ Set registers with information about hardware at index "a".
OP_EX_HWI Send an interrupt to hardware interface "a".
OP_EX_13 UNDEFINED
OP_EX_14 UNDEFINED
OP_EX_15 UNDEFINED
OP_EX_16 UNDEFINED
OP_EX_17 UNDEFINED
OP_EX_18 UNDEFINED
OP_EX_19 UNDEFINED
OP_EX_1A UNDEFINED
OP_EX_1B UNDEFINED
OP_EX_1C UNDEFINED
OP_EX_1D UNDEFINED
OP_EX_1E UNDEFINED
OP_EX_1F UNDEFINED

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

15.48 dcpu.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00018 #ifndef __DCPU_H__
00019 #define __DCPU_H__
00020
00021 #include "kerneltypes.h"
00022 #include "ll.h"
00023
00024 //-----
00046 //-----
00047 // Macros to access individual elements from within an opcode
00048 #define DCPU_NORMAL_OPCODE_MASK( x ) \

```



```

00049         ((K_USHORT)(x & 0x001F))
00050
00051 #define DCPU_EXTENDED_OPCODE_MASK( x ) \
00052         ((K_USHORT)((x >> 5) & 0x001F))
00053
00054 #define DCPU_A_MASK( x ) \
00055         ((K_USHORT)((x >> 10) & 0x003F))
00056
00057 #define DCPU_B_MASK( x ) \
00058         ((K_USHORT)((x >> 5) & 0x001F))
00059
00060 //-----
00061 // Macros to emit opcodes in the normal/extended formats
00062 #define DCPU_BUILD_NORMAL( x, y, z ) \
00063         ( ((K_USHORT)(x) & 0x001F) | ((K_USHORT)(y) & 0x001F) << 5 | ((K_USHORT)(z) & 0x003F) << 10 )
00064
00065 #define DCPU_BUILD_EXTENDED( x, y ) \
00066         ( ((K_USHORT)(x & 0x001F) << 5) | ((K_USHORT)(y & 0x003F) << 10) )
00067
00068 //-----
00072 typedef struct
00073 {
00074     union
00075     {
00076         struct
00077         {
00078             K_USHORT A;
00079             K_USHORT B;
00080             K_USHORT C;
00081             K_USHORT X;
00082             K_USHORT Y;
00083             K_USHORT Z;
00084             K_USHORT I;
00085             K_USHORT J;
00086             K_USHORT PC;
00087             K_USHORT SP;
00088             K_USHORT EX;
00089             K_USHORT IA;
00090         };
00091         K_USHORT ausRegisters[12];
00092     };
00093 } DCPU_Registers;
00094
00095 //-----
00099 typedef enum
00100 {
00101     OP_NON_BASIC = 0,
00102     OP_SET,
00103     OP_ADD,
00104     OP_SUB,
00105     OP_MUL,
00106     OP_MLI,
00107     OP_DIV,
00108     OP_DVI,
00109     OP_MOD,
00110     OP_MDI,
00111     OP_AND,
00112     OP_BOR,
00113     OP_XOR,
00114     OP_SHR,
00115     OP_ASR,
00116     OP_SHL,
00117     OP_IFB,
00118     OP_IFC,
00119     OP_IFE,
00120     OP_IFN,
00121     OP_IFG,
00122     OP_IFA,
00123     OP_IFL,
00124     OP_IFU,
00125     OP_18,
00126     OP_19,
00127     OP_ADX,
00128     OP_SBX,
00129     OP_1C,
00130     OP_1D,
00131     OP_STI,
00132     OP_STD
00133 } DCPU_OpBasic;
00134
00135 //-----
00139 typedef enum
00140 {
00141     OP_EX_RESERVED = 0,
00142     OP_EX_JSR,
00143     OP_EX_2,
00144     OP_EX_3,

```

```

00145     OP_EX_4,
00146     OP_EX_5,
00147     OP_EX_6,
00148     OP_EX_7,
00149     OP_EX_INT,
00150     OP_EX_IAG,
00151     OP_EX_IAS,
00152     OP_EX_RFI,
00153     OP_EX_IAQ,
00154     OP_EX_D,
00155     OP_EX_E,
00156     OP_EX_F,
00157     OP_EX_HWN,
00158     OP_EX_HWQ,
00159     OP_EX_HWI,
00160     OP_EX_13,
00161     OP_EX_14,
00162     OP_EX_15,
00163     OP_EX_16,
00164     OP_EX_17,
00165     OP_EX_18,
00166     OP_EX_19,
00167     OP_EX_1A,
00168     OP_EX_1B,
00169     OP_EX_1C,
00170     OP_EX_1D,
00171     OP_EX_1E,
00172     OP_EX_1F
00173 } DCPU_OpExtended;
00174
00175 //-----
00180 typedef enum
00181 {
00182     ARG_A = 0,
00183     ARG_B,
00184     ARG_C,
00185     ARG_X,
00186     ARG_Y,
00187     ARG_Z,
00188     ARG_I,
00189     ARG_J,
00190
00191     ARG_BRACKET_A,
00192     ARG_BRACKET_B,
00193     ARG_BRACKET_C,
00194     ARG_BRACKET_X,
00195     ARG_BRACKET_Y,
00196     ARG_BRACKET_Z,
00197     ARG_BRACKET_I,
00198     ARG_BRACKET_J,
00199
00200     ARG_WORD_A,
00201     ARG_WORD_B,
00202     ARG_WORD_C,
00203     ARG_WORD_X,
00204     ARG_WORD_Y,
00205     ARG_WORD_Z,
00206     ARG_WORD_I,
00207     ARG_WORD_J,
00208
00209     ARG_PUSH_POP_SP,
00210     ARG_PEEK_SP,
00211     ARG_WORD_SP,
00212     ARG_SP,
00213     ARG_PC,
00214     ARG_EX,
00215     ARG_NEXT_WORD,
00216     ARG_NEXT_LITERAL,
00217
00218     ARG_LITERAL_0,
00219     ARG_LITERAL_1,
00220     ARG_LITERAL_2,
00221     ARG_LITERAL_3,
00222     ARG_LITERAL_4,
00223     ARG_LITERAL_5,
00224     ARG_LITERAL_6,
00225     ARG_LITERAL_7,
00226     ARG_LITERAL_8,
00227     ARG_LITERAL_9,
00228     ARG_LITERAL_A,
00229     ARG_LITERAL_B,
00230     ARG_LITERAL_C,
00231     ARG_LITERAL_D,
00232     ARG_LITERAL_E,
00233     ARG_LITERAL_F,
00234     ARG_LITERAL_10,
00235     ARG_LITERAL_11,

```

```

00236     ARG_LITERAL_12,
00237     ARG_LITERAL_13,
00238     ARG_LITERAL_14,
00239     ARG_LITERAL_15,
00240     ARG_LITERAL_16,
00241     ARG_LITERAL_17,
00242     ARG_LITERAL_18,
00243     ARG_LITERAL_19,
00244     ARG_LITERAL_1A,
00245     ARG_LITERAL_1B,
00246     ARG_LITERAL_1C,
00247     ARG_LITERAL_1D,
00248     ARG_LITERAL_1E,
00249     ARG_LITERAL_1F
00250
00251 } DCPU_Argument;
00252
00253 //-----
00254 class DCPU;    // Forward declaration - required by the plugin class
00255
00256 //-----
00260 typedef void (*DCPU_Callback)(DCPU *pclVM_);
00261
00262 //-----
00267 class 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         ClearNode();
00300     }
00301
00312     void Enumerate( DCPU_Registers *pstRegisters_ )
00313     {
00314         pstRegisters_>A = (K_USHORT)(m_ulHWID & 0x0000FFFF);
00315         pstRegisters_>B = (K_USHORT)((m_ulHWID >> 16) & 0x0000FFFF);
00316         pstRegisters_>C = m_usVersion;
00317         pstRegisters_>X = (K_USHORT)(m_ulVID & 0x0000FFFF);
00318         pstRegisters_>Y = (K_USHORT)((m_ulVID >> 16) & 0x0000FFFF);
00319     }
00320
00328     void Interrupt( DCPU *pclCPU_ )
00329     {
00330         m_pfCallback(pclCPU_);
00331     }
00332
00340     K_USHORT GetDeviceNumber()
00341     {
00342         return m_usDeviceNumber;
00343     }
00344
00345     friend class DCPUPuginList;
00346 private:
00347     K_USHORT      m_usDeviceNumber;
00348     K_ULONG       m_ulHWID;
00349     K_ULONG       m_ulVID;
00350     K_USHORT      m_usVersion;
00351
00352     DCPU_Callback m_pfCallback;
00353 };
00354
00355 //-----
00360 class DCPU
00361 {
00362 public:
00376     void Init( K_USHORT *pusRAM_, K_USHORT usRAMSize_, const K_USHORT *pusROM_, K_USHORT usROMSize_ );
00377
00383     void RunOpcode();
00384
00392     DCPU_Registers *GetRegisters() { return &
m_stRegisters; }
00393
00401     void SendInterrupt( K_USHORT usMessage_ );
00402
00410     void AddPlugin( DCPUPugin *pclPlugin_ );
00411
00412 private:
00413

```

```

00414 // Basic opcodes
00415 void SET();
00416 void ADD();
00417 void SUB();
00418 void MUL();
00419 void MLI();
00420 void DIV();
00421 void DVI();
00422 void MOD();
00423 void MDI();
00424 void AND();
00425 void BOR();
00426 void XOR();
00427 void SHR();
00428 void ASR();
00429 void SHL();
00430 bool IFB();
00431 bool IFC();
00432 bool IFE();
00433 bool IFN();
00434 bool IFG();
00435 bool IFA();
00436 bool IFL();
00437 bool IFU();
00438 void ADX();
00439 void SBX();
00440 void STI();
00441 void STD();
00442
00443 // Extended opcodes
00444 void JSR();
00445 void INT();
00446 void IAG();
00447 void IAS();
00448 void RFI();
00449 void IAQ();
00450 void HWN();
00451 void HWQ();
00452 void HWI();
00453
00461 K_UCHAR GetOperand( K_UCHAR ucOpType_, K_USHORT **pusResult_ );
00462
00463
00469 void ProcessInterruptQueue();
00470
00471 DCPU_Registers m_stRegisters;
00472
00473 K_USHORT *a;
00474 K_USHORT *b;
00475
00476 K_USHORT m_usTempA;
00477
00478 K_USHORT *m_pusRAM;
00479 K_USHORT m_usRAMSize;
00480
00481 K_USHORT *m_pusROM;
00482 K_USHORT m_usROMSize;
00483
00484 K_ULONG m_ulCycleCount;
00485
00486 K_BOOL m_bInterruptQueueing;
00487 K_UCHAR m_ucQueueLevel;
00488 K_USHORT m_ausInterruptQueue[ 8 ];
00489
00490 DoubleLinkedList m_clPluginList;
00491 };
00492
00493 #endif

```

15.49 /home/moslevin/Project/R1/stage/src/debug_tokens.h File Reference

Hex codes/translation tables used for efficient string tokenization.

Macros

- #define **BLOCKING_CPP** 0x0001 /* SUBSTITUTE="blocking.cpp" */
Source file names start at 0x0000.
- #define **DRIVER_CPP** 0x0002 /* SUBSTITUTE="driver.cpp" */

- #define **KERNEL_CPP** 0x0003 /* SUBSTITUTE="kernel.cpp" */
- #define **LL_CPP** 0x0004 /* SUBSTITUTE="ll.cpp" */
- #define **MESSAGE_CPP** 0x0005 /* SUBSTITUTE="message.cpp" */
- #define **MUTEX_CPP** 0x0006 /* SUBSTITUTE="mutex.cpp" */
- #define **PROFILE_CPP** 0x0007 /* SUBSTITUTE="profile.cpp" */
- #define **QUANTUM_CPP** 0x0008 /* SUBSTITUTE="quantum.cpp" */
- #define **SCHEDULER_CPP** 0x0009 /* SUBSTITUTE="scheduler.cpp" */
- #define **SEMAPHORE_CPP** 0x000A /* SUBSTITUTE="semaphore.cpp" */
- #define **THREAD_CPP** 0x000B /* SUBSTITUTE="thread.cpp" */
- #define **THREADLIST_CPP** 0x000C /* SUBSTITUTE="threadlist.cpp" */
- #define **TIMERLIST_CPP** 0x000D /* SUBSTITUTE="timerlist.cpp" */
- #define **KERNELSWI_CPP** 0x000E /* SUBSTITUTE="kernelswi.cpp" */
- #define **KERNELTIMER_CPP** 0x000F /* SUBSTITUTE="kerneltimer.cpp" */
- #define **KPROFILE_CPP** 0x0010 /* SUBSTITUTE="kprofile.cpp" */
- #define **THREADPORT_CPP** 0x0011 /* SUBSTITUTE="threadport.cpp" */
- #define **BLOCKING_H** 0x1000 /* SUBSTITUTE="blocking.h" */

Header file names start at 0x1000.

- #define **DRIVER_H** 0x1001 /* SUBSTITUTE="driver.h" */
- #define **KERNEL_H** 0x1002 /* SUBSTITUTE="kernel.h" */
- #define **KERNELTYPES_H** 0x1003 /* SUBSTITUTE="kerneltypes.h" */
- #define **LL_H** 0x1004 /* SUBSTITUTE="ll.h" */
- #define **MANUAL_H** 0x1005 /* SUBSTITUTE="manual.h" */
- #define **MARK3CFG_H** 0x1006 /* SUBSTITUTE="mark3cfg.h" */
- #define **MESSAGE_H** 0x1007 /* SUBSTITUTE="message.h" */
- #define **MUTEX_H** 0x1008 /* SUBSTITUTE="mutex.h" */
- #define **PROFILE_H** 0x1009 /* SUBSTITUTE="profile.h" */
- #define **PROFILING_RESULTS_H** 0x100A /* SUBSTITUTE="profiling_results.h" */
- #define **QUANTUM_H** 0x100B /* SUBSTITUTE="quantum.h" */
- #define **SCHEDULER_H** 0x100C /* SUBSTITUTE="scheduler.h" */
- #define **SEMAPHORE_H** 0x100D /* SUBSTITUTE="ksemaphore.h" */
- #define **THREAD_H** 0x100E /* SUBSTITUTE="thread.h" */
- #define **THREADLIST_H** 0x100F /* SUBSTITUTE="threadlist.h" */
- #define **TIMERLIST_H** 0x1010 /* SUBSTITUTE="timerlist.h" */
- #define **KERNELSWI_H** 0x1011 /* SUBSTITUTE="kernelswi.h" */
- #define **KERNELTIMER_H** 0x1012 /* SUBSTITUTE="kerneltimer.h" */
- #define **KPROFILE_H** 0x1013 /* SUBSTITUTE="kprofile.h" */
- #define **THREADPORT_H** 0x1014 /* SUBSTITUTE="threadport.h" */
- #define **STR_PANIC** 0x2000 /* SUBSTITUTE="!Panic!" */

Indexed strings start at 0x2000.

- #define **STR_MARK3_INIT** 0x2001 /* SUBSTITUTE="Initializing Kernel Objects" */
- #define **STR_KERNEL_ENTER** 0x2002 /* SUBSTITUTE="Starting Kernel" */
- #define **STR_THREAD_START** 0x2003 /* SUBSTITUTE="Switching to First Thread" */
- #define **STR_START_ERROR** 0x2004 /* SUBSTITUTE="Error starting kernel - function should never return" */
- #define **STR_THREAD_CREATE** 0x2005 /* SUBSTITUTE="Creating Thread" */
- #define **STR_STACK_SIZE_1** 0x2006 /* SUBSTITUTE=" Stack Size: %1" */
- #define **STR_PRIORITY_1** 0x2007 /* SUBSTITUTE=" Priority: %1" */
- #define **STR_THREAD_ID_1** 0x2008 /* SUBSTITUTE=" Thread ID: %1" */
- #define **STR_ENTRYPOINT_1** 0x2009 /* SUBSTITUTE=" EntryPoint: %1" */
- #define **STR_CONTEXT_SWITCH_1** 0x200A /* SUBSTITUTE="Context Switch To Thread: %1" */
- #define **STR_IDLING** 0x200B /* SUBSTITUTE="Idling CPU" */
- #define **STR_WAKEUP** 0x200C /* SUBSTITUTE="Waking up" */
- #define **STR_SEMAPHORE_PEND_1** 0x200D /* SUBSTITUTE="Semaphore Pend: %1" */
- #define **STR_SEMAPHORE_POST_1** 0x200E /* SUBSTITUTE="Semaphore Post: %1" */

- `#define STR_MUTEX_CLAIM_1 0x200F /* SUBSTITUTE="Mutex Claim: %1" */`
- `#define STR_MUTEX_RELEASE_1 0x2010 /* SUBSTITUTE="Mutex Release: %1" */`
- `#define STR_THREAD_BLOCK_1 0x2011 /* SUBSTITUTE="Thread %1 Blocked" */`
- `#define STR_THREAD_UNBLOCK_1 0x2012-2015 /* 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" */`

15.49.1 Detailed Description

Hex codes/translation tables used for efficient string tokenization. We use this for efficiently encoding strings used for kernel traces, debug prints, etc. The upside - this is really fast and efficient for encoding strings and data. Downside? The tools need to parse this header file in order to convert the enumerated data into actual strings, decoding them.

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

15.50 debug_tokens.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00025 #ifndef __DEBUG_TOKENS_H__
00026 #define __DEBUG_TOKENS_H__
00027 //-----
00029 #define BLOCKING_CPP          0x0001      /* SUBSTITUTE="blocking.cpp" */
00030 #define DRIVER_CPP           0x0002      /* SUBSTITUTE="driver.cpp" */
00031 #define KERNEL_CPP           0x0003      /* SUBSTITUTE="kernel.cpp" */
00032 #define LL_CPP               0x0004      /* SUBSTITUTE="ll.cpp" */
00033 #define MESSAGE_CPP          0x0005      /* SUBSTITUTE="message.cpp" */
00034 #define MUTEX_CPP            0x0006      /* SUBSTITUTE="mutex.cpp" */
00035 #define PROFILE_CPP          0x0007      /* SUBSTITUTE="profile.cpp" */
00036 #define QUANTUM_CPP          0x0008      /* SUBSTITUTE="quantum.cpp" */
00037 #define SCHEDULER_CPP        0x0009      /* SUBSTITUTE="scheduler.cpp" */
00038 #define SEMAPHORE_CPP        0x000A      /* SUBSTITUTE="semaphore.cpp" */
00039 #define THREAD_CPP           0x000B      /* SUBSTITUTE="thread.cpp" */
00040 #define THREADLIST_CPP       0x000C      /* SUBSTITUTE="threadlist.cpp" */
00041 #define TIMERLIST_CPP        0x000D      /* SUBSTITUTE="timerlist.cpp" */
00042 #define KERNELSWI_CPP        0x000E      /* SUBSTITUTE="kernelswi.cpp" */
00043 #define KERNELTIMER_CPP      0x000F      /* SUBSTITUTE="kerneltimer.cpp" */
00044 #define KPROFILE_CPP         0x0010      /* SUBSTITUTE="kprofile.cpp" */
00045 #define THREADPORT_CPP       0x0011      /* SUBSTITUTE="threadport.cpp" */
00046
00047 //-----
00049 #define BLOCKING_H           0x1000      /* SUBSTITUTE="blocking.h" */
00050 #define DRIVER_H            0x1001      /* SUBSTITUTE="driver.h" */
00051 #define KERNEL_H            0x1002      /* SUBSTITUTE="kernel.h" */
00052 #define KERNELTYPES_H       0x1003      /* SUBSTITUTE="kerneltypes.h" */
00053 #define LL_H                0x1004      /* SUBSTITUTE="ll.h" */
00054 #define MANUAL_H            0x1005      /* SUBSTITUTE="manual.h" */
00055 #define MARK3CFG_H          0x1006      /* SUBSTITUTE="mark3cfg.h" */
00056 #define MESSAGE_H           0x1007      /* SUBSTITUTE="message.h" */
00057 #define MUTEX_H             0x1008      /* SUBSTITUTE="mutex.h" */
00058 #define PROFILE_H           0x1009      /* SUBSTITUTE="profile.h" */
00059 #define PROFILING_RESULTS_H 0x100A      /* SUBSTITUTE="profiling_results.h" */
00060 #define QUANTUM_H           0x100B      /* SUBSTITUTE="quantum.h" */
00061 #define SCHEDULER_H         0x100C      /* SUBSTITUTE="scheduler.h" */
00062 #define SEMAPHORE_H         0x100D      /* SUBSTITUTE="ksemaphore.h" */
00063 #define THREAD_H            0x100E      /* SUBSTITUTE="thread.h" */
00064 #define THREADLIST_H        0x100F      /* SUBSTITUTE="threadlist.h" */
00065 #define TIMERLIST_H         0x1010      /* SUBSTITUTE="timerlist.h" */

```

```

00066 #define KERNELSWI_H          0x1011      /* SUBSTITUTE="kernelswi.h */
00067 #define KERNELTIMER_H         0x1012      /* SUBSTITUTE="kerneltimer.h */
00068 #define KPROFILE_H            0x1013      /* SUBSTITUTE="kprofile.h" */
00069 #define THREADPORT_H          0x1014      /* SUBSTITUTE="threadport.h" */
00070
00071 //-----
00073 #define STR_PANIC              0x2000      /* SUBSTITUTE="!Panic!" */
00074 #define STR_MARK3_INIT        0x2001      /* SUBSTITUTE="Initializing Kernel Objects" */
00075 #define STR_KERNEL_ENTER      0x2002      /* SUBSTITUTE="Starting Kernel" */
00076 #define STR_THREAD_START      0x2003      /* SUBSTITUTE="Switching to First Thread" */
00077 #define STR_START_ERROR       0x2004      /* SUBSTITUTE="Error starting kernel - function should never
return" */
00078 #define STR_THREAD_CREATE     0x2005      /* SUBSTITUTE="Creating Thread" */
00079 #define STR_STACK_SIZE_1      0x2006      /* SUBSTITUTE=" Stack Size: %1" */
00080 #define STR_PRIORITY_1        0x2007      /* SUBSTITUTE=" Priority: %1" */
00081 #define STR_THREAD_ID_1       0x2008      /* SUBSTITUTE=" Thread ID: %1" */
00082 #define STR_ENTRYPOINT_1      0x2009      /* SUBSTITUTE=" EntryPoint: %1" */
00083 #define STR_CONTEXT_SWITCH_1  0x200A      /* SUBSTITUTE="Context Switch To Thread: %1" */
00084 #define STR_IDLING            0x200B      /* SUBSTITUTE="Idling CPU" */
00085 #define STR_WAKEUP            0x200C      /* SUBSTITUTE="Waking up" */
00086 #define STR_SEMAPHORE_PEND_1  0x200D      /* SUBSTITUTE="Semaphore Pend: %1" */
00087 #define STR_SEMAPHORE_POST_1  0x200E      /* SUBSTITUTE="Semaphore Post: %1" */
00088 #define STR_MUTEX_CLAIM_1     0x200F      /* SUBSTITUTE="Mutex Claim: %1" */
00089 #define STR_MUTEX_RELEASE_1   0x2010      /* SUBSTITUTE="Mutex Release: %1" */
00090 #define STR_THREAD_BLOCK_1    0x2011      /* SUBSTITUTE="Thread %1 Blocked" */
00091 #define STR_THREAD_UNBLOCK_1  0x2012-2015 /* 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

```

15.51 /home/moslevin/Project/R1/stage/src/draw.h File Reference

Raster graphics APIs Description: Implements basic drawing functionality.

```

#include "kerneltypes.h"
#include "font.h"
#include "colorspace.h"

```

Classes

- struct [DrawPoint_t](#)
Defines a pixel.
- struct [DrawLine_t](#)
Defines a simple line.
- struct [DrawRectangle_t](#)
Defines a rectangle.
- struct [DrawCircle_t](#)
Defines a circle.
- struct [DrawEllipse_t](#)
Defines an ellipse.
- struct [DrawBitmap_t](#)
Defines a bitmap.
- struct [DrawStamp_t](#)
Defines a 1-bit 2D bitmap of arbitrary resolution.
- struct [DrawText_t](#)
Defines a bitmap-rendered string.
- struct [TextFX_t](#)
- struct [DrawWindow_t](#)
Defines the active window - establishes boundaries for drawing on the current display.


```

00031 // Event definitions for 2D hardware accelerated graphics functions
00032 typedef enum
00033 {
00034     //--[Mandatory for a display driver]-----
00035     DISPLAY_EVENT_SET_PIXEL = 0x00,
00036     DISPLAY_EVENT_GET_PIXEL,
00037
00038     //--[Optional if supported in hardware]-----
00039     DISPLAY_EVENT_CLEAR,
00040     DISPLAY_EVENT_LINE,
00041     DISPLAY_EVENT_RECTANGLE,
00042     DISPLAY_EVENT_CIRCLE,
00043     DISPLAY_EVENT_ELLIPSE,
00044     DISPLAY_EVENT_BITMAP,
00045     DISPLAY_EVENT_STAMP,
00046     DISPLAY_EVENT_TEXT,
00047     DISPLAY_EVENT_MOVE,
00048     DISPLAY_EVENT_POLY
00049 } DisplayEvent_t;
00050
00051 //-----
00052 typedef struct
00053 {
00054     K_USHORT usX;
00055     K_USHORT usY;
00056     COLOR uColor;
00057 } DrawPoint_t;
00058
00059 //-----
00060 typedef struct
00061 {
00062     K_USHORT usX1;
00063     K_USHORT usX2;
00064     K_USHORT usY1;
00065     K_USHORT usY2;
00066     COLOR uColor;
00067 } DrawLine_t;
00068
00069 //-----
00070 typedef struct
00071 {
00072     K_USHORT usLeft;
00073     K_USHORT usTop;
00074     K_USHORT usRight;
00075     K_USHORT usBottom;
00076     COLOR uLineColor;
00077     K_BOOL bFill;
00078     COLOR uFillColor;
00079 } DrawRectangle_t;
00080
00081 //-----
00082 typedef struct
00083 {
00084     K_USHORT usX;
00085     K_USHORT usY;
00086     K_USHORT usRadius;
00087     COLOR uLineColor;
00088     K_BOOL bFill;
00089     COLOR uFillColor;
00090 } DrawCircle_t;
00091
00092 //-----
00093 typedef struct
00094 {
00095     K_USHORT usX;
00096     K_USHORT usY;
00097     K_USHORT usHeight;
00098     K_USHORT usWidth;
00099     COLOR uColor;
00100 } DrawEllipse_t;
00101
00102 //-----
00103 typedef struct
00104 {
00105     K_USHORT usX;
00106     K_USHORT usY;
00107     K_USHORT usWidth;
00108     K_USHORT usHeight;
00109     K_UCHAR ucBPP;
00110     K_UCHAR *pucData;
00111 } DrawBitmap_t;
00112
00113 //-----
00114 typedef struct
00115 {
00116     K_USHORT usX;
00117     K_USHORT usY;
00118     K_USHORT usWidth;
00119     K_USHORT usHeight;
00120     COLOR uColor;
00121     K_UCHAR *pucData;
00122 } DrawStamp_t; // monochrome stamp, bitpacked 8bpp

```

```

00139
00140 //-----
00144 typedef struct
00145 {
00146     K_USHORT usLeft;
00147     K_USHORT usTop;
00148     COLOR uColor;
00149     Font_t *pstFont;
00150     const K_CHAR *pcString;
00151 } DrawText_t;
00152
00153 //-----
00154 #define TEXTFX_FLAG_OPAQUE_BG    (0x01)
00155 #define TEXTFX_FLAG_ROTATE      (0x02)
00156 #define TEXTFX_FLAG_SCALE_X     (0x04)
00157 #define TEXTFX_FLAG_SCALE_Y     (0x08)
00158
00159 //-----
00160 typedef struct
00161 {
00162     K_UCHAR ucFlags;
00163     COLOR uBGColor;
00164     K_USHORT usRotateDeg;
00165     K_USHORT usScaleX100;
00166     K_USHORT usScaleY100;
00167 } TextFX_t;
00168
00169 //-----
00175 typedef struct
00176 {
00177     K_USHORT usLeft;
00178     K_USHORT usRight;
00179     K_USHORT usTop;
00180     K_USHORT usBottom;
00181 } DrawWindow_t;
00182
00183 //-----
00188 typedef struct
00189 {
00190     K_USHORT usSrcX;
00191     K_USHORT usSrcY;
00192     K_USHORT usDstX;
00193     K_USHORT usDstY;
00194     K_USHORT usCopyHeight;
00195     K_USHORT usCopyWidth;
00196 } DrawMove_t;
00197
00198 //-----
00204 typedef struct
00205 {
00206     K_USHORT usX;
00207     K_USHORT usY;
00208 } DrawVector_t;
00209
00210 //-----
00215 typedef struct
00216 {
00217     K_USHORT usNumPoints;
00218     COLOR uColor;
00219     K_BOOL bFill;
00220     DrawVector_t *pstVector;
00221 } DrawPoly_t;
00222
00223 #endif //__DRAW_H_

```

15.53 /home/moslevin/Project/R1/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**

15.53.1 Detailed Description

Device driver/hardware abstraction layer.

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

15.54 driver.cpp

```

00001  /*=====
00002
00003  _____
00004  | \ / | | | \ / | | | \ / | | | \ / | | |
00005  | \ / | | | \ / | | | \ / | | | \ / | | |
00006  | \ / | | | \ / | | | \ / | | | \ / | | |
00007  | \ / | | | \ / | | | \ / | | | \ / | | |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 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
00040  class DevNull : public Driver
00041  {
00042  public:
00043      virtual void Init() { SetName("/dev/null"); };
00044      virtual K_UCHAR Open() { return 0; }
00045      virtual K_UCHAR Close() { return 0; }
00046
00047      virtual K_USHORT Read( K_USHORT usBytes_,
00048          K_UCHAR *pucData_){ return 0; }
00049
00050      virtual K_USHORT Write( K_USHORT usBytes_,
00051          K_UCHAR *pucData_){ return 0; }
00052
00053      virtual K_USHORT Control( K_USHORT usEvent_,

```

```

00054         void *pvDataIn_,
00055         K_USHORT usSizeIn_,
00056         void *pvDataOut_,
00057         K_USHORT usSizeOut_ ) { return 0; }
00058
00059 };
00060
00061 //-----
00062 static DevNull clDevNull;
00063
00064 //-----
00065 static K_UCHAR DrvCmp( const K_CHAR *szStr1_, const K_CHAR *szStr2_ )
00066 {
00067     K_CHAR *szTmp1 = (K_CHAR*) szStr1_;
00068     K_CHAR *szTmp2 = (K_CHAR*) szStr2_;
00069
00070     while (*szTmp1 && *szTmp2)
00071     {
00072         if (*szTmp1++ != *szTmp2++)
00073         {
00074             return 0;
00075         }
00076     }
00077
00078     // Both terminate at the same length
00079     if (!(*szTmp1) && !(*szTmp2))
00080     {
00081         return 1;
00082     }
00083
00084     return 0;
00085 }
00086
00087 //-----
00088 void DriverList::Init()
00089 {
00090     // Ensure we always have at least one entry - a default in case no match
00091     // is found (/dev/null)
00092     clDevNull.Init();
00093     Add(&clDevNull);
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

```

15.55 /home/moslevin/Project/R1/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.

15.55.1 Detailed Description

[Driver](#) abstraction framework.

15.55.2 Intro

This is the basis of the driver framework. In the context of Mark3, drivers don't necessarily have to be based on physical hardware peripherals. They can be used to represent algorithms (such as random number generators), files, or protocol stacks. Unlike FunkOS, where driver IO is protected automatically by a mutex, we do not use this kind of protection - we leave it up to the driver implementor to do what's right in its own context. This also frees up the driver to implement all sorts of other neat stuff, like sending messages to threads associated with the driver. Drivers are implemented as character devices, with the standard array of posix-style accessor methods for reading, writing, and general driver control.

A global driver list is provided as a convenient and minimal "filesystem" structure, in which devices can be accessed by name.

15.55.3 Driver Design

A device driver needs to be able to perform the following operations: -Initialize a peripheral -Start/stop a peripheral -Handle I/O control operations -Perform various read/write operations

At the end of the day, that's pretty much all a device driver has to do, and all of the functionality that needs to be presented to the developer.

We abstract all device drivers using a base-class which implements the following methods: -Start/Open -Stop/Close -Control -Read -Write

A basic driver framework and API can thus be implemented in five function calls - that's it! You could even reduce that further by handling the initialize, start, and stop operations inside the "control" operation.

15.55.4 Driver API

In C++, we can implement this as a class to abstract these event handlers, with virtual void functions in the base class overridden by the inherited objects.

To add and remove device drivers from the global table, we use the following methods:

```
void DriverList::Add( Driver *pclDriver_ );
void DriverList::Remove( Driver *pclDriver_ );
```

[DriverList::Add\(\)](#)/[Remove\(\)](#) takes a single arguments the pointer to he object to operate on.

Once a driver has been added to the table, drivers are opened by NAME using [DriverList::FindByName\("/dev/name"\)](#). This function returns a pointer to the specified driver if successful, or to a built in /dev/null device if the path name is invalid. After a driver is open, that pointer is used for all other driver access functions.

This abstraction is incredibly useful any peripheral or service can be accessed through a consistent set of APIs, that make it easy to substitute implementations from one platform to another. Portability is ensured, the overhead is negligible, and it emphasizes the reuse of both driver and application code as separate entities.

Consider a system with drivers for I2C, SPI, and UART peripherals - under our driver framework, an application can initialize these peripherals and write a greeting to each using the same simple API functions for all drivers:

```
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](#).

15.56 driver.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00105 #include "kerneltypes.h"
00106 #include "mark3cfg.h"
00107
00108 #include "ll.h"
00109
00110 #ifndef __DRIVER_H__
00111 #define __DRIVER_H__
00112
00113 #if KERNEL_USE_DRIVER
00114
00115 class DriverList;
00116 //-----
00121 class Driver : public LinkListNode
00122 {
00123 public:
00129     virtual void Init() = 0;
00130
00138     virtual K_UCHAR Open() = 0;
00139
00147     virtual K_UCHAR Close() = 0;
00148
00164     virtual K_USHORT Read( K_USHORT usBytes_,
00165                           K_UCHAR *pucData_) = 0;
00166
00183     virtual K_USHORT Write( K_USHORT usBytes_,
00184                            K_UCHAR *pucData_) = 0;
00185
00208     virtual K_USHORT Control( K_USHORT usEvent_,
00209                              void *pvDataIn_,
00210                              K_USHORT usSizeIn_,
00211                              void *pvDataOut_,
00212                              K_USHORT usSizeOut_ ) = 0;
00213
00222     void SetName( const K_CHAR *pcName_ ) { m_pcPath = pcName_; }
00223
00231     const K_CHAR *GetPath() { return m_pcPath; }
00232
00233 private:
00234
00236     const K_CHAR *m_pcPath;
00237 };
00238
00239 //-----
00244 class DriverList
00245 {
00246 public:
00254     static void Init();
00255
00264     static void Add( Driver *pclDriver_ ) { m_clDriverList.
Add(pclDriver_); }
00265
00274     static void Remove( Driver *pclDriver_ ) { m_clDriverList.
Remove(pclDriver_); }
00275
00282     static Driver *FindByPath( const K_CHAR *m_pcPath );
00283
00284 private:
00285
00287     static DoubleLinkedList m_clDriverList;
00288 };
00289
```

```
00290 #endif //KERNEL_USE_DRIVER
00291
00292 #endif
```

15.57 /home/moslevin/Project/R1/stage/src/eventflag.cpp File Reference

Event Flag Blocking Object/IPC-Object implementation.

```
#include "mark3cfg.h"
#include "blocking.h"
#include "kernel.h"
#include "thread.h"
#include "eventflag.h"
#include "timerlist.h"
```

Functions

- void **TimedEventFlag_Callback** ([Thread](#) *pclOwner_, void *pvData_)

15.57.1 Detailed Description

Event Flag Blocking Object/IPC-Object implementation.

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

15.58 eventflag.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "mark3cfg.h"
00020 #include "blocking.h"
00021 #include "kernel.h"
00022 #include "thread.h"
00023 #include "eventflag.h"
00024
00025 #if KERNEL_USE_EVENTFLAG
00026
00027 #if KERNEL_USE_TIMEOUTS
00028 #include "timerlist.h"
00029 //-----
00030 void TimedEventFlag_Callback(Thread *pclOwner_, void *pvData_)
00031 {
00032     EventFlag *pclEventFlag = static_cast<EventFlag> (pvData_);
00033
00034     pclEventFlag->WakeMe(pclOwner_);
00035     pclOwner_->SetExpired(true);
00036     pclOwner_->SetEventFlagMask(0);
00037
00038     if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread\(\)->
GetPriority\(\))
00039     {
00040         Thread::Yield\(\);
00041     }
00042 }
00043
00044 //-----
```

```

00045 void EventFlag::WakeMe(Thread *pclChosenOne_)
00046 {
00047     Unblock(pclChosenOne_);
00048 }
00049 #endif
00050
00051 //-----
00052 #if KERNEL_USE_TIMEOUTS
00053 K_USHORT EventFlag::Wait_i(K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG
ulTimeMS_)
00054 #else
00055 K_USHORT EventFlag::Wait_i(K_USHORT usMask_, EventFlagOperation_t eMode_)
00056 #endif
00057 {
00058     bool bThreadYield = false;
00059     bool bMatch = false;
00060
00061 #if KERNEL_USE_TIMEOUTS
00062     Timer clEventTimer;
00063     bool bUseTimer = false;
00064 #endif
00065
00066     // Ensure we're operating in a critical section while we determine
00067     // whether or not we need to block the current thread on this object.
00068     CS_ENTER();
00069
00070     // Check to see whether or not the current mask matches any of the
00071     // desired bits.
00072     g_pstCurrent->SetEventFlagMask(usMask_);
00073
00074     if ((eMode_ == EVENT_FLAG_ALL) || (eMode_ == EVENT_FLAG_ALL_CLEAR))
00075     {
00076         // Check to see if the flags in their current state match all of
00077         // the set flags in the event flag group, with this mask.
00078         if ((m_usSetMask & usMask_) == usMask_)
00079         {
00080             bMatch = true;
00081             g_pstCurrent->SetEventFlagMask(usMask_);
00082         }
00083     }
00084     else if ((eMode_ == EVENT_FLAG_ANY) || (eMode_ == EVENT_FLAG_ANY_CLEAR))
00085     {
00086         // Check to see if the existing flags match any of the set flags in
00087         // the event flag group with this mask
00088         if (m_usSetMask & usMask_)
00089         {
00090             bMatch = true;
00091             g_pstCurrent->SetEventFlagMask(m_usSetMask & usMask_);
00092         }
00093     }
00094
00095     // We're unable to match this pattern as-is, so we must block.
00096     if (!bMatch)
00097     {
00098         // Reset the current thread's event flag mask & mode
00099         g_pstCurrent->SetEventFlagMask(usMask_);
00100         g_pstCurrent->SetEventFlagMode(eMode_);
00101
00102 #if KERNEL_USE_TIMEOUTS
00103         if (ulTimeMS_)
00104         {
00105             g_pstCurrent->SetExpired(false);
00106             clEventTimer.Init();
00107             clEventTimer.Start(0, ulTimeMS_, TimedEventFlag_Callback, (void*)this);
00108             bUseTimer = true;
00109         }
00110 #endif
00111
00112         // Add the thread to the object's block-list.
00113         Block(g_pstCurrent);
00114
00115         // Trigger that
00116         bThreadYield = true;
00117     }
00118
00119     // If bThreadYield is set, it means that we've blocked the current thread,
00120     // and must therefore rerun the scheduler to determine what thread to
00121     // switch to.
00122     if (bThreadYield)
00123     {
00124         // Switch threads immediately
00125         Thread::Yield();
00126     }
00127
00128     // Exit the critical section and return back to normal execution
00129     CS_EXIT();
00130

```



```

00135 #if KERNEL_USE_TIMEOUTS
00136     if (bUseTimer && bThreadYield)
00137     {
00138         clEventTimer.Stop();
00139     }
00140 #endif
00141
00142     return g_pstCurrent->GetEventFlagMask();
00143 }
00144
00145 //-----
00146 K_USHORT EventFlag::Wait(K_USHORT usMask_, EventFlagOperation_t eMode_)
00147 {
00148     #if KERNEL_USE_TIMEOUTS
00149         return Wait_i(usMask_, eMode_, 0);
00150     #else
00151         return Wait_i(usMask_, eMode_);
00152     #endif
00153 }
00154
00155 #if KERNEL_USE_TIMEOUTS
00156 //-----
00157 K_USHORT EventFlag::Wait(K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_)
00158 {
00159     return Wait_i(usMask_, eMode_, ulTimeMS_);
00160 }
00161 #endif
00162
00163 //-----
00164 void EventFlag::Set(K_USHORT usMask_)
00165 {
00166     Thread *pclPrev;
00167     Thread *pclCurrent;
00168     bool bReschedule = false;
00169     K_USHORT usNewMask;
00170
00171     CS_ENTER();
00172
00173     // Walk through the whole block list, checking to see whether or not
00174     // the current flag set now matches any/all of the masks and modes of
00175     // the threads involved.
00176
00177     m_usSetMask |= usMask_;
00178     usNewMask = m_usSetMask;
00179
00180     // Start at the head of the list, and iterate through until we hit the
00181     // "head" element in the list again. Ensure that we handle the case where
00182     // we remove the first or last elements in the list, or if there's only
00183     // one element in the list.
00184     pclCurrent = static_cast<Thread*>(m_clBlockList.GetHead());
00185
00186     // Do nothing when there are no objects blocking.
00187     if (pclCurrent)
00188     {
00189         // First loop - process every thread in the block-list and check to
00190         // see whether or not the current flags match the event-flag conditions
00191         // on the thread.
00192         do
00193         {
00194             pclPrev = pclCurrent;
00195             pclCurrent = static_cast<Thread*>(pclCurrent->GetNext());
00196
00197             // Read the thread's event mask/mode
00198             K_USHORT usThreadMask = pclPrev->GetEventFlagMask();
00199             EventFlagOperation_t eThreadMode = pclPrev->GetEventFlagMode();
00200
00201             // For the "any" mode - unblock the blocked threads if one or more bits
00202             // in the thread's bitmask match the object's bitmask
00203             if ((EVENT_FLAG_ANY == eThreadMode) || (EVENT_FLAG_ANY_CLEAR == eThreadMode))
00204             {
00205                 if (usThreadMask & m_usSetMask)
00206                 {
00207                     pclPrev->SetEventFlagMode(EVENT_FLAG_PENDING_UNBLOCK);
00208                     pclPrev->SetEventFlagMask(m_usSetMask & usThreadMask);
00209                     bReschedule = true;
00210
00211                     // If the "clear" variant is set, then clear the bits in the mask
00212                     // that caused the thread to unblock.
00213                     if (EVENT_FLAG_ANY_CLEAR == eThreadMode)
00214                     {
00215                         usNewMask &= ~ (usThreadMask & usMask_);
00216                     }
00217                 }
00218             }
00219             // For the "all" mode, every set bit in the thread's requested bitmask must
00220             // match the object's flag mask.
00221             else if ((EVENT_FLAG_ALL == eThreadMode) || (EVENT_FLAG_ALL_CLEAR == eThreadMode))

```

```

00222         {
00223             if ((usThreadMask & m_usSetMask) == usThreadMask)
00224             {
00225                 pclPrev->SetEventFlagMode(EVENT_FLAG_PENDING_UNBLOCK);
00226                 pclPrev->SetEventFlagMask(usThreadMask);
00227                 bReschedule = true;
00228
00229                 // If the "clear" variant is set, then clear the bits in the mask
00230                 // that caused the thread to unblock.
00231                 if (EVENT_FLAG_ALL_CLEAR == eThreadMode)
00232                 {
00233                     usNewMask &= ~ (usThreadMask & usMask_);
00234                 }
00235             }
00236         }
00237     }
00238     // To keep looping, ensure that there's something in the list, and
00239     // that the next item isn't the head of the list.
00240     while (pclPrev != m_clBlockList.GetTail());
00241
00242     // Second loop - go through and unblock all of the threads that
00243     // were tagged for unblocking.
00244     pclCurrent = static_cast<Thread*>(m_clBlockList.
GetHead());
00245     bool bIsTail = false;
00246     do
00247     {
00248         pclPrev = pclCurrent;
00249         pclCurrent = static_cast<Thread*>(pclCurrent->GetNext());
00250
00251         // Check to see if this is the condition to terminate the loop
00252         if (pclPrev == m_clBlockList.GetTail())
00253         {
00254             bIsTail = true;
00255         }
00256
00257         // If the first pass indicated that this thread should be
00258         // unblocked, then unblock the thread
00259         if (pclPrev->GetEventFlagMode() == EVENT_FLAG_PENDING_UNBLOCK)
00260         {
00261             Unblock(pclPrev);
00262         }
00263     }
00264     while (!bIsTail);
00265 }
00266
00267 // If we awoke any threads, re-run the scheduler
00268 if (bReschedule)
00269 {
00270     Thread::Yield();
00271 }
00272
00273 // Update the bitmask based on any "clear" operations performed along
00274 // the way
00275 m_usSetMask = usNewMask;
00276
00277 // Restore interrupts - will potentially cause a context switch if a
00278 // thread is unblocked.
00279 CS_EXIT();
00280 }
00281
00282 //-----
00283 void EventFlag::Clear(K_USHORT usMask_)
00284 {
00285     // Just clear the bitfields in the local object.
00286     CS_ENTER();
00287     m_usSetMask &= ~usMask_;
00288     CS_EXIT();
00289 }
00290
00291 //-----
00292 K_USHORT EventFlag::GetMask()
00293 {
00294     // Return the presently held event flag values in this object. Ensure
00295     // we get this within a critical section to guarantee atomicity.
00296     K_USHORT usReturn;
00297     CS_ENTER();
00298     usReturn = m_usSetMask;
00299     CS_EXIT();
00300     return usReturn;
00301 }
00302
00303 #endif // KERNEL_USE_EVENTFLAG

```

15.59 /home/moslevin/Project/R1/stage/src/eventflag.h File Reference

Event Flag Blocking Object/IPC-Object definition.

```
#include "mark3cfg.h"
#include "kernel.h"
#include "kerneltypes.h"
#include "blocking.h"
#include "thread.h"
```

Classes

- class [EventFlag](#)

The [EventFlag](#) class is a blocking object, similar to a semaphore or mutex, commonly used for synchronizing thread execution based on events occurring within the system.

15.59.1 Detailed Description

Event Flag Blocking Object/IPC-Object definition.

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

15.60 eventflag.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __EVENTFLAG_H__
00020 #define __EVENTFLAG_H__
00021
00022 #include "mark3cfg.h"
00023 #include "kernel.h"
00024 #include "kerneltypes.h"
00025 #include "blocking.h"
00026 #include "thread.h"
00027
00028 #if KERNEL_USE_EVENTFLAG
00029
00030 //-----
00046 class EventFlag : public BlockingObject
00047 {
00048 public:
00052 void Init() { m_usSetMask = 0; m_clBlockList.Init(); }
00053
00061 K_USHORT Wait(K_USHORT usMask_, EventFlagOperation_t eMode_);
00062
00063 #if KERNEL_USE_TIMEOUTS
00064
00072 K_USHORT Wait(K_USHORT usMask_, EventFlagOperation_t eMode_, K_ULONG ulTimeMS_);
00073
00074 void WakeMe(Thread *pClOwner_);
00075
00076 #endif
00077
00083 void Set(K_USHORT usMask_);
00084
00089 void Clear(K_USHORT usMask_);
00090
00095 K_USHORT GetMask();
00096
```



```

00048 void *BlockHeap::Create( void *pvHeap_, K_USHORT usSize_, K_USHORT usBlockSize_ )
00049 {
00050     K_USHORT usNodeCount = usSize_ /
00051         (usBlockSize_ + sizeof(LinkListNode) + sizeof(void*));
00052     K_ADDR adNode = (K_ADDR)pvHeap_;
00053     K_ADDR adMaxNode = (K_ADDR)((K_ADDR)pvHeap_ + (K_ADDR)usSize_);
00054     m_clList.Init();
00055
00056     // Create a heap (linked-list nodes + byte pool) in the middle of
00057     // the data blob
00058     for (K_USHORT i = 0; i < usNodeCount; i++)
00059     {
00060         // Create a pointer back to the source list.
00061         BlockHeap **pclTemp = (BlockHeap**) (adNode + sizeof(
LinkListNode));
00062         *pclTemp = (BlockHeap*)(this);
00063
00064         // Add the node to the block list
00065         m_clList.Add( (LinkListNode*)adNode );
00066
00067         // Move the pointer in the pool to point to the next block to allocate
00068         adNode += (usBlockSize_ + sizeof(LinkListNode) + sizeof(
BlockHeap*));
00069
00070         // Bail if we would be going past the end of the allocated space...
00071         if ((K_ULONG)adNode >= (K_ULONG)adMaxNode)
00072         {
00073             break;
00074         }
00075     }
00076     m_usBlocksFree = usNodeCount;
00077
00078     // Return pointer to end of heap (used for heap-chaining)
00079     return (void*)adNode;
00080 }
00081
00082 //-----
00083 void *BlockHeap::Alloc()
00084 {
00085     LinkListNode *pclNode = m_clList.GetHead();
00086
00087     // Return the first node from the head of the list
00088     if (pclNode)
00089     {
00090         m_clList.Remove( pclNode );
00091         m_usBlocksFree--;
00092
00093         // Account for block-management metadata
00094         return (void*)((K_ADDR)pclNode + sizeof(LinkListNode) + sizeof(void*));
00095     }
00096
00097     // Or null, if the heap is empty.
00098     return 0;
00099 }
00100
00101 //-----
00102 void BlockHeap::Free( void* pvData_ )
00103 {
00104     // Compute the address of the original object (class metadata included)
00105     LinkListNode *pclNode = (LinkListNode*)((K_ADDR)pvData_ - sizeof(
LinkListNode) - sizeof(void*));
00106
00107     // Add the object back to the block data pool
00108     m_clList.Add(pclNode);
00109     m_usBlocksFree++;
00110 }
00111
00112 //-----
00113 void FixedHeap::Create( void *pvHeap_, HeapConfig *pclHeapConfig_ )
00114 {
00115     K_USHORT i = 0;
00116     void *pvTemp = pvHeap_;
00117     while( pclHeapConfig_[i].m_usBlockSize != 0)
00118     {
00119         pvTemp = pclHeapConfig_[i].m_clHeap.Create
00120             (pvTemp,
00121              (pclHeapConfig_[i].m_usBlockSize + sizeof(LinkListNode) + sizeof(void*)) *
00122              pclHeapConfig_[i].m_usBlockCount,
00123              pclHeapConfig_[i].m_usBlockSize );
00124         i++;
00125     }
00126     m_paclHeaps = pclHeapConfig_;
00127 }
00128
00129 //-----
00130 void *FixedHeap::Alloc( K_USHORT usSize_ )
00131 {

```

```

00132     void *pvRet = 0;
00133     K_USHORT i = 0;
00134
00135     // Go through all heaps, trying to find the smallest one that
00136     // has a free item to satisfy the allocation
00137     while (m_paclHeaps[i].m_usBlockSize != 0)
00138     {
00139         CS_ENTER();
00140         if ((m_paclHeaps[i].m_usBlockSize >= usSize_) && m_paclHeaps[i].m_clHeap.
IsFree() )
00141         {
00142             // Found a match
00143             pvRet = m_paclHeaps[i].m_clHeap.Alloc();
00144         }
00145         CS_EXIT();
00146
00147         // Return an object if found
00148         if (pvRet)
00149         {
00150             return pvRet;
00151         }
00152         i++;
00153     }
00154
00155     // Or null on no-match
00156     return pvRet;
00157 }
00158
00159 //-----
00160 void FixedHeap::Free( void *pvNode_ )
00161 {
00162     // Compute the pointer to the block-heap this block belongs to, and
00163     // return it.
00164     CS_ENTER();
00165     BlockHeap **pclHeap = (BlockHeap**) ((K_ADDR)pvNode_ - sizeof(
BlockHeap*));
00166     (*pclHeap)->Free (pvNode_);
00167     CS_EXIT();
00168 }
00169
00170

```

15.63 /home/moslevin/Project/R1/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.

15.63.1 Detailed Description

Fixed-block-size heaps.

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

15.64 fixed_heap.h

```

00001 /*=====

```

```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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

```

15.65 /home/moslevin/Project/R1/stage/src/font.h File Reference

Font structure definitions.

```

#include "kerneltypes.h"
#include "fontport.h"

```



```
#include "kerneltypes.h"
#include "graphics.h"
#include "draw.h"
#include "driver.h"
#include "colorspace.h"
#include "font.h"
#include <stdio.h>
```

15.67.1 Detailed Description

Generic graphics driver implementation.

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

15.68 graphics.cpp

```

00001 /*-----
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*
00014 #include "kerneltypes.h"
00015 #include "graphics.h"
00016 #include "draw.h"
00017 #include "driver.h"
00018 #include "colorspace.h"
00019 #include "font.h"
00020
00021 #include <stdio.h>
00022
00023 //-----
00024 void GraphicsDriver::ClearScreen()
00025 {
00026     DrawPoint_t stPoint;
00027     stPoint.uColor = COLOR_BLACK;
00028
00029     for (stPoint.usX = 0; stPoint.usX < m_usResX; stPoint.usX++)
00030     {
00031         for (stPoint.usY = 0; stPoint.usY < m_usResY; stPoint.usY++)
00032         {
00033             // Pixel Write
00034             DrawPixel(&stPoint);
00035         }
00036     }
00037 //-----
00038 void GraphicsDriver::Point(DrawPoint_t *pstPoint_)
00039 {
00040     DrawPixel(pstPoint_);
00041 }
00042 //-----
00043 void GraphicsDriver::Line(DrawLine_t *pstLine_)
00044 {
00045     // Bresenham Line drawing algorithm, adapted from:
00046     // www.cs.unc.edu/~mcmillan/compl36/Lecture6/Lines.html
00047
00048     DrawPoint_t stPoint;
00049     K_SHORT sX1 = (K_SHORT)pstLine_>usX1;
00050     K_SHORT sX2 = (K_SHORT)pstLine_>usX2;
00051     K_SHORT sY1 = (K_SHORT)pstLine_>usY1;
00052     K_SHORT sY2 = (K_SHORT)pstLine_>usY2;
00053     K_SHORT sDeltaY = sY2 - sY1;
00054     K_SHORT sDeltaX = sX2 - sX1;
00055     K_CHAR cStepx, cStepy;
00056     stPoint.uColor = pstLine_>uColor;
00057

```

```

00065     if (sDeltaY < 0)
00066     {
00067         sDeltaY = -sDeltaY;
00068         cStepy = -1;
00069     }
00070     else
00071     {
00072         cStepy = 1;
00073     }
00074
00075     if (sDeltaX < 0)
00076     {
00077         sDeltaX = -sDeltaX;
00078         cStepx = -1;
00079     }
00080     else
00081     {
00082         cStepx = 1;
00083     }
00084
00085     // Scale by a factor of 2 in each direction
00086     sDeltaY <<= 1;
00087     sDeltaX <<= 1;
00088
00089     stPoint.usX = sX1;
00090     stPoint.usY = sY1;
00091     DrawPixel(&stPoint);
00092
00093     if (sDeltaX > sDeltaY)
00094     {
00095         K_SHORT sFraction = sDeltaY - (sDeltaX >> 1);
00096
00097         while (sX1 != sX2)
00098         {
00099             if (sFraction >= 0)
00100             {
00101                 sY1 += cStepy;
00102                 sFraction -= sDeltaX;
00103             }
00104             sX1 += cStepx;
00105             sFraction += sDeltaY;
00106
00107             stPoint.usX = sX1;
00108             stPoint.usY = sY1;
00109             DrawPixel(&stPoint);
00110         }
00111     }
00112     else
00113     {
00114         K_SHORT sFraction = sDeltaX - (sDeltaY >> 1);
00115         while (sY1 != sY2)
00116         {
00117             if (sFraction >= 0)
00118             {
00119                 sX1 += cStepx;
00120                 sFraction -= sDeltaY;
00121             }
00122             sY1 += cStepy;
00123             sFraction += sDeltaX;
00124
00125             stPoint.usX = sX1;
00126             stPoint.usY = sY1;
00127             DrawPixel(&stPoint);
00128         }
00129     }
00130 }
00131
00132 //-----
00133 void GraphicsDriver::Rectangle(DrawRectangle_t *pstRectangle_)
00134 {
00135     DrawPoint_t stPoint;
00136
00137     // if drawing a background fill color (optional)
00138     if (pstRectangle_>bFill == true)
00139     {
00140         stPoint.uColor = pstRectangle_>uFillColor;
00141         for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00142         {
00143             for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00144             {
00145                 DrawPixel(&stPoint);
00146             }
00147         }
00148     }
00149 }

```

```

00150     // Draw four orthogonal lines...
00151     stPoint.uColor = pstRectangle_>uLineColor;
00152     stPoint.usY = pstRectangle_>usTop;
00153     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00154     {
00155         DrawPixel(&stPoint);
00156     }
00157
00158     stPoint.usY = pstRectangle_>usBottom;
00159     for (stPoint.usX = pstRectangle_>usLeft; stPoint.usX <= pstRectangle_>
usRight; stPoint.usX++)
00160     {
00161         DrawPixel(&stPoint);
00162     }
00163
00164     stPoint.usX = pstRectangle_>usLeft;
00165     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00166     {
00167         DrawPixel(&stPoint);
00168     }
00169
00170     stPoint.usX = pstRectangle_>usRight;
00171     for (stPoint.usY = pstRectangle_>usTop; stPoint.usY <= pstRectangle_>
usBottom; stPoint.usY++)
00172     {
00173         DrawPixel(&stPoint);
00174     }
00175 }
00176
00177 //-----
00178 void GraphicsDriver::Circle(DrawCircle_t *pstCircle_)
00179 {
00180     DrawPoint_t stPoint;
00181     K_SHORT sX;
00182     K_SHORT sY;
00183     K_ULONG ulRadSquare;
00184
00185     K_ULONG ulXSquare;
00186     K_ULONG ulYSquare;
00187
00188     // Get the radius squared value...
00189     ulRadSquare = (K_ULONG)pstCircle_>usRadius;
00190     ulRadSquare *= ulRadSquare;
00191
00192     // Look at the upper-right quarter of the circle
00193     for (sX = 0; sX <= (K_SHORT)pstCircle_>usRadius; sX++)
00194     {
00195         ulXSquare = (K_ULONG)sX;
00196         ulXSquare *= ulXSquare;
00197         for (sY = 0; sY <= (K_SHORT)pstCircle_>usRadius; sY++)
00198         {
00199             ulYSquare = (K_ULONG)sY;
00200             ulYSquare *= ulYSquare;
00201
00202             // if filled...
00203             if (pstCircle_>bFill == true)
00204             {
00205                 stPoint.uColor = pstCircle_>uFillColor;
00206                 if (ulXSquare + ulYSquare <= ulRadSquare)
00207                 {
00208                     // Draw the fill color at the appropriate locations (quadrature...)
00209                     stPoint.usX = pstCircle_>usX + sX;
00210                     stPoint.usY = pstCircle_>usY + sY;
00211                     DrawPixel(&stPoint);
00212                     stPoint.usX = pstCircle_>usX - sX;
00213                     stPoint.usY = pstCircle_>usY + sY;
00214                     DrawPixel(&stPoint);
00215                     stPoint.usX = pstCircle_>usX + sX;
00216                     stPoint.usY = pstCircle_>usY - sY;
00217                     DrawPixel(&stPoint);
00218                     stPoint.usX = pstCircle_>usX - sX;
00219                     stPoint.usY = pstCircle_>usY - sY;
00220                     DrawPixel(&stPoint);
00221                 }
00222             }
00223             // Check for edge...
00224             if (
00225                 ((ulXSquare + ulYSquare) >= (ulRadSquare-pstCircle_>usRadius)) &&
00226                 ((ulXSquare + ulYSquare) <= (ulRadSquare+pstCircle_>usRadius))
00227             )
00228             {
00229                 stPoint.uColor = pstCircle_>uLineColor;
00230
00231                 // Draw the fill color at the appropriate locations (quadrature...)
00232                 stPoint.usX = pstCircle_>usX + sX;

```

```

00233         stPoint.usY = pstCircle->usY + sY;
00234         DrawPixel(&stPoint);
00235         stPoint.usX = pstCircle->usX - sX;
00236         stPoint.usY = pstCircle->usY + sY;
00237         DrawPixel(&stPoint);
00238         stPoint.usX = pstCircle->usX + sX;
00239         stPoint.usY = pstCircle->usY - sY;
00240         DrawPixel(&stPoint);
00241         stPoint.usX = pstCircle->usX - sX;
00242         stPoint.usY = pstCircle->usY - sY;
00243         DrawPixel(&stPoint);
00244     }
00245 }
00246 }
00247 }
00248
00249 //-----
00250 void GraphicsDriver::Ellipse(DrawEllipse_t *pstEllipse_)
00251 {
00252     DrawPoint_t stPoint;
00253     K_SHORT sX;
00254     K_SHORT sY;
00255     K_ULONG ulRadius;
00256     K_ULONG ulHSquare;
00257     K_ULONG ulVSquare;
00258     K_ULONG ulXSquare;
00259     K_ULONG ulYSquare;
00260
00261     ulHSquare = (K_ULONG)pstEllipse->usWidth;
00262     ulHSquare *= ulHSquare;
00263
00264     ulVSquare = (K_ULONG)pstEllipse->usHeight;
00265     ulVSquare *= ulVSquare;
00266
00267     ulRadius = ulHSquare * ulVSquare;
00268
00269     for (sX = 0; sX <= (K_SHORT)pstEllipse->usWidth; sX++)
00270     {
00271         ulXSquare = (K_ULONG)sX;
00272         ulXSquare *= ulXSquare;
00273         ulXSquare *= ulHSquare;
00274
00275         for (sY = 0; sY <= (K_SHORT)pstEllipse->usHeight; sY++)
00276         {
00277             ulYSquare = (K_ULONG)sY;
00278             ulYSquare *= ulYSquare;
00279             ulYSquare *= ulVSquare;
00280
00281             if ((ulXSquare + ulYSquare) <= ulRadius)
00282             {
00283                 // Draw the fill color at the appropriate locations (quadrature...)
00284                 stPoint.usX = pstEllipse->usX + sX;
00285                 stPoint.usY = pstEllipse->usY + sY;
00286                 DrawPixel(&stPoint);
00287                 stPoint.usX = pstEllipse->usX - sX;
00288                 stPoint.usY = pstEllipse->usY + sY;
00289                 DrawPixel(&stPoint);
00290                 stPoint.usX = pstEllipse->usX + sX;
00291                 stPoint.usY = pstEllipse->usY - sY;
00292                 DrawPixel(&stPoint);
00293                 stPoint.usX = pstEllipse->usX - sX;
00294                 stPoint.usY = pstEllipse->usY - sY;
00295                 DrawPixel(&stPoint);
00296             }
00297         }
00298     }
00299 }
00300
00301 //-----
00302 void GraphicsDriver::Bitmap(DrawBitmap_t *pstBitmap_)
00303 {
00304     K_USHORT usRow;
00305     K_USHORT usCol;
00306
00307     K_USHORT usIndex;
00308
00309     K_UCHAR ucRed = 0;
00310     K_UCHAR ucBlue = 0;
00311     K_UCHAR ucGreen = 0;
00312
00313     DrawPoint_t stPoint;
00314
00315     usIndex = 0;
00316     for (usRow = pstBitmap->usY; usRow < (pstBitmap->usY + pstBitmap->
00317         usHeight); usRow++)
00318     {
00319         for (usCol = pstBitmap->usX; usCol < (pstBitmap->usX + pstBitmap->

```

```

        usWidth); usCol++)
    {
00319         {
00320
00321             stPoint.usX = usCol;
00322             stPoint.usY = usRow;
00323
00324             // Build the color based on the bitmap value... This algorithm
00325             // is slow, but it automatically converts any 8/16/24 bit bitmap into the
00326             // current colorspace defined...
00327             switch(pstBitmap->ucBPP)
00328             {
00329                 case 1:
00330                 {
00331                     // 3:2:3, RGB
00332                     ucRed   = ((pstBitmap->pucData[usIndex]) & 0xE0) << 1;
00333                     ucGreen = ((pstBitmap->pucData[usIndex]) & 0x18) << 3;
00334                     ucBlue  = ((pstBitmap->pucData[usIndex]) & 0x07) << 5;
00335                 }
00336                 break;
00337                 case 2:
00338                 {
00339                     K_USHORT usTemp;
00340                     usTemp = pstBitmap->pucData[usIndex];
00341                     usTemp <= 8;
00342                     usTemp |= pstBitmap->pucData[usIndex + 1];
00343
00344                     // 5:6:5, RGB
00345                     ucRed   = (K_UCHAR)((usTemp >> 11) & 0x001F) << 3;
00346                     ucGreen = (K_UCHAR)((usTemp >> 5) & 0x003F) << 2;
00347                     ucBlue  = (K_UCHAR)(usTemp & 0x001F) << 3;
00348                 }
00349                 break;
00350                 case 3:
00351                 {
00352                     K_ULONG ulTemp;
00353                     ulTemp = pstBitmap->pucData[usIndex];
00354                     ulTemp <= 8;
00355                     ulTemp |= pstBitmap->pucData[usIndex + 1];
00356                     ulTemp <= 8;
00357                     ulTemp |= pstBitmap->pucData[usIndex + 2];
00358
00359                     // 8:8:8 RGB
00360                     ucRed   = (K_UCHAR)((ulTemp & 0x00FF0000) >> 16);
00361                     ucGreen = (K_UCHAR)((ulTemp & 0x0000FF00) >> 8);
00362                     ucBlue  = (K_UCHAR)((ulTemp & 0x000000FF));
00363                 }
00364                 break;
00365                 default:
00366                     break;
00367             }
00368
00369             // Convert the R,G,B values into the correct colorspace for display
00370             #if DRAW_COLOR_2BIT
00371                 //1-bit
00372                 ucRed >>= 7;
00373                 ucGreen >>= 7;
00374                 ucBlue >>= 7;
00375             #elif DRAW_COLOR_8BIT
00376                 //3:2:3 R:G:B
00377                 ucRed >>= 5;
00378                 ucGreen >>= 6;
00379                 ucBlue >>= 5;
00380             #elif DRAW_COLOR_16BIT
00381                 //5:6:5 R:G:B
00382                 ucRed >>= 3;
00383                 ucGreen >>= 2;
00384                 ucBlue >>= 3;
00385             #elif DRAW_COLOR_24BIT
00386                 // No conversion required
00387             #endif
00388             // Build the color.
00389             stPoint.uColor = RGB_COLOR(ucRed,ucGreen,ucBlue);
00390
00391             // Draw the point.
00392             DrawPixel(&stPoint);
00393
00394             // Stamps are opaque, don't fill in the BG
00395             usIndex += m_ucBPP / 8;
00396         }
00397     }
00398 }
00399
00400 //-----
00401 void GraphicsDriver::Stamp(DrawStamp_t *pstStamp_)
00402 {
00403     K_USHORT usRow;
00404     K_USHORT usCol;

```

```

00405     K_USHORT usShift;
00406     K_USHORT usIndex;
00407     DrawPoint_t stPoint;
00408
00409     usIndex = 0;
00410     for (usRow = pstStamp_->usY; usRow < (pstStamp_->usY + pstStamp_->
usHeight); usRow++)
00411     {
00412         usShift = 0x80;
00413         for (usCol = pstStamp_->usX; usCol < (pstStamp_->usX + pstStamp_->
usWidth); usCol++)
00414         {
00415             // If the packed bit in the bitmap is a "1", draw the color.
00416             if (pstStamp_->pucData[usIndex] & usShift)
00417             {
00418                 stPoint.usX = usCol;
00419                 stPoint.usY = usRow;
00420                 stPoint.uColor = pstStamp_->uColor;
00421                 DrawPixel(&stPoint);
00422             }
00423             // Stamps are opaque, don't fill in the BG
00424
00425             // Shift to the next bit in the field
00426             usShift >>= 1;
00427
00428             // Rollover - next bit in the bitmap.
00429             // This obviously works best for stamps that are multiples of 8x8
00430             if (usShift == 0)
00431             {
00432                 usShift = 0x80;
00433                 usIndex++;
00434             }
00435         }
00436     }
00437 }
00438
00439 //-----
00440 void GraphicsDriver::Move( DrawMove_t *pstMove_ )
00441 {
00442     DrawPoint_t stPoint;
00443     K_LONG sX;
00444     K_LONG sY;
00445     K_LONG sXInc = 0;
00446     K_LONG sYInc = 0;
00447
00448     K_BOOL bLeftToRight = false;
00449     K_BOOL bTopToBottom = false;
00450
00451     if (pstMove_->usSrcX > pstMove_->usDstX)
00452     {
00453         bLeftToRight = true;
00454     }
00455     if (pstMove_->usSrcY > pstMove_->usDstY)
00456     {
00457         bTopToBottom = true;
00458     }
00459
00460     if (bLeftToRight)
00461     {
00462         sXInc++;
00463     }
00464     else
00465     {
00466         sXInc--;
00467         pstMove_->usSrcX += pstMove_->usCopyWidth - 1;
00468         pstMove_->usDstX += pstMove_->usCopyWidth - 1;
00469     }
00470
00471     if (bTopToBottom)
00472     {
00473         sYInc++;
00474     }
00475     else
00476     {
00477         sYInc--;
00478         pstMove_->usSrcY += pstMove_->usCopyHeight - 1;
00479         pstMove_->usDstY += pstMove_->usCopyHeight - 1;
00480     }
00481
00482     // Hideously inefficient memory move...
00483     for (sX = 0; sX < pstMove_->usCopyWidth; sX++)
00484     {
00485         for (sY = 0; sY < pstMove_->usCopyHeight; sY++)
00486         {
00487             // Read from source (value read into the point struct)
00488             stPoint.usY = (K_USHORT) ((K_LONG)pstMove_->usSrcY + ((K_LONG)sY * sYInc));
00489             stPoint.usX = (K_USHORT) ((K_LONG)pstMove_->usSrcX + ((K_LONG)sX * sXInc));

```

```

00490         ReadPixel(&stPoint);
00491
00492         // Copy to dest
00493         stPoint.usY = (K_USHORT)((K_LONG)pstMove_>usDstY + ((K_LONG)sY * sYInc));
00494         stPoint.usX = (K_USHORT)((K_LONG)pstMove_>usDstX + ((K_LONG)sX * sXInc));
00495         DrawPixel(&stPoint);
00496     }
00497 }
00498 }
00499
00500 //-----
00501 void GraphicsDriver::Text(DrawText_t *pstText_)
00502 {
00503     K_USHORT usX, usY;
00504     K_USHORT usStartX;
00505     K_USHORT usStartY;
00506     K_USHORT usCharOffsetX;
00507     K_USHORT usCharIndex = 0;
00508     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00509     DrawPoint_t stPoint;
00510
00511     // set the color for this element.
00512     stPoint.uColor = pstText_>uColor;
00513
00514     usCharOffsetX = 0;
00515
00516     // Draw every character in the string, one at a time
00517     while (pstText_>pcString[usCharIndex] != 0)
00518     {
00519         K_USHORT usOffset = 0;
00520
00521         K_UCHAR ucWidth;
00522         K_UCHAR ucHeight;
00523         K_UCHAR ucVOffset;
00524         K_UCHAR ucBitmask;
00525
00526         // Read the glyphs from memory until we arrive at the one we wish to print
00527         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00528         {
00529             // Glyphs are variable-sized for efficiency - to look up a particular
00530             // glyph, we must traverse all preceding glyphs in the list
00531             ucWidth = Font_ReadByte(usOffset, pucData);
00532             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00533
00534             // Adjust the offset to point to the next glyph
00535             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00536                 + (sizeof(Glyph_t) - 1);
00537         }
00538
00539         // Header information: glyph size and vertical offset
00540         ucWidth = Font_ReadByte(usOffset++, pucData);
00541         ucHeight = Font_ReadByte(usOffset++, pucData);
00542         ucVOffset = Font_ReadByte(usOffset++, pucData);
00543
00544         usStartY = pstText_>usTop + (K_USHORT)ucVOffset;
00545         usStartX = pstText_>usLeft;
00546
00547         // Draw the font from left->right, top->bottom
00548         for (usY = usStartY;
00549              usY < usStartY + (K_USHORT)ucHeight;
00550              usY++)
00551         {
00552             K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
00553             ucBitmask = 0x80;
00554
00555             for (usX = usCharOffsetX + usStartX;
00556                  usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
00557                  usX++)
00558             {
00559                 if (!ucBitmask)
00560                 {
00561                     ucBitmask = 0x80;
00562                     usOffset++;
00563                     ucTempChar = Font_ReadByte(usOffset, pucData);
00564                 }
00565
00566                 if (ucTempChar & ucBitmask)
00567                 {
00568                     // Update the location
00569                     stPoint.usX = usX;
00570                     stPoint.usY = usY;
00571
00572                     // Draw the point.
00573                     DrawPixel(&stPoint);
00574                 }
00575             }
00576             ucBitmask >>= 1;

```

```

00577         }
00578
00579         usOffset++;
00580     }
00581
00582     // Next character
00583     usCharIndex++;
00584     usCharOffsetX += (K_USHORT)ucWidth + 1;
00585 }
00586 }
00587
00588 //-----
00589 void GraphicsDriver::TextFX(DrawText_t *pstText_,
    TextFX_t *pstFX_ )
00590 {
00591     K_USHORT usX, usY;
00592     K_USHORT usPartialX = 0;
00593     K_USHORT usPartialY = 0;
00594     K_USHORT usStartX;
00595     K_USHORT usStartY;
00596     K_USHORT usCharOffsetX;
00597     K_USHORT usCharIndex = 0;
00598     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00599     DrawPoint_t stPoint;
00600
00601     // set the color for this element.
00602     stPoint.uColor = pstText_>uColor;
00603
00604     usCharOffsetX = 0;
00605
00606     // Draw every character in the string, one at a time
00607     while (pstText_>pcString[usCharIndex] != 0)
00608     {
00609         K_USHORT usOffset = 0;
00610
00611         K_UCHAR ucWidth;
00612         K_UCHAR ucHeight;
00613         K_UCHAR ucVOffset;
00614         K_UCHAR ucBitmask;
00615
00616         // Read the glyphs from memory until we arrive at the one we wish to print
00617         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00618         {
00619             // Glyphs are variable-sized for efficiency - to look up a particular
00620             // glyph, we must traverse all preceding glyphs in the list
00621             ucWidth = Font_ReadByte(usOffset, pucData);
00622             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00623
00624             // Adjust the offset to point to the next glyph
00625             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00626                 + (sizeof(Glyph_t) - 1);
00627         }
00628
00629         // Header information: glyph size and vertical offset
00630         ucWidth = Font_ReadByte(usOffset++, pucData);
00631         ucHeight = Font_ReadByte(usOffset++, pucData);
00632         ucVOffset = Font_ReadByte(usOffset++, pucData);
00633
00634         usStartY = pstText_>usTop + (K_USHORT)ucVOffset;
00635         usStartX = pstText_>usLeft;
00636
00637         // Draw the font from left->right, top->bottom
00638         for ( usY = usStartY;
00639             usY < usStartY + (K_USHORT)ucHeight;
00640             usY++ )
00641         {
00642             K_UCHAR ucTempChar = Font_ReadByte(usOffset, pucData);
00643             ucBitmask = 0x80;
00644             usPartialY = 0;
00645             usPartialX = 0;
00646
00647             K_USHORT usTempPartialX = 0;
00648
00649             for ( usX = usCharOffsetX + usStartX;
00650                 usX < usCharOffsetX + usStartX + (K_USHORT)ucWidth;
00651                 usX++ )
00652             {
00653                 K_USHORT usTempPartialY = 0;
00654                 usPartialY = 0;
00655                 if (!ucBitmask)
00656                 {
00657                     ucBitmask = 0x80;
00658                     usOffset++;
00659                     ucTempChar = Font_ReadByte(usOffset, pucData);
00660                 }
00661
00662                 if ((ucTempChar & ucBitmask) || (pstFX_>ucFlags &

```



```

TEXTFX_FLAG_OPAQUE_BG))
00663     {
00664         // usX and usY represent the untransformed data...
00665         // we need usStartX, usStartY, usDeltaX, usDeltaY to proceed.
00666         K_USHORT usDeltaX = (usX - pstText_>usLeft);
00667         K_USHORT usDeltaY = (usY - pstText_>usTop);
00668
00669         // Compute "unadjusted" pixels for normal or scaled
00670         K_USHORT usRawX, usRawY;
00671
00672         if (pstFX_>ucFlags & TEXTFX_FLAG_SCALE_X)
00673         {
00674             usRawX = usStartX + (((usDeltaX * pstFX_>usScaleX100))/100);
00675             usTempPartialX = pstFX_>usScaleX100;
00676         }
00677         else
00678         {
00679             usRawX = usX;
00680             usTempPartialX = 100;
00681         }
00682         usTempPartialX += usPartialX;
00683
00684         if (pstFX_>ucFlags & TEXTFX_FLAG_SCALE_Y)
00685         {
00686             usRawY = usStartY + (((usDeltaY * pstFX_>usScaleY100))/100);
00687             usTempPartialY = pstFX_>usScaleY100;
00688         }
00689         else
00690         {
00691             usRawY = usY;
00692             usTempPartialY = 100;
00693         }
00694         usTempPartialY += usPartialY;
00695
00696         K_USHORT usBLAH = usTempPartialX;
00697
00698         if (!(ucTempChar & ucBitmask))
00699         {
00700             stPoint.uColor = pstFX_>uBGColor;
00701         }
00702         else
00703         {
00704             stPoint.uColor = pstText_>uColor;
00705         }
00706
00707         stPoint.usX = usRawX;
00708         while (usTempPartialX >= 50)
00709         {
00710             stPoint.usY = usRawY;
00711             usBLAH = usTempPartialY;
00712             while (usBLAH >= 50)
00713             {
00714                 DrawPixel(&stPoint);
00715                 stPoint.usY++;
00716                 if (usBLAH >= 100)
00717                 {
00718                     usBLAH -= 100;
00719                 }
00720                 else
00721                 {
00722                     usBLAH = 0;
00723                 }
00724             }
00725             stPoint.usX++;
00726             if (usTempPartialX >= 100)
00727             {
00728                 usTempPartialX -= 100;
00729             }
00730             else
00731             {
00732                 usTempPartialX = 0;
00733             }
00734         }
00735         usPartialX = (usTempPartialX % 100);
00736         usPartialY = (usTempPartialY % 100);
00737     }
00738
00739     ucBitmask >>= 1;
00740 }
00741
00742     usOffset++;
00743 }
00744
00745 // Next character
00746 usCharIndex++;
00747

```

```

00750         usCharOffsetX += (K_USHORT)ucWidth + 1;
00751     }
00752 }
00753
00754 //-----
00755 K_USHORT GraphicsDriver::TextWidth(DrawText_t *pstText_)
00756 {
00757     K_USHORT usCharOffsetX;
00758     K_USHORT usCharIndex = 0;
00759     K_USHORT usX;
00760     K_UCHAR *pucData = (K_UCHAR*)pstText_>pstFont->pucFontData;
00761
00762     usCharOffsetX = 0;
00763
00764     // Draw every character in the string, one at a time
00765     while (pstText_>pcString[usCharIndex] != 0)
00766     {
00767         K_USHORT usOffset = 0;
00768
00769         K_UCHAR ucWidth;
00770         K_UCHAR ucHeight;
00771
00772         // Read the glyphs from memory until we arrive at the one we wish to print
00773         for (usX = 0; usX < pstText_>pcString[usCharIndex]; usX++)
00774         {
00775             // Glyphs are variable-sized for efficiency - to look up a particular
00776             // glyph, we must traverse all preceding glyphs in the list
00777             ucWidth = Font_ReadByte(usOffset, pucData);
00778             ucHeight = Font_ReadByte(usOffset + 1, pucData);
00779
00780             // Adjust the offset to point to the next glyph
00781             usOffset += (((K_USHORT)ucWidth + 7) >> 3) * (K_USHORT)ucHeight)
00782                 + (sizeof(Glyph_t) - 1);
00783         }
00784
00785         // Header information: glyph size and vertical offset
00786         ucWidth = Font_ReadByte(usOffset, pucData);
00787         usOffset += (sizeof(Glyph_t) - 1);
00788
00789         // Next character
00790         usCharIndex++;
00791         usCharOffsetX += (K_USHORT)ucWidth + 1;
00792     }
00793
00794     return usCharOffsetX;
00795 }
00796
00797 //-----
00798 void GraphicsDriver::TriangleWire(DrawPoly_t *pstPoly_)
00799 {
00800     DrawLine_t stLine;
00801
00802     stLine.uColor = pstPoly_>uColor;
00803
00804     stLine.usX1 = pstPoly_>pstVector[0].usX;
00805     stLine.usY1 = pstPoly_>pstVector[0].usY;
00806     stLine.usX2 = pstPoly_>pstVector[1].usX;
00807     stLine.usY2 = pstPoly_>pstVector[1].usY;
00808     Line(&stLine);
00809
00810     stLine.usX1 = pstPoly_>pstVector[1].usX;
00811     stLine.usY1 = pstPoly_>pstVector[1].usY;
00812     stLine.usX2 = pstPoly_>pstVector[2].usX;
00813     stLine.usY2 = pstPoly_>pstVector[2].usY;
00814     Line(&stLine);
00815
00816     stLine.usX1 = pstPoly_>pstVector[2].usX;
00817     stLine.usY1 = pstPoly_>pstVector[2].usY;
00818     stLine.usX2 = pstPoly_>pstVector[0].usX;
00819     stLine.usY2 = pstPoly_>pstVector[0].usY;
00820     Line(&stLine);
00821 }
00822 //-----
00823 void GraphicsDriver::TriangleFill(DrawPoly_t *pstPoly_)
00824 {
00825     // Drawing a raster-filled triangle:
00826     K_UCHAR ucMaxEdge = 0;
00827     K_UCHAR ucMinEdge1 = 0, ucMinEdge2 = 0;
00828     K_SHORT sMax = 0;
00829     K_SHORT sTemp;
00830
00831     K_SHORT sDeltaX1, sDeltaX2;
00832     K_SHORT sDeltaY1, sDeltaY2;
00833     K_CHAR cStepX1, cStepX2;
00834     K_CHAR cStepY;
00835     K_SHORT sX1, sX2, sX3, sY1, sY2, sY3;
00836     K_SHORT sTempX1, sTempY1, sTempX2, sTempY2;

```

```

00837     K_SHORT sFraction1;
00838     K_SHORT sFraction2;
00839     K_SHORT i;
00840     DrawPoint_t stPoint;
00841
00842     // Figure out which line segment is the longest
00843     sTemp = (K_SHORT)pstPoly->pstVector[0].usY - (K_SHORT)pstPoly->
pstVector[1].usY;
00844     if( sTemp < 0 )      { sTemp = -sTemp; }
00845     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 0; ucMinEdge1 = 1; ucMinEdge2 = 2; }
00846
00847     sTemp = (K_SHORT)pstPoly->pstVector[1].usY - (K_SHORT)pstPoly->
pstVector[2].usY;
00848     if( sTemp < 0 )      { sTemp = -sTemp; }
00849     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 1; ucMinEdge1 = 2; ucMinEdge2 = 0; }
00850
00851     sTemp = (K_SHORT)pstPoly->pstVector[2].usY - (K_SHORT)pstPoly->
pstVector[0].usY;
00852     if( sTemp < 0 )      { sTemp = -sTemp; }
00853     if( sTemp > sMax ) { sMax = sTemp; ucMaxEdge = 2; ucMinEdge1 = 0; ucMinEdge2 = 1; }
00854
00855     // Label the vectors and copy into temporary signed buffers
00856     sX1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usX;
00857     sX2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usX;
00858     sX3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usX;
00859
00860     sY1 = (K_SHORT)pstPoly->pstVector[ucMaxEdge].usY;
00861     sY2 = (K_SHORT)pstPoly->pstVector[ucMinEdge1].usY;
00862     sY3 = (K_SHORT)pstPoly->pstVector[ucMinEdge2].usY;
00863
00864     // Figure out whether or not we're drawing up-down or down-up
00865     sDeltaY1 = sY1 - sY2;
00866     if (sDeltaY1 < 0) { cStepY = -1; sDeltaY1 = -sDeltaY1; } else { cStepY = 1; }
00867
00868     sDeltaX1 = sX1 - sX2;
00869     if (sDeltaX1 < 0) { cStepX1 = -1; sDeltaX1 = -sDeltaX1; } else { cStepX1 = 1; }
00870
00871     sDeltaY2 = sY1 - sY3;
00872     if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1; }
00873
00874     sDeltaX2 = sX1 - sX3;
00875     if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 = 1; }
00876
00877     sDeltaX1 <=<=1;
00878     sDeltaX2 <=<=1;
00879     sDeltaY1 <=<=1;
00880     sDeltaY2 <=<=1;
00881
00882     sFraction1 = sDeltaX1;// - (sDeltaY1 >> 1);
00883     sFraction2 = sDeltaX2;// - (sDeltaY2 >> 1);
00884
00885     sTempY1 = sY1;
00886     sTempY2 = sY1;
00887     sTempX1 = sX1;
00888     sTempX2 = sX1;
00889
00890     stPoint.uColor = pstPoly->uColor;
00891
00892     if( sDeltaY2 != 0 )
00893     {
00894         while (sTempY2 != sY3)
00895         {
00896             stPoint.usY = sTempY2;
00897             if( sTempX1 < sTempX2 ) {
00898                 for( i = sTempX1; i <= sTempX2; i++) {
00899                     stPoint.usX = i;
00900                     Point(&stPoint);
00901                 }
00902             } else {
00903                 for( i = sTempX2; i <= sTempX1; i++ ) {
00904                     stPoint.usX = i;
00905                     Point(&stPoint);
00906                 }
00907             }
00908
00909             while (sFraction2 >= sDeltaY2)
00910             {
00911                 sTempX2 -= cStepX2;
00912                 sFraction2 -= sDeltaY2;
00913             }
00914             sTempY2 -= cStepY;
00915             sFraction2 += sDeltaX2;
00916
00917             while (sFraction1 >= sDeltaY1)
00918             {
00919                 sTempX1 -= cStepX1;
00920                 sFraction1 -= sDeltaY1;

```

```

00921         }
00922         sTempY1 -= cStepY;
00923         sFraction1 += sDeltaX1;
00924     }
00925 }
00926
00927 sDeltaY2 = sY3 - sY2;
00928 sDeltaX2 = sX3 - sX2;
00929
00930 if (sDeltaX2 < 0) { cStepX2 = -1; sDeltaX2 = -sDeltaX2; } else { cStepX2 = 1; }
00931 if (sDeltaY2 < 0) { cStepY = -1; sDeltaY2 = -sDeltaY2; } else { cStepY = 1; }
00932
00933 sDeltaX2 <= 1;
00934 sDeltaY2 <= 1;
00935
00936 sFraction2 = sDeltaX2; // - (sDeltaY2 >> 1);
00937
00938 sTempY2 = sY3;
00939 sTempX2 = sX3;
00940
00941 if( sDeltaY2 != 0)
00942 {
00943     while (sTempY2 != sY2)
00944     {
00945         stPoint.usY = sTempY2;
00946         if( sTempX1 < sTempX2 ) {
00947             for( i = sTempX1; i <= sTempX2; i++) {
00948                 stPoint.usX = i;
00949                 Point(&stPoint);
00950             }
00951         } else {
00952             for( i = sTempX2; i <= sTempX1; i++ ) {
00953                 stPoint.usX = i;
00954                 Point(&stPoint);
00955             }
00956         }
00957         while (sFraction2 >= sDeltaY2)
00958         {
00959             sTempX2 -= cStepX2;
00960             sFraction2 -= sDeltaY2;
00961         }
00962         sTempY2 -= cStepY;
00963         sFraction2 += sDeltaX2;
00964         while (sFraction1 >= sDeltaY1)
00965         {
00966             sTempX1 -= cStepX1;
00967             sFraction1 -= sDeltaY1;
00968         }
00969         sTempY1 -= cStepY;
00970         sFraction1 += sDeltaX1;
00971     }
00972 }
00973 }
00974 }
00975 }
00976
00977 //-----
00978 void GraphicsDriver::Polygon(DrawPoly_t *pstPoly_)
00979 {
00980     K_USHORT i,j,k;
00981     K_BOOL bState = false;
00982
00983     DrawPoly_t stTempPoly;
00984     DrawVector_t astTempVec[3];
00985
00986     if (pstPoly_>usNumPoints < 3)
00987     {
00988         return;
00989     }
00990
00991     stTempPoly.uColor = pstPoly_>uColor;
00992     stTempPoly.bFill = pstPoly_>bFill;
00993     stTempPoly.pstVector = astTempVec;
00994     stTempPoly.usNumPoints = 3;
00995
00996     astTempVec[0].usX = pstPoly_>pstVector[0].usX;
00997     astTempVec[1].usX = pstPoly_>pstVector[1].usX;
00998     astTempVec[0].usY = pstPoly_>pstVector[0].usY;
00999     astTempVec[1].usY = pstPoly_>pstVector[1].usY;
01000
01001     j = 2;
01002     astTempVec[2].usX = pstPoly_>pstVector[pstPoly_>usNumPoints - 1].usX;
01003     astTempVec[2].usY = pstPoly_>pstVector[pstPoly_>usNumPoints - 1].usY;
01004
01005     k = pstPoly_>usNumPoints - 2;
01006
01007     if( pstPoly_>bFill )

```

```

01008     {
01009         TriangleFill(&stTempPoly);
01010     }
01011     else
01012     {
01013         TriangleWire(&stTempPoly);
01014     }
01015
01016     // Filled polygon/wireframe polygon using triangle decomp.
01017     for(i = 0; i < pstPoly->usNumPoints - 3; i++)
01018     {
01019         astTempVec[0].usX = astTempVec[1].usX;
01020         astTempVec[1].usX = astTempVec[2].usX;
01021         astTempVec[0].usY = astTempVec[1].usY;
01022         astTempVec[1].usY = astTempVec[2].usY;
01023
01024         if( !bState )
01025         {
01026             bState = true;
01027             astTempVec[2].usX = pstPoly->pstVector[j].usX;
01028             astTempVec[2].usY = pstPoly->pstVector[j].usY;
01029             j++;
01030         }
01031         else
01032         {
01033             bState = false;
01034             astTempVec[2].usX = pstPoly->pstVector[k].usX;
01035             astTempVec[2].usY = pstPoly->pstVector[k].usY;
01036             k--;
01037         }
01038         if( pstPoly->bFill )
01039         {
01040             TriangleFill(&stTempPoly);
01041         }
01042         else
01043         {
01044             TriangleWire(&stTempPoly);
01045         }
01046     }
01047 }
01048
01049 //-----
01050 void GraphicsDriver::SetWindow(DrawWindow_t *pstWindow_)
01051 {
01052     if ((pstWindow_->usLeft <= pstWindow_->usRight) &&
01053         (pstWindow_->usRight < m_usResX) &&
01054         (pstWindow_->usLeft < m_usResX))
01055     {
01056         m_usLeft = pstWindow_->usLeft;
01057         m_usRight = pstWindow_->usRight;
01058     }
01059
01060     if ((pstWindow_->usTop <= pstWindow_->usBottom) &&
01061         (pstWindow_->usTop < m_usTop) &&
01062         (pstWindow_->usBottom < m_usBottom))
01063     {
01064         m_usTop = pstWindow_->usTop;
01065         m_usBottom = pstWindow_->usBottom;
01066     }
01067 }
01068
01069 //-----
01070 void GraphicsDriver::ClearWindow()
01071 {
01072     {
01073         m_usLeft = 0;
01074         m_usTop = 0;
01075         m_usRight = m_usResX - 1;
01076         m_usBottom = m_usResY - 1;
01077     }

```

15.69 /home/moslevin/Project/R1/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.

15.69.1 Detailed Description

Graphics driver class declaration.

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

15.70 graphics.h

```

00001  /*=====
00002
00003  |-----|-----|-----|-----|-----|-----|-----|-----|
00004  | \    / | \    / | \    / | \    / | \    / | \    / | \    / |
00005  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00006  |   \/   |   \/   |   \/   |   \/   |   \/   |   \/   |   \/   |
00007  |_____|   |_____|   |_____|   |_____|   |_____|   |_____|   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 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
00161     void TextFX(DrawText_t *pstText_, TextFX_t *pstFX_);
00170
00171     virtual K_USHORT TextWidth(DrawText_t *pstText_);
00178
00184     void SetWindow( DrawWindow_t *pstWindow_ );
00185
00191     void ClearWindow();
00192 protected:
00193
00194     K_USHORT m_usResX;
00195     K_USHORT m_usResY;
00196
00197     K_USHORT m_usLeft;
00198     K_USHORT m_usTop;
00199     K_USHORT m_usRight;
00200     K_USHORT m_usBottom;
00201
00202     K_UCHAR m_ucBPP;
00203 };
00204
00205 #endif
00206

```

15.71 /home/moslevin/Project/R1/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"
```

15.71.1 Detailed Description

Graphical User Interface classes and data structure definitions.

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

15.72 gui.cpp

```

00001 /*-----
00002
00003 |-----|-----|-----|-----|-----|-----|
00004 | | | | | | | | | | | | | | | | | | | | | | | |
00005 | | | | | | | | | | | | | | | | | | | | | | | |
00006 | | | | | | | | | | | | | | | | | | | | | | | |
00007 |-----|-----|-----|-----|-----|-----|
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00014 #include "message.h"
00015 #include "kerneltypes.h"
00016 #include "gui.h"
00017 #include "system_heap.h"
00018 #include "fixed_heap.h"
00019 #include "memutil.h"
00020
00021 //-----
00022 void GuiWindow::AddControl( GuiControl *pclControl_,
00023                             GuiControl *pclParent_ )
00024 {
00025     GUI_DEBUG_PRINT("GuiWindow::AddControl\n");
00026
00027     m_clControlList.Add(static_cast<LinkListNode*>(pclControl_));

```

```

00032     m_pclInFocus = pclControl_;
00033     m_ucControlCount++;
00034
00035     pclControl_->SetParentWindow(this);
00036     pclControl_->SetParentControl(pclParent_);
00037 }
00038
00039 //-----
00040 void GuiWindow::RemoveControl( GuiControl *pclControl_ )
00041 {
00042     GUI_DEBUG_PRINT("GuiWindow::RemoveControl\n");
00043     if (pclControl_->GetPrev())
00044     {
00045         m_pclInFocus = static_cast<GuiControl*>(pclControl_->
00046         GetPrev());
00047     }
00048     else if (pclControl_->GetNext())
00049     {
00050         m_pclInFocus = static_cast<GuiControl*>(pclControl_->
00051         GetNext());
00052     }
00053     {
00054         m_pclInFocus = NULL;
00055     }
00056     m_clControlList.Remove(static_cast<LinkListNode*>(pclControl_));
00057     m_ucControlCount--;
00058 }
00059
00060 //-----
00061 K_UCHAR GuiWindow::GetMaxZOrder()
00062 {
00063     GUI_DEBUG_PRINT("GuiWindow::GetMaxZOrder\n");
00064
00065     LinkListNode *pclTempNode;
00066     K_UCHAR ucZ = 0;
00067     K_UCHAR ucTempZ;
00068
00069     pclTempNode = m_clControlList.GetHead();
00070
00071     while (pclTempNode)
00072     {
00073         ucTempZ = (static_cast<GuiControl*>(pclTempNode))->GetZOrder();
00074         if (ucTempZ > ucZ)
00075         {
00076             ucZ = ucTempZ;
00077         }
00078         pclTempNode = pclTempNode->GetNext();
00079     }
00080
00081     return ucZ;
00082 }
00083
00084 //-----
00085 void GuiWindow::Redraw( K_BOOL bRedrawAll_ )
00086 {
00087     GUI_DEBUG_PRINT("GuiWindow::Redraw\n");
00088
00089     K_UCHAR ucControlsLeft = m_ucControlCount;
00090     K_UCHAR ucCurrentZ = 0;
00091     K_UCHAR ucMaxZ;
00092
00093     ucMaxZ = GetMaxZOrder();
00094
00095     // While there are still controls left to process (and we're less than
00096     // the maximum Z-order, just a sanity check.), redraw each object that
00097     // has its stale flag set, or all controls if the bRedrawAll_ parameter
00098     // is true.
00099     while (ucControlsLeft && (ucCurrentZ <= ucMaxZ))
00100     {
00101         LinkListNode *pclTempNode;
00102
00103         pclTempNode = m_clControlList.GetHead();
00104         while (pclTempNode)
00105         {
00106             GuiControl* pclTempControl = static_cast<GuiControl*>(pclTempNode);
00107             if (pclTempControl->GetZOrder() == ucCurrentZ)
00108             {
00109                 if ((bRedrawAll_) || (pclTempControl->IsStale()))
00110                 {
00111                     pclTempControl->Draw();
00112                     pclTempControl->ClearStale();
00113                 }
00114
00115                 ucControlsLeft--;
00116             }

```



```

00117
00118         pclTempNode = pclTempNode->GetNext();
00119     }
00120     ucCurrentZ++;
00121 }
00122 GUI_DEBUG_PRINT(" Current Z: %d\n", ucCurrentZ);
00123 GUI_DEBUG_PRINT(" Controls Left: %d\n", ucControlsLeft);
00124 }
00125
00126 //-----
00127 void GuiWindow::InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT
usWidth_, K_USHORT usHeight_ )
00128 {
00129     LinkListNode *pclTempNode;
00130     K_USHORT usLeft1, usLeft2, usRight1, usRight2, usTop1, usTop2, usBottom1, usBottom2;
00131
00132     pclTempNode = m_clControlList.GetHead();
00133
00134     usLeft1 = usLeft_;
00135     usRight1 = usLeft_ + usWidth_ - 1;
00136     usTop1 = usTop_;
00137     usBottom1 = usTop_ + usHeight_ - 1;
00138
00139     while (pclTempNode)
00140     {
00141         GuiControl *pclControl = static_cast<GuiControl*>(pclTempNode);
00142         K_USHORT usX, usY;
00143
00144         bool bMatch = false;
00145
00146         // Get the absolute display coordinates
00147         pclControl->GetControlOffset(&usX, &usY);
00148
00149
00150         usLeft2 = pclControl->GetLeft() + usX;
00151         usRight2 = usLeft2 + pclControl->GetWidth() - 1;
00152         usTop2 = pclControl->GetTop() + usY;
00153         usBottom2 = usTop2 + pclControl->GetHeight() - 1;
00154
00155         // If the control has any pixels in the bounding box.
00156         if (
00157             (
00158                 (
00159                     (usLeft1 >= usLeft2) &&
00160                     (usLeft1 <= usRight2)
00161                 ) ||
00162                 (
00163                     (usRight1 >= usLeft2) &&
00164                     (usRight1 <= usRight2)
00165                 ) ||
00166                 ((usLeft1 <= usLeft2) && (usRight1 >= usRight2))
00167             ) &&
00168             (
00169                 (
00170                     (usTop1 >= usTop2) &&
00171                     (usTop1 <= usBottom2)
00172                 ) ||
00173                 (
00174                     (usBottom1 >= usTop2) &&
00175                     (usBottom1 <= usBottom2)
00176                 ) ||
00177                 ((usTop1 <= usTop2) && (usBottom1 >= usBottom2))
00178             )
00179         )
00180         {
00181             bMatch = true;
00182         }
00183         else if(
00184             (
00185                 (
00186                     (usLeft2 >= usLeft1) &&
00187                     (usLeft2 <= usRight1)
00188                 ) ||
00189                 (
00190                     (usRight2 >= usLeft1) &&
00191                     (usRight2 <= usRight1)
00192                 ) ||
00193                 ((usLeft2 <= usLeft1) && (usRight2 >= usRight1))
00194             ) &&
00195             (
00196                 (
00197                     (usTop2 >= usTop1) &&
00198                     (usTop2 <= usBottom1)
00199                 ) ||
00200                 (
00201                     (usBottom2 >= usTop1) &&
00202                     (usBottom2 <= usBottom1)

```

```

00203         ) ||
00204         ((usTop2 <= usTop1) && (usBottom2 >= usBottom1))
00205     )
00206 )
00207 {
00208     bMatch = true;
00209 }
00210
00211
00212 if (bMatch)
00213 {
00214     pclControl->SetStale();
00215
00216     // Invalidate all child controls as well (since redrawing a parent could cause them to
disappear)
00217     GuiControl *pclChild = static_cast<GuiControl*>(
m_clControlList.GetHead());
00218
00219     // Go through all controls and check for parental ancestry
00220     while (pclChild)
00221     {
00222         GuiControl *pclParent = static_cast<GuiControl*>(pclChild->
GetParentControl());
00223
00224         // If this control is a descendant of the current control at some level
00225         while (pclParent)
00226         {
00227             if (pclParent == pclControl)
00228             {
00229                 // Set the control as stale
00230                 pclChild->SetStale();
00231                 break;
00232             }
00233             pclParent = pclParent->GetParentControl();
00234         }
00235
00236         pclChild = static_cast<GuiControl*>((static_cast<
LinkListNode*>(pclChild))->GetNext());
00237     }
00238 }
00239
00240 pclTempNode = pclTempNode->GetNext();
00241 }
00242 }
00243
00244 //-----
00245 void GuiWindow::ProcessEvent( GuiEvent_t *pstEvent_ )
00246 {
00247     GUI_DEBUG_PRINT("GuiWindow::ProcessEvent\n");
00248
00249     // If the event is for broadcast - send it to all controls,
00250     // without regard to order.
00251     if ((TARGET_ID_BROADCAST == pstEvent_->ucTargetID)
00252         || (TARGET_ID_BROADCAST_Z == pstEvent_->ucTargetID))
00253     {
00254         GUI_DEBUG_PRINT(" TARGET_ID_BROADCAST(_Z)\n");
00255
00256         LinkListNode *pclTempNode;
00257         pclTempNode = m_clControlList.GetHead();
00258
00259         while (pclTempNode)
00260         {
00261             GuiReturn_t eRet;
00262             eRet = (static_cast<GuiControl*>(pclTempNode))->ProcessEvent(pstEvent_);
00263             if (GUI_EVENT_CONSUMED == eRet)
00264             {
00265                 break;
00266             }
00267             pclTempNode = pclTempNode->GetNext();
00268         }
00269     }
00270     // Send the event only to the currently-selected object.
00271     else if (TARGET_ID_FOCUS == pstEvent_->ucTargetID)
00272     {
00273         GUI_DEBUG_PRINT(" TARGET_ID_FOCUS\n");
00274         GuiReturn_t eReturn = GUI_EVENT_OK;
00275
00276         // Try to let the control process the event on its own
00277         if (m_pclInFocus)
00278         {
00279             eReturn = m_pclInFocus->ProcessEvent(pstEvent_);
00280         }
00281
00282         // If the event was not consumed, use default logic to process the event
00283         if (GUI_EVENT_CONSUMED != eReturn)
00284         {
00285             if (EVENT_TYPE_KEYBOARD == pstEvent_->ucEventType)

```

```

00286         {
00287             if (KEYCODE_TAB == pstEvent_>stKey.ucKeyCode)
00288             {
00289                 if (pstEvent_>stKey.bKeyState)
00290                 {
00291                     CycleFocus(true);
00292                 }
00293             }
00294         }
00295         else if (EVENT_TYPE_JOYSTICK == pstEvent_>
ucEventType)
00296         {
00297             if (pstEvent_>stJoystick.bUp || pstEvent_>
stJoystick.bLeft)
00298             {
00299                 // Cycle focus *backwards*
00300                 CycleFocus(false);
00301             }
00302             else if (pstEvent_>stJoystick.bRight || pstEvent_>
stJoystick.bDown)
00303             {
00304                 // Cycle focus *forewards*
00305                 CycleFocus(true);
00306             }
00307         }
00308     }
00309 }
00310 else if (TARGET_ID_HIGH_Z == pstEvent_>ucTargetID)
00311 {
00312     GUI_DEBUG_PRINT("  TARGET_ID_HIGH_Z\n");
00313     K_USHORT usTargetX, usTargetY;
00314     K_USHORT usOffsetX, usOffsetY;
00315     K_UCHAR ucMaxZ = 0;
00316     LinkListNode *pclTempNode;
00317     pclTempNode = m_clControlList.GetHead();
00318     switch (pstEvent_>ucEventType)
00319     {
00320     case EVENT_TYPE_MOUSE:
00321     case EVENT_TYPE_TOUCH:
00322     {
00323         GuiControl *pclTargetControl = NULL;
00324         // Read the target X/Y coordinates out of the event struct
00325         if (EVENT_TYPE_TOUCH == pstEvent_>ucEventType)
00326         {
00327             usTargetX = pstEvent_>stTouch.usX;
00328             usTargetY = pstEvent_>stTouch.usY;
00329         }
00330         else
00331         {
00332             usTargetX = pstEvent_>stMouse.usX;
00333             usTargetY = pstEvent_>stMouse.usY;
00334         }
00335         // Go through every control on the window, checking to see if the
00336         // event falls within the bounding box
00337         while (pclTempNode)
00338         {
00339             GuiControl *pclControl = (static_cast<GuiControl*>(pclTempNode));
00340             pclControl->GetControlOffset(&usOffsetX, &usOffsetY);
00341             // Compare event coordinates to bounding box (with offsets)
00342             if ( ((usTargetX >= (usOffsetX + pclControl->GetLeft()) &&
(usTargetX <= (usOffsetX + pclControl->GetLeft() + pclControl->
GetWidth() - 1)))) &&
00343                 ((usTargetY >= (usOffsetY + pclControl->GetTop()) &&
(usTargetY <= (usOffsetY + pclControl->GetTop() + pclControl->
GetHeight() - 1)))) )
00344             {
00345                 // If this control is higher in Z-Order, set this as the newest
00346                 // candidate control to accept the event
00347                 if (pclControl->GetZOrder() >= ucMaxZ)
00348                 {
00349                     pclTargetControl = pclControl;
00350                     ucMaxZ = pclControl->GetZOrder();
00351                 }
00352             }
00353             pclTempNode = pclTempNode->GetNext();
00354         }
00355         // If a suitable control was found on the event surface, pass the event off
00356         // for processing.
00357     }
00358 }

```

```

00368         if (pclTargetControl)
00369         {
00370             // If the selected control is different from the current in-focus
00371             // control, then deactivate that control.
00372             if (m_pclInFocus && (m_pclInFocus != pclTargetControl))
00373             {
00374                 m_pclInFocus->Activate(false);
00375                 m_pclInFocus = NULL;
00376             }
00377             (static_cast<GuiControl*>(pclTargetControl))->ProcessEvent(pstEvent_);
00378         }
00379     }
00380     break;
00381 default:
00382     break;
00383 }
00384 }
00385 }
00386 //-----
00387 void GuiWindow::SetFocus( GuiControl *pclControl_ )
00388 {
00389     GUI_DEBUG_PRINT("GuiWindow::SetFocus\n");
00390     m_pclInFocus = pclControl_;
00391 }
00392 }
00393 //-----
00394 void GuiWindow::CycleFocus( bool bForward_ )
00395 {
00396     GUI_DEBUG_PRINT("GuiWindow::CycleFocus\n");
00397     // Set starting point and cached copy of current nodes
00398     LinkListNode *pclTempNode = static_cast<GuiControl*>(
00399         m_clControlList.GetHead());
00400     LinkListNode *pclStartNode = m_pclInFocus;
00401     if (bForward_)
00402     {
00403         // If there isn't a current focus node, set the focus to the beginning
00404         // of the list
00405         if (!m_pclInFocus)
00406         {
00407             m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00408             if (!m_pclInFocus)
00409             {
00410                 return;
00411             }
00412             pclTempNode = static_cast<GuiControl*>(m_pclInFocus);
00413             pclStartNode = NULL;
00414         }
00415         else
00416         {
00417             // Deactivate the control that's losing focus
00418             static_cast<GuiControl*>(m_pclInFocus)->Activate(false);
00419             // Otherwise start with the next node
00420             pclStartNode = pclStartNode->GetNext();
00421         }
00422         // Go through the whole control list and find the next one to accept
00423         // the focus
00424         while (pclTempNode && pclTempNode != pclStartNode)
00425         {
00426             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00427             {
00428                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00429                 m_pclInFocus->Activate(true);
00430                 SetFocus(m_pclInFocus);
00431                 return;
00432             }
00433             pclTempNode = pclTempNode->GetNext();
00434         }
00435         pclTempNode = static_cast<GuiControl*>(m_clControlList.
00436             GetHead());
00437         while (pclTempNode && pclTempNode != pclStartNode)
00438         {
00439             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00440             {
00441                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00442                 m_pclInFocus->Activate(true);
00443                 SetFocus(m_pclInFocus);
00444                 return;
00445             }
00446             pclTempNode = pclTempNode->GetNext();
00447         }
00448     }
00449 }

```

```

00453     else
00454     {
00455         pclTempNode = static_cast<GuiControl*>(m_clControlList.
GetTail());
00456         pclStartNode = m_pclInFocus;
00457
00458         // If there isn't a current focus node, set the focus to the end
00459         // of the list
00460         if (!m_pclInFocus)
00461         {
00462             m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00463             if (!m_pclInFocus)
00464             {
00465                 return;
00466             }
00467             pclTempNode = static_cast<GuiControl*>(m_pclInFocus);
00468             pclStartNode = NULL;
00469         }
00470         else
00471         {
00472             // Deactivate the control that's losing focus
00473             static_cast<GuiControl*>(m_pclInFocus)->Activate(false);
00474
00475             // Otherwise start with the previous node
00476             pclStartNode = pclStartNode->GetPrev();
00477         }
00478
00479         // Go through the whole control list and find the next one to accept
00480         // the focus
00481         while (pclTempNode && pclTempNode != pclStartNode)
00482         {
00483             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00484             {
00485                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00486                 m_pclInFocus->Activate(true);
00487                 SetFocus(m_pclInFocus);
00488                 return;
00489             }
00490             pclTempNode = pclTempNode->GetPrev();
00491         }
00492
00493         pclTempNode = static_cast<GuiControl*>(m_clControlList.
GetTail());
00494         while (pclTempNode && pclTempNode != pclStartNode)
00495         {
00496             if (static_cast<GuiControl*>(pclTempNode)->AcceptsFocus())
00497             {
00498                 m_pclInFocus = static_cast<GuiControl*>(pclTempNode);
00499                 m_pclInFocus->Activate(true);
00500                 SetFocus(m_pclInFocus);
00501                 return;
00502             }
00503             pclTempNode = pclTempNode->GetPrev();
00504         }
00505     }
00506 }
00507 //-----
00508 GuiWindow *GuiEventSurface::FindWindowByName( const K_CHAR *
szName_ )
00509 {
00510     LinkListNode *pclTempNode = static_cast<LinkListNode*>(
m_clWindowList.GetHead());
00511
00512     while (pclTempNode)
00513     {
00514         if (MemUtil::CompareStrings( szName_, static_cast<GuiWindow*>(pclTempNode)->
GetName()))
00515         {
00516             return static_cast<GuiWindow*>(pclTempNode);
00517         }
00518         pclTempNode = pclTempNode->GetNext();
00519     }
00520
00521     return NULL;
00522 }
00523
00524 //-----
00525 void GuiEventSurface::AddWindow( GuiWindow *pclWindow_ )
00526 {
00527     GUI_DEBUG_PRINT("GuiEventSurface::AddWindow\n");
00528     m_clWindowList.Add(static_cast<LinkListNode*>(pclWindow_));
00529 }
00530
00531 //-----
00532 void GuiEventSurface::RemoveWindow( GuiWindow *pclWindow_ )
00533 {
00534

```

```

00535     GUI_DEBUG_PRINT("GuiEventSurface::RemoveWindow\n");
00536
00537     m_clWindowList.Remove(static_cast<LinkListNode*>(pclWindow_));
00538 }
00539
00540 //-----
00541 K_BOOL GuiEventSurface::SendEvent( GuiEvent_t *pstEvent_ )
00542 {
00543     GUI_DEBUG_PRINT("GuiEventSurface::SendEvent\n");
00544
00545     // Allocate a message from the global message pool
00546     Message *pclMessage = GlobalMessagePool::Pop();
00547
00548     // No messages available? Return a failure
00549     if (!pclMessage)
00550     {
00551         return false;
00552     }
00553
00554     // Allocate a copy of the event from the heap
00555     GuiEvent_t *pstEventCopy = static_cast<GuiEvent_t*>(
00556         SystemHeap::Alloc(sizeof(GuiEvent_t)));
00557
00558     // If the allocation fails, push the message back to the global pool and bail
00559     if (!pstEventCopy)
00560     {
00561         GlobalMessagePool::Push(pclMessage);
00562         return false;
00563     }
00564
00565     // Copy the source event into the destination event buffer
00566     CopyEvent(pstEventCopy, pstEvent_);
00567
00568     // Set the new event as the message payload
00569     pclMessage->SetData(static_cast<void*>(pstEventCopy));
00570
00571     // Send the event to the message queue
00572     m_clMessageQueue.Send(pclMessage);
00573
00574     return true;
00575 }
00576 //-----
00577 K_BOOL GuiEventSurface::ProcessEvent()
00578 {
00579     GUI_DEBUG_PRINT("GuiEventSurface::ProcessEvent\n");
00580
00581     // read the event from the queue (blocking call)
00582     Message *pclMessage = m_clMessageQueue.Receive();
00583     GuiEvent_t stLocalEvent;
00584
00585     // If we failed to get something from the queue,
00586     // bail out
00587     if (!pclMessage)
00588     {
00589         return false;
00590     }
00591
00592     // Copy the event data from the message into a local copy
00593     CopyEvent(&stLocalEvent,
00594         static_cast<GuiEvent_t*>(pclMessage->GetData()));
00595
00596     // Free the message and event as soon as possible, since
00597     // they are shared system resources
00598     SystemHeap::Free(pclMessage->GetData());
00599     GlobalMessagePool::Push(pclMessage);
00600
00601     // Special case check - target ID is the highest Z-ordered window(s) ONLY.
00602     if (stLocalEvent.ucTargetID == TARGET_ID_BROADCAST_Z)
00603     {
00604         LinkListNode* pclTempNode = m_clWindowList.
00605         GetHead();
00606         K_UCHAR ucMaxZ = 0;
00607
00608         while (pclTempNode)
00609         {
00610             if (ucMaxZ < (static_cast<GuiWindow*>(pclTempNode))->GetZOrder() )
00611             {
00612                 ucMaxZ = static_cast<GuiWindow*>(pclTempNode)->GetZOrder();
00613             }
00614             pclTempNode = pclTempNode->GetNext();
00615         }
00616
00617         // Iterate through all windows again - may have multiple windows
00618         // at the same z-order.
00619         pclTempNode = m_clWindowList.GetHead();
00620         while (pclTempNode)

```

```

00620     {
00621         if (ucMaxZ == (static_cast<GuiWindow*>(pclTempNode))->GetZOrder())
00622         {
00623             (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&stLocalEvent);
00624         }
00625         pclTempNode = pclTempNode->GetNext();
00626     }
00627 }
00628 // Broadcast the event - sending it to *all* windows. Let the individual
00629 // windows figure out what to do with the events.
00630 else
00631 {
00632     LinkListNode* pclTempNode = m_clWindowList.
00633     GetHead();
00634     while (pclTempNode)
00635     {
00636         (static_cast<GuiWindow*>(pclTempNode))->ProcessEvent(&stLocalEvent);
00637         pclTempNode = pclTempNode->GetNext();
00638     }
00639 }
00640 // Return out
00641 return true;
00642 }
00643
00644 //-----
00645 void GuiEventSurface::CopyEvent( GuiEvent_t *pstDst_,
00646     GuiEvent_t *pstSrc_ )
00647 {
00648     GUI_DEBUG_PRINT("GuiEventSurface::CopyEvent\n");
00649     K_UCHAR *pucDst_ = (K_UCHAR*)pstDst_;
00650     K_UCHAR *pucSrc_ = (K_UCHAR*)pstSrc_;
00651     K_UCHAR i;
00652     for (i = 0; i < sizeof(GuiEvent_t); i++)
00653     {
00654         *pucDst_++ = *pucSrc_++;
00655     }
00656 }
00657 //-----
00658 void GuiEventSurface::InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_,
00659     K_USHORT usWidth_, K_USHORT usHeight_ )
00660 {
00661     LinkListNode* pclTempNode = m_clWindowList.GetHead();
00662     while (pclTempNode)
00663     {
00664         (static_cast<GuiWindow*>(pclTempNode))->InvalidateRegion(usLeft_, usTop_, usWidth_,
00665             usHeight_);
00666         pclTempNode = pclTempNode->GetNext();
00667     }
00668 }
00669 //-----
00670 void GuiControl::GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ )
00671 {
00672     GUI_DEBUG_PRINT("GuiControl::GetControlOffset\n");
00673     GuiControl *pclTempControl = m_pclParentControl;
00674     *pusX_ = 0;
00675     *pusY_ = 0;
00676     while (pclTempControl)
00677     {
00678         *pusX_ += pclTempControl->GetLeft();
00679         *pusY_ += pclTempControl->GetTop();
00680         pclTempControl = pclTempControl->GetParentControl();
00681     }
00682     if (m_pclParentWindow)
00683     {
00684         *pusX_ += m_pclParentWindow->GetLeft();
00685         *pusY_ += m_pclParentWindow->GetTop();
00686     }
00687 }

```

15.73 /home/moslevin/Project/R1/stage/src/gui.h File Reference

Graphical User Interface classes and data structure declarations.

```
#include "kerneltypes.h"
#include "ll.h"
#include "driver.h"
#include "graphics.h"
#include "message.h"
#include "keycodes.h"
```

Classes

- struct [KeyEvent_t](#)
Keyboard UI event structure definition.
- struct [MouseEvent_t](#)
Mouse UI event structure.
- struct [TouchEvent_t](#)
Touch UI event structure.
- struct [JoystickEvent_t](#)
Joystick UI event structure.
- struct [TimerEvent_t](#)
Timer UI event structure.
- struct [GuiEvent_t](#)
Composite UI event structure.
- class [GuiWindow](#)
Basic Window Class.
- class [GuiEventSurface](#)
GUI Event Surface Object.
- class [GuiControl](#)
GUI Control Base Class.
- class [StubControl](#)
Stub control class, used for testing out the GUI framework where certain controls have not yet been implemented.

Macros

- #define [GUI_DEBUG](#) (0)
- #define [GUI_DEBUG_PRINT](#)(...)
- #define [EVENT_STATE_UP](#) (0)
Event state definitions, used for determining whether or not a button or key is in the "up" or "down" contact state.
- #define [EVENT_STATE_DOWN](#) (1)
- #define [MAX_WINDOW_CONTROLS](#) (251)
Maximum number of controls per window.
- #define [TARGET_ID_BROADCAST_Z](#) (252)
Broadcast event to all controls in the topmost window.
- #define [TARGET_ID_BROADCAST](#) (253)
Send event to all controls in all windows.
- #define [TARGET_ID_FOCUS](#) (254)
Send event to the in-focus control.
- #define [TARGET_ID_HIGH_Z](#) (255)
Send event to the highest Z-order control.

Enumerations

- enum `GuiEventType_t` {
 `EVENT_TYPE_KEYBOARD`, `EVENT_TYPE_MOUSE`, `EVENT_TYPE_TOUCH`, `EVENT_TYPE_JOYSTICK`,
 `EVENT_TYPE_TIMER`, `EVENT_TYPE_COUNT` }

 Enumeration defining the various UI event codes.
- enum `GuiReturn_t` {
 `GUI_EVENT_OK` = 0, `GUI_EVENT_CONSUMED`, `GUI_EVENT_CANCEL`, `GUI_EVENT_RETRY`,
 `GUI_EVENT_COUNT` }

15.73.1 Detailed Description

Graphical User Interface classes and data structure declarations.

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

15.73.2 Enumeration Type Documentation

15.73.2.1 enum GuiEventType_t

Enumeration defining the various UI event codes.

Enumerator

EVENT_TYPE_KEYBOARD Keypress event.

EVENT_TYPE_MOUSE Mouse movement or click event.

EVENT_TYPE_TOUCH Touchscreen movement event.

EVENT_TYPE_JOYSTICK Joystick event.

EVENT_TYPE_TIMER [Timer](#) event.

EVENT_TYPE_COUNT Count of different event types supported.

Definition at line 65 of file gui.h.

15.73.2.2 enum GuiReturn_t

Enumerator

GUI_EVENT_OK No problem.

GUI_EVENT_CONSUMED Event was consumed.

GUI_EVENT_CANCEL Event processing canceled.

GUI_EVENT_RETRY Retry processing the event.

Definition at line 203 of file gui.h.

15.74 gui.h

```

00001 /*=====
00002
00003  _____
00004  |   |   |   |   |   |   |   |   |   |   |
00005  |   |   |   |   |   |   |   |   |   |   |
00006  |   |   |   |   |   |   |   |   |   |   |
00007  |   |   |   |   |   |   |   |   |   |   |
00008  |   |   |   |   |   |   |   |   |   |   |
00009  --[Mark3 Realtime Platform]-----
00010

```

```

00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __GUI_H__
00020 #define __GUI_H__
00021
00022 #include "kerneltypes.h"
00023 #include "ll.h"
00024 #include "driver.h"
00025 #include "graphics.h"
00026
00027 #include "message.h"
00028
00029 #include "keycodes.h"
00030
00031 #define GUI_DEBUG                (0)
00032
00033 #if GUI_DEBUG
00034     #include <stdio.h>
00035     #include <stdlib.h>
00036     #include <string.h>
00037
00038     #define GUI_DEBUG_PRINT        printf
00039 #else
00040 #define GUI_DEBUG_PRINT(...)
00041 #endif
00042
00043
00044 //-----
00049 #define EVENT_STATE_UP            (0)
00050 #define EVENT_STATE_DOWN        (1)
00051
00052 //-----
00053 #define MAX_WINDOW_CONTROLS      (251)
00054
00055 #define TARGET_ID_BROADCAST_Z    (252)
00056 #define TARGET_ID_BROADCAST      (253)
00057 #define TARGET_ID_FOCUS          (254)
00058 #define TARGET_ID_HIGH_Z        (255)
00059
00060
00061 //-----
00065 typedef enum
00066 {
00067     EVENT_TYPE_KEYBOARD,
00068     EVENT_TYPE_MOUSE,
00069     EVENT_TYPE_TOUCH,
00070     EVENT_TYPE_JOYSTICK,
00071     EVENT_TYPE_TIMER,
00072 }
00073     EVENT_TYPE_COUNT
00074 } GuiEventType_t;
00075
00076 //-----
00080 typedef struct
00081 {
00082     K_UCHAR ucKeyCode;
00083     union
00084     {
00085         K_UCHAR ucFlags;
00086         struct
00087         {
00088             unsigned int bKeyState:1;
00089             unsigned int bShiftState:1;
00090             unsigned int bCtrlState:1;
00091             unsigned int bAltState:1;
00092             unsigned int bWinState:1;
00093             unsigned int bFnState:1;
00094         };
00095     };
00096 } KeyEvent_t;
00097
00098 //-----
00102 typedef struct
00103 {
00104     K_USHORT usX;
00105     K_USHORT usY;
00106     union
00107     {
00108         K_UCHAR ucFlags;
00109         struct
00110         {
00111             unsigned int bLeftState:1;
00112             unsigned int bRightState:1;
00113             unsigned int bMiddleState:1;
00114             unsigned int bScrollUp:1;
00115

```

```

00116         unsigned int bScrollDown:1;
00117     };
00118 };
00119 } MouseEvent_t;
00120
00121 //-----
00125 typedef struct
00126 {
00127     K_USHORT usX;
00128     K_USHORT usY;
00129
00130     union
00131     {
00132         K_USHORT ucFlags;
00133         struct
00134         {
00135             unsigned int bTouch:1;
00136         };
00137     };
00138 } TouchEvent_t;
00139
00140 //-----
00144 typedef struct
00145 {
00146     union
00147     {
00148         K_USHORT usRawData;
00149         struct
00150         {
00151             unsigned int bUp:1;
00152             unsigned int bDown:1;
00153             unsigned int bLeft:1;
00154             unsigned int bRight:1;
00155
00156             unsigned int bButton1:1;
00157             unsigned int bButton2:1;
00158             unsigned int bButton3:1;
00159             unsigned int bButton4:1;
00160             unsigned int bButton5:1;
00161             unsigned int bButton6:1;
00162             unsigned int bButton7:1;
00163             unsigned int bButton8:1;
00164             unsigned int bButton9:1;
00165             unsigned int bButton10:1;
00166
00167             unsigned int bSelect:1;
00168             unsigned int bStart:1;
00169         };
00170     };
00171 } JoystickEvent_t;
00172
00173 //-----
00177 typedef struct
00178 {
00179     K_USHORT usTicks;
00180 } TimerEvent_t;
00181
00182 //-----
00187 typedef struct
00188 {
00189     K_UCHAR ucEventType;
00190     K_UCHAR ucTargetID;
00191     union
00192     {
00193         KeyEvent_t      stKey;
00194         MouseEvent_t     stMouse;
00195         TouchEvent_t     stTouch;
00196         JoystickEvent_t  stJoystick;
00197         TimerEvent_t     stTimer;
00198     };
00199
00200 } GuiEvent_t;
00201
00202 //-----
00203 typedef enum
00204 {
00205     GUI_EVENT_OK = 0,
00206     GUI_EVENT_CONSUMED,
00207     GUI_EVENT_CANCEL,
00208     GUI_EVENT_RETRY,
00209 } GUI_EVENT_COUNT
00210 } GuiReturn_t;
00211
00212 class GuiControl;
00213
00214 //-----

```

```

00223 class GuiWindow : public LinkListNode
00224 {
00225
00226 public:
00231     void Init()
00232     {
00233         m_ucControlCount = 0;
00234         m_pclDriver = NULL;
00235         m_szName = "";
00236         ClearNode();
00237     }
00238
00245     void SetDriver( GraphicsDriver *pclDriver_ ) {
m_pclDriver = pclDriver_; }
00246
00253     GraphicsDriver *GetDriver() { return m_pclDriver; }
00254
00266     void AddControl( GuiControl *pclControl_, GuiControl *pclParent_ );
00267
00275     void RemoveControl( GuiControl *pclControl_ );
00276
00284     K_UCHAR GetMaxZOrder();
00285
00294     void Redraw( K_BOOL bRedrawAll_ );
00295
00302     void ProcessEvent( GuiEvent_t *pstEvent_ );
00303
00312     void SetFocus( GuiControl *pclControl_ );
00313
00324     K_BOOL IsInFocus( GuiControl *pclControl_ )
00325     {
00326         if (m_pclInFocus == pclControl_)
00327         {
00328             return true;
00329         }
00330         return false;
00331     }
00332
00338     void SetTop( K_USHORT usTop_ )          { m_usTop = usTop_; }
00339
00345     void SetLeft( K_USHORT usLeft_ )         { m_usLeft = usLeft_; }
00346
00352     void SetHeight( K_USHORT usHeight_ ) { m_usHeight = usHeight_; }
00353
00359     void SetWidth( K_USHORT usWidth_ )       { m_usWidth = usWidth_; }
00360
00366     K_USHORT GetTop()                       { return m_usTop; }
00367
00373     K_USHORT GetLeft()                      { return m_usLeft; }
00374
00380     K_USHORT GetHeight()                    { return m_usHeight; }
00381
00387     K_USHORT GetWidth()                     { return m_usWidth; }
00388
00392     K_UCHAR GetZOrder()                     { return m_ucZ; }
00393
00397     void SetZOrder( K_UCHAR ucZ_ ) { m_ucZ = ucZ_; }
00398
00406     void CycleFocus( bool bForward_ );
00407
00411     void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }
00412
00416     const K_CHAR *GetName() { return m_szName; }
00417
00423     void InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT
usHeight_ );
00424
00425 private:
00426     K_USHORT m_usTop;
00427     K_USHORT m_usLeft;
00428     K_USHORT m_usHeight;
00429     K_USHORT m_usWidth;
00430
00431     K_UCHAR m_ucZ;
00432     const K_CHAR *m_szName;
00433
00434     DoubleLinkedList m_clControlList;
00435     GuiControl *m_pclInFocus;
00436     K_UCHAR m_ucControlCount;
00437     GraphicsDriver *m_pclDriver;
00438 };
00439
00440 //-----
00453 class GuiEventSurface
00454 {
00455 public:
00460     void Init() { m_clMessageQueue.Init(); }

```

```

00461
00467     void AddWindow( GuiWindow *pclWindow_ );
00468
00474     void RemoveWindow( GuiWindow *pclWindow_ );
00475
00483     K_BOOL SendEvent( GuiEvent_t *pstEvent_ );
00484
00489     K_BOOL ProcessEvent();
00490
00494     K_UCHAR GetEventCount() { return m_clMessageQueue.
GetCount(); }
00495
00499     GuiWindow *FindWindowByName( const K_CHAR *szName_ );
00500
00506     void InvalidateRegion( K_USHORT usLeft_, K_USHORT usTop_, K_USHORT usWidth_, K_USHORT
usHeight_ );
00507
00508 private:
00515     void CopyEvent( GuiEvent_t *pstDst_, GuiEvent_t *pstSrc_ );
00516
00517 private:
00521     DoubleLinkedList m_clWindowList;
00522
00526     MessageQueue     m_clMessageQueue;
00527 };
00528
00529 //-----
00539 class GuiControl : public LinkListNode
00540 {
00541 public:
00548     virtual void Init() = 0;
00549
00555     virtual void Draw() = 0;
00556
00564     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) = 0;
00565
00571     void SetTop( K_USHORT usTop_ )          { m_usTop = usTop_; }
00572
00578     void SetLeft( K_USHORT usLeft_ )        { m_usLeft = usLeft_; }
00579
00585     void SetHeight( K_USHORT usHeight_ ) { m_usHeight = usHeight_; }
00586
00592     void SetWidth( K_USHORT usWidth_ )      { m_usWidth = usWidth_; }
00593
00599     void SetZOrder( K_UCHAR ucZ_ )          { m_ucZOrder = ucZ_; }
00600
00607     void SetControlIndex( K_UCHAR ucIdx_ ) { m_ucControlIndex = ucIdx_; }
00608
00614     K_USHORT GetTop()                      { return m_usTop; }
00615
00621     K_USHORT GetLeft()                    { return m_usLeft; }
00622
00628     K_USHORT GetHeight()                  { return m_usHeight; }
00629
00635     K_USHORT GetWidth()                   { return m_usWidth; }
00636
00642     K_UCHAR GetZOrder()                   { return m_ucZOrder; }
00643
00649     K_UCHAR GetControlIndex()             { return m_ucControlIndex; }
00650
00656     K_BOOL IsStale()                      { return m_bStale; }
00657
00669     void GetControlOffset( K_USHORT *pusX_, K_USHORT *pusY_ );
00670
00678     K_BOOL IsInFocus()
00679     {
00680         return m_pclParentWindow->IsInFocus(this);
00681     }
00682
00690     virtual void Activate( bool bActivate_ ) = 0;
00691
00692 protected:
00693     friend class GuiWindow;
00694     friend class GuiEventSurface;
00695
00707     void SetParentControl( GuiControl *pclParent_ ) {
m_pclParentControl = pclParent_; }
00708
00718     void SetParentWindow( GuiWindow *pclWindow_ ) {
m_pclParentWindow = pclWindow_; }
00719
00726     GuiControl *GetParentControl()           { return
m_pclParentControl; }
00727
00734     GuiWindow *GetParentWindow()            { return
m_pclParentWindow; }
00735

```

```

00742     void ClearStale()                                { m_bStale = false; }
00743
00747     void SetStale()                                { m_bStale = true; }
00748
00752     void SetAcceptFocus( bool bFocus_ )            {
m_bAcceptsFocus = bFocus_; }
00753
00757     bool AcceptsFocus()                            { return
m_bAcceptsFocus; }
00758 private:
00760     K_BOOL      m_bStale;
00761
00763     K_BOOL      m_bAcceptsFocus;
00764
00767     K_UCHAR     m_ucZOrder;
00768
00771     K_UCHAR     m_ucControlIndex;
00772
00774     K_USHORT    m_usTop;
00775
00777     K_USHORT    m_usLeft;
00778
00780     K_USHORT    m_usWidth;
00781
00783     K_USHORT    m_usHeight;
00784
00786     GuiControl *m_pclParentControl;
00787
00789     GuiWindow   *m_pclParentWindow;
00790 };
00791
00792 //-----
00797 class StubControl : public GuiControl
00798 {
00799 public:
00800     virtual void Init() { }
00801     virtual void Draw() { }
00802     virtual GuiReturn_t ProcessEvent( GuiEvent_t *pstEvent_ ) { return
GUI_EVENT_OK; }
00803     virtual void Activate( bool bActivate_ ) { }
00804 };
00805
00806 #endif
00807

```

15.75 /home/moslevin/Project/R1/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

15.79 /home/moslevin/Project/R1/stage/src/kernel_debug.h File Reference

```
#include "debug_tokens.h"
#include "mark3cfg.h"
#include "tracebuffer.h"
#include "kernel_aware.h"
#include "panic_codes.h"
#include "kernel.h"
```

- #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)

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

```
00001 /*=====
```

```
00002
```

```
00003      _   _   _   _   _   _   _   _   _   _   _   _   _   _
```

```
00004 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
```

```
00005 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
```

```
00006 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
```

```
00007 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
```

```
00008
```

```
00009 --[Mark3 Realtime Platform]-----
```

```
00010
```

```
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
```

```
00012 See license.txt for more information
```

```
00013 =====*/
```

```
00020 #ifndef __KERNEL_DEBUG_H__
```

```
00021 #define __KERNEL_DEBUG_H__
```

```

00022
00023 #include "debug_tokens.h"
00024 #include "mark3cfg.h"
00025 #include "tracebuffer.h"
00026 #include "kernel_aware.h"
00027 #include "panic_codes.h"
00028 #include "kernel.h"
00029 //-----
00030 #if (KERNEL_USE_DEBUG && !KERNEL_AWARE_SIMULATION)
00031
00032 //-----
00033 #define __FILE_ID__          STR_UNDEFINED
00034
00035 //-----
00036 #define KERNEL_TRACE( x ) \
00037 { \
00038     K_USHORT ausMsg__[5]; \
00039     ausMsg__[0] = 0xACDC; \
00040     ausMsg__[1] = __FILE_ID__; \
00041     ausMsg__[2] = __LINE__; \
00042     ausMsg__[3] = TraceBuffer::Increment(); \
00043     ausMsg__[4] = (K_USHORT)(x); \
00044     TraceBuffer::Write(ausMsg__, 5); \
00045 };
00046
00047 //-----
00048 #define KERNEL_TRACE_1( x, arg1 ) \
00049 { \
00050     K_USHORT ausMsg__[6]; \
00051     ausMsg__[0] = 0xACDC; \
00052     ausMsg__[1] = __FILE_ID__; \
00053     ausMsg__[2] = __LINE__; \
00054     ausMsg__[3] = TraceBuffer::Increment(); \
00055     ausMsg__[4] = (K_USHORT)(x); \
00056     ausMsg__[5] = arg1; \
00057     TraceBuffer::Write(ausMsg__, 6); \
00058 };
00059
00060 //-----
00061 #define KERNEL_TRACE_2( x, arg1, arg2 ) \
00062 { \
00063     K_USHORT ausMsg__[7]; \
00064     ausMsg__[0] = 0xACDC; \
00065     ausMsg__[1] = __FILE_ID__; \
00066     ausMsg__[2] = __LINE__; \
00067     ausMsg__[3] = TraceBuffer::Increment(); \
00068     ausMsg__[4] = (K_USHORT)(x); \
00069     ausMsg__[5] = arg1; \
00070     ausMsg__[6] = arg2; \
00071     TraceBuffer::Write(ausMsg__, 7); \
00072 };
00073
00074 //-----
00075 #define KERNEL_ASSERT( x ) \
00076 { \
00077     if( ( x ) == false ) \
00078     { \
00079         K_USHORT ausMsg__[5]; \
00080         ausMsg__[0] = 0xACDC; \
00081         ausMsg__[1] = __FILE_ID__; \
00082         ausMsg__[2] = __LINE__; \
00083         ausMsg__[3] = TraceBuffer::Increment(); \
00084         ausMsg__[4] = STR_ASSERT_FAILED; \
00085         TraceBuffer::Write(ausMsg__, 5); \
00086         Kernel::Panic(PANIC_ASSERT_FAILED); \
00087     } \
00088 };
00089
00090 #elif (KERNEL_USE_DEBUG && KERNEL_AWARE_SIMULATION)
00091 //-----
00092 #define __FILE_ID__          STR_UNDEFINED
00093
00094 //-----
00095 #define KERNEL_TRACE( x ) \
00096 { \
00097     Kernel_Aware::Trace( __FILE_ID__, __LINE__, x ); \
00098 };
00099
00100 //-----
00101 #define KERNEL_TRACE_1( x, arg1 ) \
00102 { \
00103     Kernel_Aware::Trace( __FILE_ID__, __LINE__, x, arg1 ); \
00104 };
00105
00106 //-----
00107 #define KERNEL_TRACE_2( x, arg1, arg2 ) \
00108 { \

```

15.81 /home/moslevin/Project/R1/stage/src/kernelswi.cpp File Reference

```
#include "kerntypes.h"
#include "kernelswi.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

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

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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     bool bEnabled = (EIMSK & (1 << INT0)) != 0;
00053     EIMSK &= ~(1 << INT0);
00054     return bEnabled;
00055 }
00056
00057 //-----
00058 void KernelSWI::RI(bool 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 }

```

15.83 /home/moslevin/Project/R1/stage/src/kernelswi.h File Reference

[Kernel](#) Software interrupt declarations.

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

Classes

- class [KernelSWI](#)

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

15.83.1 Detailed Description

[Kernel](#) Software interrupt declarations.

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

15.84 kernelswi.h

```

00001  /*=====
00002
00003  _____
00004  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00005  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00006  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00007  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |  \  /  |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00023  #include "kerneltypes.h"
00024  #ifndef __KERNELSWI_H_
00025  #define __KERNELSWI_H_
00026
00027  //-----
00032  class KernelSWI
00033  {
00034  public:
00041      static void Config(void);
00042
00048      static void Start(void);
00049
00055      static void Stop(void);
00056
00062      static void Clear(void);
00063
00069      static void Trigger(void);
00070
00078      static K_UCHAR DI();
00079
00087      static void RI(bool bEnable_);
00088  };
00089
00090
00091  #endif // __KERNELSWI_H_

```

15.85 /home/moslevin/Project/R1/stage/src/kerneltimer.cpp File Reference

[Kernel Timer](#) Implementation for ATmega328p.

```

#include "kerneltypes.h"
#include "kerneltimer.h"
#include "mark3cfg.h"
#include <avr/io.h>
#include <avr/interrupt.h>

```

Macros

- `#define TCCR1B_INIT ((1 << WGM12) | (1 << CS12))`
- `#define TIMER_IMSK (1 << OCIE1A)`
- `#define TIMER_IFR (1 << OCF1A)`

15.85.1 Detailed Description

[Kernel Timer](#) Implementation for ATmega328p.

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

15.86 kerneltimer.cpp

```

00001  /*=====

```

```

00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "kerneltimer.h"
00023 #include "mark3cfg.h"
00024
00025 #include <avr/io.h>
00026 #include <avr/interrupt.h>
00027
00028 #define TCCR1B_INIT ((1 << WGM12) | (1 << CS12))
00029 #define TIMER_IMSK (1 << OCIE1A)
00030 #define TIMER_IFR (1 << OCF1A)
00031
00032 //-----
00033 void KernelTimer::Config(void)
00034 {
00035     TCCR1B = TCCR1B_INIT;
00036 }
00037
00038 //-----
00039 void KernelTimer::Start(void)
00040 {
00041     #if !KERNEL_TIMERS_TICKLESS
00042         TCCR1B = ((1 << WGM12) | (1 << CS11) | (1 << CS10));
00043         OCR1A = ((SYSTEM_FREQ / 1000) / 64);
00044     #else
00045         TCCR1B |= (1 << CS12);
00046     #endif
00047
00048     TCNT1 = 0;
00049     TIFR1 &= ~TIMER_IFR;
00050     TIMSK1 |= TIMER_IMSK;
00051 }
00052
00053 //-----
00054 void KernelTimer::Stop(void)
00055 {
00056     #if KERNEL_TIMERS_TICKLESS
00057         TIFR1 &= ~TIMER_IFR;
00058         TIMSK1 &= ~TIMER_IMSK;
00059         TCCR1B &= ~(1 << CS12);    // Disable count...
00060         TCNT1 = 0;
00061         OCR1A = 0;
00062     #endif
00063 }
00064
00065 //-----
00066 K_USHORT KernelTimer::Read(void)
00067 {
00068     #if KERNEL_TIMERS_TICKLESS
00069         volatile K_USHORT usRead1;
00070         volatile K_USHORT usRead2;
00071
00072         do {
00073             usRead1 = TCNT1;
00074             usRead2 = TCNT1;
00075         } while (usRead1 != usRead2);
00076
00077         return usRead1;
00078     #else
00079         return 0;
00080     #endif
00081 }
00082
00083 //-----
00084 K_ULONG KernelTimer::SubtractExpiry(K_ULONG ulInterval_)
00085 {
00086     #if KERNEL_TIMERS_TICKLESS
00087         OCR1A -= (K_USHORT)ulInterval_;
00088         return (K_ULONG)OCR1A;
00089     #else
00090         return 0;
00091     #endif
00092 }
00093
00094 //-----
00095 K_ULONG KernelTimer::TimeToExpiry(void)

```

```

00096 {
00097 #if KERNEL_TIMERS_TICKLESS
00098     K_USHORT usRead = KernelTimer::Read();
00099     K_USHORT usOCR1A = OCR1A;
00100
00101     if (usRead >= usOCR1A)
00102     {
00103         return 0;
00104     }
00105     else
00106     {
00107         return (K_ULONG) (usOCR1A - usRead);
00108     }
00109 #else
00110     return 0;
00111 #endif
00112 }
00113
00114 //-----
00115 K_ULONG KernelTimer::GetOvertime(void)
00116 {
00117     return KernelTimer::Read();
00118 }
00119
00120 //-----
00121 K_ULONG KernelTimer::SetExpiry(K_ULONG ulInterval_)
00122 {
00123 #if KERNEL_TIMERS_TICKLESS
00124     K_USHORT usSetInterval;
00125     if (ulInterval_ > 65535)
00126     {
00127         usSetInterval = 65535;
00128     }
00129     else
00130     {
00131         usSetInterval = (K_USHORT)ulInterval_ ;
00132     }
00133     OCR1A = usSetInterval;
00134     return (K_ULONG)usSetInterval;
00135 #else
00136     return 0;
00137 #endif
00138 }
00139
00140 //-----
00141 void KernelTimer::ClearExpiry(void)
00142 {
00143 #if KERNEL_TIMERS_TICKLESS
00144     OCR1A = 65535; // Clear the compare value
00145 #endif
00146 }
00147
00148 //-----
00149 K_UCHAR KernelTimer::DI(void)
00150 {
00151 #if KERNEL_TIMERS_TICKLESS
00152     bool bEnabled = ((TIMSK1 & (TIMER_IMSK)) != 0);
00153     TIFR1 &= ~TIMER_IFR; // Clear interrupt flags
00154     TIMSK1 &= ~TIMER_IMSK; // Disable interrupt
00155     return bEnabled;
00156 #else
00157     return 0;
00158 #endif
00159 }
00160
00161 //-----
00162 void KernelTimer::EI(void)
00163 {
00164     KernelTimer::RI(0);
00165 }
00166
00167 //-----
00168 void KernelTimer::RI(bool bEnable_)
00169 {
00170 #if KERNEL_TIMERS_TICKLESS
00171     if (bEnable_)
00172     {
00173         TIMSK1 |= (1 << OCIE1A); // Enable interrupt
00174     }
00175     else
00176     {
00177         TIMSK1 &= ~(1 << OCIE1A);
00178     }
00179 #endif
00180 }

```

15.87 /home/moslevin/Project/R1/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))

15.87.1 Detailed Description

[Kernel Timer](#) Class declaration.

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

15.88 kerneltimer.h

```
00001 /*=====
00002
00003
00004 | | | | | | | | | | | | | | | | | | | | | |
00005 | | | | | | | | | | | | | | | | | | | | | |
00006 | | | | | | | | | | | | | | | | | | | | | |
00007 | | | | | | | | | | | | | | | | | | | | | |
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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(bool 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_
```

15.89 /home/moslevin/Project/R1/stage/src/kerneltypes.h File Reference

Basic data type primitives used throughout the OS.

```
#include <stdint.h>
```

Macros

- #define **K_BOOL** uint8_t
- #define **K_CHAR** char
- #define **K_UCHAR** uint8_t
- #define **K_USHORT** uint16_t
- #define **K_SHORT** int16_t
- #define **K_ULONG** uint32_t
- #define **K_LONG** int32_t
- #define **K_ADDR** uint32_t
- #define **K_WORD** uint32_t

Typedefs

- `typedef void(* panic_func_t)(K_USHORT usPanicCode_)`

Enumerations

- enum **EventFlagOperation_t** {
EVENT_FLAG_ALL, **EVENT_FLAG_ANY**, **EVENT_FLAG_ALL_CLEAR**, **EVENT_FLAG_ANY_CLEAR**,
EVENT_FLAG_MODES, **EVENT_FLAG_PENDING_UNBLOCK**}

15.89.1 Detailed Description

Basic data type primitives used throughout the OS.

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

15.90 kerneltypes.h

[illegible]

```

00013 =====*/
00019 #include <stdint.h>
00020
00021 #ifndef __KERNELTYPES_H__
00022 #define __KERNELTYPES_H__
00023
00024 #if defined(bool)
00025     #define K_BOOL          bool
00026 #else
00027     #define K_BOOL          uint8_t
00028 #endif
00029
00030 #define K_CHAR              char
00031 #define K_UCHAR             uint8_t
00032 #define K_USHORT            uint16_t
00033 #define K_SHORT             int16_t
00034 #define K_ULONG             uint32_t
00035 #define K_LONG              int32_t
00036
00037 #if !defined(K_ADDR)
00038     #define K_ADDR          uint32_t
00039 #endif
00040 #if !defined(K_WORD)
00041     #define K_WORD          uint32_t
00042 #endif
00043
00044 //-----
00045 typedef void (*panic_func_t)( K_USHORT usPanicCode_ );
00046
00047 //-----
00048 typedef enum
00049 {
00050     EVENT_FLAG_ALL,
00051     EVENT_FLAG_ANY,
00052     EVENT_FLAG_ALL_CLEAR,
00053     EVENT_FLAG_ANY_CLEAR,
00054     EVENT_FLAG_MODES,
00055     EVENT_FLAG_PENDING_UNBLOCK
00056 } EventFlagOperation_t;
00057
00058
00059 #endif

```

15.91 /home/moslevin/Project/R1/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 }

```

15.91.1 Detailed Description

Standard ASCII keyboard codes.

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

15.92 keycodes.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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__

```

[illegible]

```

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

```

15.95 /home/moslevin/Project/R1/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)

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ksemaphore.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_)`

15.97.1 Detailed Description

[Semaphore](#) Blocking-Object Implementation.

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

15.98 ksemaphore.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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 "ksemaphore.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_TIMEOUTS
00037 #include "timerlist.h"
00038
00039 //-----
00040 void TimedSemaphore_Callback(Thread *pclOwner_, void *pvData_)
00041 {
00042     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_);
00043
00044     // Indicate that the semaphore has expired on the thread
00045     pclOwner_->SetExpired(true);
00046
00047     // Wake up the thread that was blocked on this semaphore.
00048     pclSemaphore->WakeMe(pclOwner_);
00049
00050     if (pclOwner_->GetPriority() > Scheduler::GetCurrentThread()->
        GetPriority())
00051     {
00052         Thread::Yield();
00053     }
```



```

00054 }
00055
00056 //-----
00057 void Semaphore::WakeMe(Thread *pclChosenOne_)
00058 {
00059     // Remove from the semaphore waitlist and back to its ready list.
00060     Unblock(pclChosenOne_);
00061 }
00062
00063 #endif // KERNEL_USE_TIMEOUTS
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
00092     m_clBlockList.Init();
00093 }
00094
00095 //-----
00096 bool Semaphore::Post()
00097 {
00098     KERNEL_TRACE_1( STR_SEMAPHORE_POST_1, (K_USHORT)g_pstCurrent->GetID() );
00099
00100     bool bThreadWake = 0;
00101     K_BOOL bBail = false;
00102     // Increment the semaphore count - we can mess with threads so ensure this
00103     // is in a critical section. We don't just disable the scheduler since
00104     // we want to be able to do this from within an interrupt context as well.
00105     CS_ENTER();
00106
00107     // If nothing is waiting for the semaphore
00108     if (m_clBlockList.GetHead() == NULL)
00109     {
00110         // Check so see if we've reached the maximum value in the semaphore
00111         if (m_usValue < m_usMaxValue)
00112         {
00113             // Increment the count value
00114             m_usValue++;
00115         }
00116         else
00117         {
00118             // Maximum value has been reached, bail out.
00119             bBail = true;
00120         }
00121     }
00122     else
00123     {
00124         // Otherwise, there are threads waiting for the semaphore to be
00125         // posted, so wake the next one (highest priority goes first).
00126         bThreadWake = WakeNext();
00127     }
00128
00129     CS_EXIT();
00130
00131     // If we weren't able to increment the semaphore count, fail out.
00132     if (bBail)
00133     {
00134         return false;
00135     }
00136
00137     // if bThreadWake was set, it means that a higher-priority thread was
00138     // woken. Trigger a context switch to ensure that this thread gets
00139     // to execute next.

```

```

00140     if (bThreadWake)
00141     {
00142         Thread::Yield();
00143     }
00144     return true;
00145 }
00146
00147 //-----
00148 #if KERNEL_USE_TIMEOUTS
00149 bool Semaphore::Pend_i( K_ULONG ulWaitTimeMS_ )
00150 #else
00151 void Semaphore::Pend_i( void )
00152 #endif
00153 {
00154     KERNEL_TRACE_1( STR_SEMAPHORE_PEND_1, (K_USHORT)g_pstCurrent->GetID() );
00155
00156 #if KERNEL_USE_TIMEOUTS
00157     Timer clSemTimer;
00158     bool bUseTimer = false;
00159 #endif
00160
00161     // Once again, messing with thread data - ensure
00162     // we're doing all of these operations from within a thread-safe context.
00163     CS_ENTER();
00164
00165     // Check to see if we need to take any action based on the semaphore count
00166     if (m_usValue != 0)
00167     {
00168         // The semaphore count is non-zero, we can just decrement the count
00169         // and go along our merry way.
00170         m_usValue--;
00171     }
00172     else
00173     {
00174         // The semaphore count is zero - we need to block the current thread
00175         // and wait until the semaphore is posted from elsewhere.
00176 #if KERNEL_USE_TIMEOUTS
00177         if (ulWaitTimeMS_)
00178         {
00179             g_pstCurrent->SetExpired(false);
00180             clSemTimer.Init();
00181             clSemTimer.Start(0, ulWaitTimeMS_, TimedSemaphore_Callback, (void*)this);
00182             bUseTimer = true;
00183         }
00184 #endif
00185         Block(g_pstCurrent);
00186
00187         // Switch Threads immediately
00188         Thread::Yield();
00189     }
00190
00191     CS_EXIT();
00192
00193 #if KERNEL_USE_TIMEOUTS
00194     if (bUseTimer)
00195     {
00196         clSemTimer.Stop();
00197         return (g_pstCurrent->GetExpired() == 0);
00198     }
00199     return true;
00200 #endif
00201 }
00202
00203 //-----
00204 // Redirect the untimed pend API to the timed pend, with a null timeout.
00205 void Semaphore::Pend()
00206 {
00207     #if KERNEL_USE_TIMEOUTS
00208         Pend_i(0);
00209     #else
00210         Pend_i();
00211     #endif
00212 }
00213
00214 #if KERNEL_USE_TIMEOUTS
00215 //-----
00216 bool Semaphore::Pend( K_ULONG ulWaitTimeMS_ )
00217 {
00218     return Pend_i( ulWaitTimeMS_ );
00219 }
00220 #endif
00221
00222 //-----
00223 K_USHORT Semaphore::GetCount()
00224 {
00225     K_USHORT usRet;
00226     CS_ENTER();

```

```

00227     usRet = m_usValue;
00228     CS_EXIT();
00229     return usRet;
00230 }
00231
00232 #endif

```

15.99 /home/moslevin/Project/R1/stage/src/ksemaphore.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.

15.99.1 Detailed Description

Semaphore Blocking Object class declarations.

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

15.100 ksemaphore.h

```

00001 /*=====
00002
00003  _____
00004  |   /   \   |   /   \   |   /   \   |   /   \   |   /   \   |
00005  |  /     \  |  /     \  |  /     \  |  /     \  |  /     \  |
00006  | /       \ | /       \ | /       \ | /       \ | /       \ |
00007  | \       / | \       / | \       / | \       / | \       / |
00008  |  \     /  |  \     /  |  \     /  |  \     /  |  \     /  |
00009  --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #ifndef __KSEMAPHORE_H__
00023 #define __KSEMAPHORE_H__
00024
00025 #include "kerneltypes.h"
00026 #include "mark3cfg.h"
00027
00028 #include "blocking.h"
00029 #include "threadlist.h"
00030
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
00059     bool Post();
00060
00067     void Pend();
00068
00069
00081     K_USHORT GetCount();
00082
00083 #if KERNEL_USE_TIMEOUTS
00084

```



```

00030 #endif
00031 #define __FILE_ID__      LL_CPP
00032
00033 //-----
00034 void LinkListNode::ClearNode()
00035 {
00036     next = NULL;
00037     prev = NULL;
00038 }
00039
00040 //-----
00041 void DoubleLinkedList::Add(LinkListNode *node_)
00042 {
00043     KERNEL_ASSERT( node_ );
00044
00045     // Add a node to the end of the linked list.
00046     if (!m_pstHead)
00047     {
00048         // If the list is empty, initilize the nodes
00049         m_pstHead = node_;
00050         m_pstTail = node_;
00051
00052         m_pstHead->prev = NULL;
00053         m_pstTail->next = NULL;
00054         return;
00055     }
00056
00057     // Move the tail node, and assign it to the new node just passed in
00058     m_pstTail->next = node_;
00059     node_->prev = m_pstTail;
00060     node_->next = NULL;
00061     m_pstTail = node_;
00062 }
00063
00064 //-----
00065 void DoubleLinkedList::Remove(LinkListNode *node_)
00066 {
00067     KERNEL_ASSERT( node_ );
00068
00069     if (node_->prev)
00070     {
00071         #if SAFE_UNLINK
00072             if (node_->prev->next != node_)
00073             {
00074                 Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00075             }
00076         #endif
00077         node_->prev->next = node_->next;
00078     }
00079     if (node_->next)
00080     {
00081         #if SAFE_UNLINK
00082             if (node_->next->prev != node_)
00083             {
00084                 Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00085             }
00086         #endif
00087         node_->next->prev = node_->prev;
00088     }
00089     if (node_ == m_pstHead)
00090     {
00091         m_pstHead = node_->next;
00092     }
00093     if (node_ == m_pstTail)
00094     {
00095         m_pstTail = node_->prev;
00096     }
00097
00098     node_->ClearNode();
00099 }
00100
00101 //-----
00102 void CircularLinkedList::Add(LinkListNode *node_)
00103 {
00104     KERNEL_ASSERT( node_ );
00105
00106     // Add a node to the end of the linked list.
00107     if (!m_pstHead)
00108     {
00109         // If the list is empty, initilize the nodes
00110         m_pstHead = node_;
00111         m_pstTail = node_;
00112
00113         m_pstHead->prev = m_pstHead;
00114         m_pstHead->next = m_pstHead;
00115         return;
00116     }

```

```

00117
00118 // Move the tail node, and assign it to the new node just passed in
00119 m_pstTail->next = node_;
00120 node_>prev = m_pstTail;
00121 node_>next = m_pstHead;
00122 m_pstTail = node_;
00123 m_pstHead->prev = node_;
00124 }
00125
00126 //-----
00127 void CircularLinkedList::Remove(LinkListNode *node_)
00128 {
00129     KERNEL_ASSERT( node_ );
00130
00131     // Check to see if this is the head of the list...
00132     if ((node_ == m_pstHead) && (m_pstHead == m_pstTail))
00133     {
00134         // Clear the head and tail pointers - nothing else left.
00135         m_pstHead = NULL;
00136         m_pstTail = NULL;
00137         return;
00138     }
00139
00140 #if SAFE_UNLINK
00141     // Verify that all nodes are properly connected
00142     if ((node_>prev->next != node_) || (node_>next->prev != node_))
00143     {
00144         Kernel::Panic(PANIC_LIST_UNLINK_FAILED);
00145     }
00146 #endif
00147
00148     // This is a circularly linked list - no need to check for connection,
00149     // just remove the node.
00150     node_>next->prev = node_>prev;
00151     node_>prev->next = node_>next;
00152
00153     if (node_ == m_pstHead)
00154     {
00155         m_pstHead = m_pstHead->next;
00156     }
00157     if (node_ == m_pstTail)
00158     {
00159         m_pstTail = m_pstTail->prev;
00160     }
00161     node_>ClearNode();
00162 }
00163
00164 //-----
00165 void CircularLinkedList::PivotForward()
00166 {
00167     if (m_pstHead)
00168     {
00169         m_pstHead = m_pstHead->next;
00170         m_pstTail = m_pstTail->next;
00171     }
00172 }
00173
00174 //-----
00175 void CircularLinkedList::PivotBackward()
00176 {
00177     if (m_pstHead)
00178     {
00179         m_pstHead = m_pstHead->prev;
00180         m_pstTail = m_pstTail->prev;
00181     }
00182 }

```

15.103 /home/moslevin/Project/R1/stage/src/ll.h File Reference

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

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

Classes

- class [LinkListNode](#)

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)`

15.103.1 Detailed Description

Core linked-list declarations, used by all kernel list types. At the heart of RTOS data structures are linked lists. Having a robust and efficient set of linked-list types that we can use as a foundation for building the rest of our kernel types allows us to keep our RTOS code efficient and logically-separated.

So what data types rely on these linked-list classes?

-Threads -ThreadLists -The [Scheduler](#) -Timers, -The [Timer Scheduler](#) -Blocking objects (Semaphores, Mutexes, etc...)

Pretty much everything in the kernel uses these linked lists. By having objects inherit from the base linked-list node type, we're able to leverage the double and circular linked-list classes to manager virtually every object type in the system without duplicating code. These functions are very efficient as well, allowing for very deterministic behavior in our code.

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

15.104 ll.h

```

00001  /*=====
00002
00003  _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00006  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00007  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00043  #ifndef __LL_H__
00044  #define __LL_H__
00045
00046  #include "kerneltypes.h"
00047
00048  //-----
00049  #ifndef NULL
00050  #define NULL          (0)
00051  #endif
00052
00053  //-----
00059  class LinkedList;
00060  class DoubleLinkedList;
00061  class CircularLinkedList;
00062
00063  //-----
00068  class LinkedListNode
00069  {
00070  protected:
00071
00072      LinkedListNode *next;
00073      LinkedListNode *prev;
00074
00075      LinkedListNode() { }
00076

```

```

00082     void ClearNode();
00083
00084 public:
00092     LinkListNode *GetNext(void) { return next; }
00093
00101     LinkListNode *GetPrev(void) { return prev; }
00102
00103     friend class LinkList;
00104     friend class DoubleLinkList;
00105     friend class CircularLinkList;
00106 };
00107
00108 //-----
00112 class LinkList
00113 {
00114 protected:
00115     LinkListNode *m_pstHead;
00116     LinkListNode *m_pstTail;
00117
00118 public:
00122     void Init(){ m_pstHead = NULL; m_pstTail = NULL; }
00123
00131     virtual void Add(LinkListNode *node_) = 0;
00132
00140     virtual void Remove(LinkListNode *node_) = 0;
00141
00149     LinkListNode *GetHead() { return m_pstHead; }
00150
00158     LinkListNode *GetTail() { return m_pstTail; }
00159 };
00160
00161 //-----
00165 class DoubleLinkList : public LinkList
00166 {
00167 public:
00171     DoubleLinkList() { m_pstHead = NULL; m_pstTail = NULL; }
00172
00180     virtual void Add(LinkListNode *node_);
00181
00189     virtual void Remove(LinkListNode *node_);
00190 };
00191
00192 //-----
00196 class CircularLinkList : public LinkList
00197 {
00198 public:
00199     CircularLinkList() { m_pstHead = NULL; m_pstTail = NULL; }
00200
00208     virtual void Add(LinkListNode *node_);
00209
00217     virtual void Remove(LinkListNode *node_);
00218
00225     void PivotForward();
00226
00233     void PivotBackward();
00234 };
00235
00236 #endif

```

15.105 /home/moslevin/Project/R1/stage/src/manual.h File Reference

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

15.105.1 Detailed Description

Ascii-format documentation, used by doxygen to create various printable and viewable forms.

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

15.106 manual.h

```

00001 /*=====
00002
00003  _ _ _ _ _
00004  | | | | |

```


15.107 /home/moslevin/Project/R1/stage/src/mark3cfg.h File Reference

Macros

- Generated on Thu Mar 5 2015 21:04:26 for Mark3 Realtime Kernel by Doxygen

Provides support for atomic operations, including addition, subtraction, set, and test-and-set.

- `#define SAFE_UNLINK (0)`

"Safe unlinking" performs extra checks on data to make sure that there are no consistencies when performing operations on linked lists.

- `#define KERNEL_AWARE_SIMULATION (1)`

Include support for kernel-aware simulation.

15.107.1 Detailed Description

Mark3 [Kernel](#) Configuration. This file is used to configure the kernel for your specific application in order to provide the optimal set of features for a given use case.

Since you only pay the price (code space/RAM) for the features you use, you can usually find a sweet spot between features and resource usage by picking and choosing features a-la-carte. This config file is written in an "interactive" way, in order to minimize confusion about what each option provides, and to make dependencies obvious.

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

15.107.2 Macro Definition Documentation

15.107.2.1 `#define GLOBAL_MESSAGE_POOL_SIZE (8)`

If Messages are enabled, define the size of the default kernel message pool.

Messages can be manually added to the message pool, but this mechanism is more convenient and automatic. All message queues share their message objects from this global pool to maximize efficiency and simplify data management.

Definition at line 144 of file [mark3cfg.h](#).

15.107.2.2 `#define KERNEL_AWARE_SIMULATION (1)`

Include support for kernel-aware simulation.

Enabling this feature adds advanced profiling, trace, and environment-aware debugging and diagnostic functionality when Mark3-based applications are run on the flavr AVR simulator.

Definition at line 210 of file [mark3cfg.h](#).

15.107.2.3 `#define KERNEL_TIMERS_TICKLESS (1)`

If you've opted to use the kernel timers module, you have an option as to which timer implementation to use: Tick-based or Tick-less.

Tick-based timers provide a "traditional" RTOS timer implementation based on a fixed-frequency timer interrupt. While this provides very accurate, reliable timing, it also means that the CPU is being interrupted far more often than may be necessary (as not all timer ticks result in "real work" being done).

Tick-less timers still rely on a hardware timer interrupt, but uses a dynamic expiry interval to ensure that the interrupt is only called when the next timer expires. This increases the complexity of the timer interrupt handler, but reduces the number and frequency.

Note that the CPU port ([kerneltimer.cpp](#)) must be implemented for the particular timer variant desired.

Definition at line 62 of file [mark3cfg.h](#).

15.107.2.4 `#define KERNEL_USE_ATOMIC (0)`

Provides support for atomic operations, including addition, subtraction, set, and test-and-set.

Add/Sub/Set contain 8, 16, and 32-bit variants.

Definition at line 194 of file [mark3cfg.h](#).

15.107.2.5 `#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 176 of file [mark3cfg.h](#).

15.107.2.6 `#define KERNEL_USE_EVENTFLAG (1)`

Provides additional event-flag based blocking.

This relies on an additional per-thread flag-mask to be allocated, which adds 2 bytes to the size of each thread object.

Definition at line 123 of file [mark3cfg.h](#).

15.107.2.7 `#define KERNEL_USE_MESSAGE (1)`

Enable inter-thread messaging using message queues.

This is the preferred mechanism for IPC for serious multi-threaded communications; generally anywhere a semaphore or event-flag is insufficient.

Definition at line 131 of file [mark3cfg.h](#).

15.107.2.8 `#define KERNEL_USE_PROFILER (1)`

Provides extra classes for profiling the performance of code.

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

Definition at line 182 of file [mark3cfg.h](#).

15.107.2.9 `#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 92 of file [mark3cfg.h](#).

15.107.2.10 `#define KERNEL_USE_SEMAPHORE (1)`

Do you want the ability to use counting/binary semaphores for thread synchronization? Enabling this features provides fully-blocking semaphores and enables all API functions declared in [semaphore.h](#).

If you have to pick one blocking mechanism, this is the one to choose.

Definition at line 109 of file [mark3cfg.h](#).


```

00013 =====*/
00029 #ifndef __MARK3CFG_H__
00030 #define __MARK3CFG_H__
00031
00041 #define KERNEL_USE_TIMERS (1)
00042
00061 #if KERNEL_USE_TIMERS
00062     #define KERNEL_TIMERS_TICKLESS (1)
00063 #endif
00064
00076 #if KERNEL_USE_TIMERS
00077     #define KERNEL_USE_TIMEOUTS (1)
00078 #else
00079     #define KERNEL_USE_TIMEOUTS (0)
00080 #endif
00081
00091 #if KERNEL_USE_TIMERS
00092     #define KERNEL_USE_QUANTUM (1)
00093 #else
00094     #define KERNEL_USE_QUANTUM (0)
00095 #endif
00096
00101 #define THREAD_QUANTUM_DEFAULT (4)
00102
00109 #define KERNEL_USE_SEMAPHORE (1)
00110
00116 #define KERNEL_USE_MUTEX (1)
00117
00123 #define KERNEL_USE_EVENTFLAG (1)
00124
00130 #if KERNEL_USE_SEMAPHORE
00131     #define KERNEL_USE_MESSAGE (1)
00132 #else
00133     #define KERNEL_USE_MESSAGE (0)
00134 #endif
00135
00143 #if KERNEL_USE_MESSAGE
00144     #define GLOBAL_MESSAGE_POOL_SIZE (8)
00145 #endif
00146
00151 #if KERNEL_USE_TIMERS && KERNEL_USE_SEMAPHORE
00152     #define KERNEL_USE_SLEEP (1)
00153 #else
00154     #define KERNEL_USE_SLEEP (0)
00155 #endif
00156
00161 #define KERNEL_USE_DRIVER (1)
00162
00168 #define KERNEL_USE_THREADNAME (0)
00169
00176 #define KERNEL_USE_DYNAMIC_THREADS (1)
00177
00182 #define KERNEL_USE_PROFILER (1)
00183
00188 #define KERNEL_USE_DEBUG (0)
00189
00194 #define KERNEL_USE_ATOMIC (0)
00195
00202 #define SAFE_UNLINK (0)
00203
00210 #define KERNEL_AWARE_SIMULATION (1)
00211
00212 #endif

```

15.109 /home/moslevin/Project/R1/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"

```

15.109.1 Detailed Description

Implementation of memory, string, and conversion routines.


```

00088     }
00089 }
00090
00091 //-----
00092 void MemUtil::DecimalToHex( K_ULONG ulData_, char *szText_ )
00093 {
00094     K_ULONG ulTmp = ulData_;
00095     K_ULONG ulMax = 1;
00096     K_ULONG ulCompare = 0x0010;
00097
00098     KERNEL_ASSERT( szText_ );
00099
00100     while (ulData_ > ulCompare && ulMax < 8)
00101     {
00102         ulMax++;
00103         ulCompare <= 4;
00104     }
00105
00106     ulTmp = ulData_;
00107     szText_[ulMax] = 0;
00108     while (ulMax--)
00109     {
00110         if ((ulTmp & 0x0F) <= 9)
00111         {
00112             szText_[ulMax] = '0' + (ulTmp & 0x0F);
00113         }
00114         else
00115         {
00116             szText_[ulMax] = 'A' + ((ulTmp & 0x0F) - 10);
00117         }
00118         ulTmp >>= 4;
00119     }
00120 }
00121 //-----
00122 void MemUtil::DecimalToString( K_UCHAR ucData_, char *szText_ )
00123 {
00124     K_UCHAR ucTmp = ucData_;
00125     K_UCHAR ucMax;
00126
00127     KERNEL_ASSERT(szText_);
00128
00129     // Find max index to print...
00130     if (ucData_ >= 100)
00131     {
00132         ucMax = 3;
00133     }
00134     else if (ucData_ >= 10)
00135     {
00136         ucMax = 2;
00137     }
00138     else
00139     {
00140         ucMax = 1;
00141     }
00142
00143     szText_[ucMax] = 0;
00144     while (ucMax--)
00145     {
00146         szText_[ucMax] = '0' + (ucTmp % 10);
00147         ucTmp /= 10;
00148     }
00149 }
00150
00151 //-----
00152 void MemUtil::DecimalToString( K_USHORT usData_, char *szText_ )
00153 {
00154     K_USHORT usTmp = usData_;
00155     K_USHORT usMax = 1;
00156     K_USHORT usCompare = 10;
00157
00158     KERNEL_ASSERT(szText_);
00159
00160     while (usData_ >= usCompare && usMax < 5)
00161     {
00162         usCompare *= 10;
00163         usMax++;
00164     }
00165
00166     szText_[usMax] = 0;
00167     while (usMax--)
00168     {
00169         szText_[usMax] = '0' + (usTmp % 10);
00170         usTmp /= 10;
00171     }
00172 }
00173
00174 //-----

```

```

00175 void MemUtil::DecimalToString( K_ULONG ulData_, char *szText_ )
00176 {
00177     K_ULONG ulTmp = ulData_;
00178     K_ULONG ulMax = 1;
00179     K_ULONG ulCompare = 10;
00180
00181     KERNEL_ASSERT(szText_);
00182
00183     while (ulData_ >= ulCompare && ulMax < 12)
00184     {
00185         ulCompare *= 10;
00186         ulMax++;
00187     }
00188
00189     szText_[ulMax] = 0;
00190     while (ulMax--)
00191     {
00192         szText_[ulMax] = '0' + (ulTmp % 10);
00193         ulTmp/=10;
00194     }
00195 }
00196
00197 //-----
00198 // Basic checksum routines
00199 K_UCHAR MemUtil::Checksum8( const void *pvSrc_, K_USHORT usLen_ )
00200 {
00201     K_UCHAR ucRet = 0;
00202     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00203
00204     KERNEL_ASSERT(pvSrc_);
00205
00206     // 8-bit CRC, computed byte at a time
00207     while (usLen_--)
00208     {
00209         ucRet += *pcData++;
00210     }
00211     return ucRet;
00212 }
00213
00214 //-----
00215 K_USHORT MemUtil::Checksum16( const void *pvSrc_, K_USHORT usLen_ )
00216 {
00217     K_USHORT usRet = 0;
00218     K_UCHAR *pcData = (K_UCHAR*)pvSrc_;
00219
00220     KERNEL_ASSERT(pvSrc_);
00221
00222     // 16-bit CRC, computed byte at a time
00223     while (usLen_--)
00224     {
00225         usRet += *pcData++;
00226     }
00227     return usRet;
00228 }
00229
00230 //-----
00231 // Basic string routines
00232 K_USHORT MemUtil::StringLength( const char *szStr_ )
00233 {
00234     K_UCHAR *pcData = (K_UCHAR*)szStr_;
00235     K_USHORT usLen = 0;
00236
00237     KERNEL_ASSERT(szStr_);
00238
00239     while (*pcData++)
00240     {
00241         usLen++;
00242     }
00243     return usLen;
00244 }
00245
00246 //-----
00247 bool MemUtil::CompareStrings( const char *szStr1_, const char *szStr2_ )
00248 {
00249     char *szTmp1 = (char*) szStr1_;
00250     char *szTmp2 = (char*) szStr2_;
00251
00252     KERNEL_ASSERT(szStr1_);
00253     KERNEL_ASSERT(szStr2_);
00254
00255     while (*szTmp1 && *szTmp2)
00256     {
00257         if (*szTmp1++ != *szTmp2++)
00258         {
00259             return false;
00260         }
00261     }

```



```

00262
00263 // Both terminate at the same length
00264 if (!(*szTmp1) && !(*szTmp2))
00265 {
00266     return true;
00267 }
00268
00269 return false;
00270 }
00271
00272 //-----
00273 void MemUtil::CopyMemory( void *pvDst_, const void *pvSrc_, K_USHORT usLen_ )
00274 {
00275     char *szDst = (char*) pvDst_;
00276     char *szSrc = (char*) pvSrc_;
00277
00278     KERNEL_ASSERT(pvDst_);
00279     KERNEL_ASSERT(pvSrc_);
00280
00281     // Run through the strings verifying that each character matches
00282     // and the lengths are the same.
00283     while (usLen_--)
00284     {
00285         *szDst++ = *szSrc++;
00286     }
00287 }
00288
00289 //-----
00290 void MemUtil::CopyString( char *szDst_, const char *szSrc_ )
00291 {
00292     char *szDst = (char*) szDst_;
00293     char *szSrc = (char*) szSrc_;
00294
00295     KERNEL_ASSERT(szDst_);
00296     KERNEL_ASSERT(szSrc_);
00297
00298     // Run through the strings verifying that each character matches
00299     // and the lengths are the same.
00300     while (*szSrc)
00301     {
00302         *szDst++ = *szSrc++;
00303     }
00304 }
00305
00306 //-----
00307 K_SHORT MemUtil::StringSearch( const char *szBuffer_, const char *szPattern_ )
00308 {
00309     char *szTmpPat = (char*)szPattern_;
00310     K_SHORT i16Idx = 0;
00311     K_SHORT i16Start;
00312     KERNEL_ASSERT( szBuffer_ );
00313     KERNEL_ASSERT( szPattern_ );
00314
00315     // Run through the big buffer looking for a match of the pattern
00316     while (szBuffer_[i16Idx])
00317     {
00318         // Reload the pattern
00319         i16Start = i16Idx;
00320         szTmpPat = (char*)szPattern_;
00321         while (*szTmpPat && szBuffer_[i16Idx])
00322         {
00323             if (*szTmpPat != szBuffer_[i16Idx])
00324             {
00325                 break;
00326             }
00327             szTmpPat++;
00328             i16Idx++;
00329         }
00330         // Made it to the end of the pattern, it's a match.
00331         if (*szTmpPat == '\0')
00332         {
00333             return i16Start;
00334         }
00335         i16Idx++;
00336     }
00337
00338     return -1;
00339 }
00340
00341 //-----
00342 bool MemUtil::CompareMemory( const void *pvMem1_, const void *pvMem2_, K_USHORT
usLen_ )
00343 {
00344     char *szTmp1 = (char*) pvMem1_;
00345     char *szTmp2 = (char*) pvMem2_;
00346
00347     KERNEL_ASSERT(pvMem1_);

```

```

00348     KERNEL_ASSERT(pvMem2_);
00349
00350     // Run through the strings verifying that each character matches
00351     // and the lengths are the same.
00352     while (usLen_--)
00353     {
00354         if (*szTmp1++ != *szTmp2++)
00355         {
00356             return false;
00357         }
00358     }
00359     return true;
00360 }
00361
00362 //-----
00363 void MemUtil::SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_ )
00364 {
00365     char *szDst = (char*)pvDst_;
00366
00367     KERNEL_ASSERT(pvDst_);
00368
00369     while (usLen_--)
00370     {
00371         *szDst++ = ucVal_;
00372     }
00373 }
00374
00375 //-----
00376 K_UCHAR MemUtil::Tokenize( const K_CHAR *szBuffer_, Token_t *pastTokens_, K_UCHAR
ucMaxTokens_ )
00377 {
00378     K_UCHAR ucCurrArg = 0;
00379     K_UCHAR ucLastArg = 0;
00380     K_UCHAR i = 0;
00381
00382     bool bEscape = false;
00383
00384     KERNEL_ASSERT(szBuffer_);
00385     KERNEL_ASSERT(pastTokens_);
00386
00387     while (szBuffer_[i])
00388     {
00389         //-- Handle unescaped quotes
00390         if (szBuffer_[i] == '\"')
00391         {
00392             if (bEscape)
00393             {
00394                 bEscape = false;
00395             }
00396             else
00397             {
00398                 bEscape = true;
00399             }
00400             i++;
00401             continue;
00402         }
00403
00404         //-- Handle all escaped chars - by ignoring them
00405         if (szBuffer_[i] == '\\')
00406         {
00407             i++;
00408             if (szBuffer_[i])
00409             {
00410                 i++;
00411             }
00412             continue;
00413         }
00414
00415         //-- Process chars based on current escape characters
00416         if (bEscape)
00417         {
00418             // Everything within the quote is treated as literal, but escaped chars are still treated the
00419             same
00420             i++;
00421             continue;
00422         }
00423
00424         //-- Non-escaped case
00425         if (szBuffer_[i] != ' ' )
00426         {
00427             i++;
00428             continue;
00429         }
00430
00431         pastTokens_[ucCurrArg].pcToken = &(szBuffer_[ucLastArg]);
00432         pastTokens_[ucCurrArg].ucLen = i - ucLastArg;
00433         ucCurrArg++;

```

```

00433         if (ucCurrArg >= ucMaxTokens_)
00434         {
00435             return ucMaxTokens_;
00436         }
00437
00438         i++;
00439         while (szBuffer_[i] && szBuffer_[i] == ' ')
00440         {
00441             i++;
00442         }
00443
00444         ucLastArg = i;
00445     }
00446     if (i && !szBuffer_[i] && (i - ucLastArg))
00447     {
00448         pastTokens_[ucCurrArg].pcToken = &(szBuffer_[ucLastArg]);
00449         pastTokens_[ucCurrArg].ucLen = i - ucLastArg;
00450         ucCurrArg++;
00451     }
00452     return ucCurrArg;
00453 }
00454
00455

```

15.111 /home/moslevin/Project/R1/stage/src/memutil.h File Reference

Utility class containing memory, string, and conversion routines.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "kernel_debug.h"

```

Classes

- struct [Token_t](#)
Token descriptor struct format.
- class [MemUtil](#)
String and Memory manipulation class.

15.111.1 Detailed Description

Utility class containing memory, string, and conversion routines.

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

15.112 memutil.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #ifndef __MEMUTIL_H__
00022 #define __MEMUTIL_H__
00023
00024 #include "kerneltypes.h"
00025 #include "mark3cfg.h"
00026 #include "kernel_debug.h"
00027

```

```

00028 //-----
00032 typedef struct
00033 {
00034     const K_CHAR *pcToken;
00035     K_UCHAR ucLen;
00036 } Token_t;
00037
00038 //-----
00047 class MemUtil
00048 {
00049
00050 public:
00051
00052     //-----
00061     static void DecimalToHex( K_UCHAR ucData_, char *szText_ );
00062     static void DecimalToHex( K_USHORT usData_, char *szText_ );
00063     static void DecimalToHex( K_ULONG ulData_, char *szText_ );
00064
00065     //-----
00074     static void DecimalToString( K_UCHAR ucData_, char *szText_ );
00075     static void DecimalToString( K_USHORT usData_, char *szText_ );
00076     static void DecimalToString( K_ULONG ulData_, char *szText_ );
00077
00078     //-----
00088     static K_UCHAR Checksum8( const void *pvSrc_, K_USHORT usLen_ );
00089
00090     //-----
00100     static K_USHORT Checksum16( const void *pvSrc_, K_USHORT usLen_ );
00101
00102     //-----
00112     static K_USHORT StringLength( const char *szStr_ );
00113
00114     //-----
00124     static bool CompareStrings( const char *szStr1_, const char *szStr2_ );
00125
00126     //-----
00136     static void CopyMemory( void *pvDst_, const void *pvSrc_, K_USHORT usLen_ );
00137
00138     //-----
00147     static void CopyString( char *szDst_, const char *szSrc_ );
00148
00149     //-----
00159     static K_SHORT StringSearch( const char *szBuffer_, const char *szPattern_ );
00160
00161     //-----
00173     static bool CompareMemory( const void *pvMem1_, const void *pvMem2_, K_USHORT usLen_ );
00174
00175     //-----
00185     static void SetMemory( void *pvDst_, K_UCHAR ucVal_, K_USHORT usLen_ );
00186
00187     //-----
00197     static K_UCHAR Tokenize( const char *szBuffer_, Token_t *pastTokens_, K_UCHAR
ucMaxTokens_ );
00198 };
00199
00200
00201 #endif //__MEMUTIL_H__
00202
00203
00204
00205

```

15.113 /home/moslevin/Project/R1/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`

15.113.1 Detailed Description

Inter-thread communications via message passing.

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

15.114 message.cpp

```

00001  /*=====
00002
00003  _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00006  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00007  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 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_TIMEOUTS
00039      #include "timerlist.h"
00040  #endif
00041
00042  Message GlobalMessagePool::m_aclMessagePool[8];
00043  DoubleLinkedList GlobalMessagePool::m_clList;
00044
00045  //-----
00046  void GlobalMessagePool::Init()
00047  {
00048      K_UCHAR i;
00049      GlobalMessagePool::m_clList.Init();
00050      for (i = 0; i < GLOBAL_MESSAGE_POOL_SIZE; i++)
00051      {
00052          GlobalMessagePool::m_aclMessagePool[i].Init();
00053          GlobalMessagePool::m_clList.Add(&(GlobalMessagePool::m_aclMessagePool[i]));
00054      }
00055  }
00056
00057  //-----
00058  void GlobalMessagePool::Push( Message *pclMessage_ )
00059  {
00060      KERNEL_ASSERT( pclMessage_ );
00061
00062      CS_ENTER();
00063
00064      GlobalMessagePool::m_clList.Add(pclMessage_);
00065
00066      CS_EXIT();
00067  }
00068
00069  //-----
00070  Message *GlobalMessagePool::Pop()
00071  {
00072      Message *pclRet;
00073      CS_ENTER();
00074
00075      pclRet = static_cast<Message*>( GlobalMessagePool::m_clList.GetHead() );

```

```

00076     if (0 != pclRet)
00077     {
00078         GlobalMessagePool::m_clList.Remove( static_cast<LinkListNode*>( pclRet ) );
00079     }
00080
00081     CS_EXIT();
00082     return pclRet;
00083 }
00084
00085 //-----
00086 void MessageQueue::Init()
00087 {
00088     m_clSemaphore.Init(0, GLOBAL_MESSAGE_POOL_SIZE);
00089 }
00090
00091 //-----
00092 Message *MessageQueue::Receive()
00093 {
00094     #if KERNEL_USE_TIMEOUTS
00095         return Receive_i(0);
00096     #else
00097         return Receive_i();
00098     #endif
00099 }
00100
00101 //-----
00102 #if KERNEL_USE_TIMEOUTS
00103 Message *MessageQueue::Receive( K_ULONG ulTimeWaitMS_ )
00104 {
00105     return Receive_i( ulTimeWaitMS_ );
00106 }
00107 #endif
00108
00109 //-----
00110 #if KERNEL_USE_TIMEOUTS
00111 Message *MessageQueue::Receive_i( K_ULONG ulTimeWaitMS_ )
00112 #else
00113 Message *MessageQueue::Receive_i( void )
00114 #endif
00115 {
00116     Message *pclRet;
00117
00118     // Block the current thread on the counting semaphore
00119     #if KERNEL_USE_TIMEOUTS
00120     if (!m_clSemaphore.Pend(ulTimeWaitMS_))
00121     {
00122         return NULL;
00123     }
00124     #else
00125     m_clSemaphore.Pend();
00126     #endif
00127
00128     CS_ENTER();
00129
00130     // Pop the head of the message queue and return it
00131     pclRet = static_cast<Message*>( m_clLinkList.GetHead() );
00132     m_clLinkList.Remove( static_cast<Message*>(pclRet) );
00133
00134     CS_EXIT();
00135
00136     return pclRet;
00137 }
00138
00139 //-----
00140 void MessageQueue::Send( Message *pclSrc_ )
00141 {
00142     KERNEL_ASSERT( pclSrc_ );
00143
00144     CS_ENTER();
00145
00146     // Add the message to the head of the linked list
00147     m_clLinkList.Add( pclSrc_ );
00148
00149     // Post the semaphore, waking the blocking thread for the queue.
00150     m_clSemaphore.Post();
00151
00152     CS_EXIT();
00153 }
00154
00155 //-----
00156 K_USHORT MessageQueue::GetCount()
00157 {
00158     return m_clSemaphore.GetCount();
00159 }
00160 #endif //KERNEL_USE_MESSAGE

```

15.115 /home/moslevin/Project/R1/stage/src/message.h File Reference

Inter-thread communication via message-passing.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "ll.h"
#include "ksemaphore.h"
#include "timerlist.h"
```

Classes

- class [Message](#)
Class to provide message-based IPC services in the kernel.
- class [GlobalMessagePool](#)
Implements a list of message objects shared between all threads.
- class [MessageQueue](#)
List of messages, used as the channel for sending and receiving messages between threads.

15.115.1 Detailed Description

Inter-thread communication via message-passing. Embedded systems guru Jack Ganssle once said that without a robust form of interprocess communications (IPC), an RTOS is just a toy. Mark3 implements a form of IPC to provide safe and flexible messaging between threads.

Using kernel-managed IPC offers significant benefits over other forms of data sharing (i.e. Global variables) in that it avoids synchronization issues and race conditions common to the practice. Using IPC also enforces a more disciplined coding style that keeps threads decoupled from one another and minimizes global data, preventing careless and hard-to-debug errors.

15.115.2 Using Messages, Queues, and the Global Message Pool

```
// Declare a message queue shared between two threads
MessageQueue my_queue;

int main()
{
    ...
    // Initialize the message queue
    my_queue.init();
    ...
}

void Thread1()
{
    // Example TX thread - sends a message every 10ms
    while(1)
    {
        // Grab a message from the global message pool
        Message *tx_message = GlobalMessagePool::Pop();

        // Set the message data/parameters
        tx_message->SetCode( 1234 );
        tx_message->SetData( NULL );

        // Send the message on the queue.
        my_queue.Send( tx_message );
        Thread::Sleep(10);
    }
}

void Thread2()
{
    while()
    {
        // Blocking receive - wait until we have messages to process
```

```

    Message *rx_message = my_queue.Recv();

    // Do something with the message data...

    // Return back into the pool when done
    GlobalMessagePool::Push(rx_message);
}
}

```

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

15.116 message.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00080 #ifndef __MESSAGE_H__
00081 #define __MESSAGE_H__
00082
00083 #include "kerneltypes.h"
00084 #include "mark3cfg.h"
00085
00086 #include "ll.h"
00087 #include "ksemaphore.h"
00088
00089 #if KERNEL_USE_MESSAGE
00090
00091 #if KERNEL_USE_TIMEOUTS
00092     #include "timerlist.h"
00093 #endif
00094
00095 //-----
00099 class Message : public LinkListNode
00100 {
00101 public:
00107     void Init() { ClearNode(); m_pvData = NULL; m_usCode = 0; }
00108
00116     void SetData( void *pvData_ ) { m_pvData = pvData_; }
00117
00125     void *GetData() { return m_pvData; }
00126
00134     void SetCode( K_USHORT usCode_ ) { m_usCode = usCode_; }
00135
00143     K_USHORT GetCode() { return m_usCode; }
00144 private:
00145
00147     void *m_pvData;
00148
00150     K_USHORT m_usCode;
00151 };
00152
00153 //-----
00157 class GlobalMessagePool
00158 {
00159 public:
00165     static void Init();
00166
00176     static void Push( Message *pclMessage_ );
00177
00186     static Message *Pop();
00187
00188 private:
00190     static Message m_aclMessagePool[
        GLOBAL_MESSAGE_POOL_SIZE];
00191
00193     static DoubleLinkedList m_clList;
00194 };
00195
00196 //-----
00201 class MessageQueue
00202 {
00203 public:

```


15.117 /home/moslevin/Project/R1/stage/src/mutex.cpp File Reference

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

- #define **__FILE_ID__** MUTEX_CPP

- void **TimedMutex_Callback** (Thread *pclOwner_, void *pvData_)

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

```

00001 /*-----
00002
00003      _ _ _ _ _
00004  | | | | | | | | | |
00005  | | | | | | | | | |
00006  | | | | | | | | | |
00007  | | | | | | | | | |
00008
00009  --[Mark3 Realtime Platform]-----

```

```

00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "kerneltypes.h"
00021 #include "mark3cfg.h"
00022
00023 #include "blocking.h"
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_TIMEOUTS
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     pclOwner_->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()->
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 priority
00078     if (pclChosenOne->GetPriority() >= Scheduler::GetCurrentThread()
->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     m_ucRecurse = 0;        // Reset recurse count
00093 }
00094
00095 //-----
00096 #if KERNEL_USE_TIMEOUTS
00097 bool Mutex::Claim_i(K_ULONG ulWaitTimeMS_)
00098 #else
00099 void Mutex::Claim_i(void)
00100 #endif

```

```

00101 {
00102     KERNEL_TRACE_1( STR_MUTEX_CLAIM_1, (K_USHORT)g_pstCurrent->GetID() );
00103
00104 #if KERNEL_USE_TIMEOUTS
00105     Timer clTimer;
00106     bool bUseTimer = false;
00107 #endif
00108
00109     // Disable the scheduler while claiming the mutex - we're dealing with all
00110     // sorts of private thread data, can't have a thread switch while messing
00111     // with internal data structures.
00112     Scheduler::SetScheduler(0);
00113
00114     // Check to see if the mutex is claimed or not
00115     if (m_bReady != 0)
00116     {
00117         // Mutex isn't claimed, claim it.
00118         m_bReady = 0;
00119         m_ucRecurse = 0;
00120         m_ucMaxPri = g_pstCurrent->GetPriority();
00121         m_pclOwner = g_pstCurrent;
00122
00123         Scheduler::SetScheduler(1);
00124
00125 #if KERNEL_USE_TIMEOUTS
00126         return true;
00127 #else
00128         return;
00129 #endif
00130     }
00131
00132     // If the mutex is already claimed, check to see if this is the owner thread,
00133     // since we allow the mutex to be claimed recursively.
00134     if (g_pstCurrent == m_pclOwner)
00135     {
00136         // Ensure that we haven't exceeded the maximum recursive-lock count
00137         KERNEL_ASSERT( (m_ucRecurse < 255) );
00138         m_ucRecurse++;
00139
00140         // Increment the lock count and bail
00141         Scheduler::SetScheduler(1);
00142 #if KERNEL_USE_TIMEOUTS
00143         return true;
00144 #else
00145         return;
00146 #endif
00147     }
00148
00149     // The mutex is claimed already - we have to block now. Move the
00150     // current thread to the list of threads waiting on the mutex.
00151 #if KERNEL_USE_TIMEOUTS
00152     if (ulWaitTimeMS_)
00153     {
00154         g_pstCurrent->SetExpired(false);
00155         clTimer.Init();
00156         clTimer.Start(0, ulWaitTimeMS_, (TimerCallback_t)TimedMutex_Callback, (void*)this);
00157         bUseTimer = true;
00158     }
00159 #endif
00160     Block(g_pstCurrent);
00161
00162     // Check if priority inheritance is necessary. We do this in order
00163     // to ensure that we don't end up with priority inversions in case
00164     // multiple threads are waiting on the same resource.
00165     if (m_ucMaxPri <= g_pstCurrent->GetPriority())
00166     {
00167         m_ucMaxPri = g_pstCurrent->GetPriority();
00168
00169         Thread *pclTemp = static_cast<Thread*>(m_clBlockList.GetHead());
00170         while (pclTemp)
00171         {
00172             pclTemp->InheritPriority(m_ucMaxPri);
00173             if (pclTemp == static_cast<Thread*>(m_clBlockList.GetTail()))
00174             {
00175                 break;
00176             }
00177             pclTemp = static_cast<Thread*>(pclTemp->GetNext());
00178         }
00179         m_pclOwner->InheritPriority(m_ucMaxPri);
00180     }
00181
00182     // Done with thread data -reenable the scheduler
00183     Scheduler::SetScheduler(1);
00184
00185     // Switch threads if this thread acquired the mutex
00186     Thread::Yield();
00187

```

```

00188 #if KERNEL_USE_TIMEOUTS
00189     if (bUseTimer)
00190     {
00191         clTimer.Stop();
00192         return (g_pstCurrent->GetExpired() == 0);
00193     }
00194     return true;
00195 #endif
00196 }
00197
00198 //-----
00199 void Mutex::Claim(void)
00200 {
00201     #if KERNEL_USE_TIMEOUTS
00202         Claim_i(0);
00203     #else
00204         Claim_i();
00205     #endif
00206 }
00207
00208 //-----
00209 #if KERNEL_USE_TIMEOUTS
00210 bool Mutex::Claim(K_ULONG ulWaitTimeMS_)
00211 {
00212     return Claim_i(ulWaitTimeMS_);
00213 }
00214 #endif
00215
00216 //-----
00217 void Mutex::Release()
00218 {
00219     KERNEL_TRACE_1( STR_MUTEX_RELEASE_1, (K_USHORT)g_pstCurrent->GetID() );
00220
00221     bool bSchedule = 0;
00222
00223     // Disable the scheduler while we deal with internal data structures.
00224     Scheduler::SetScheduler(0);
00225
00226     // This thread had better be the one that owns the mutex currently...
00227     KERNEL_ASSERT( (g_pstCurrent == m_pclOwner) );
00228
00229     // If the owner had claimed the lock multiple times, decrease the lock
00230     // count and return immediately.
00231     if (m_ucRecurse)
00232     {
00233         m_ucRecurse--;
00234         Scheduler::SetScheduler(1);
00235         return;
00236     }
00237
00238     // Restore the thread's original priority
00239     if (g_pstCurrent->GetCurPriority() != g_pstCurrent->
GetPriority())
00240     {
00241         g_pstCurrent->SetPriority(g_pstCurrent->GetPriority());
00242
00243         // In this case, we want to reschedule
00244         bSchedule = 1;
00245     }
00246
00247     // No threads are waiting on this semaphore?
00248     if (m_clBlockList.GetHead() == NULL)
00249     {
00250         // Re-initialize the mutex to its default values
00251         m_bReady = 1;
00252         m_ucMaxPri = 0;
00253         m_pclOwner = NULL;
00254     }
00255     else
00256     {
00257         // Wake the highest priority Thread pending on the mutex
00258         if(WakeNext())
00259         {
00260             // Switch threads if it's higher or equal priority than the current thread
00261             bSchedule = 1;
00262         }
00263     }
00264
00265     // Must enable the scheduler again in order to switch threads.
00266     Scheduler::SetScheduler(1);
00267     if(bSchedule)
00268     {
00269         // Switch threads if a higher-priority thread was woken
00270         Thread::Yield();
00271     }
00272 }
00273

```

```
00274 #endif //KERNEL_USE_MUTEX
```

15.119 /home/moslevin/Project/R1/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](#).

15.119.1 Detailed Description

Mutual exclusion class declaration. Resource locks are implemented using mutual exclusion semaphores (Mutex_t). Protected blocks can be placed around any resource that may only be accessed by one thread at a time. If additional threads attempt to access the protected resource, they will be placed in a wait queue until the resource becomes available. When the resource becomes available, the thread with the highest original priority claims the resource and is activated. Priority inheritance is included in the implementation to prevent priority inversion. Always ensure that you claim and release your mutex objects consistently, otherwise you may end up with a deadlock scenario that's hard to debug.

15.119.2 Initializing

Initializing a mutex object by calling:

```
clMutex.Init();
```

15.119.3 Resource protection example

```
clMutex.Claim();
...
<resource protected block>
...
clMutex.Release();
```

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

15.120 mutex.h

```
00001 /*=====
```

```
00002
```

```
00003      |_____|       |_____|       |_____|       |_____|
```

```
00004      || \ / || || \ / || || \ / || || \ / || ||
```

```
00005      || \ / || || \ / || || \ / || || \ / || ||
```

```
00006      || \ / || || \ / || || \ / || || \ / || ||
```

```
00007      |_____|       |_____|       |_____|       |_____|
```

```
00008
```

```
00009 --[Mark3 Realtime Platform]-----
```

```
00010
```

```
00011 Copyright (c) 2012-2015 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_TIMEOUTS
00061 #include "timerlist.h"
00062 #endif
00063
00064 //-----
00068 class Mutex : public BlockingObject
00069 {
00070 public:
00077     void Init();
00078
00085     void Claim();
00086
00087 #if KERNEL_USE_TIMEOUTS
00088
00097     bool Claim(K_ULONG ulWaitTimeMS_);
00098
00111     void WakeMe( Thread *pclOwner_ );
00112
00113 #endif
00114
00121     void Release();
00122
00123 private:
00124
00130     K_UCHAR WakeNext();
00131
00132
00133 #if KERNEL_USE_TIMEOUTS
00134     bool Claim_i( K_ULONG ulWaitTimeMS_ );
00135 #else
00136     void Claim_i(void);
00137 #endif
00138
00139     K_UCHAR m_ucRecurse;
00140     K_UCHAR m_bReady;
00141     K_UCHAR m_ucMaxPri;
00142     Thread *m_pclOwner;
00143
00144 };
00145
00146 #endif //KERNEL_USE_MUTEX
00147
00148 #endif //__MUTEX_H_
00149

```

15.121 /home/moslevin/Project/R1/stage/src/nlfs.cpp File Reference

Nice Little Filesystem (NLFS) implementation for Mark3.

```

#include "kerneltypes.h"
#include "nlfs.h"
#include "nlfs_file.h"
#include "memutil.h"
#include "nlfs_config.h"

```

15.121.1 Detailed Description

Nice Little Filesystem (NLFS) implementation for Mark3.

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

15.122 nlfs.cpp

```

00001  /*=====
00002
00003  _____
00004  |   |   |   |   |   |   |   |   |   |   |
00005  |   |   |   |   |   |   |   |   |   |   |
00006  |   |   |   |   |   |   |   |   |   |   |
00007  |   |   |   |   |   |   |   |   |   |   |
00008  |   |   |   |   |   |   |   |   |   |   |
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00019  #include "kerneltypes.h"
00020  #include "nlfs.h"
00021  #include "nlfs_file.h"
00022  #include "memutil.h"
00023  #include "nlfs_config.h"
00024
00025  //-----
00026  K_CHAR NLFS::Find_Last_Slash( const char *szPath_ )
00027  {
00028      K_UCHAR ucLastSlash = 0;
00029      K_UCHAR i = 0;
00030      while (szPath_[i])
00031      {
00032          if (szPath_[i] == '/')
00033          {
00034              ucLastSlash = i;
00035          }
00036          i++;
00037      }
00038      return ucLastSlash;
00039  }
00040
00041  //-----
00042  K_BOOL NLFS::File_Names_Match( const K_CHAR *szPath_,
00043                                NLFS_Node_t *pstNode_ )
00044  {
00045      K_UCHAR ucLastSlash = Find_Last_Slash( szPath_ );
00046      K_UCHAR i;
00047      ucLastSlash++;
00048      for (i = 0; i < FILE_NAME_LENGTH; i++)
00049      {
00050          if (!szPath_[ucLastSlash+i] || !pstNode_>stFileNode.
00051              acFileName[i])
00052          {
00053              break;
00054          }
00055          if (szPath_[ucLastSlash+i] != pstNode_>stFileNode.acFileName[i])
00056          {
00057              return false;
00058          }
00059      }
00060      if (szPath_[ucLastSlash+i] != pstNode_>stFileNode.acFileName[i])
00061      {
00062          return false;
00063      }
00064      return true;
00065  }
00066
00067  //-----
00068  void NLFS::Print_File_Details( K_USHORT usNode_ )
00069  {
00070      NLFS_Node_t stFileNode;
00071      Read_Node(usNode_, &stFileNode);
00072
00073      DEBUG_PRINT(" Name      : %16s\n", stFileNode.stFileNode.
00074          acFileName);
00075      DEBUG_PRINT(" Next Peer : %d\n", stFileNode.stFileNode.
00076          usNextPeer);
00077      DEBUG_PRINT(" Prev Peer : %d\n", stFileNode.stFileNode.
00078          usPrevPeer);
00079      DEBUG_PRINT(" User|Group : %d|%d\n", stFileNode.stFileNode.ucUser,
00080          stFileNode.stFileNode.ucGroup);
00081      DEBUG_PRINT(" Permissions: %04X\n", stFileNode.stFileNode.usPerms);
00082      DEBUG_PRINT(" Parent     : %d\n", stFileNode.stFileNode.
00083          usParent);
00084      DEBUG_PRINT(" First Child: %d\n", stFileNode.stFileNode.usChild);
00085      DEBUG_PRINT(" Alloc Size : %d\n", stFileNode.stFileNode.
00086          ulAllocSize);

```

```

00083     DEBUG_PRINT(" File Size : %d\n" , stFileNode.stFileNode.
        ulFileSize);
00084
00085     DEBUG_PRINT(" First Block: %d\n" , stFileNode.stFileNode.
        ulFirstBlock);
00086     DEBUG_PRINT(" Last Block : %d\n" , stFileNode.stFileNode.
        ulLastBlock);
00087 }
00088
00089 //-----
00090 void NLFS::Print_Dir_Details( K_USHORT usNode_ )
00091 {
00092     NLFS_Node_t stFileNode;
00093     Read_Node(usNode_, &stFileNode);
00094
00095     DEBUG_PRINT(" Name      : %16s\n" , stFileNode.stFileNode.
        acFileName);
00096     DEBUG_PRINT(" Next Peer : %d\n" , stFileNode.stFileNode.
        usNextPeer);
00097     DEBUG_PRINT(" Prev Peer : %d\n" , stFileNode.stFileNode.
        usPrevPeer);
00098     DEBUG_PRINT(" User|Group : %d|%d\n" , stFileNode.stFileNode.ucUser,
        stFileNode.stFileNode.ucGroup);
00099     DEBUG_PRINT(" Permissions: %04X\n" , stFileNode.stFileNode.
        usPerms);
00100     DEBUG_PRINT(" Parent      : %d\n" , stFileNode.stFileNode.
        usParent);
00101     DEBUG_PRINT(" First Child: %d\n" , stFileNode.stFileNode.usChild);
00102 }
00103
00104 //-----
00105 void NLFS::Print_Free_Details( K_USHORT usNode_ )
00106 {
00107     NLFS_Node_t stFileNode;
00108     Read_Node(usNode_, &stFileNode);
00109
00110     DEBUG_PRINT(" Next Free : %d\n" , stFileNode.stFileNode.
        usNextPeer );
00111 }
00112
00113 //-----
00114 void NLFS::Print_Node_Details( K_USHORT usNode_ )
00115 {
00116     NLFS_Node_t stTempNode;
00117     Read_Node(usNode_, &stTempNode);
00118
00119     DEBUG_PRINT("\nNode: %d\n"
        " Node Type: ", usNode_);
00120     switch (stTempNode.eBlockType)
00121     {
00122     case NLFS_NODE_FREE:
00123         DEBUG_PRINT( "Free\n" );
00124         Print_Free_Details(usNode_);
00125         break;
00126     case NLFS_NODE_ROOT:
00127         DEBUG_PRINT( "Root Block\n" );
00128         break;
00129     case NLFS_NODE_FILE:
00130         DEBUG_PRINT( "File\n" );
00131         Print_File_Details(usNode_);
00132         break;
00133     case NLFS_NODE_DIR:
00134         DEBUG_PRINT( "Directory\n" );
00135         Print_Dir_Details(usNode_);
00136         break;
00137     default:
00138         break;
00139     }
00140 }
00141
00142 //-----
00143 K_USHORT NLFS::Pop_Free_Node(void)
00144 {
00145     K_USHORT usRetVal = m_stLocalRoot.usNextFreeNode;
00146     NLFS_Node_t stFileNode;
00147
00148     if (INVALID_NODE == usRetVal)
00149     {
00150         return 0;
00151     }
00152
00153     // Update Claimed node
00154     Read_Node(usRetVal, &stFileNode);
00155     m_stLocalRoot.usNextFreeNode = stFileNode.
        stFileNode.usNextPeer;
00156     stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00157     DEBUG_PRINT("Node %d allocated, next free %d\n", usRetVal, m_stLocalRoot.

```



```

        usNextFreeNode);
00160     Write_Node(usRetVal, &stFileNode);
00161
00162     //Update root node
00163     Read_Node(FS_CONFIG_BLOCK, &stFileNode);
00164     stFileNode.stRootNode.usNextFreeNode = m_stLocalRoot.
usNextFreeNode;
00165     stFileNode.stRootNode.usNumFilesFree--;
00166     Write_Node(FS_CONFIG_BLOCK, &stFileNode);
00167
00168     return usRetVal;
00169 }
00170
00171 //-----
00172 void NLFS::Push_Free_Node(K_USHORT usNode_)
00173 {
00174     NLFS_Node_t stFileNode;
00175
00176     Read_Node(usNode_, &stFileNode);
00177     stFileNode.stFileNode.usNextPeer = m_stLocalRoot.
usNextFreeNode;
00178     m_stLocalRoot.usNextFreeNode = usNode_;
00179
00180     Write_Node(usNode_, &stFileNode);
00181
00182     DEBUG_PRINT("Node %d freed\n", usNode_);
00183
00184     //Update root node
00185     Read_Node(FS_CONFIG_BLOCK, &stFileNode);
00186     stFileNode.stRootNode.usNextFreeNode = m_stLocalRoot.
usNextFreeNode;
00187     stFileNode.stRootNode.usNumFilesFree++;
00188     Write_Node(FS_CONFIG_BLOCK, &stFileNode);
00189 }
00190
00191 //-----
00192 K_ULONG NLFS::Pop_Free_Block(void)
00193 {
00194     K_ULONG ulRetVal = m_stLocalRoot.ulNextFreeBlock;
00195     NLFS_Block_t stFileBlock;
00196     NLFS_Node_t stFileNode;
00197
00198     if ((INVALID_BLOCK == ulRetVal) || (0 == m_stLocalRoot.
ulNumBlocksFree))
00199     {
00200         DEBUG_PRINT("Out of data blocks\n");
00201         return 0;
00202     }
00203
00204     Read_Block_Header(ulRetVal, &stFileBlock);
00205
00206     m_stLocalRoot.ulNextFreeBlock = stFileBlock.
ulNextBlock;
00207     m_stLocalRoot.ulNumBlocksFree--;
00208     stFileBlock.ulNextBlock = INVALID_BLOCK;
00209
00210     Write_Block_Header(ulRetVal, &stFileBlock);
00211
00212     Read_Node(FS_CONFIG_BLOCK, &stFileNode);
00213
00214     stFileNode.stRootNode.ulNextFreeBlock =
m_stLocalRoot.ulNextFreeBlock;
00215     stFileNode.stRootNode.ulNumBlocksFree--;
00216
00217     Write_Node(FS_CONFIG_BLOCK, &stFileNode);
00218
00219     DEBUG_PRINT("Allocated block %d, next free %d\n", ulRetVal, m_stLocalRoot.
ulNextFreeBlock);
00220     return ulRetVal;
00221 }
00222
00223 //-----
00224 void NLFS::Push_Free_Block(K_ULONG ulBlock_)
00225 {
00226     NLFS_Block_t stFileBlock;
00227     NLFS_Node_t stFileNode;
00228
00229     Read_Block_Header(ulBlock_, &stFileBlock);
00230
00231     stFileBlock.ulNextBlock = m_stLocalRoot.
ulNextFreeBlock;
00232     m_stLocalRoot.ulNextFreeBlock = ulBlock_;
00233
00234     Write_Block_Header(ulBlock_, &stFileBlock);
00235
00236     Read_Node(FS_CONFIG_BLOCK, &stFileNode);
00237     stFileNode.stRootNode.ulNextFreeBlock =

```

```

    m_stLocalRoot.ulNextFreeBlock;
00238     stFileNode.stRootNode.ulNumBlocksFree++;
00239     Write_Node(FS_CONFIG_BLOCK , &stFileNode);
00240
00241     DEBUG_PRINT("Block %d freed\n", ulBlock_);
00242 }
00243
00244 //-----
00245 K_ULONG NLFSS::Append_Block_To_Node(NLFSS_Node_t *pstFile_ )
00246 {
00247     K_ULONG ulBlock;
00248     NLFSS_Block_t stFileBlock;
00249
00250     // Allocate a new block
00251     ulBlock = Pop_Free_Block();
00252     if (ulBlock == INVALID_BLOCK)
00253     {
00254         return -1;
00255     }
00256
00257     // Initialize the block
00258     DEBUG_PRINT("reading block header\n");
00259     Read_Block_Header(ulBlock, &stFileBlock);
00260     stFileBlock.ulNextBlock = INVALID_BLOCK;
00261     stFileBlock.uAllocated = 1;
00262
00263     DEBUG_PRINT("writing block header\n");
00264     Write_Block_Header(ulBlock, &stFileBlock);
00265
00266     // Update the previous last-block links (if there is one)
00267     DEBUG_PRINT("updating previous block %d\n", pstFile_>stFileNode.
ulLastBlock);
00268     if (pstFile_>stFileNode.ulLastBlock != INVALID_BLOCK)
00269     {
00270         Read_Block_Header(pstFile_>stFileNode.
ulLastBlock, &stFileBlock);
00271         stFileBlock.ulNextBlock = ulBlock;
00272         Write_Block_Header(pstFile_>stFileNode.
ulLastBlock, &stFileBlock);
00273     }
00274     else
00275     {
00276         DEBUG_PRINT(" previous block is invalid, setting as first\n");
00277         pstFile_>stFileNode.ulFirstBlock = ulBlock;
00278     }
00279
00280     pstFile_>stFileNode.ulLastBlock = ulBlock;
00281     pstFile_>stFileNode.ulAllocSize += m_stLocalRoot.
ulBlockSize;
00282
00283     RootSync();
00284
00285     return ulBlock;
00286 }
00287
00288 //-----
00289 K_USHORT NLFSS::Find_Parent_Dir(const K_CHAR *szPath_)
00290 {
00291     int i, j;
00292     K_UCHAR ucLastSlash = 0;
00293     K_USHORT usRetVal;
00294     K_CHAR szTempName[FILE_NAME_LENGTH];
00295     NLFSS_Node_t stFileNode;
00296     K_USHORT usTempPeer;
00297
00298     Read_Node(FS_ROOT_BLOCK, &stFileNode );
00299
00300     usRetVal = FS_ROOT_BLOCK;
00301
00302     if (szPath_[0] != '/')
00303     {
00304         DEBUG_PRINT("Only fully-qualified paths are supported.  Bailing\n");
00305         return -1;
00306     }
00307
00308     // Starting from the root fs_block (which is the mount point...)
00309     ucLastSlash = Find_Last_Slash(szPath_);
00310
00311     // a) Search for each "/" if we've got more than one...
00312     if (0 == ucLastSlash)
00313     {
00314         return usRetVal;
00315     }
00316
00317     usTempPeer = stFileNode.stFileNode.usChild;
00318     Read_Node(usTempPeer, &stFileNode );
00319

```

```

00320     i = 1;
00321     while (szPath_[i] && i < ucLastSlash)
00322     {
00323         NLFS_Node_t stTempNode;
00324         K_BOOL bMatch = false;
00325
00326         j = 0;
00327         MemUtil::SetMemory(szTempName, 0, FILE_NAME_LENGTH);
00328
00329         while (szPath_[i] && (szPath_[i] != '/') && j < FILE_NAME_LENGTH)
00330         {
00331             szTempName[j] = szPath_[i];
00332             i++;
00333             j++;
00334         }
00335         DEBUG_PRINT("Checking %s\n", szTempName );
00336         if (j == FILE_NAME_LENGTH && szPath_[i] != '/')
00337         {
00338             DEBUG_PRINT("Directory name too long, invalid\n");
00339             return -1;
00340         }
00341         else if (szPath_[i] != '/')
00342         {
00343             i++;
00344             continue;
00345         }
00346
00347         // Check to see if there's a valid peer with this name...
00348         while (INVALID_NODE != usTempPeer)
00349         {
00350             Read_Node(usTempPeer, &stTempNode);
00351             if (NLFS_NODE_DIR == stTempNode.eBlockType)
00352             {
00353                 if (true == MemUtil::CompareStrings(stTempNode.
00354 stFileName.acFileName, szTempName))
00355                 {
00356                     bMatch = true;
00357                     break;
00358                 }
00359                 usTempPeer = stTempNode.stFileName.usNextPeer;
00360             }
00361
00362             // Matched the folder name descend into the folder
00363             if (bMatch)
00364             {
00365                 DEBUG_PRINT("Matched folder: %s, node %d\n", szTempName, usTempPeer);
00366
00367                 usRetVal = usTempPeer;
00368
00369                 usTempPeer = stTempNode.stFileName.usChild;
00370                 if (INVALID_NODE != usTempPeer)
00371                 {
00372                     DEBUG_PRINT("Entering subdirectory %d\n", usTempPeer);
00373                     Read_Node(usTempPeer, &stFileNode);
00374                 }
00375                 else
00376                 {
00377                     break;
00378                 }
00379             }
00380             // Failed to match the folder name, bail
00381             else
00382             {
00383                 DEBUG_PRINT("Could not match folder name, bailing\n");
00384                 usRetVal = -1;
00385                 break;
00386             }
00387
00388             if (i >= ucLastSlash)
00389             {
00390                 break;
00391             }
00392             i++;
00393         }
00394
00395         if (i == ucLastSlash)
00396         {
00397             // No more folders to traverse - we're successful.
00398             DEBUG_PRINT("Found root path for %s\n with node %d\n", szPath_, usRetVal);
00399             return usRetVal;
00400         }
00401         return INVALID_NODE;
00402     }
00403
00404 //-----
00405 K_USHORT NLFS::Find_File(const K_CHAR *szPath_)

```

```

00406 {
00407     NLFS_Node_t stTempNode;
00408     NLFS_Node_t stTempDir;
00409
00410     K_USHORT usTempNode;
00411
00412     K_USHORT usParentDir = Find_Parent_Dir(szPath_);
00413
00414     if (INVALID_NODE == usParentDir)
00415     {
00416         DEBUG_PRINT("invalid root dir\n");
00417         return INVALID_NODE;
00418     }
00419
00420     Read_Node(usParentDir, &stTempDir);
00421
00422     if (INVALID_NODE == stTempDir.stFileNode.usChild)
00423     {
00424         return INVALID_NODE;
00425     }
00426
00427     usTempNode = stTempDir.stFileNode.usChild;
00428
00429     // See if there are matching child nodes
00430     while (INVALID_NODE != usTempNode)
00431     {
00432         Read_Node(usTempNode, &stTempNode);
00433
00434         if (true == File_Names_Match(szPath_, &stTempNode ))
00435         {
00436             DEBUG_PRINT("matched file: %16s, node %d\n",
00437                 stTempNode.stFileNode.acFileName, usTempNode);
00438             return usTempNode;
00439         }
00440
00441         usTempNode = stTempNode.stFileNode.usNextPeer;
00442     }
00443     DEBUG_PRINT("couldn't match file: %s\n", szPath_);
00444     return INVALID_NODE;
00445 }
00446
00447 //-----
00448 void NLFS::Print(void)
00449 {
00450     K_USHORT i;
00451     for (i = 0; i < m_stLocalRoot.usNumFiles; i++)
00452     {
00453         Print_Node_Details(i);
00454     }
00455 }
00456
00457 //-----
00458 void NLFS::Set_Node_Name( NLFS_Node_t *pstFileNode_, const char *szPath_ )
00459 {
00460     K_UCHAR i, j;
00461     K_UCHAR ucLastSlash = 0;
00462
00463     // Search for the last "/", that's where we stop looking.
00464     i = 0;
00465     while (szPath_[i])
00466     {
00467         if (szPath_[i] == '/')
00468         {
00469             ucLastSlash = i;
00470         }
00471         i++;
00472     }
00473
00474     // Parse out filename
00475     i = ucLastSlash + 1;
00476     j = 0;
00477     while (szPath_[i] && j < FILE_NAME_LENGTH)
00478     {
00479         pstFileNode_>stFileNode.acFileName[j] = szPath_[i];
00480         j++;
00481         i++;
00482     }
00483     if (!szPath_[i]) // if no extension, we're done.
00484     {
00485         return;
00486     }
00487 }
00488
00489 //-----
00490 K_USHORT NLFS::Create_File_i(const K_CHAR *szPath_,
00491     NLFS_Type_t eType_ )
00492 {

```

```

00492     K_USHORT usNode;
00493     K_USHORT usRootNodes;
00494
00495     NLFS_Node_t stFileNode;
00496     NLFS_Node_t stParentNode;
00497     NLFS_Node_t stPeerNode;
00498
00499     // Tricky part - directory traversal
00500     usRootNodes = Find_Parent_Dir(szPath_);
00501
00502     if (INVALID_NODE == usRootNodes)
00503     {
00504         DEBUG_PRINT("Unable to find path - bailing\n");
00505         return INVALID_NODE;
00506     }
00507
00508     usNode = Pop_Free_Node();
00509     if (!usNode)
00510     {
00511         DEBUG_PRINT("Unable to allocate node. Failing\n");
00512         return INVALID_NODE;
00513     }
00514     DEBUG_PRINT("New file using node %d\n", usNode);
00515
00516     // File node allocated, do something with it...
00517     // Set the file's name and extension
00518
00519     Read_Node(usNode, &stFileNode);
00520
00521     // Set the file path
00522     Set_Node_Name(&stFileNode, szPath_);
00523
00524     // Set block as in-use as a file
00525     stFileNode.eBlockType = eType_;
00526
00527     // Zero-out the file
00528     stFileNode.stFileNode.ulFileSize = 0;
00529
00530     // Set the default user and group, as well as perms
00531     stFileNode.stFileNode.ucUser = 0;
00532     stFileNode.stFileNode.ucGroup = 0;
00533     stFileNode.stFileNode.usPerms = PERM_U_ALL | PERM_G_ALL | PERM_O_ALL;
00534
00535     stFileNode.stFileNode.usChild = INVALID_NODE;
00536     stFileNode.stFileNode.usParent = usRootNodes;
00537
00538     // Update the parent node.
00539     Read_Node(usRootNodes, &stParentNode);
00540
00541     DEBUG_PRINT("Parent's root child: %d\n", stParentNode.stFileNode.
usChild );
00542     // Insert node at the beginning of the peer list
00543     if (INVALID_NODE != stParentNode.stFileNode.usChild)
00544     {
00545         stFileNode.stFileNode.usNextPeer = stParentNode.
stFileNode.usChild;
00546         stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00547
00548         // Update the peer node.
00549         Read_Node(stFileNode.stFileNode.usNextPeer , &stPeerNode);
00550
00551         stPeerNode.stFileNode.usPrevPeer = usNode;
00552         stParentNode.stFileNode.usChild = usNode;
00553
00554         DEBUG_PRINT("updating peer's prev: %d\n", stPeerNode.stFileNode.
usPrevPeer);
00555         Write_Node(stFileNode.stFileNode.usNextPeer, &stPeerNode);
00556     }
00557     else
00558     {
00559         stParentNode.stFileNode.usChild = usNode;
00560         stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00561         stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00562     }
00563
00564     Write_Node(usNode, &stFileNode);
00565     Write_Node(usRootNodes, &stParentNode);
00566
00567     RootSync();
00568
00569     return usNode;
00570 }
00571
00572 //-----
00573 K_USHORT NLFS::Create_File( const K_CHAR *szPath_ )
00574 {
00575

```

```

00576     if (INVALID_NODE != Find_File(szPath_))
00577     {
00578         DEBUG_PRINT("Create_File: File already exists\n");
00579         return INVALID_NODE;
00580     }
00581
00582     return Create_File_i( szPath_, NLFS_NODE_FILE );
00583 }
00584
00585 //-----
00586 K_USHORT NLFS::Create_Dir( const K_CHAR *szPath_ )
00587 {
00588     if (INVALID_NODE != Find_File(szPath_))
00589     {
00590         DEBUG_PRINT("Create_Dir: Dir already exists!\n");
00591         return INVALID_NODE;
00592     }
00593
00594     return Create_File_i(szPath_, NLFS_NODE_DIR );
00595 }
00596
00597 //-----
00598 void NLFS::Cleanup_Node_Links(K_USHORT usNode_,
00599                               NLFS_Node_t *pstNode_)
00600 {
00601     DEBUG_PRINT("Cleanup_Node_Links: Entering\n");
00602
00603     if (INVALID_NODE != pstNode_>stFileNode.usParent)
00604     {
00605         NLFS_Node_t stParent;
00606         DEBUG_PRINT("Cleanup_Node_Links: Parent Node: %d\n", pstNode_>
00607 stFileNode.usParent);
00608         Read_Node(pstNode_>stFileNode.usParent, &stParent);
00609
00610         DEBUG_PRINT("0\n");
00611         if (stParent.stFileNode.usChild == usNode_)
00612         {
00613             DEBUG_PRINT("1\n");
00614             stParent.stFileNode.usChild = pstNode_>stFileNode.
00615 usNextPeer;
00616             Write_Node(pstNode_>stFileNode.usParent, &stParent);
00617             DEBUG_PRINT("2\n");
00618         }
00619
00620         DEBUG_PRINT("a\n");
00621         if ( (INVALID_NODE != pstNode_>stFileNode.usNextPeer) ||
00622             (INVALID_NODE != pstNode_>stFileNode.usPrevPeer) )
00623         {
00624             NLFS_Node_t stNextPeer;
00625             NLFS_Node_t stPrevPeer;
00626
00627             DEBUG_PRINT("b\n");
00628             if (INVALID_NODE != pstNode_>stFileNode.usNextPeer)
00629             {
00630                 DEBUG_PRINT("c\n");
00631                 Read_Node(pstNode_>stFileNode.usNextPeer, &stNextPeer);
00632                 DEBUG_PRINT("d\n");
00633             }
00634
00635             if (INVALID_NODE != pstNode_>stFileNode.usPrevPeer)
00636             {
00637                 DEBUG_PRINT("e\n");
00638                 Read_Node(pstNode_>stFileNode.usPrevPeer, &stPrevPeer);
00639                 DEBUG_PRINT("f\n");
00640             }
00641
00642             if (INVALID_NODE != pstNode_>stFileNode.usNextPeer)
00643             {
00644                 DEBUG_PRINT("g\n");
00645                 stNextPeer.stFileNode.usPrevPeer = pstNode_>
00646 stFileNode.usPrevPeer;
00647                 Write_Node(pstNode_>stFileNode.usNextPeer, &stNextPeer);
00648                 DEBUG_PRINT("h\n");
00649             }
00650
00651             if (INVALID_NODE != pstNode_>stFileNode.usPrevPeer)
00652             {
00653                 DEBUG_PRINT("i\n");
00654                 stPrevPeer.stFileNode.usNextPeer = pstNode_>
00655 stFileNode.usNextPeer;
00656                 Write_Node(pstNode_>stFileNode.usPrevPeer, &stPrevPeer);
00657                 DEBUG_PRINT("j\n");
00658             }
00659         }
00660
00661         pstNode_>stFileNode.usParent = INVALID_NODE;
00662         pstNode_>stFileNode.usPrevPeer = INVALID_NODE;

```

```

00658     pstNode->stFileNode.usNextPeer = INVALID_NODE;
00659 }
00660
00661 //-----
00662 K_USHORT NLFS::Delete_Folder(const K_CHAR *szPath_)
00663 {
00664     K_USHORT usNode = Find_File(szPath_);
00665     NLFS_Node_t stNode;
00666
00667     if (INVALID_NODE == usNode)
00668     {
00669         DEBUG_PRINT("Delete_Folder: File not found!\n");
00670         return INVALID_NODE;
00671     }
00672     if (FS_ROOT_BLOCK == usNode || FS_CONFIG_BLOCK == usNode)
00673     {
00674         DEBUG_PRINT("Delete_Folder: Cannot delete root!\n");
00675         return INVALID_NODE;
00676     }
00677
00678     Read_Node(usNode, &stNode);
00679
00680     if (NLFS_NODE_FILE == stNode.eBlockType)
00681     {
00682         DEBUG_PRINT("Delete_Folder: Path is not a Folder (is it a file?)");
00683         return INVALID_NODE;
00684     }
00685
00686     if (INVALID_NODE != stNode.stFileNode.usChild)
00687     {
00688         DEBUG_PRINT("Delete_Folder: Folder is not empty!");
00689         return INVALID_NODE;
00690     }
00691
00692     Cleanup_Node_Links(usNode, &stNode);
00693
00694     stNode.eBlockType = NLFS_NODE_FREE;
00695
00696     Write_Node(usNode, &stNode);
00697     Push_Free_Node(usNode);
00698
00699     RootSync();
00700
00701     return usNode;
00702 }
00703
00704 //-----
00705 K_USHORT NLFS::Delete_File( const K_CHAR *szPath_)
00706 {
00707     K_USHORT usNode = Find_File(szPath_);
00708     K_ULONG ulCurr;
00709     K_ULONG ulPrev;
00710     NLFS_Node_t stNode;
00711     NLFS_Block_t stBlock;
00712
00713     if (INVALID_NODE == usNode)
00714     {
00715         DEBUG_PRINT("Delete_File: File not found!\n");
00716         return INVALID_NODE;
00717     }
00718     if (FS_ROOT_BLOCK == usNode || FS_CONFIG_BLOCK == usNode)
00719     {
00720         DEBUG_PRINT("Delete_File: Cannot delete root!\n");
00721         return INVALID_NODE;
00722     }
00723
00724     Read_Node(usNode, &stNode);
00725
00726     if (NLFS_NODE_DIR == stNode.eBlockType)
00727     {
00728         DEBUG_PRINT("Delete_File: Path is not a file (is it a directory?)");
00729         return INVALID_NODE;
00730     }
00731
00732     Cleanup_Node_Links(usNode, &stNode);
00733     ulCurr = stNode.stFileNode.ulFirstBlock;
00734
00735     while (INVALID_BLOCK != ulCurr)
00736     {
00737         Read_Block_Header(ulCurr, &stBlock);
00738
00739         ulPrev = ulCurr;
00740         ulCurr = stBlock.ulNextBlock;
00741
00742         Push_Free_Block(ulPrev);
00743     }
00744

```

```

00745     stNode.eBlockType = NLFS_NODE_FREE;
00746
00747     Write_Node(usNode, &stNode);
00748     Push_Free_Node(usNode);
00749
00750     RootSync();
00751
00752     return usNode;
00753 }
00754
00755 //-----
00756 void NLFS::Format(NLFS_Host_t *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_,
K_USHORT usDataBlockSize_)
00757 {
00758     K_ULONG i;
00759     K_ULONG ulNumBlocks;
00760
00761     NLFS_Node_t stFileNode;
00762     NLFS_Block_t stFileBlock;
00763
00764     // Compute number of data blocks (based on FS Size and the number of file blocks)
00765     ulTotalSize_ -= ((K_ULONG)usNumFiles_) * sizeof(stFileNode);
00766     ulNumBlocks = ulTotalSize_ / (((K_ULONG)usDataBlockSize_) + (sizeof(stFileBlock) - 1) + 3) & ~3;
00767
00768     DEBUG_PRINT("Number of blocks %d\n", ulNumBlocks);
00769
00770     // Set up the local_pointer -> this is used for the low-level, platform-specific
00771     // bits, allowing the FS to be used on RAM buffers, EEPROM's, networks, etc.
00772     m_puHost = puHost_;
00773
00774     // Set the local copies of the data block byte-offset, as well as the data-block size
00775     m_stLocalRoot.usNumFiles = usNumFiles_;
00776     m_stLocalRoot.usNumFilesFree = m_stLocalRoot.usNumFiles - 2;
00777     m_stLocalRoot.usNextFreeNode = 2;
00778
00779     m_stLocalRoot.ulNumBlocks = ulNumBlocks;
00780     m_stLocalRoot.ulNumBlocksFree = ulNumBlocks;
00781     m_stLocalRoot.ulNextFreeBlock = 0;
00782
00783     m_stLocalRoot.ulBlockSize = (((K_ULONG)usDataBlockSize_) + 3) & ~3;
00784     m_stLocalRoot.ulBlockOffset = (((K_ULONG)usNumFiles_) * sizeof(
NLFS_Node_t));
00785     m_stLocalRoot.ulDataOffset = m_stLocalRoot.ulBlockOffset
00786     + (((K_ULONG)ulNumBlocks) * sizeof(
NLFS_Block_t));
00787
00788     // Create root data block node
00789     MemUtil::CopyMemory(&(stFileNode.stRootNode), &
m_stLocalRoot, sizeof(m_stLocalRoot));
00790     stFileNode.eBlockType = NLFS_NODE_ROOT;
00791
00792     DEBUG_PRINT("Writing root node\n");
00793     Write_Node(0, &stFileNode);
00794     DEBUG_PRINT("Done\n");
00795
00796     // Create root mount point (directory)
00797     MemUtil::SetMemory(&stFileNode, 0, sizeof(stFileNode));
00798     stFileNode.eBlockType = NLFS_NODE_DIR;
00799
00800     stFileNode.stFileNode.acFileName[0] = '/';
00801
00802     stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00803     stFileNode.stFileNode.usPrevPeer = INVALID_NODE;
00804     stFileNode.stFileNode.ucGroup = 0;
00805     stFileNode.stFileNode.ucUser = 0;
00806     stFileNode.stFileNode.usPerms = PERM_U_ALL | PERM_G_ALL | PERM_O_ALL;
00807
00808     stFileNode.stFileNode.usParent = INVALID_NODE;
00809     stFileNode.stFileNode.usChild = INVALID_NODE;
00810
00811     stFileNode.stFileNode.ulAllocSize = 0;
00812     stFileNode.stFileNode.ulFileSize = 0;
00813
00814     stFileNode.stFileNode.ulFirstBlock = INVALID_BLOCK;
00815     stFileNode.stFileNode.ulLastBlock = INVALID_BLOCK;
00816
00817     DEBUG_PRINT("Writing mount point\n");
00818     Write_Node(1, &stFileNode);
00819     DEBUG_PRINT("Done\n");
00820
00821     stFileNode.stFileNode.acFileName[0] = 0;
00822     // Format nodes
00823     for (i = 2; i < usNumFiles_; i++)
00824     {
00825         stFileNode.eBlockType = NLFS_NODE_FREE;

```



```

00826         if (i != usNumFiles_ - 1)
00827         {
00828             stFileNode.stFileNode.usNextPeer = (K_USHORT)(i + 1);
00829         }
00830         else
00831         {
00832             stFileNode.stFileNode.usNextPeer = INVALID_NODE;
00833         }
00834
00835         Write_Node(i, &stFileNode);
00836     }
00837     DEBUG_PRINT("File nodes formatted\n");
00838
00839     // Format file blocks
00840     MemUtil::SetMemory(&stFileBlock, 0, sizeof(stFileBlock));
00841
00842     DEBUG_PRINT("Writing file blocks\n");
00843     for (i = 0; i < ulNumBlocks; i++)
00844     {
00845         if (i == ulNumBlocks - 1)
00846         {
00847             stFileBlock.ulNextBlock = INVALID_BLOCK;
00848         }
00849         else
00850         {
00851             stFileBlock.ulNextBlock = i + 1;
00852         }
00853
00854         Write_Block_Header(i, &stFileBlock);
00855     }
00856 }
00857
00858 //-----
00859 void NLFS::Mount(NLFS_Host_t *puHost_)
00860 {
00861     NLFS_Node_t stRootNode;
00862
00863     m_puHost = puHost_;
00864     DEBUG_PRINT("Remounting FS %X - reading config node\n", puHost_);
00865
00866     // Reload the root block into the local cache
00867     Read_Node(FS_CONFIG_BLOCK, &stRootNode);
00868
00869     DEBUG_PRINT("Copying config node\n");
00870     MemUtil::CopyMemory(&m_stLocalRoot, &(stRootNode.
00871 stRootNode), sizeof(m_stLocalRoot));
00872
00873     DEBUG_PRINT("Block Size", m_stLocalRoot.ulBlockSize );
00874     DEBUG_PRINT("Data Offset", m_stLocalRoot.ulDataOffset );
00875     DEBUG_PRINT("Block Offset", m_stLocalRoot.ulBlockOffset );
00876 }
00877
00878 //-----
00879 void NLFS::RootSync()
00880 {
00881     NLFS_Node_t stRootNode;
00882
00883     MemUtil::CopyMemory(&(stRootNode.stRootNode), &
00884 m_stLocalRoot, sizeof(m_stLocalRoot));
00885     stRootNode.eBlockType = NLFS_NODE_ROOT;
00886     Write_Node(FS_CONFIG_BLOCK, &stRootNode);
00887 }
00888
00889 //-----
00890 K_USHORT NLFS::GetFirstChild( K_USHORT usNode_ )
00891 {
00892     NLFS_Node_t stTemp;
00893     if (!usNode_ || INVALID_NODE == usNode_)
00894     {
00895         return INVALID_NODE;
00896     }
00897     Read_Node(usNode_, &stTemp);
00898
00899     if (stTemp.eBlockType != NLFS_NODE_DIR)
00900     {
00901         return INVALID_NODE;
00902     }
00903
00904     return stTemp.stFileNode.usChild;
00905 }
00906
00907 //-----
00908 K_USHORT NLFS::GetNextPeer( K_USHORT usNode_ )
00909 {
00910     NLFS_Node_t stTemp;
00911     if (!usNode_ || INVALID_NODE == usNode_)

```

```

00912     {
00913         return INVALID_NODE;
00914     }
00915     Read_Node(usNode_, &stTemp);
00916     return stTemp.stFileNode.usNextPeer;
00917 }
00918
00919 //-----
00920 K_BOOL NLFS::GetStat( K_USHORT usNode_, NLFS_File_Stat_t *pstStat_)
00921 {
00922     NLFS_Node_t stTemp;
00923     if (!usNode_ || INVALID_NODE == usNode_)
00924     {
00925         return false;
00926     }
00927     Read_Node(usNode_, &stTemp);
00928     pstStat_->ulAllocSize = stTemp.stFileNode.ulAllocSize;
00929     pstStat_->ulFileSize = stTemp.stFileNode.ulFileSize;
00930     pstStat_->ucGroup = stTemp.stFileNode.ucGroup;
00931     pstStat_->ucUser = stTemp.stFileNode.ucUser;
00932     pstStat_->usPerms = stTemp.stFileNode.usPerms;
00933     MemUtil::CopyMemory(pstStat_->acFileName, stTemp.
        stFileNode.acFileName, 16);
00934     return true;
00935 }
00936

```

15.123 /home/moslevin/Project/R1/stage/src/nlfs.h File Reference

Nice Little Filesystem (NLFS) - a simple, embeddable filesystem.

```

#include "kerneltypes.h"
#include <stdint.h>

```

Classes

- struct [NLFS_File_Node_t](#)
Data structure for the "file" FS-node type.
- struct [NLFS_Root_Node_t](#)
Data structure for the Root-configuration FS-node type.
- struct [NLFS_Node_t](#)
Filesystem node data structure.
- struct [NLFS_Block_t](#)
Block data structure.
- union [NLFS_Host_t](#)
Union used for managing host-specific pointers/data-types.
- struct [NLFS_File_Stat_t](#)
Structure used to report the status of a given file.
- class [NLFS](#)
Nice Little File System class.

Macros

- #define [PERM_UX](#) (0x0001)
Permission bit definitions.
- #define [PERM_UW](#) (0x0002)
- #define [PERM_UR](#) (0x0004)
- #define [PERM_U_ALL](#) ([PERM_UX](#) | [PERM_UW](#) | [PERM_UR](#))
- #define [PERM_GX](#) (0x0008)
- #define [PERM_GW](#) (0x0010)

- `#define PERM_GR (0x0020)`
- `#define PERM_G_ALL (PERM_GX | PERM_GW | PERM_GR)`
- `#define PERM_OX (0x0040)`
- `#define PERM_OW (0x0080)`
- `#define PERM_OR (0x0100)`
- `#define PERM_O_ALL (PERM_OX | PERM_OW | PERM_OR)`
- `#define INVALID_BLOCK (0xFFFFFFFF)`
- `#define INVALID_NODE (0xFFFF)`
- `#define FILE_NAME_LENGTH (16)`
- `#define FS_CONFIG_BLOCK (0)`
- `#define FS_ROOT_BLOCK (1)`

Enumerations

- `enum NLFS_Type_t {
NLFS_NODE_FREE, NLFS_NODE_ROOT, NLFS_NODE_FILE, NLFS_NODE_DIR,
FILE_BLOCK_COUNTS }`

Enumeration describing the various types of filesystem nodes used by [NLFS](#).

15.123.1 Detailed Description

Nice Little Filesystem ([NLFS](#)) - a simple, embeddable filesystem. Introduction to the Nice-Little-Filesystem ([NLFS](#))
[NLFS](#) is yet-another filesystem intended for use in embedded applications.

It is intended to be portable, lightweight, and flexible in terms of supporting different types of physical storage media. In order to ensure that it's easily embeddable, there are no external library dependencies, aside from library code provided elsewhere in Mark3 (namely the [MemUtil](#) utility class). Balancing code-size with features and functionality is also a tradeoff - [NLFS](#) supports basic operations (create file, create directory, read, write, seek, and delete), without a lot of other bells and whistles. One other feature built into the filesystem is posix-style user-group permissions. While the APIs in the [NLFS](#) classes do not enforce permissions explicitly, application-specific implementations of [NLFS](#) can enforce permissions based on facilities based on the security mechanisms built into the host OS.

The original purpose of this filesystem was to provide a flexible way of packaging files for read-only use within Mark3 (such as scripts and compiled DCPU-16 objects). However, there are all sorts of purposes for this type of filesystem - essentially, any application where a built-in file manifest or resource container format.

[NLFS](#) is a block-based filesystem, composed of three separate regions of data structures within a linearly-addressed blob of storage. These regions are represented on the physical storage in the following order:

[File Nodes][Data Block Headers][Block Data]

The individual regions are as follows:

1) File Nodes

This region is composed of a linear array of equally-sized file-node ([NLFS_Node_t](#)) structures, starting at byte offset 0 in the underlying media.

Each node defines a particular file or directory within the filesystem. Because of the linear layout of the filesystem, the file nodes are all pre-allocated during the time of filesystem creation. As a result, care should be taken to ensure enough file nodes are allocated to meet the needs of your application, without wasting space in the filesystem for nodes that will never be needed.

The first two nodes (node 0 and node 1) are special in the [NLFS](#) implementation.

Node 0 is also known as the root filesystem node. This block contains a different internal data structure from other file nodes, and stores the configuration information for the particular filesystem, such as the number of file nodes, file blocks, block sizes, as well as indexes of the first free file and block nodes in the filesystem. With this information, it is possible to re-mount a filesystem created once in another location.

Node 1 is the mount-point for the filesystem, and is the root directory under which all other files and directories are found. By default Node 1 is simply named `"/"`.

2) Block Headers

The block header region of the system comes after the file node region, and consists of a linear array of block node data structures. All storage in a filesystem not allocated towards file nodes is automatically allocated towards data blocks, and for each data block allocated, there is a block node data structure allocated within the block node region.

The `NLFS_Block_t` data structure contains a link to the next node in a block chain. If the block is free, the link points to the index of the next free block in the filesystem. If allocated, the link points to the index of the next block in the file. This structure also contains flags which indicate whether or not a block is free or allocated, and other flags used for filesystem continuity checks.

3) Block Data

The block data region is the last linear range in the filesystem, and consists of equally-sized blocks in the filesystem. Each block consists of a region of raw physical storage, without any additional metadata.

The contents of any files read or written to the filesystem is stored within the blocks in this region.

The `NLFS` Class has a number of virtual methods, which require that a user provides an implementation appropriate for the underlying physical storage medium from within a class inheriting `NLFS.s`

An example implementation for a RAM-based filesystem is provided in the `NLFS_RAM` class located within `nlfs_ram.-cpp`.

Definition in file `nlfs.h`.

15.123.2 Enumeration Type Documentation

15.123.2.1 enum `NLFS_Type_t`

Enumeration describing the various types of filesystem nodes used by `NLFS`.

A filesystem node is a fixed-sized data structure consisting of a type specifier, and a union of the data structures representing each possible block type.

Enumerator

`NLFS_NODE_FREE` File node is free.
`NLFS_NODE_ROOT` Root filesystem descriptor.
`NLFS_NODE_FILE` File node.
`NLFS_NODE_DIR` Directory node.

Definition at line 152 of file `nlfs.h`.

15.124 `nlfs.h`

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00108 #ifndef __NLFS_H__
00109 #define __NLFS_H__
00110
00111 #include "kerneltypes.h"
```

```

00112 #include <stdint.h>
00113
00114 class NLFS_File;
00115
00116 //-----
00120 #define PERM_UX      (0x0001)
00121 #define PERM_UW      (0x0002)
00122 #define PERM_UR      (0x0004)
00123 #define PERM_U_ALL   ( PERM_UX | PERM_UW | PERM_UR )
00124
00125 #define PERM_GX      (0x0008)
00126 #define PERM_GW      (0x0010)
00127 #define PERM_GR      (0x0020)
00128 #define PERM_G_ALL   ( PERM_GX | PERM_GW | PERM_GR )
00129
00130 #define PERM_OX      (0x0040)
00131 #define PERM_OW      (0x0080)
00132 #define PERM_OR      (0x0100)
00133 #define PERM_O_ALL   ( PERM_OX | PERM_OW | PERM_OR )
00134
00135 //-----
00136 #define INVALID_BLOCK (0xFFFFFFFF)
00137 #define INVALID_NODE   (0xFFFF)
00138
00139 //-----
00140 #define FILE_NAME_LENGTH (16)
00141
00142 #define FS_CONFIG_BLOCK (0)
00143 #define FS_ROOT_BLOCK   (1)
00144
00145 //-----
00152 typedef enum
00153 {
00154     NLFS_NODE_FREE,
00155     NLFS_NODE_ROOT,
00156     NLFS_NODE_FILE,
00157     NLFS_NODE_DIR,
00158 } --
00159     FILE_BLOCK_COUNTS
00160 } NLFS_Type_t;
00161
00162 //-----
00168 typedef struct
00169 {
00170     K_CHAR      acFileName[16];
00171
00172     K_USHORT     usNextPeer;
00173     K_USHORT     usPrevPeer;
00174
00175     K_UCHAR      ucGroup;
00176     K_UCHAR      ucUser;
00177     K_USHORT     usPerms;
00178
00179     K_USHORT     usParent;
00180     K_USHORT     usChild;
00181
00182 } -- File-specific
00183     K_ULONG      ulAllocSize;
00184     K_ULONG      ulFileSize;
00185
00186     K_ULONG      ulFirstBlock;
00187     K_ULONG      ulLastBlock;
00188 } NLFS_File_Node_t;
00189
00190 //-----
00194 typedef struct
00195 {
00196     K_USHORT     usNumFiles;
00197     K_USHORT     usNumFilesFree;
00198     K_USHORT     usNextFreeNode;
00199
00200     K_ULONG      ulNumBlocks;
00201     K_ULONG      ulNumBlocksFree;
00202     K_ULONG      ulNextFreeBlock;
00203
00204     K_ULONG      ulBlockSize;
00205     K_ULONG      ulBlockOffset;
00206     K_ULONG      ulDataOffset;
00207 } NLFS_Root_Node_t;
00208
00209 //-----
00215 typedef struct
00216 {
00217     NLFS_Type_t    eBlockType;
00218
00219     union          // Depending on the block type, we use one of the following
00220     {

```

```

00221         NLFS_Root_Node_t      stRootNode;
00222         NLFS_File_Node_t      stFileNode;
00223     };
00224 } NLFS_Node_t;
00225
00226 //-----
00232 typedef struct
00233 {
00234     K_ULONG      ulNextBlock;
00235     union
00236     {
00237         K_UCHAR      ucFlags;
00238         struct
00239         {
00240             unsigned int      uAllocated;
00241             unsigned int      uCheckBit;
00242         };
00243     };
00244 } NLFS_Block_t;
00245
00246 //-----
00253 typedef union
00254 {
00255     void *pvData;
00256     uint32_t u32Data;
00257     uint64_t u64Data;
00258     K_ADDR kaData;
00259 } NLFS_Host_t;
00260
00261 //-----
00266 typedef struct
00267 {
00268     K_ULONG      ulAllocSize;
00269     K_ULONG      ulFileSize;
00270     K_USHORT      usPerms;
00271     K_UCHAR      ucUser;
00272     K_UCHAR      ucGroup;
00273     K_CHAR      acFileName[16];
00274 } NLFS_File_Stat_t;
00275
00276 //-----
00280 class NLFS
00281 {
00282 friend class NLFS_File;
00283 public:
00284
00311     void Format(NLFS_Host_t *puHost_, K_ULONG ulTotalSize_, K_USHORT usNumFiles_, K_USHORT
        usDataBlockSize_);
00312
00318     void Mount(NLFS_Host_t *puHost_);
00319
00326     K_USHORT Create_File(const K_CHAR *szPath_);
00327
00334     K_USHORT Create_Dir(const K_CHAR *szPath_);
00335
00341     K_USHORT Delete_File(const K_CHAR *szPath_);
00342
00348     K_USHORT Delete_Folder(const K_CHAR *szPath_);
00349
00356     void Cleanup_Node_Links(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00357
00364     K_USHORT Find_Parent_Dir(const K_CHAR *szPath_);
00365
00371     K_USHORT Find_File(const K_CHAR *szPath_);
00372
00376     void Print(void);
00377
00382     K_ULONG GetBlockSize(void) { return m_stLocalRoot.
        ulBlockSize; }
00383
00388     K_ULONG GetNumBlocks(void) { return m_stLocalRoot.
        ulNumBlocks; }
00389
00395     K_ULONG GetNumBlocksFree(void) { return m_stLocalRoot.
        ulNumBlocksFree; }
00396
00401     K_ULONG GetNumFiles(void) { return m_stLocalRoot.
        usNumFiles; }
00402
00407     K_USHORT GetNumFilesFree(void) { return m_stLocalRoot.
        usNumFilesFree; }
00408
00409
00417     K_USHORT GetFirstChild(K_USHORT usNode_);
00418

```

```

00424     K_USHORT GetNextPeer( K_USHORT usNode_ );
00425
00432     K_BOOL GetStat( K_USHORT usNode_, NLFS_File_Stat_t *pstStat_);
00433
00434     protected:
00435
00442     K_CHAR Find_Last_Slash(const K_CHAR *szPath_);
00443
00451     K_BOOL File_Names_Match(const K_CHAR *szPath_, NLFS_Node_t *pstNode_);
00452
00459     virtual void Read_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_) = 0;
00460
00467     virtual void Write_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_) = 0;
00468
00475     virtual void Read_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstBlock_) = 0;
00476
00483     virtual void Write_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstFileBlock_) = 0;
00484
00494     virtual void Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_) =
0;
00495
00506     virtual void Write_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_)
= 0;
00507
00514     void RootSync();
00515
00520     void Repair() {}
00521
00526     void Print_Free_Details( K_USHORT usNode_);
00527
00528
00533     void Print_File_Details(K_USHORT usNode_);
00534
00539     void Print_Dir_Details(K_USHORT usNode_);
00540
00546     void Print_Node_Details(K_USHORT usNode_);
00547
00552     void Push_Free_Node(K_USHORT usNode_);
00553
00558     K_USHORT Pop_Free_Node(void);
00559
00565     void Push_Free_Block(K_ULONG ulBlock_);
00566
00572     K_ULONG Pop_Free_Block(void);
00573
00579     K_ULONG Append_Block_To_Node(NLFS_Node_t *pstFile_);
00580
00587     K_USHORT Create_File_i(const K_CHAR *szPath_, NLFS_Type_t eType_);
00588
00594     void Set_Node_Name( NLFS_Node_t *pstFileNode_, const K_CHAR *szPath_ );
00595
00596     NLFS_Host_t *m_puHost;
00597     NLFS_Root_Node_t m_stLocalRoot;
00598 };
00599
00600 #endif

```

15.125 /home/moslevin/Project/R1/stage/src/nlfs_config.h File Reference

NLFS configuration parameters.

Macros

- #define **DEBUG** 0
- #define **DEBUG_PRINT**(...)

15.125.1 Detailed Description

NLFS configuration parameters.

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


```

00027 {
00028     K_USHORT usNode;
00029     usNode = pclFS_>Find_File(szPath_);
00030
00031     if (INVALID_NODE == usNode)
00032     {
00033         DEBUG_PRINT("file does not exist in path\n");
00034         if (eMode_ & NLFS_FILE_CREATE)
00035         {
00036             DEBUG_PRINT("Attempt to create\n");
00037             usNode = pclFS_>Create_File(szPath_);
00038             if (INVALID_NODE == usNode)
00039             {
00040                 DEBUG_PRINT("unable to create node in path\n");
00041                 return -1;
00042             }
00043         }
00044         else
00045         {
00046             return -1;
00047         }
00048     }
00049
00050     DEBUG_PRINT("Current Node: %d\n", usNode);
00051
00052     m_pclFileSystem = pclFS_;
00053     m_pclFileSystem->Read_Node(usNode, &m_stNode);
00054
00055     m_usFile = usNode;
00056
00057     if (eMode_ & NLFS_FILE_APPEND)
00058     {
00059         if (!(eMode_ & NLFS_FILE_WRITE))
00060         {
00061             DEBUG_PRINT("Open file for append in read-only mode? Why!\n");
00062             return -1;
00063         }
00064         if (-1 == Seek(m_stNode.stFileNode.ulFileSize))
00065         {
00066             DEBUG_PRINT("file open failed - error seeking to EOF for append\n");
00067             return -1;
00068         }
00069     }
00070
00071     else if (eMode_ & NLFS_FILE_TRUNCATE)
00072     {
00073         if (!(eMode_ & NLFS_FILE_WRITE))
00074         {
00075             DEBUG_PRINT("Truncate file in read-only mode? Why!\n");
00076             return -1;
00077         }
00078
00079         K_ULONG ulCurr = m_stNode.stFileNode.ulFirstBlock;
00080         K_ULONG ulPrev = ulCurr;
00081
00082         // Go through and clear all blocks allocated to the file
00083         while (INVALID_BLOCK != ulCurr)
00084         {
00085             NLFS_Block_t stBlock;
00086             pclFS_>Read_Block_Header(ulCurr, &stBlock);
00087
00088             ulPrev = ulCurr;
00089             ulCurr = stBlock.ulNextBlock;
00090
00091             pclFS_>Push_Free_Block(ulPrev);
00092         }
00093
00094         m_ulOffset = 0;
00095         m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00096     }
00097     else
00098     {
00099         // Open file to beginning of file, regardless of mode.
00100         m_ulOffset = 0;
00101         m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00102     }
00103
00104     m_ucFlags = eMode_;
00105
00106     DEBUG_PRINT("Current Block: %d\n", m_ulCurrentBlock);
00107     DEBUG_PRINT("file open OK\n");
00108     return 0;
00109 }
00110
00111 //-----

```

```

00112 int NLFS_File::Seek(K_ULONG ulOffset_)
00113 {
00114     NLFS_Block_t stBlock;
00115     m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00116     m_ulOffset = ulOffset_;
00117
00118     if (INVALID_NODE == m_usFile)
00119     {
00120         DEBUG_PRINT("Error - invalid file");
00121         return -1;
00122     }
00123
00124     if (INVALID_BLOCK == m_ulCurrentBlock)
00125     {
00126         DEBUG_PRINT("Invalid block\n");
00127         m_ulOffset = 0;
00128         return -1;
00129     }
00130
00131     m_pclFileSystem->Read_Block_Header(
m_ulCurrentBlock, &stBlock);
00132
00133     while (ulOffset_ >= m_pclFileSystem->GetBlockSize())
00134     {
00135         ulOffset_ -= m_pclFileSystem->GetBlockSize();
00136         m_ulCurrentBlock = stBlock.ulNextBlock;
00137         if ((ulOffset_) && (INVALID_BLOCK == m_ulCurrentBlock))
00138         {
00139             m_ulCurrentBlock = m_stNode.stFileNode.
ulFirstBlock;
00140             m_ulOffset = 0;
00141             return -1;
00142         }
00143         m_pclFileSystem->Read_Block_Header(
m_ulCurrentBlock, &stBlock);
00144     }
00145
00146     m_ulOffset = ulOffset_;
00147     return 0;
00148 }
00149
00150 //-----
00151 int NLFS_File::Read(void *pvBuf_, K_ULONG ulLen_)
00152 {
00153     K_ULONG ulBytesLeft;
00154     K_ULONG ulOffset;
00155     K_ULONG ulRead = 0;
00156     K_BOOL bBail = false;
00157
00158     K_CHAR *szCharBuf = (K_CHAR*)pvBuf_;
00159
00160     if (INVALID_NODE == m_usFile)
00161     {
00162         DEBUG_PRINT("Error - invalid file");
00163         return -1;
00164     }
00165
00166     if (!(NLFS_FILE_READ & m_ucFlags))
00167     {
00168         DEBUG_PRINT("Error - file not open for read\n");
00169         return -1;
00170     }
00171
00172     DEBUG_PRINT("Reading: %d bytes from file\n", ulLen_);
00173     while (ulLen_ && !bBail)
00174     {
00175         ulOffset = m_ulOffset & (m_pclFileSystem->
GetBlockSize() - 1);
00176         ulBytesLeft = m_pclFileSystem->GetBlockSize() - ulOffset;
00177         if (ulBytesLeft > ulLen_)
00178         {
00179             ulBytesLeft = ulLen_;
00180         }
00181         if (m_ulOffset + ulBytesLeft >= m_stNode.stFileNode.
ulFileSize)
00182         {
00183             ulBytesLeft = m_stNode.stFileNode.ulFileSize -
m_ulOffset;
00184             bBail = true;
00185         }
00186
00187         DEBUG_PRINT(" %d bytes left in block, %d len, %x block\n", ulBytesLeft, ulLen_,
m_ulCurrentBlock);
00188         if (ulBytesLeft && ulLen_ && (INVALID_BLOCK != m_ulCurrentBlock))
00189         {
00190             m_pclFileSystem->Read_Block(

```

```

        m_ulCurrentBlock, ulOffset, (void*)szCharBuf, ulBytesLeft );
00191
00192         ulRead += ulBytesLeft;
00193         ulLen_ -= ulBytesLeft;
00194         szCharBuf += ulBytesLeft;
00195         m_ulOffset += ulBytesLeft;
00196         DEBUG_PRINT( "%d bytes to go\n", ulLen_);
00197     }
00198     if (ulLen_)
00199     {
00200         DEBUG_PRINT("reading next node\n");
00201         NLFS_Block_t stBlock;
00202         m_pclFileSystem->Read_Block_Header(
00203             m_ulCurrentBlock, &stBlock);
00204         m_ulCurrentBlock = stBlock.ulNextBlock;
00205     }
00206     if (INVALID_BLOCK == m_ulCurrentBlock)
00207     {
00208         break;
00209     }
00210 }
00211 }
00212 DEBUG_PRINT("Return :%d bytes read\n", ulRead);
00213 return ulRead;
00214 }
00215
00216 //-----
00217 int NLFS_File::Write(void *pvBuf_, K_ULONG ulLen_)
00218 {
00219     K_ULONG ulBytesLeft;
00220     K_ULONG ulOffset;
00221     K_ULONG ulWritten = 0;
00222     K_CHAR *szCharBuf = (K_CHAR*)pvBuf_;
00223
00224     if (INVALID_NODE == m_usFile)
00225     {
00226         DEBUG_PRINT("Error - invalid file");
00227         return -1;
00228     }
00229
00230     if (!(NLFS_FILE_WRITE & m_ucFlags))
00231     {
00232         DEBUG_PRINT("Error - file not open for write\n");
00233         return -1;
00234     }
00235
00236     DEBUG_PRINT("writing: %d bytes to file\n", ulLen_);
00237     while (ulLen_)
00238     {
00239         ulOffset = m_ulOffset & (m_pclFileSystem->
00240             GetBlockSize() - 1);
00241         ulBytesLeft = m_pclFileSystem->GetBlockSize() - ulOffset;
00242         if (ulBytesLeft > ulLen_)
00243         {
00244             ulBytesLeft = ulLen_;
00245         }
00246         if (ulBytesLeft && ulLen_ && (INVALID_BLOCK != m_ulCurrentBlock))
00247         {
00248             m_pclFileSystem->Write_Block(
00249                 m_ulCurrentBlock, ulOffset, (void*)szCharBuf, ulBytesLeft );
00250             ulWritten += ulBytesLeft;
00251             ulLen_ -= ulBytesLeft;
00252             szCharBuf += ulBytesLeft;
00253             m_stNode.stFileNode.ulFileSize += ulBytesLeft;
00254             m_ulOffset += ulBytesLeft;
00255             DEBUG_PRINT( "%d bytes to go\n", ulLen_);
00256         }
00257         if (!ulLen_)
00258         {
00259             m_pclFileSystem->Write_Node(m_usFile, &
00260                 m_stNode);
00261         }
00262         else
00263         {
00264             DEBUG_PRINT("appending\n");
00265             m_ulCurrentBlock = m_pclFileSystem->
00266                 Append_Block_To_Node(&m_stNode);
00267         }
00268         DEBUG_PRINT("writing node to file\n");
00269         m_pclFileSystem->Write_Node(m_usFile, &
00270             m_stNode);
00271     }
00272     return ulWritten;
00273 }
00274 }
00275
00276

```

```

00271 //-----
00272 int NLFS_File::Close(void)
00273 {
00274     m_usFile = INVALID_NODE;
00275     m_ulCurrentBlock = INVALID_BLOCK;
00276     m_ulOffset = 0;
00277     m_ucFlags = 0;
00278     return 0;
00279 }

```

15.129 /home/moslevin/Project/R1/stage/src/nlfs_file.h File Reference

NLFS file access class.

```

#include "kerneltypes.h"
#include "nlfs.h"
#include "nlfs_config.h"

```

Classes

- class [NLFS_File](#)
The *NLFS_File* class.

Typedefs

- typedef K_UCHAR **NLFS_File_Mode_t**

Enumerations

- enum [NLFS_File_Mode](#) {
[NLFS_FILE_CREATE](#) = 0x01, [NLFS_FILE_APPEND](#) = 0x02, [NLFS_FILE_TRUNCATE](#) = 0x04, [NLFS_FILE_READ](#) = 0x08,
[NLFS_FILE_WRITE](#) = 0x10 }

15.129.1 Detailed Description

NLFS file access class.

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

15.129.2 Enumeration Type Documentation

15.129.2.1 enum NLFS_File_Mode

Enumerator

NLFS_FILE_CREATE Create the file if it does not exist.
NLFS_FILE_APPEND Open to end of file.
NLFS_FILE_TRUNCATE Truncate file size to 0-bytes.
NLFS_FILE_READ Open file for read.
NLFS_FILE_WRITE Open file for write.

Definition at line 27 of file [nlfs_file.h](#).

15.130 nlfs_file.h

```

00001  /*=====
00002
00003  _____
00004  |   \   |   |   |   |   |   |   |   |
00005  |   \   |   |   |   |   |   |   |   |
00006  |   \   |   |   |   |   |   |   |   |
00007  |   \   |   |   |   |   |   |   |   |
00008  |   \   |   |   |   |   |   |   |   |
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00019  #ifndef __NLFS_FILE_H
00020  #define __NLFS_FILE_H
00021
00022  #include "kerneltypes.h"
00023  #include "nlfs.h"
00024  #include "nlfs_config.h"
00025
00026  //-----
00027  typedef enum
00028  {
00029      NLFS_FILE_CREATE = 0x01,
00030      NLFS_FILE_APPEND = 0x02,
00031      NLFS_FILE_TRUNCATE = 0x04,
00032      NLFS_FILE_READ = 0x08,
00033      NLFS_FILE_WRITE = 0x10
00034  } NLFS_File_Mode;
00035  typedef K_UCHAR NLFS_File_Mode_t;
00036
00037  //-----
00045  class NLFS_File
00046  {
00047
00048  public:
00056      int      Open(NLFS *pclFS_, const K_CHAR *szPath_, NLFS_File_Mode_t eMode_);
00057
00064      int      Read(void *pvBuf_, K_ULONG ulLen_);
00065
00073      int      Write(void *pvBuf_, K_ULONG ulLen_);
00074
00080      int      Seek(K_ULONG ulOffset_);
00081
00086      int      Close(void);
00087
00088  private:
00089      NLFS      *m_pclFileSystem;
00090      K_ULONG    m_ulOffset;
00091      K_ULONG    m_ulCurrentBlock;
00092      K_USHORT    m_usFile;
00093      NLFS_File_Mode_t m_ucFlags;
00094      NLFS_Node_t m_stNode;
00095  };
00096
00097  #endif // __NLFS_FILE_H

```

15.131 /home/moslevin/Project/R1/stage/src/nlfs_ram.cpp File Reference

RAM-based Nice Little Filesystem (NLFS) driver.

```

#include "nlfs.h"
#include "nlfs_ram.h"
#include "memutil.h"
#include "nlfs_config.h"

```

15.131.1 Detailed Description

RAM-based Nice Little Filesystem (NLFS) driver.

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

15.132 nlfs_ram.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #include "nlfs.h"
00020 #include "nlfs_ram.h"
00021 #include "memutil.h"
00022 #include "nlfs_config.h"
00023
00024 //-----
00025 void NLFS_RAM::Read_Node( K_USHORT usNode_, NLFS_Node_t *pstFileNode_)
00026 {
00027     NLFS_Node_t *pstFileNode = (NLFS_Node_t*)(m_puHost->kaData
00028                                             + (usNode_ * sizeof(
00029 NLFS_Node_t)));
00029
00030     MemUtil::CopyMemory(pstFileNode_, pstFileNode, sizeof(
00031 NLFS_Node_t));
00032 }
00033 //-----
00034 void NLFS_RAM::Write_Node(K_USHORT usNode_, NLFS_Node_t *pstFileNode_)
00035 {
00036     NLFS_Node_t *pstFileNode = (NLFS_Node_t*)(m_puHost->kaData
00037                                             + (usNode_ * sizeof(
00038 NLFS_Node_t)));
00039
00040     MemUtil::CopyMemory(pstFileNode, pstFileNode_, sizeof(
00041 NLFS_Node_t));
00042 }
00043 //-----
00043 void NLFS_RAM::Read_Block_Header(K_ULONG ulBlock_,
00044 NLFS_Block_t *pstFileBlock_)
00045 {
00046     NLFS_Block_t *pstFileBlock = (NLFS_Block_t*)(
00047 m_puHost->kaData
00048                                             + m_stLocalRoot.
00049 ulBlockOffset
00050                                             + (ulBlock_ * sizeof(
00051 NLFS_Block_t)));
00052
00053     MemUtil::CopyMemory(pstFileBlock_, pstFileBlock, sizeof(
00054 NLFS_Block_t));
00055 }
00056 //-----
00053 void NLFS_RAM::Write_Block_Header(K_ULONG ulBlock_,
00054 NLFS_Block_t *pstFileBlock_)
00055 {
00056     NLFS_Block_t *pstFileBlock = (NLFS_Block_t*)(
00057 m_puHost->kaData
00058                                             + m_stLocalRoot.
00059 ulBlockOffset
00060                                             + (ulBlock_ * sizeof(
00061 NLFS_Block_t)));
00062
00063     MemUtil::CopyMemory(pstFileBlock, pstFileBlock_, sizeof(
00064 NLFS_Block_t));
00065 }
00066 //-----
00063 void NLFS_RAM::Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG
00064 ulLen_)
00065 {
00066     void *pvSrc_ = (void*)( m_puHost->kaData
00067                             + m_stLocalRoot.ulDataOffset
00068                             + ulOffset_
00069                             + (ulBlock_ * m_stLocalRoot.ulBlockSize) );
00070     MemUtil::CopyMemory(pvData_, pvSrc_, (K_USHORT)ulLen_);
00071 }
00072 //-----
00073 void NLFS_RAM::Write_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG
00074 ulLen_)

```

```

00074 {
00075     void *pvDst_ = (void*)( m_puHost->kaData
00076                             + m_stLocalRoot.ulDataOffset
00077                             + ulOffset_
00078                             + (ulBlock_ * m_stLocalRoot.ulBlockSize) );
00079     MemUtil::CopyMemory(pvDst_, pvData_, (K_USHORT)ulLen_);
00080 }

```

15.133 /home/moslevin/Project/R1/stage/src/nlfs_ram.h File Reference

RAM-based Nice Little Filesystem (NLFS) driver.

```
#include "nlfs.h"
```

Classes

- class [NLFS_RAM](#)

The *NLFS_RAM* class.

15.133.1 Detailed Description

RAM-based Nice Little Filesystem (NLFS) driver.

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

15.134 nlfs_ram.h

```

00001 /*=====
00002
00003 |-----|-----|-----|-----|-----|-----|-----|-----|
00004 | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / | \ / |
00005 |  / \ |  / \ |  / \ |  / \ |  / \ |  / \ |  / \ |  / \ |
00006 |-----|-----|-----|-----|-----|-----|-----|-----|
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2013 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __NLFS_RAM_H
00020 #define __NLFS_RAM_H
00021
00022 #include "nlfs.h"
00023
00031 class NLFS_RAM : public NLFS
00032 {
00033 private:
00034
00041     virtual void Read_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00042
00049     virtual void Write_Node(K_USHORT usNode_, NLFS_Node_t *pstNode_);
00050
00057     virtual void Read_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstBlock_);
00058
00065     virtual void Write_Block_Header(K_ULONG ulBlock_,
NLFS_Block_t *pstFileBlock_);
00066
00076     virtual void Read_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_);
00077
00088     void Write_Block(K_ULONG ulBlock_, K_ULONG ulOffset_, void *pvData_, K_ULONG ulLen_);
00089
00090 };
00091
00092 #endif // NLFS_RAM_H

```



```

00081     if (m_usIterations)
00082     {
00083         return m_ulCumulative / (K_ULONG)m_usIterations;
00084     }
00085     return 0;
00086 }
00087
00088 //-----
00089 K_ULONG ProfileTimer::GetCurrent()
00090 {
00091
00092     if (m_bActive)
00093     {
00094         K_USHORT usCurrent;
00095         K_ULONG ulEpoch;
00096         CS_ENTER();
00097         usCurrent = Profiler::Read();
00098         ulEpoch = Profiler::GetEpoch();
00099         CS_EXIT();
00100         return ComputeCurrentTicks(usCurrent, ulEpoch);
00101     }
00102     return m_ulCurrentIteration;
00103 }
00104
00105 //-----
00106 K_ULONG ProfileTimer::ComputeCurrentTicks(K_USHORT usCurrent_, K_ULONG
ulEpoch_)
00107 {
00108     K_ULONG ulTotal;
00109     K_ULONG ulOverflows;
00110
00111     ulOverflows = ulEpoch_ - m_ulInitialEpoch;
00112
00113     // More than one overflow...
00114     if (ulOverflows > 1)
00115     {
00116         ulTotal = ((K_ULONG)(ulOverflows-1) * TICKS_PER_OVERFLOW)
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

```

15.139 /home/moslevin/Project/R1/stage/src/profile.h File Reference

High-precision profiling timers.

```

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

```

Classes

- class [ProfileTimer](#)

Profiling timer.

15.139.1 Detailed Description

High-precision profiling timers. Enables the profiling and instrumentation of performance-critical code. Multiple timers can be used simultaneously to enable system-wide performance metrics to be computed in a lightweight manner.

Usage:

```
ProfileTimer clMyTimer;
int i;

clMyTimer.Init();

// Profile the same block of code ten times
for (i = 0; i < 10; i++)
{
    clMyTimer.Start();
    ...
    //Block of code to profile
    ...
    clMyTimer.Stop();
}

// Get the average execution time of all iterations
ulAverageTimer = clMyTimer.GetAverage();

// Get the execution time from the last iteration
ulLastTimer = clMyTimer.GetCurrent();
```

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

15.140 profile.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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;
```



```

00033 #define __FILE_ID__      QUANTUM_CPP
00034
00035 #if KERNEL_USE_QUANTUM
00036
00037 //-----
00038 static volatile K_BOOL bAddQuantumTimer;    // Indicates that a timer add is pending
00039
00040 //-----
00041 Timer Quantum::m_clQuantumTimer;    // The global timernodelist_t object
00042 K_UCHAR Quantum::m_bActive;
00043 K_UCHAR Quantum::m_bInTimer;
00044 //-----
00045 static void QuantumCallback(Thread *pclThread_, void *pvData_)
00046 {
00047     // Validate thread pointer, check that source/destination match (it's
00048     // in its real priority list). Also check that this thread was part of
00049     // the highest-running priority level.
00050     if (pclThread_>GetPriority() >= Scheduler::GetCurrentThread()->
        GetPriority())
00051     {
00052         if (pclThread_>GetCurrent()->GetHead() != pclThread_>
            GetCurrent()->GetTail() )
00053         {
00054             bAddQuantumTimer = true;
00055             pclThread_>GetCurrent()->PivotForward();
00056         }
00057     }
00058 }
00059
00060 //-----
00061 void Quantum::SetTimer(Thread *pclThread_)
00062 {
00063     m_clQuantumTimer.SetIntervalMSeconds(pclThread_>
        GetQuantum());
00064     m_clQuantumTimer.SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00065     m_clQuantumTimer.SetData(NULL);
00066     m_clQuantumTimer.SetCallback((TimerCallback_t)QuantumCallback);
00067     m_clQuantumTimer.SetOwner(pclThread_);
00068 }
00069
00070 //-----
00071 void Quantum::AddThread(Thread *pclThread_)
00072 {
00073     if (m_bActive)
00074     {
00075         return;
00076     }
00077
00078     // If this is called from the timer callback, queue a timer add...
00079     if (m_bInTimer)
00080     {
00081         bAddQuantumTimer = true;
00082         return;
00083     }
00084
00085     // If this isn't the only thread in the list.
00086     if ( pclThread_>GetCurrent()->GetHead() !=
        pclThread_>GetCurrent()->GetTail() )
00087     {
00088
00089
00090         Quantum::SetTimer(pclThread_);
00091         TimerScheduler::Add(&m_clQuantumTimer);
00092         m_bActive = 1;
00093     }
00094 }
00095
00096 //-----
00097 void Quantum::RemoveThread(void)
00098 {
00099     if (!m_bActive)
00100     {
00101         return;
00102     }
00103
00104     // Cancel the current timer
00105     TimerScheduler::Remove(&m_clQuantumTimer);
00106     m_bActive = 0;
00107 }
00108
00109 //-----
00110 void Quantum::UpdateTimer(void)
00111 {
00112     // If we have to re-add the quantum timer (more than 2 threads at the
00113     // high-priority level...)
00114     if (bAddQuantumTimer)
00115     {
00116         // Trigger a thread yield - this will also re-schedule the

```

```

00117         // thread *and* reset the round-robin scheduler.
00118         Thread::Yield();
00119         bAddQuantumTimer = false;
00120     }
00121 }
00122
00123 #endif //KERNEL_USE_QUANTUM

```

15.143 /home/moslevin/Project/R1/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.

15.143.1 Detailed Description

[Thread Quantum](#) declarations for Round-Robin Scheduling.

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

15.144 quantum.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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
00075     static void SetInTimer(void) { m_bInTimer = true; }
00076
00082     static void ClearInTimer(void) { m_bInTimer = false; }
00083

```

```

00084 private:
00096     static void SetTimer( Thread *pclThread_ );
00097
00098     static Timer m_clQuantumTimer;
00099     static K_UCHAR m_bActive;
00100     static K_UCHAR m_bInTimer;
00101 };
00102
00103 #endif //KERNEL_USE_QUANTUM
00104
00105 #endif

```

15.145 /home/moslevin/Project/R1/stage/src/rand_lfsr.h File Reference

Linear Feedback Shift Register based Pseudo-Random Number Generator.

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

Classes

- class [PseudoRandom](#)
The *PseudoRandom* class.

15.145.1 Detailed Description

Linear Feedback Shift Register based Pseudo-Random Number Generator.

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

15.146 rand_lfsr.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2014 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00019 #ifndef __RAND_LFSR_H_
00020 #define __RAND_LFSR_H_
00021
00022 #include "kerneltypes.h"
00023
00024 //-----
00036 //-----
00037 class PseudoRandom
00038 {
00039 public:
00045     PseudoRandom();
00046
00056     void Seed( K_ULONG ulSeed1_, K_ULONG ulSeed2_ );
00057
00065     K_USHORT GetRandom();
00066
00067 private:
00068
00078     K_ULONG Shift( K_ULONG *pulLFSR_, K_ULONG ulMask_ );
00079
00080     K_ULONG m_ulLFSR1;
00081     K_ULONG m_ulLFSR2;
00082
00083 };
00084

```

```
00085 #endif
00086
```

15.147 /home/moslevin/Project/R1/stage/src/scheduler.cpp File Reference

Strict-Priority + Round-Robin thread scheduler implementation.

```
#include "kerneltypes.h"
#include "ll.h"
#include "scheduler.h"
#include "thread.h"
#include "threadport.h"
#include "kernel_debug.h"
```

Macros

- `#define __FILE_ID__ SCHEDULER_CPP`

Variables

- `Thread * g_pstNext`
- `Thread * g_pstCurrent`
- `K_UCHAR g_ucFlag`
- `static const K_UCHAR aucCLZ [16] = {255,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3}`

15.147.1 Detailed Description

Strict-Priority + Round-Robin thread scheduler implementation.

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

15.148 scheduler.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "ll.h"
00024 #include "scheduler.h"
00025 #include "thread.h"
00026 #include "threadport.h"
00027 #include "kernel_debug.h"
00028 //-----
00029 #if defined __FILE_ID__
00030     #undef __FILE_ID__
00031 #endif
00032 #define __FILE_ID__ SCHEDULER_CPP
00033
00034 //-----
00035 Thread *g_pstNext;
00036 Thread *g_pstCurrent;
00037
```



```

00038 //-----
00039 K_BOOL Scheduler::m_bEnabled;
00040 K_BOOL Scheduler::m_bQueuedSchedule;
00041
00042 ThreadList Scheduler::m_clStopList;
00043 ThreadList Scheduler::m_aclPriorities[ NUM_PRIORITIES ];
00044 K_UCHAR Scheduler::m_ucPriFlag;
00045
00046 K_UCHAR g_ucFlag;
00047 //-----
00048 static const K_UCHAR aucCLZ[16] = {255,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3};
00049
00050 //-----
00051 void Scheduler::Init()
00052 {
00053     m_ucPriFlag = 0;
00054     for (int i = 0; i < NUM_PRIORITIES; i++)
00055     {
00056         m_aclPriorities[i].SetPriority(i);
00057         m_aclPriorities[i].SetFlagPointer(&
00058             m_ucPriFlag);
00059     }
00059     g_ucFlag = m_ucPriFlag;
00060     m_bQueuedSchedule = false;
00061 }
00062
00063 //-----
00064 void Scheduler::Schedule()
00065 {
00066     K_UCHAR ucPri = 0;
00067
00068     // Figure out what priority level has ready tasks (8 priorities max)
00069     ucPri = aucCLZ[m_ucPriFlag >> 4 ];
00070     if (ucPri == 0xFF) { ucPri = aucCLZ[m_ucPriFlag & 0x0F]; }
00071     else { ucPri += 4; }
00072
00073     // Get the thread node at this priority.
00074     g_pstNext = (Thread*)( m_aclPriorities[ucPri].GetHead() );
00075     g_ucFlag = m_ucPriFlag;
00076
00077     KERNEL_TRACE_1( STR_SCHEDULE_1, (K_USHORT)g_pstNext->GetID() );
00078 }
00079
00080 //-----
00081 void Scheduler::Add(Thread *pclThread_)
00082 {
00083     m_aclPriorities[pclThread_->GetPriority()].Add(pclThread_);
00084     g_ucFlag = m_ucPriFlag;
00085 }
00086
00087 //-----
00088 void Scheduler::Remove(Thread *pclThread_)
00089 {
00090     m_aclPriorities[pclThread_->GetPriority()].Remove(pclThread_);
00091     g_ucFlag = m_ucPriFlag;
00092 }
00093
00094 //-----
00095 K_BOOL Scheduler::SetScheduler(K_BOOL bEnable_)
00096 {
00097     K_BOOL bRet ;
00098     CS_ENTER();
00099     bRet = m_bEnabled;
00100     m_bEnabled = bEnable_;
00101     // If there was a queued scheduler event, dequeue and trigger an
00102     // immediate Yield
00103     if (m_bEnabled && m_bQueuedSchedule)
00104     {
00105         m_bQueuedSchedule = false;
00106         Thread::Yield();
00107     }
00108     CS_EXIT();
00109     return bRet;
00110 }

```

15.149 /home/moslevin/Project/R1/stage/src/scheduler.h File Reference

[Thread](#) scheduler function declarations.


```

00051 #include "threadport.h"
00052
00053 extern Thread *g_pstNext;
00054 extern Thread *g_pstCurrent;
00055
00056 #define NUM_PRIORITIES (8)
00057 //-----
00062 class Scheduler
00063 {
00064 public:
00070     static void Init();
00071
00079     static void Schedule();
00080
00088     static void Add(Thread *pclThread_);
00089
00098     static void Remove(Thread *pclThread_);
00099
00112     static K_BOOL SetScheduler(K_BOOL bEnable_);
00113
00119     static Thread *GetCurrentThread(){ return g_pstCurrent; }
00120
00127     static Thread *GetNextThread(){ return g_pstNext; }
00128
00137     static ThreadList *GetThreadList(K_UCHAR ucPriority_){ return &
m_aclPriorities[ucPriority_]; }
00138
00145     static ThreadList *GetStopList(){ return &m_clStopList; }
00146
00155     static K_UCHAR IsEnabled(){ return m_bEnabled; }
00156
00157     static void QueueScheduler() { m_bQueuedSchedule = true; }
00158
00159 private:
00161     static K_BOOL m_bEnabled;
00162
00164     static K_BOOL m_bQueuedSchedule;
00165
00167     static ThreadList m_clStopList;
00168
00170     static ThreadList m_aclPriorities[NUM_PRIORITIES];
00171
00173     static K_UCHAR m_ucPriFlag;
00174 };
00175 #endif
00176

```

15.151 /home/moslevin/Project/R1/stage/src/screen.cpp File Reference

Higher level window management framework.

```

#include "kerneltypes.h"
#include "screen.h"
#include "gui.h"
#include "memutil.h"

```

15.151.1 Detailed Description

Higher level window management framework.

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

15.152 screen.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008

```

```

00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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

```

15.153 /home/moslevin/Project/R1/stage/src/screen.h File Reference

Higher level window management framework.

```

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

```

Classes

- class [Screen](#)
- class [ScreenList](#)
- class [ScreenManager](#)

15.153.1 Detailed Description

Higher level window management framework.

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

15.154 screen.h

```

00001  /*-----
00002
00003
00004
00005
00006
00007
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*-/
00014  #ifndef __SCREEN_H__
00015  #define __SCREEN_H__
00016
00017  #include "kerneltypes.h"
00018  #include "gui.h"
00019  #include "ll.h"
00020
00021  //-----
00022  class ScreenList;
00023  class ScreenManager;
00024
00025  //-----
00026  class Screen : public LinkListNode
00027  {
00028  public:
00029      Screen() { ClearNode(); }
00030
00031      void Activate() { Create(); }
00032
00033      void Deactivate() { Destroy(); }
00034
00035      void SetWindowAffinity( const K_CHAR *szWindowName_ );
00036
00037      void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }
00038
00039      const K_CHAR *GetName() { return m_szName; }
00040
00041  protected:
00042      friend class ScreenManager;
00043
00044      void SetManager( ScreenManager *pclScreenManager_ );
00045
00046      const K_CHAR *m_szName;
00047      ScreenManager *m_pclScreenManager;
00048      GuiWindow *m_pclWindow;
00049
00050  private:
00051      virtual void Create() = 0;
00052      virtual void Destroy() = 0;
00053
00054  };
00055
00056  //-----
00057  class ScreenList
00058  {
00059  public:
00060      ScreenList() { m_clList.Init(); }
00061
00062      void Add( Screen *pclScreen_ ) { m_clList.Add(pclScreen_); }
00063
00064      void Remove( Screen *pclScreen_ ) { m_clList.Remove(pclScreen_); }
00065
00066      Screen *GetHead() { return static_cast<Screen*>(
00067          m_clList.GetHead()); }
00068
00069  private:
00070      DoubleLinkedList m_clList;
00071  };
00072
00073  //-----
00074  class ScreenManager
00075  {
00076  public:
00077      ScreenManager() { m_pclSurface = NULL; }
00078
00079      void AddScreen( Screen *pclScreen_ ) { m_clScreenList.
00080          Add(pclScreen_);
00081
00082          pclScreen_>SetManager(this); }
00083
00084  };
00085
00086  */

```



```

00043         i++;
00044     }
00045     return 0;
00046 }
00047
00048 //-----
00049 void ShellSupport::UnescapeToken( Token_t *pstToken_, K_CHAR *szDest_ )
00050 {
00051     const K_CHAR *szSrc = pstToken_>pcToken;
00052     int i;
00053     int j = 0;
00054     for (i = 0; i < pstToken_>ucLen; i++)
00055     {
00056         //-- Escape characters
00057         if ('\\' == szSrc[i])
00058         {
00059             i++;
00060             if (i >= pstToken_>ucLen)
00061             {
00062                 break;
00063             }
00064             switch (szSrc[i])
00065             {
00066                 case 't':
00067                     szDest_[j++] = '\t';
00068                     break;
00069                 case 'r':
00070                     szDest_[j++] = '\r';
00071                     break;
00072                 case 'n':
00073                     szDest_[j++] = '\n';
00074                     break;
00075                 case ' ':
00076                     szDest_[j++] = ' ';
00077                     break;
00078                 case '\\':
00079                     szDest_[j++] = '\\';
00080                     break;
00081                 case '\"':
00082                     szDest_[j++] = '\"';
00083                     break;
00084                 default:
00085                     break;
00086             }
00087             //-- Unescaped quotes
00088             else if ('\"' == szSrc[i])
00089             {
00090                 continue;
00091             }
00092             //-- Everything else
00093             else
00094             {
00095                 szDest_[j++] = szSrc[i];
00096             }
00097         }
00098         //-- Null-terminate the string
00099         szDest_[j] = '\0';
00100     }
00101 }
00102
00103 //-----
00104 Option_t *ShellSupport::CheckForOption(
    CommandLine_t *pstCommand_, const K_CHAR *szOption_ )
00105 {
00106     K_CHAR i;
00107     K_UCHAR tmp_len;
00108     for (i = 0; i < pstCommand_>ucNumOptions; i++)
00109     {
00110         tmp_len = MIN(MemUtil::StringLength(szOption_), pstCommand_>
astOptions[i].pstStart->ucLen);
00111         if (true == MemUtil::CompareMemory( (const void*)szOption_,
00112                                             (const void*)(pstCommand_>astOptions[i].
pstStart->pcToken),
00113                                             tmp_len ) )
00114         {
00115             {
00116                 return &(pstCommand_>astOptions[i]);
00117             }
00118         }
00119     }
00120     return 0;
00121 }
00122 //-----
00123 K_CHAR ShellSupport::TokensToCommandLine(
    Token_t *pastTokens_, K_UCHAR ucTokens_, CommandLine_t *pstCommand_)
00124 {
00125     K_CHAR count = 0;

```

```

00126     K_CHAR token = 0;
00127     K_CHAR option = 0;
00128     pstCommand_>ucNumOptions = 0;
00129
00130     if (!ucTokens_)
00131     {
00132         return -1;
00133     }
00134
00135     // Command is a single token...
00136     pstCommand_>pstCommand = &pastTokens_[0];
00137
00138     // Parse out options
00139     token = 1;
00140     while (token < ucTokens_ && option < 12)
00141     {
00142         pstCommand_>astOptions[option].pstStart = &pastTokens_[token];
00143         count = 1;
00144         token++;
00145         while (token < ucTokens_ && pastTokens_[token].pcToken[0] != '-')
00146         {
00147             token++;
00148             count++;
00149         }
00150         pstCommand_>astOptions[option].ucCount = count;
00151         option++;
00152     }
00153
00154     pstCommand_>ucNumOptions = option;
00155     pstCommand_>ucTokenCount = ucTokens_;
00156     pstCommand_>pastTokenList = pastTokens_;
00157     return option;
00158 }

```

15.157 /home/moslevin/Project/R1/stage/src/shell_support.h File Reference

Support functions & data structures useful in implementing a shell.

```

#include "kerneltypes.h"
#include "memutil.h"

```

Classes

- struct [Option_t](#)
Structure used to represent a command-line option with its arguments.
- struct [CommandLine_t](#)
Structure containing multiple representations for command-line data.
- struct [ShellCommand_t](#)
Data structure defining a lookup table correlating a command name to its handler function.
- class [ShellSupport](#)
The [ShellSupport](#) class features utility functions which handle token processing, option/parameter lookup, and functions making it generally trivial to implement a lightweight custom shell.

Macros

- `#define MIN(x, y) (((x) < (y)) ? (x) : (y))`
Utility macro used to return the lesser of two values/objects.
- `#define MAX(x, y) (((x) > (y)) ? (x) : (y))`
Utility macro used to return the greater of two values/objects.

Typedefs

- typedef `K_CHAR(* fp_internal_command)(CommandLine_t *pstCommandLine_)`
Function pointer type used to represent shell commands, as implemented by users of this infrastructure.

Generated on Thu Mar 5 2015 21:04:26 for Mark3 Realtime Kernel by Doxygen

```

00097     while (usIdx < usSize)
00098     {
00099         usIdx += m_pclDriver->Write(usSize, &aucBuf[usIdx]);
00100     }
00101 }
00102
00103 //-----
00104 K_USHORT Slip::ReadData(K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_)
00105 {
00106     K_USHORT usReadCount;
00107     K_UCHAR ucTempCount;
00108     K_USHORT usValid = 0;
00109     K_USHORT usCRC;
00110     K_USHORT usCRC_Calc = 0;
00111     K_USHORT usLen;
00112     K_UCHAR *pucSrc = (K_UCHAR*)aucBuf_;
00113     K_UCHAR *pucDst = (K_UCHAR*)aucBuf_;
00114
00115     usReadCount = m_pclDriver->Read(usLen_, (K_UCHAR*)aucBuf_);
00116
00117     while (usReadCount)
00118     {
00119         K_UCHAR ucRead;
00120         ucTempCount = DecodeByte(&ucRead, pucSrc);
00121
00122         *pucDst = ucRead;
00123
00124         // Encountered a FRAMING_BYTE - end of message
00125         if (!ucTempCount)
00126         {
00127             break;
00128         }
00129
00130         // Add to the CRC
00131         usCRC_Calc += ucRead;
00132
00133         // Adjust iterators, source, and destination pointers.
00134         usReadCount -= ucTempCount;
00135         pucSrc += ucTempCount;
00136         pucDst++;
00137         usValid++;
00138     }
00139
00140     // Ensure we have enough data to try a match.
00141     if (usValid < 5) {
00142         return 0;
00143     }
00144
00145     usCRC_Calc -= aucBuf_[usValid-2];
00146     usCRC_Calc -= aucBuf_[usValid-1];
00147
00148     usLen = ((K_USHORT)aucBuf_[1]) << 8;
00149     usLen += ((K_USHORT)aucBuf_[2]);
00150     usCRC = ((K_USHORT)aucBuf_[usValid-2]) << 8;
00151     usCRC += ((K_USHORT)aucBuf_[usValid-1]);
00152
00153     if (usCRC != usCRC_Calc)
00154     {
00155         return 0;
00156     }
00157
00158     *pucChannel_ = aucBuf_[0];
00159
00160     return usLen;
00161 }
00162
00163 //-----
00164 void Slip::WriteData(K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_)
00165 {
00166     K_UCHAR aucTmp[2];
00167     K_USHORT usCRC = 0;
00168
00169     // Lightweight protocol built on-top of SLIP.
00170     // 1) Channel ID (8-bit)
00171     // 2) Data Size (16-bit)
00172     // 3) Data blob
00173     // 4) CRC16 (16-bit)
00174     aucTmp[0] = FRAMING_BYTE;
00175     while( !m_pclDriver->Write(1, aucTmp) ) {}
00176
00177     if (!usLen_) // Read to end-of-line (\0)
00178     {
00179         K_UCHAR *pucBuf = (K_UCHAR*)aucBuf_;
00180         while (*pucBuf != '\0')
00181         {
00182             usLen_++;
00183             pucBuf++;

```

```

00184     }
00185 }
00186
00187 WriteByte(ucChannel_);
00188 usCRC = ucChannel_;
00189
00190 WriteByte((K_UCHAR) (usLen_ >> 8));
00191 usCRC += (usLen_ >> 8);
00192
00193 WriteByte((K_UCHAR) (usLen_ & 0x00FF));
00194 usCRC += (usLen_ & 0x00FF);
00195
00196 while (usLen_--)
00197 {
00198     WriteByte(*aucBuf_);
00199     usCRC += (K_USHORT)*aucBuf_;
00200     aucBuf_++;
00201 }
00202
00203 WriteByte((K_UCHAR) (usCRC >> 8));
00204 WriteByte((K_UCHAR) (usCRC & 0x00FF));
00205
00206 aucTmp[0] = FRAMING_BYTE;
00207 while( !m_pclDriver->Write(1, aucTmp) ) {}
00208 }
00209
00210 //-----
00211 void Slip::SendAck()
00212 {
00213     WriteByte(ACchar);
00214 }
00215
00216 //-----
00217 void Slip::SendNack()
00218 {
00219     WriteByte(NACchar);
00220 }
00221
00222 //-----
00223 void Slip::WriteVector(K_UCHAR ucChannel_, SlipDataVector *astData_,
00224     K_USHORT usLen_)
00225 {
00226     K_UCHAR aucTmp[2];
00227     K_USHORT usCRC = 0;
00228     K_UCHAR i, j;
00229     K_USHORT usTotalLen = 0;
00230
00231     // Calculate the total length of all message fragments
00232     for (i = 0; i < usLen_; i++)
00233     {
00234         usTotalLen += astData_[i].ucSize;
00235     }
00236
00237     // Send a FRAMING_BYTE to start framing a message
00238     aucTmp[0] = FRAMING_BYTE;
00239     while( !m_pclDriver->Write(1, aucTmp) ) {}
00240
00241     // Write a the channel
00242     WriteByte(ucChannel_);
00243     usCRC = ucChannel_;
00244
00245     // Write the length
00246     WriteByte((K_UCHAR) (usTotalLen >> 8));
00247     usCRC += (usTotalLen >> 8);
00248
00249     WriteByte((K_UCHAR) (usTotalLen & 0x00FF));
00250     usCRC += (usTotalLen & 0x00FF);
00251
00252     // Write the message fragments
00253     for (i = 0; i < usLen_; i++)
00254     {
00255         K_UCHAR *aucBuf = astData_[i].pucData;
00256         for (j = 0; j < astData_[i].ucSize; j++)
00257         {
00258             WriteByte(*aucBuf);
00259             usCRC += (K_USHORT)*aucBuf;
00260             aucBuf++;
00261         }
00262     }
00263
00264     // Write the CRC
00265     WriteByte((K_UCHAR) (usCRC >> 8));
00266     WriteByte((K_UCHAR) (usCRC & 0x00FF));
00267
00268     // Write the end-of-message
00269     aucTmp[0] = FRAMING_BYTE;
00270     while( !m_pclDriver->Write(1, aucTmp) ) {}

```

```
00270 }
```

15.161 /home/moslevin/Project/R1/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](#) }

15.161.1 Detailed Description

Serial Line IP framing code. Also includes code to frame data in FunkenSlip format for use with [SlipTerm](#) on a host PC.

FunkenSlip uses SLIP-framed messages with a pre-defined packet format as follows:

[Channel][Size][Data Buffer][CRC8]

Channel is 1 byte, indicating the type of data carried in the message

Size is 2 bytes, indicating the length of the binary blob that follows

Data Buffer is n bytes, and contains the raw packet data.

CRC16 is 2 byte, Providing an error detection mechanism

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

15.161.2 Enumeration Type Documentation

15.161.2.1 enum SlipChannel

Enumerator

- [SLIP_CHANNEL_TERMINAL](#)** ASCII text mode terminal.
- [SLIP_CHANNEL_UNISCOPE](#)** Uniscope VM command channel.
- [SLIP_CHANNEL_NVM](#)** Non-volatile memory configuration.
- [SLIP_CHANNEL_RESET](#)** Channel used to reset the device...
- [SLIP_CHANNEL_GRAPHICS](#)** Encoded drawing commands.
- [SLIP_CHANNEL_HID](#)** HID commands.

Definition at line 41 of file [slip.h](#).

15.162 slip.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00034 #include "kerneltypes.h"
00035 #include "driver.h"
00036
00037 #ifndef __SLIP_H__
00038 #define __SLIP_H__
00039
00040 //-----
00041 typedef enum
00042 {
00043     SLIP_CHANNEL_TERMINAL = 0,
00044     SLIP_CHANNEL_UNISCOPE,
00045     SLIP_CHANNEL_NVM,
00046     SLIP_CHANNEL_RESET,
00047     SLIP_CHANNEL_GRAPHICS,
00048     SLIP_CHANNEL_HID,
00049 } SLIP_CHANNEL_COUNT;
00050 } SlipChannel;
00051
00052 //-----
00053 typedef struct
00054 {
00055     K_UCHAR ucSize;
00056     K_UCHAR *pucData;
00057 }SlipDataVector;
00058
00059 //-----
00070 class Slip
00071 {
00072 public:
00073     void SetDriver( Driver *pclDriver_ ){ m_pclDriver = pclDriver_; }
00074     Driver *GetDriver() { return m_pclDriver; }
00075     static K_USHORT EncodeByte( K_UCHAR ucChar_, K_UCHAR *aucBuf_ );
00076     static K_USHORT DecodeByte( K_UCHAR *ucChar_, const K_UCHAR *aucBuf_ );
00077     void WriteData( K_UCHAR ucChannel_, const K_CHAR *aucBuf_, K_USHORT usLen_ );
00078     K_USHORT ReadData( K_UCHAR *pucChannel_, K_CHAR *aucBuf_, K_USHORT usLen_ );
00079     void WriteVector( K_UCHAR ucChannel_, SlipDataVector *astData_, K_USHORT
usLen_ );
00080     void SendAck();
00081     void SendNack();
00082 private:
00083     void WriteByte(K_UCHAR ucData_);
00084     Driver *m_pclDriver;
00085 };
00086 #endif

```

15.163 /home/moslevin/Project/R1/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_)

15.163.1 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](#).

15.163.2 Function Documentation

15.163.2.1 static void SlipMux_CallBack ([Driver](#) * [pclDriver_](#)) [static]

Parameters

pclDriver_	Pointer to the driver data for the port triggering the callback
----------------------------	---

Definition at line 43 of file [slip_mux.cpp](#).

15.164 slip_mux.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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(const K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT
    usTxSize_, K_UCHAR *aucTx_)
00060 {
00061     m_pclDriver = DriverList::FindByPath(pcDriverPath_);
00062     m_pclMessageQueue = NULL;
00063
00064     m_clSlip.SetDriver(m_pclDriver);
00065     m_clSlipSem.Init(0, 1);
00066
00067     m_pclDriver->Control(CMD_SET_BUFFERS, (void*)aucRx_, usRxSize_, (void*)aucTx_, usTxSize_);
00068     m_pclDriver->Control(CMD_SET_RX_CALLBACK, (void*)SlipMux_CallBack, 0, 0, 0);
00069     {
00070         K_UCHAR ucEscape = 192;
00071         m_pclDriver->Control(CMD_SET_RX_ESCAPE, (void*)&ucEscape, 1, 0, NULL);
00072     }
00073 }
00074
00075 //-----
00076 void SlipMux::InstallHandler( K_UCHAR ucChannel_, Slip_Channel pfHandler_ )
00077 {
00078     if (pfHandler_)
00079     {
00080         m_apfChannelHandlers[ucChannel_] = pfHandler_;
00081     }
00082 }
00083
00084 //-----
00085 void SlipMux::MessageReceive(void)
00086 {
00087     K_USHORT usLen;
00088     K_UCHAR ucChannel;
00089
00090     usLen = m_clSlip.ReadData( &ucChannel, (K_CHAR*)m_aucData, SLIP_BUFFER_SIZE );
00091     if (usLen && (m_apfChannelHandlers[ucChannel] != NULL))
00092     {
00093         m_apfChannelHandlers[ucChannel]( m_pclDriver, ucChannel, &(m_aucData[3]), usLen);
00094     }
00095
00096     // Re-enable the driver once we're done.
00097     m_pclDriver->Control( CMD_SET_RX_ENABLE, 0, 0, 0, 0 );
00098 }
00099

```

15.165 /home/moslevin/Project/R1/stage/src/slip_mux.h File Reference

FunkenSlip Channel Multiplexer.

```

#include "kerneltypes.h"
#include "driver.h"
#include "ksemaphore.h"
#include "message.h"
#include "slip.h"

```

Classes

- class [SlipMux](#)

Static-class which implements a multiplexed stream of SLIP data over a single interface.

Macros

- #define **SLIP_BUFFER_SIZE** (32)
- #define **SLIP_RX_MESSAGE_ID** (0xD00D)

Typedefs

- typedef void(* **Slip_Channel**)(Driver *pclDriver_, K_UCHAR ucChannel_, K_UCHAR *pucData_, K_USHORT usLen_)

15.165.1 Detailed Description

FunkenSlip Channel Multiplexer. Demultiplexes FunkenSlip packets transmitted over a single serial channel

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

15.166 slip_mux.h

```

00001  /*=====
00002
00003  _____|_____|_____|_____|_____|_____
00004  |         / \      / \      / \      / \      / \
00005  |        /   \    /   \    /   \    /   \    /   \
00006  |       /_____\  /_____\  /_____\  /_____\  /_____\
00007  |_____/         \_____/         \_____/         \_____/
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00021  #include "kerneltypes.h"
00022  #include "driver.h"
00023  #include "ksemaphore.h"
00024  #include "message.h"
00025  #include "slip.h"
00026
00027  #ifndef __SLIP_MUX_H__
00028  #define __SLIP_MUX_H__
00029
00030  //-----
00031  #define SLIP_BUFFER_SIZE      (32)
00032
00033  #define SLIP_RX_MESSAGE_ID    (0xD00D)
00034
00035  //-----
00036  typedef void (*Slip_Channel)( Driver *pclDriver_, K_UCHAR ucChannel_, K_UCHAR *pucData_, K_USHORT
usLen_ );
00037
00038  //-----
00043  class SlipMux
00044  {
00045  public:
00065      static void Init(const K_CHAR *pcDriverPath_, K_USHORT usRxSize_, K_UCHAR *aucRx_, K_USHORT
usTxSize_, K_UCHAR *aucTx_);
00066
00075      static void InstallHandler( K_UCHAR ucChannel_, Slip_Channel pfHandler_ );
00076
00084      static void MessageReceive();
00085
00091      static Driver *GetDriver(){ return m_pclDriver; }
00092
00099      static MessageQueue *GetQueue(){ return m_pclMessageQueue; }
00100
00108      static void SetQueue( MessageQueue *pclMessageQueue_ )
00109      { m_pclMessageQueue = pclMessageQueue_; }
00110
00111
00117      static Slip *GetSlip(){ return &m_clSlip; }
00118
00119  private:
00120      static MessageQueue *m_pclMessageQueue;
00121      static Driver *m_pclDriver;
00122      static Slip_Channel m_apfChannelHandlers[SLIP_CHANNEL_COUNT];
00123      static K_UCHAR m_aucData[SLIP_BUFFER_SIZE];
00124      static Semaphore m_clSlipSem;
00125      static Slip m_clSlip;
00126  };
00127
00128  #endif

```

15.167 /home/moslevin/Project/R1/stage/src/slipterm.cpp File Reference

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

```
#include "kerneltypes.h"
#include "slip.h"
#include "slipterm.h"
```

15.167.1 Detailed Description

Serial debug interface using SLIP protocol, and FunkenSlip multiplexing.

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

15.168 slipterm.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "slip.h"
00023 #include "slipterm.h"
00024
00025 //-----
00026 void SlipTerm::Init()
00027 {
00028     m_clSlip.SetDriver( DriverList::FindByPath("/dev/tty" ) );
00029
00030     m_ucVerbosity = SEVERITY_DEBUG;
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 void SlipTerm::PrintLn( const char *szLine_ )
00044 {
00045     SlipDataVector astData[2];
00046     astData[0].pucData = (K_UCHAR*)szLine_;
00047     astData[0].ucSize = StrLen(szLine_);
00048     astData[1].pucData = (K_UCHAR*)"\\r\\n";
00049     astData[1].ucSize = 2;
00050
00051     m_clSlip.WriteVector(SLIP_CHANNEL_TERMINAL, astData, 2);
00052 }
00053 //-----
00054 void SlipTerm::PrintLn( K_UCHAR ucSeverity_, const char *szLine_ )
00055 {
00056     if (ucSeverity_ <= m_ucVerbosity)
00057     {
00058         PrintLn( szLine_ );
00059     }
00060 }
00061
00062 }
```

15.169 /home/moslevin/Project/R1/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__**

15.169.1 Detailed Description

Serial debug interface using SLIP serial, and Funkenslip serial port multiplexing.

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

15.170 slipterm.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "driver.h"
00023 #include "slip.h"
00024
00025 //-----
00026 #define SEVERITY_DEBUG          4
00027 #define SEVERITY_INFO          3
00028 #define SEVERITY_WARN          2
00029 #define SEVERITY_CRITICAL      1
00030 #define SEVERITY_CATASTROPHIC  0
00031
00032 //-----
00033 #ifndef __SLIPTERM_H__
00034 #define __SLIPTERM_H__
00035
00040 class SlipTerm
00041 {
00042 public:
00050     void Init();
00051
00060     void PrintLn( const char *szLine_ );
00061
```

```

00072     void PrintLn( K_UCHAR ucSeverity_, const char *szLine_ );
00073
00081     void SetVerbosity( K_UCHAR ucLevel_ ) { m_ucVerbosity = ucLevel_; }
00082 private:
00090     K_USHORT StrLen( const char *szString_ );
00091
00092     K_UCHAR m_ucVerbosity;
00093
00095     Slip m_clSlip;
00096 };
00097
00098 #endif

```

15.171 /home/moslevin/Project/R1/stage/src/system_heap.cpp File Reference

Global system-heap implementation.

```

#include "kerneltypes.h"
#include "system_heap_config.h"
#include "system_heap.h"

```

15.171.1 Detailed Description

Global system-heap implementation. Provides a system-wide malloc/free paradigm allocation scheme.

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

15.172 system_heap.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00021 #include "kerneltypes.h"
00022 #include "system_heap_config.h"
00023 #include "system_heap.h"
00024
00025 #if USE_SYSTEM_HEAP
00026
00027 //-----
00028 K_UCHAR SystemHeap::m_pucRawHeap[ HEAP_RAW_SIZE ];
00029 HeapConfig SystemHeap::m_pclSystemHeapConfig[
00030     HEAP_NUM_SIZES + 1];
00031 FixedHeap SystemHeap::m_clSystemHeap;
00032 bool SystemHeap::m_bInit;
00033
00034 //-----
00035 void SystemHeap::Init( void )
00036 {
00037     #if HEAP_NUM_SIZES > 0
00038         m_pclSystemHeapConfig[0].m_usBlockSize =
00039             HEAP_BLOCK_SIZE_1;
00040         m_pclSystemHeapConfig[0].m_usBlockCount =
00041             HEAP_BLOCK_COUNT_1;
00042     #endif
00043     #if HEAP_NUM_SIZES > 1
00044         m_pclSystemHeapConfig[1].m_usBlockSize = HEAP_BLOCK_SIZE_2;
00045         m_pclSystemHeapConfig[1].m_usBlockCount = HEAP_BLOCK_COUNT_2;
00046     #endif
00047     #if HEAP_NUM_SIZES > 2
00048         m_pclSystemHeapConfig[2].m_usBlockSize = HEAP_BLOCK_SIZE_3;
00049         m_pclSystemHeapConfig[2].m_usBlockCount = HEAP_BLOCK_COUNT_3;
00050     #endif
00051 }

```

```

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].
00122     m_usBlockSize = 0;
00123     m_pclSystemHeapConfig[HEAP_NUM_SIZES].
00124     m_usBlockCount = 0;
00125
00126     m_clSystemHeap.Create((void*)m_pucRawHeap,
00127     m_pclSystemHeapConfig);
00128
00129     m_bInit = true;
00130 }
00131
00132 //-----
00133 void *SystemHeap::Alloc(K_USHORT usSize_)
00134 {

```

```

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

```

15.173 /home/moslevin/Project/R1/stage/src/system_heap.h File Reference

Global system-heap implementation.

```

#include "system_heap_config.h"
#include "fixed_heap.h"

```

Classes

- class [SystemHeap](#)

The [SystemHeap](#) class implements a heap which is accessible from all components in the system.

Macros

- #define [HEAP_RAW_SIZE_1](#) (([HEAP_BLOCK_SIZE_1](#) + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_1](#))
Really ugly computations used to auto-size the heap footprint based on the user-configuration data.
- #define [HEAP_RAW_SIZE_2](#) (([HEAP_BLOCK_SIZE_2](#) + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_2](#))
- #define [HEAP_RAW_SIZE_3](#) (([HEAP_BLOCK_SIZE_3](#) + sizeof([LinkListNode](#)) + sizeof(void*)) * [HEAP_BLOCK_COUNT_3](#))
- #define [HEAP_RAW_SIZE_4](#) 0
- #define [HEAP_RAW_SIZE_5](#) 0
- #define [HEAP_RAW_SIZE_6](#) 0
- #define [HEAP_RAW_SIZE_7](#) 0
- #define [HEAP_RAW_SIZE_8](#) 0
- #define [HEAP_RAW_SIZE_9](#) 0
- #define [HEAP_RAW_SIZE_10](#) 0
- #define [HEAP_RAW_SIZE_11](#) 0
- #define [HEAP_RAW_SIZE_12](#) 0
- #define [HEAP_RAW_SIZE_13](#) 0
- #define [HEAP_RAW_SIZE_14](#) 0
- #define [HEAP_RAW_SIZE_15](#) 0
- #define [HEAP_RAW_SIZE_16](#) 0
- #define [HEAP_RAW_SIZE_17](#) 0
- #define [HEAP_RAW_SIZE_18](#) 0
- #define [HEAP_RAW_SIZE_19](#) 0
- #define [HEAP_RAW_SIZE_20](#) 0
- #define [HEAP_RAW_SIZE_21](#) 0
- #define [HEAP_RAW_SIZE](#)


```

        HEAP_BLOCK_COUNT_1 )
00036 #else
00037     #define HEAP_RAW_SIZE_1 0
00038 #endif
00039
00040 #if HEAP_NUM_SIZES > 1
00041     #define HEAP_RAW_SIZE_2 ((HEAP_BLOCK_SIZE_2 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_2 )
00042 #else
00043     #define HEAP_RAW_SIZE_2 0
00044 #endif
00045
00046 #if HEAP_NUM_SIZES > 2
00047     #define HEAP_RAW_SIZE_3 ((HEAP_BLOCK_SIZE_3 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_3 )
00048 #else
00049     #define HEAP_RAW_SIZE_3 0
00050 #endif
00051
00052 #if HEAP_NUM_SIZES > 3
00053     #define HEAP_RAW_SIZE_4 ((HEAP_BLOCK_SIZE_4 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_4 )
00054 #else
00055     #define HEAP_RAW_SIZE_4 0
00056 #endif
00057
00058 #if HEAP_NUM_SIZES > 4
00059     #define HEAP_RAW_SIZE_5 ((HEAP_BLOCK_SIZE_5 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_5 )
00060 #else
00061     #define HEAP_RAW_SIZE_5 0
00062 #endif
00063
00064 #if HEAP_NUM_SIZES > 5
00065     #define HEAP_RAW_SIZE_6 ((HEAP_BLOCK_SIZE_6 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_6 )
00066 #else
00067     #define HEAP_RAW_SIZE_6 0
00068 #endif
00069
00070 #if HEAP_NUM_SIZES > 6
00071     #define HEAP_RAW_SIZE_7 ((HEAP_BLOCK_SIZE_7 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_7 )
00072 #else
00073     #define HEAP_RAW_SIZE_7 0
00074 #endif
00075
00076 #if HEAP_NUM_SIZES > 7
00077     #define HEAP_RAW_SIZE_8 ((HEAP_BLOCK_SIZE_8 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_8 )
00078 #else
00079     #define HEAP_RAW_SIZE_8 0
00080 #endif
00081
00082 #if HEAP_NUM_SIZES > 8
00083     #define HEAP_RAW_SIZE_9 ((HEAP_BLOCK_SIZE_9 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_9 )
00084 #else
00085     #define HEAP_RAW_SIZE_9 0
00086 #endif
00087
00088 #if HEAP_NUM_SIZES > 9
00089     #define HEAP_RAW_SIZE_10 ((HEAP_BLOCK_SIZE_10 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_10 )
00090 #else
00091     #define HEAP_RAW_SIZE_10 0
00092 #endif
00093
00094 #if HEAP_NUM_SIZES > 10
00095     #define HEAP_RAW_SIZE_11 ((HEAP_BLOCK_SIZE_11 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_11 )
00096 #else
00097     #define HEAP_RAW_SIZE_11 0
00098 #endif
00099
00100 #if HEAP_NUM_SIZES > 11
00101     #define HEAP_RAW_SIZE_12 ((HEAP_BLOCK_SIZE_12 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_12 )
00102 #else
00103     #define HEAP_RAW_SIZE_12 0
00104 #endif
00105
00106 #if HEAP_NUM_SIZES > 12
00107     #define HEAP_RAW_SIZE_13 ((HEAP_BLOCK_SIZE_13 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_13 )
00108 #else
00109     #define HEAP_RAW_SIZE_13 0

```

```

00110 #endif
00111
00112 #if HEAP_NUM_SIZES > 13
00113     #define HEAP_RAW_SIZE_14 ((HEAP_BLOCK_SIZE_14 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_14 )
00114 #else
00115     #define HEAP_RAW_SIZE_14 0
00116 #endif
00117
00118 #if HEAP_NUM_SIZES > 14
00119     #define HEAP_RAW_SIZE_15 ((HEAP_BLOCK_SIZE_15 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_15 )
00120 #else
00121     #define HEAP_RAW_SIZE_15 0
00122 #endif
00123
00124 #if HEAP_NUM_SIZES > 15
00125     #define HEAP_RAW_SIZE_16 ((HEAP_BLOCK_SIZE_16 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_16 )
00126 #else
00127     #define HEAP_RAW_SIZE_16 0
00128 #endif
00129
00130 #if HEAP_NUM_SIZES > 16
00131     #define HEAP_RAW_SIZE_17 ((HEAP_BLOCK_SIZE_17 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_17 )
00132 #else
00133     #define HEAP_RAW_SIZE_17 0
00134 #endif
00135
00136 #if HEAP_NUM_SIZES > 17
00137     #define HEAP_RAW_SIZE_18 ((HEAP_BLOCK_SIZE_18 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_18 )
00138 #else
00139     #define HEAP_RAW_SIZE_18 0
00140 #endif
00141
00142 #if HEAP_NUM_SIZES > 18
00143     #define HEAP_RAW_SIZE_19 ((HEAP_BLOCK_SIZE_19 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_19 )
00144 #else
00145     #define HEAP_RAW_SIZE_19 0
00146 #endif
00147
00148 #if HEAP_NUM_SIZES > 19
00149     #define HEAP_RAW_SIZE_20 ((HEAP_BLOCK_SIZE_20 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_20 )
00150 #else
00151     #define HEAP_RAW_SIZE_20 0
00152 #endif
00153
00154 #if HEAP_NUM_SIZES > 20
00155     #define HEAP_RAW_SIZE_21 ((HEAP_BLOCK_SIZE_21 + sizeof(LinkListNode) + sizeof(void*)) *
        HEAP_BLOCK_COUNT_21 )
00156 #else
00157     #define HEAP_RAW_SIZE_21 0
00158 #endif
00159
00160 //-----
00161 #define HEAP_RAW_SIZE \
00162     HEAP_RAW_SIZE_1 + \
00163     HEAP_RAW_SIZE_2 + \
00164     HEAP_RAW_SIZE_3 + \
00165     HEAP_RAW_SIZE_4 + \
00166     HEAP_RAW_SIZE_5 + \
00167     HEAP_RAW_SIZE_6 + \
00168     HEAP_RAW_SIZE_7 + \
00169     HEAP_RAW_SIZE_8 + \
00170     HEAP_RAW_SIZE_9 + \
00171     HEAP_RAW_SIZE_10 + \
00172     HEAP_RAW_SIZE_11 + \
00173     HEAP_RAW_SIZE_12 + \
00174     HEAP_RAW_SIZE_13 + \
00175     HEAP_RAW_SIZE_14 + \
00176     HEAP_RAW_SIZE_15 + \
00177     HEAP_RAW_SIZE_16 + \
00178     HEAP_RAW_SIZE_17 + \
00179     HEAP_RAW_SIZE_18 + \
00180     HEAP_RAW_SIZE_19 + \
00181     HEAP_RAW_SIZE_20 + \
00182     HEAP_RAW_SIZE_21
00183
00184 //-----
00189 class SystemHeap
00190 {
00191 public:
00195     static void Init(void);

```

```

00196
00203     static void* Alloc(K_USHORT usSize_);
00204
00209     static void Free(void *pvData_);
00210
00211 private:
00212     static K_UCHAR m_pucRawHeap[ HEAP_RAW_SIZE ];
00213     static HeapConfig m_pclSystemHeapConfig[
00214         HEAP_NUM_SIZES + 1 ];
00214     static FixedHeap m_clSystemHeap;
00215     static bool m_bInit;
00216 };
00217
00218 #endif // USE_SYSTEM_HEAP
00219
00220 #endif // __SYSTEM_HEAP_H__

```

15.175 /home/moslevin/Project/R1/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)`

15.175.1 Detailed Description

System heap configuration - defines the block sizes and counts used to fulfill system/service allocations.

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

15.175.2 Macro Definition Documentation

15.175.2.1 #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](#).

15.176 system_heap_config.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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

```

15.177 /home/moslevin/Project/R1/stage/src/thread.cpp File Reference

Platform-Independent thread class Definition.

```
#include "kerneltypes.h"
#include "mark3cfg.h"
#include "thread.h"
#include "scheduler.h"
#include "kernelswi.h"
#include "timerlist.h"
#include "ksemaphore.h"
#include "quantum.h"
#include "kernel.h"
#include "kernel_debug.h"
```

Macros

- `#define __FILE_ID__ THREAD_CPP`

Functions

- static void `ThreadSleepCallback` (`Thread *pclOwner_`, void `*pvData_`)

This callback is used to wake up a thread once the interval has expired.

15.177.1 Detailed Description

Platform-Independent thread class Definition.

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

15.178 thread.cpp

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024
00025 #include "thread.h"
00026 #include "scheduler.h"
00027 #include "kernelswi.h"
00028 #include "timerlist.h"
00029 #include "ksemaphore.h"
00030 #include "quantum.h"
00031 #include "kernel.h"
00032 #include "kernel_debug.h"
00033
00034 //-----
00035 #if defined __FILE_ID__
00036     #undef __FILE_ID__
00037 #endif
00038 #define __FILE_ID__      THREAD_CPP
00039
00040 //-----
00041 void Thread::Init( K_WORD *pwStack_,
00042                   K_USHORT usStackSize_,
00043                   K_UCHAR ucPriority_,
00044                   ThreadEntry_t pfEntryPoint_,
00045                   void *pvArg_ )
00046 {
```

```

00047     static K_UCHAR ucThreadID = 0;
00048
00049     KERNEL_ASSERT( pwStack_ );
00050     KERNEL_ASSERT( pfEntryPoint_ );
00051
00052     ClearNode();
00053
00054     m_ucThreadID = ucThreadID++;
00055
00056     KERNEL_TRACE_1( STR_STACK_SIZE_1, usStackSize_ );
00057     KERNEL_TRACE_1( STR_PRIORITY_1, (K_UCHAR)ucPriority_ );
00058     KERNEL_TRACE_1( STR_THREAD_ID_1, (K_USHORT)m_ucThreadID );
00059     KERNEL_TRACE_1( STR_ENTRYPOINT_1, (K_USHORT)pfEntryPoint_ );
00060
00061     // Initialize the thread parameters to their initial values.
00062     m_pwStack = pwStack_;
00063     m_pwStackTop = TOP_OF_STACK(pwStack_, usStackSize_);
00064
00065     m_usStackSize = usStackSize_;
00066
00067     #if KERNEL_USE_QUANTUM
00068         m_usQuantum = 4;
00069     #endif
00070
00071     m_ucPriority = ucPriority_;
00072     m_ucCurPriority = m_ucPriority;
00073     m_pfEntryPoint = pfEntryPoint_;
00074     m_pvArg = pvArg_;
00075
00076     #if KERNEL_USE_THREADNAME
00077         m_szName = NULL;
00078     #endif
00079
00080     // Call CPU-specific stack initialization
00081     ThreadPort::InitStack(this);
00082
00083     // Add to the global "stop" list.
00084     CS_ENTER();
00085     m_pclOwner = Scheduler::GetThreadList(
m_ucPriority);
00086     m_pclCurrent = Scheduler::GetStopList();
00087     m_pclCurrent->Add(this);
00088     CS_EXIT();
00089 }
00090
00091 //-----
00092 void Thread::Start(void)
00093 {
00094     // Remove the thread from the scheduler's "stopped" list, and add it
00095     // to the scheduler's ready list at the proper priority.
00096     KERNEL_TRACE_1( STR_THREAD_START_1, (K_USHORT)m_ucThreadID );
00097
00098     CS_ENTER();
00099     Scheduler::GetStopList()->Remove(this);
00100     Scheduler::Add(this);
00101     m_pclOwner = Scheduler::GetThreadList(
m_ucPriority);
00102     m_pclCurrent = m_pclOwner;
00103
00104     if (Kernel::IsStarted())
00105     {
00106         if (m_ucPriority >= Scheduler::GetCurrentThread()->
GetCurPriority())
00107         {
00108             #if KERNEL_USE_QUANTUM
00109                 // Deal with the thread Quantum
00110                 Quantum::RemoveThread();
00111                 Quantum::AddThread(this);
00112             #endif
00113         }
00114         if (m_ucPriority > Scheduler::GetCurrentThread()->
GetPriority())
00115         {
00116             Thread::Yield();
00117         }
00118     }
00119     CS_EXIT();
00120 }
00121
00122 //-----
00123 void Thread::Stop()
00124 {
00125     bool bReschedule = 0;
00126
00127     CS_ENTER();
00128
00129     // If a thread is attempting to stop itself, ensure we call the scheduler

```

```

00130     if (this == Scheduler::GetCurrentThread())
00131     {
00132         bReschedule = true;
00133     }
00134
00135     // Add this thread to the stop-list (removing it from active scheduling)
00136     Scheduler::Remove(this);
00137     m_pclOwner = Scheduler::GetStopList();
00138     m_pclCurrent = m_pclOwner;
00139     m_pclOwner->Add(this);
00140
00141     CS_EXIT();
00142
00143     if (bReschedule)
00144     {
00145         Thread::Yield();
00146     }
00147 }
00148
00149 #if KERNEL_USE_DYNAMIC_THREADS
00150 //-----
00151 void Thread::Exit()
00152 {
00153     bool bReschedule = 0;
00154
00155     KERNEL_TRACE_1( STR_THREAD_EXIT_1, m_ucThreadID );
00156
00157     CS_ENTER();
00158
00159     // If this thread is the actively-running thread, make sure we run the
00160     // scheduler again.
00161     if (this == Scheduler::GetCurrentThread())
00162     {
00163         bReschedule = 1;
00164     }
00165
00166     // Remove the thread from scheduling
00167     m_pclCurrent->Remove(this);
00168
00169 #if KERNEL_USE_TIMERS
00170     // Just to be safe - attempt to remove the thread's timer
00171     // from the timer-scheduler (does no harm if it isn't
00172     // in the timer-list)
00173     TimerScheduler::Remove(&m_clTimer);
00174 #endif
00175
00176     CS_EXIT();
00177
00178     if (bReschedule)
00179     {
00180         // Choose a new "next" thread if we must
00181         Thread::Yield();
00182     }
00183 }
00184 #endif
00185
00186 #if KERNEL_USE_SLEEP
00187 //-----
00188 static void ThreadSleepCallback( Thread *pclOwner_, void *pvData_ )
00189 {
00190     Semaphore *pclSemaphore = static_cast<Semaphore*>(pvData_);
00191     // Post the semaphore, which will wake the sleeping thread.
00192     pclSemaphore->Post();
00193 }
00194
00195 //-----
00196 void Thread::Sleep(K_ULONG ulTimeMs_)
00197 {
00198     Semaphore clSemaphore;
00199     Timer *pclTimer = g_pstCurrent->GetTimer();
00200
00201     // Create a semaphore that this thread will block on
00202     clSemaphore.Init(0, 1);
00203
00204     // Create a one-shot timer that will call a callback that posts the
00205     // semaphore, waking our thread.
00206     pclTimer->Init();
00207     pclTimer->SetIntervalMSeconds(ulTimeMs_);
00208     pclTimer->SetCallback(ThreadSleepCallback);
00209     pclTimer->SetData((void*)&clSemaphore);
00210     pclTimer->SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00211
00212     // Add the new timer to the timer scheduler, and block the thread
00213     TimerScheduler::Add(pclTimer);
00214     clSemaphore.Pend();
00215 }
00216
00217

```

```

00218 //-----
00219 void Thread::USleep(K_ULONG ulTimeUs_)
00220 {
00221     Semaphore clSemaphore;
00222     Timer *pclTimer = g_pstCurrent->GetTimer();
00223
00224     // Create a semaphore that this thread will block on
00225     clSemaphore.Init(0, 1);
00226
00227     // Create a one-shot timer that will call a callback that posts the
00228     // semaphore, waking our thread.
00229     pclTimer->Init();
00230     pclTimer->SetIntervalUSeconds(ulTimeUs_);
00231     pclTimer->SetCallback(ThreadSleepCallback);
00232     pclTimer->SetData((void*)&clSemaphore);
00233     pclTimer->SetFlags(TIMERLIST_FLAG_ONE_SHOT);
00234
00235     // Add the new timer to the timer scheduler, and block the thread
00236     TimerScheduler::Add(pclTimer);
00237     clSemaphore.Pend();
00238 }
00239 #endif // KERNEL_USE_SLEEP
00240
00241 //-----
00242 K_USHORT Thread::GetStackSlack()
00243 {
00244     K_USHORT usCount = 0;
00245
00246     CS_ENTER();
00247
00248     for (usCount = 0; usCount < m_usStackSize; usCount++)
00249     {
00250         if (m_pwStack[usCount] != 0xFF)
00251         {
00252             break;
00253         }
00254     }
00255
00256     CS_EXIT();
00257
00258     return usCount;
00259 }
00260
00261 //-----
00262 void Thread::Yield()
00263 {
00264     CS_ENTER();
00265
00266     // Run the scheduler
00267     if (Scheduler::IsEnabled())
00268     {
00269         Scheduler::Schedule();
00270
00271         // Only switch contexts if the new task is different than the old task
00272         if (Scheduler::GetCurrentThread() !=
00273             Scheduler::GetNextThread())
00274         {
00275             #if KERNEL_USE_QUANTUM
00276                 // new thread scheduled. Stop current quantum timer (if it exists),
00277                 // and restart it for the new thread (if required).
00278                 Quantum::RemoveThread();
00279                 Quantum::AddThread(g_pstNext);
00280             #endif
00281             Thread::ContextSwitchSWI();
00282         }
00283     }
00284     else
00285     {
00286         Scheduler::QueueScheduler();
00287     }
00288
00289     CS_EXIT();
00290 }
00291
00292 //-----
00293 void Thread::SetPriorityBase(K_UCHAR ucPriority_)
00294 {
00295     GetCurrent()->Remove(this);
00296
00297     SetCurrent(Scheduler::GetThreadList(
00298         m_ucPriority));
00299
00300     GetCurrent()->Add(this);
00301 }
00302 //-----
00303 void Thread::SetPriority(K_UCHAR ucPriority_)

```



```

00304 {
00305     bool bSchedule = 0;
00306     CS_ENTER();
00307     // If this is the currently running thread, it's a good idea to reschedule
00308     // Or, if the new priority is a higher priority than the current thread's.
00309     if ((g_pstCurrent == this) || (ucPriority_ > g_pstCurrent->GetPriority()))
00310     {
00311         bSchedule = 1;
00312     }
00313     Scheduler::Remove(this);
00314     CS_EXIT();
00315
00316     m_ucCurPriority = ucPriority_;
00317     m_ucPriority = ucPriority_;
00318
00319     CS_ENTER();
00320     Scheduler::Add(this);
00321     CS_EXIT();
00322
00323     if (bSchedule)
00324     {
00325         if (Scheduler::IsEnabled())
00326         {
00327             CS_ENTER();
00328             Scheduler::Schedule();
00329             #if KERNEL_USE_QUANTUM
00330             // new thread scheduled. Stop current quantum timer (if it exists),
00331             // and restart it for the new thread (if required).
00332             Quantum::RemoveThread();
00333             Quantum::AddThread(g_pstNext);
00334             #endif
00335             CS_EXIT();
00336             Thread::ContextSwitchSWI();
00337         }
00338         else
00339         {
00340             Scheduler::QueueScheduler();
00341         }
00342     }
00343 }
00344
00345 //-----
00346 void Thread::InheritPriority(K_UCHAR ucPriority_)
00347 {
00348     SetOwner(Scheduler::GetThreadList(ucPriority_));
00349     m_ucCurPriority = ucPriority_;
00350 }
00351
00352 //-----
00353 void Thread::ContextSwitchSWI()
00354 {
00355     // Call the context switch interrupt if the scheduler is enabled.
00356     if (Scheduler::IsEnabled() == 1)
00357     {
00358         KERNEL_TRACE_1(STR_CONTEXT_SWITCH_1, (K_USHORT)g_pstNext->GetID());
00359         KernelSWI::Trigger();
00360     }
00361 }
00362
00363 #if KERNEL_USE_TIMERS
00364 //-----
00365 Timer *Thread::GetTimer() { return &
    m_clTimer; }
00366 //-----
00367
00368 void Thread::SetExpired(K_BOOL bExpired_) { m_bExpired = bExpired_; }
00369 //-----
00370
00371 K_BOOL Thread::GetExpired() { return m_bExpired; }
00372 #endif
00373

```

15.179 /home/moslevin/Project/R1/stage/src/thread.h File Reference

Platform independent thread class declarations.


```

00058 {
00059 public:
00079     void Init(K_WORD *paucStack_,
00080               K_USHORT usStackSize_,
00081               K_UCHAR ucPriority_,
00082               ThreadEntry_t pfEntryPoint_,
00083               void *pvArg_ );
00084
00092     void Start();
00093
00094
00101     void Stop();
00102
00103 #if KERNEL_USE_THREADNAME
00104
00113     void SetName(const K_CHAR *szName_) { m_szName = szName_; }
00114
00121     const K_CHAR* GetName() { return m_szName; }
00122 #endif
00123
00132     ThreadList *GetOwner(void) { return m_pclOwner; }
00133
00141     ThreadList *GetCurrent(void) { return m_pclCurrent; }
00142
00151     K_UCHAR GetPriority(void) { return m_ucPriority; }
00152
00160     K_UCHAR GetCurPriority(void) { return m_ucCurPriority; }
00161
00162 #if KERNEL_USE_QUANTUM
00163
00170     void SetQuantum( K_USHORT usQuantum_ ) { m_usQuantum = usQuantum_; }
00171
00179     K_USHORT GetQuantum(void) { return m_usQuantum; }
00180 #endif
00181
00189     void SetCurrent( ThreadList *pclNewList_ ) {
m_pclCurrent = pclNewList_; }
00190
00198     void SetOwner( ThreadList *pclNewList_ ) { m_pclOwner = pclNewList_; }
00199
00200
00213     void SetPriority(K_UCHAR ucPriority_);
00214
00224     void InheritPriority(K_UCHAR ucPriority_);
00225
00226 #if KERNEL_USE_DYNAMIC_THREADS
00227
00238     void Exit();
00239 #endif
00240
00241 #if KERNEL_USE_SLEEP
00242
00250     static void Sleep(K_ULONG ulTimeMs_);
00251
00260     static void USleep(K_ULONG ulTimeUs_);
00261 #endif
00262
00270     static void Yield(void);
00271
00279     void SetID( K_UCHAR ucID_ ) { m_ucThreadID = ucID_; }
00280
00288     K_UCHAR GetID() { return m_ucThreadID; }
00289
00290
00303     K_USHORT GetStackSlack();
00304
00305 #if KERNEL_USE_EVENTFLAG
00306
00313     K_USHORT GetEventFlagMask() { return m_usFlagMask; }
00314
00319     void SetEventFlagMask(K_USHORT usMask_) { m_usFlagMask = usMask_; }
00320
00326     void SetEventFlagMode(EventFlagOperation_t eMode_ ) {
m_eFlagMode = eMode_; }
00327
00332     EventFlagOperation_t GetEventFlagMode() { return m_eFlagMode; }
00333 #endif
00334
00335 #if KERNEL_USE_TIMERS
00336
00339     Timer *GetTimer();
00340     void SetExpired( K_BOOL bExpired_ );
00341     K_BOOL GetExpired();
00342 #endif
00343
00344     friend class ThreadPort;
00345

```

```

00346 private:
00354     static void ContextSwitchSWI(void);
00355
00360     void SetPriorityBase(K_UCHAR ucPriority_);
00361
00363     K_WORD *m_pwStackTop;
00364
00366     K_WORD *m_pwStack;
00367
00369     K_USHORT m_usStackSize;
00370
00371     #if KERNEL_USE_QUANTUM
00372         K_USHORT m_usQuantum;
00374     #endif
00375
00377     K_UCHAR m_ucThreadID;
00378
00380     K_UCHAR m_ucPriority;
00381
00383     K_UCHAR m_ucCurPriority;
00384
00386     ThreadEntry_t m_pfEntryPoint;
00387
00389     void *m_pvArg;
00390
00391     #if KERNEL_USE_THREADNAME
00392         const K_CHAR *m_szName;
00394     #endif
00395
00396     #if KERNEL_USE_EVENTFLAG
00397         K_USHORT m_usFlagMask;
00399
00401         EventFlagOperation_t m_eFlagMode;
00402     #endif
00403
00404     #if KERNEL_USE_TIMERS
00405         Timer m_clTimer;
00407         K_BOOL m_bExpired;
00408     #endif
00409
00411     ThreadList *m_pclCurrent;
00412
00414     ThreadList *m_pclOwner;
00415 };
00416
00417 #endif

```

15.181 /home/moslevin/Project/R1/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`

15.181.1 Detailed Description

[Thread](#) linked-list definitions.

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

15.182 threadlist.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "tl.h"
00024 #include "threadlist.h"
00025 #include "thread.h"
00026 #include "kernel_debug.h"
00027 //-----
00028 #if defined __FILE_ID__
00029 #undef __FILE_ID__
00030 #endif
00031 #define __FILE_ID__      THREADLIST_CPP
00032
00033 //-----
00034 void ThreadList::SetPriority(K_UCHAR ucPriority_)
00035 {
00036     m_ucPriority = ucPriority_;
00037 }
00038
00039 //-----
00040 void ThreadList::SetFlagPointer( K_UCHAR *pucFlag_)
00041 {
00042     m_pucFlag = pucFlag_;
00043 }
00044
00045 //-----
00046 void ThreadList::Add(LinkListNode *node_) {
00047     CircularLinkList::Add(node_);
00048
00049     // If the head of the list isn't empty,
00050     if (m_pstHead != NULL)
00051     {
00052         // We've specified a bitmap for this threadlist
00053         if (m_pucFlag)
00054         {
00055             // Set the flag for this priority level
00056             *m_pucFlag |= (1 << m_ucPriority);
00057         }
00058     }
00059 }
00060
00061 //-----
00062 void ThreadList::Add(LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_)
00063 {
00064     // Set the threadlist's priority level, flag pointer, and then add the
00065     // thread to the threadlist
00066     SetPriority(ucPriority_);
00067     SetFlagPointer(pucFlag_);
00068     Add(node_);
00069 }
00070 //-----
00071 void ThreadList::Remove(LinkListNode *node_) {
00072     // Remove the thread from the list
00073     CircularLinkList::Remove(node_);
00074
00075     // If the list is empty...
00076     if (!m_pstHead)
00077     {
00078         // Clear the bit in the bitmap at this priority level
00079         if (m_pucFlag)
00080         {
00081             *m_pucFlag &= ~(1 << m_ucPriority);
00082         }
00083     }
00084 }
00085
00086 //-----
00087 Thread *ThreadList::HighestWaiter()
00088 {
00089     Thread *pclTemp = static_cast<Thread*>(GetHead());
00090     Thread *pclChosen = pclTemp;
00091

```



```

00049     void SetPriority(K_UCHAR ucPriority_);
00050
00059     void SetFlagPointer(K_UCHAR *pucFlag_);
00060
00068     void Add(LinkListNode *node_);
00069
00083     void Add(LinkListNode *node_, K_UCHAR *pucFlag_, K_UCHAR ucPriority_);
00084
00092     void Remove(LinkListNode *node_);
00093
00101     Thread *HighestWaiter();
00102 private:
00103
00105     K_UCHAR m_ucPriority;
00106
00108     K_UCHAR *m_pucFlag;
00109 };
00110
00111 #endif
00112

```

15.185 /home/moslevin/Project/R1/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.

Variables

- **Thread** * **g_pstCurrentThread**
- **naked**

15.185.1 Detailed Description

ATMega328p Multithreading.

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

15.186 threadport.cpp

```

00001 /*=====
00002     _____

```

```

00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00022 #include "kerneltypes.h"
00023 #include "mark3cfg.h"
00024 #include "thread.h"
00025 #include "threadport.h"
00026 #include "kernelswi.h"
00027 #include "kerneltimer.h"
00028 #include "timerlist.h"
00029 #include "quantum.h"
00030 #include <avr/io.h>
00031 #include <avr/interrupt.h>
00032
00033 //-----
00034 Thread *g_pstCurrentThread;
00035
00036 //-----
00037 void ThreadPort::InitStack(Thread *pclThread_)
00038 {
00039     // Initialize the stack for a Thread
00040     K_USHORT usAddr;
00041     K_UCHAR *pucStack;
00042     K_USHORT i;
00043
00044     // Get the address of the thread's entry function
00045     usAddr = (K_USHORT)(pclThread_>m_pfEntryPoint);
00046
00047     // Start by finding the bottom of the stack
00048     pucStack = (K_UCHAR*)pclThread_>m_pwStackTop;
00049
00050     // clear the stack, and initialize it to a known-default value (easier
00051     // to debug when things go sour with stack corruption or overflow)
00052     for (i = 0; i < pclThread_>m_usStackSize; i++)
00053     {
00054         pclThread_>m_pwStack[i] = 0xFF;
00055     }
00056
00057     // Our context starts with the entry function
00058     PUSH_TO_STACK(pucStack, (K_UCHAR)(usAddr & 0x00FF));
00059     PUSH_TO_STACK(pucStack, (K_UCHAR)((usAddr >> 8) & 0x00FF));
00060
00061     // R0
00062     PUSH_TO_STACK(pucStack, 0x00); // R0
00063
00064     // Push status register and R1 (which is used as a constant zero)
00065     PUSH_TO_STACK(pucStack, 0x80); // SR
00066     PUSH_TO_STACK(pucStack, 0x00); // R1
00067
00068     // Push other registers
00069     for (i = 2; i <= 23; i++) //R2-R23
00070     {
00071         PUSH_TO_STACK(pucStack, i);
00072     }
00073
00074     // Assume that the argument is the only stack variable
00075     PUSH_TO_STACK(pucStack, (K_UCHAR)((K_USHORT)(pclThread_>
m_pvArg)) & 0x00FF)); //R24
00076     PUSH_TO_STACK(pucStack, (K_UCHAR)((K_USHORT)(pclThread_>
m_pvArg)>>8) & 0x00FF)); //R25
00077
00078     // Push the rest of the registers in the context
00079     for (i = 26; i <=31; i++)
00080     {
00081         PUSH_TO_STACK(pucStack, i);
00082     }
00083
00084     // Set the top o' the stack.
00085     pclThread_>m_pwStackTop = (K_UCHAR*)pucStack;
00086
00087     // That's it! the thread is ready to run now.
00088 }
00089
00090 //-----
00091 static void Thread_Switch(void)
00092 {
00093     g_pstCurrent = g_pstNext;
00094 }
00095

```



```

00096
00097 //-----
00098 void ThreadPort::StartThreads()
00099 {
00100     KernelSWI::Config();           // configure the task switch SWI
00101     KernelTimer::Config();         // configure the kernel timer
00102
00103     Scheduler::SetScheduler(1);    // enable the scheduler
00104     Scheduler::Schedule();         // run the scheduler - determine the first
                                     thread to run
00105
00106     Thread_Switch();               // Set the next scheduled thread to the current thread
00107
00108     KernelTimer::Start();          // enable the kernel timer
00109     KernelSWI::Start();            // enable the task switch SWI
00110
00111     // Restore the context...
00112     Thread_RestoreContext();       // restore the context of the first running thread
00113     ASM("reti");                  // return from interrupt - will return to the first scheduled thread
00114 }
00115
00116 //-----
00121 //-----
00122 ISR(INT0_vect) __attribute__ ( ( signal, naked ) );
00123 ISR(INT0_vect)
00124 {
00125     Thread_SaveContext();          // Push the context (registers) of the current task
00126     Thread_Switch();              // Switch to the next task
00127     Thread_RestoreContext();       // Pop the context (registers) of the next task
00128     ASM("reti");                  // Return to the next task
00129 }
00130
00131 //-----
00136 //-----
00137 ISR(TIMER1_COMPA_vect)
00138 {
00139     #if KERNEL_USE_TIMERS
00140         TimerScheduler::Process();
00141     #endif
00142     #if KERNEL_USE_QUANTUM
00143         Quantum::UpdateTimer();
00144     #endif
00145 }

```

15.187 /home/moslevin/Project/R1/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

Generated on Thu Mar 5 2015 21:04:26 for Mark3 Realtime Kernel by Doxygen

```

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_

```

15.189 /home/moslevin/Project/R1/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"
#include "quantum.h"

```

Macros

- #define __FILE_ID__ TIMERLIST_CPP

15.189.1 Detailed Description

Timer data structure + scheduler implementations.

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

15.190 timerlist.cpp

```

00001  /*=====
00002
00003  _____
00004  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00005  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00006  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00007  |   \   /   |   \   /   |   \   /   |   \   /   |   \   /   |
00008
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 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  #include "quantum.h"
00030  //-----
00031  #if defined __FILE_ID__
00032      #undef __FILE_ID__
00033  #endif
00034  #define __FILE_ID__      TIMERLIST_CPP
00035
00036  #if KERNEL_USE_TIMERS
00037
00038  //-----
00039  TimerList TimerScheduler::m_clTimerList;
00040
00041  //-----
00042  void TimerList::Init(void)
00043  {
00044      m_bTimerActive = 0;
00045      m_ulNextWakeup = 0;
00046  }
00047
00048  //-----
00049  void TimerList::Add(Timer *pclListNode_)
00050  {
00051      #if KERNEL_TIMERS_TICKLESS
00052          bool bStart = 0;
00053      #endif
00054
00055      K_LONG lDelta;
00056      CS_ENTER();
00057
00058      #if KERNEL_TIMERS_TICKLESS
00059          if (GetHead() == NULL)
00060          {
00061              bStart = 1;
00062          }
00063      #endif
00064
00065      pclListNode_->ClearNode();
00066      DoubleLinkedList::Add(pclListNode_);
00067
00068      // Set the initial timer value
00069      pclListNode_>m_ulTimeLeft = pclListNode_>m_ulInterval;
00070
00071      #if KERNEL_TIMERS_TICKLESS
00072          if (!bStart)
00073          {
00074              // If the new interval is less than the amount of time remaining...
00075              lDelta = KernelTimer::TimeToExpiry() - pclListNode_>

```

```

        m_ulInterval;
00076
00077         if (lDelta > 0)
00078         {
00079             // Set the new expiry time on the timer.
00080             m_ulNextWakeup = KernelTimer::SubtractExpiry( (K_ULONG)
lDelta);
00081         }
00082     }
00083     else
00084     {
00085         m_ulNextWakeup = pclListNode->m_ulInterval;
00086         KernelTimer::SetExpiry(m_ulNextWakeup);
00087         KernelTimer::Start();
00088     }
00089 #endif
00090
00091     // Set the timer as active.
00092     pclListNode->m_ucFlags |= TIMERLIST_FLAG_ACTIVE;
00093     CS_EXIT();
00094 }
00095
00096 //-----
00097 void TimerList::Remove(Timer *pclLinkListNode_)
00098 {
00099     CS_ENTER();
00100
00101     DoubleLinkedList::Remove(pclLinkListNode_);
00102
00103     #if KERNEL_TIMERS_TICKLESS
00104     if (this->GetHead() == NULL)
00105     {
00106         KernelTimer::Stop();
00107     }
00108     #endif
00109
00110     CS_EXIT();
00111 }
00112
00113 //-----
00114 void TimerList::Process(void)
00115 {
00116     #if KERNEL_TIMERS_TICKLESS
00117     K_ULONG ulNewExpiry;
00118     K_ULONG ulOvertime;
00119     bool bContinue;
00120     #endif
00121
00122     Timer *pclNode;
00123     Timer *pclPrev;
00124
00125     #if KERNEL_USE_QUANTUM
00126     Quantum::SetInTimer();
00127     #endif
00128     #if KERNEL_TIMERS_TICKLESS
00129     // Clear the timer and its expiry time - keep it running though
00130     KernelTimer::ClearExpiry();
00131     do
00132     {
00133     #endif
00134         pclNode = static_cast<Timer*>(GetHead());
00135         pclPrev = NULL;
00136
00137         #if KERNEL_TIMERS_TICKLESS
00138         bContinue = 0;
00139         ulNewExpiry = MAX_TIMER_TICKS;
00140         #endif
00141
00142         // Subtract the elapsed time interval from each active timer.
00143         while (pclNode)
00144         {
00145             // Active timers only...
00146             if (pclNode->m_ucFlags & TIMERLIST_FLAG_ACTIVE)
00147             {
00148                 // Did the timer expire?
00149                 #if KERNEL_TIMERS_TICKLESS
00150                 if (pclNode->m_ulTimeLeft <= m_ulNextWakeup)
00151                 #else
00152                 pclNode->m_ulTimeLeft--;
00153                 if (0 == pclNode->m_ulTimeLeft)
00154                 #endif
00155                 {
00156                     // Yes - set the "callback" flag - we'll execute the callbacks later
00157                     pclNode->m_ucFlags |= TIMERLIST_FLAG_CALLBACK;
00158
00159                     if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
00160                     {

```

```

00161             // If this was a one-shot timer, deactivate the timer.
00162             pclNode->m_ucFlags |= TIMERLIST_FLAG_EXPIRED;
00163             pclNode->m_ucFlags &= ~TIMERLIST_FLAG_ACTIVE;

00164         }
00165         else
00166         {
00167             // Reset the interval timer.
00168             // I think we're good though...
00169             pclNode->m_ulTimeLeft = pclNode->
00170             m_ulInterval;
00171
00172             #if KERNEL_TIMERS_TICKLESS
00173             // If the time remaining (plus the length of the tolerance interval)
00174             // is less than the next expiry interval, set the next expiry interval.
00175             if ((pclNode->m_ulTimeLeft + pclNode->
00176             m_ulTimerTolerance) < ulNewExpiry)
00177             {
00178                 ulNewExpiry = pclNode->m_ulTimeLeft + pclNode->
00179                 m_ulTimerTolerance;
00180             }
00181             #endif
00182             #if KERNEL_TIMERS_TICKLESS
00183             else
00184             {
00185                 // Not expiring, but determine how K_LONG to run the next timer interval for.
00186                 pclNode->m_ulTimeLeft -= m_ulNextWakeup;
00187                 if (pclNode->m_ulTimeLeft < ulNewExpiry)
00188                 {
00189                     ulNewExpiry = pclNode->m_ulTimeLeft;
00190                 }
00191             }
00192             #endif
00193         }
00194         pclNode = static_cast<Timer*>(pclNode->GetNext());
00195     }
00196
00197     // Process the expired timers callbacks.
00198     pclNode = static_cast<Timer*>(GetHead());
00199     while (pclNode)
00200     {
00201         pclPrev = NULL;
00202
00203         // If the timer expired, run the callbacks now.
00204         if (pclNode->m_ucFlags & TIMERLIST_FLAG_CALLBACK)
00205         {
00206             // Run the callback. these callbacks must be very fast...
00207             pclNode->m_pfCallback( pclNode->m_pclOwner, pclNode->
00208             m_pvData );
00209             pclNode->m_ucFlags &= ~TIMERLIST_FLAG_CALLBACK;
00210
00211             // If this was a one-shot timer, let's remove it.
00212             if (pclNode->m_ucFlags & TIMERLIST_FLAG_ONE_SHOT)
00213             {
00214                 pclPrev = pclNode;
00215             }
00216             pclNode = static_cast<Timer*>(pclNode->GetNext());
00217
00218             // Remove one-shot-timers
00219             if (pclPrev)
00220             {
00221                 Remove(pclPrev);
00222             }
00223         }
00224
00225         #if KERNEL_TIMERS_TICKLESS
00226         // Check to see how much time has elapsed since the time we
00227         // acknowledged the interrupt...
00228         ulOvertime = KernelTimer::GetOvertime();
00229
00230         if( ulOvertime >= ulNewExpiry ) {
00231             m_ulNextWakeup = ulOvertime;
00232             bContinue = 1;
00233         }
00234
00235         // If it's taken longer to go through this loop than would take us to
00236         // the next expiry, re-run the timing loop
00237     } while (bContinue);
00238
00239     // This timer elapsed, but there's nothing more to do...
00240     // Turn the timer off.
00241     if (ulNewExpiry >= MAX_TIMER_TICKS)
00242     {

```

```

00244         KernelTimer::Stop();
00245     }
00246     else
00247     {
00248         // Update the timer with the new "Next Wakeup" value, plus whatever
00249         // overtime has accumulated since the last time we called this handler
00250         m_ulNextWakeup = KernelTimer::SetExpiry(ulNewExpiry +
00251         ulOvertime);
00252     }
00253 #endif
00254 #if KERNEL_USE_QUANTUM
00255     Quantum::ClearInTimer();
00256 #endif
00257 }
00258 //-----
00259 void Timer::Start( bool bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *
pvData_ )
00260 {
00261     SetIntervalMSeconds(ulIntervalMs_);
00262     m_pfCallback = pfCallback_;
00263     m_pvData = pvData_;
00264     if (!bRepeat_)
00265     {
00266         m_ucFlags = TIMERLIST_FLAG_ONE_SHOT;
00267     }
00268     else
00269     {
00270         m_ucFlags = 0;
00271     }
00272     m_pclOwner = Scheduler::GetCurrentThread();
00273     TimerScheduler::Add(this);
00274 }
00275 //-----
00276 void Timer::Start( bool bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_,
TimerCallback_t pfCallback_, void *pvData_ )
00277 {
00278     m_ulTimerTolerance = MSECONDS_TO_TICKS(ulToleranceMs_);
00279     Start(bRepeat_, ulIntervalMs_, pfCallback_, pvData_);
00280 }
00281 //-----
00282 void Timer::Stop()
00283 {
00284     TimerScheduler::Remove(this);
00285 }
00286 //-----
00287 void Timer::SetIntervalTicks( K_ULONG ulTicks_ )
00288 {
00289     m_ulInterval = ulTicks_;
00290 }
00291 //-----
00292 void Timer::SetIntervalSeconds( K_ULONG ulSeconds_ )
00293 {
00294     m_ulInterval = SECONDS_TO_TICKS(ulSeconds_);
00295 }
00296 //-----
00297 void Timer::SetIntervalMSeconds( K_ULONG ulMSeconds_ )
00298 {
00299     m_ulInterval = MSECONDS_TO_TICKS(ulMSeconds_);
00300 }
00301 //-----
00302 void Timer::SetIntervalUSeconds( K_ULONG ulUSeconds_ )
00303 {
00304     m_ulInterval = USECONDS_TO_TICKS(ulUSeconds_);
00305 }
00306 //-----
00307 void Timer::SetTolerance(K_ULONG ulTicks_)
00308 {
00309     m_ulTimerTolerance = ulTicks_;
00310 }
00311 //-----
00312 #endif //KERNEL_USE_TIMERS

```


15.191 /home/moslevin/Project/R1/stage/src/timerlist.h File Reference

[Timer](#) list and timer-scheduling declarations.

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

Classes

- class [Timer](#)
Timer - an event-driven execution context based on a specified time interval.
- class [TimerList](#)
TimerList class - a doubly-linked-list of timer objects.
- class [TimerScheduler](#)
"Static" Class used to interface a global TimerList with the rest of the kernel.

Macros

- #define [TIMERLIST_FLAG_ONE_SHOT](#) (0x01)
Timer is one-shot.
- #define [TIMERLIST_FLAG_ACTIVE](#) (0x02)
Timer is currently active.
- #define [TIMERLIST_FLAG_CALLBACK](#) (0x04)
Timer is pending a callback.
- #define [TIMERLIST_FLAG_EXPIRED](#) (0x08)
Timer is actually expired.
- #define [MAX_TIMER_TICKS](#) (0x7FFFFFFF)
Maximum value to set.
- #define [SECONDS_TO_TICKS](#)(x) (((K_ULONG)x) * TIMER_FREQ))
- #define [MSECONDS_TO_TICKS](#)(x) (((((K_ULONG)x) * (TIMER_FREQ/100)) + 5) / 10))
- #define [USECONDS_TO_TICKS](#)(x) (((((K_ULONG)x) * TIMER_FREQ) + 50000) / 1000000))
- #define [MIN_TICKS](#) (3)
The minimum tick value to set.

Typedefs

- typedef void(* [TimerCallback_t](#))([Thread](#) *pclOwner_, void *pvData_)

15.191.1 Detailed Description

[Timer](#) list and timer-scheduling declarations. These classes implements a linked list of timer objects attached to the global kernel timer. Unlike other kernels which use a fully-synchronous "tick-based" timing mechanism, where the OS timing facilities are based on a fixed-frequency timer (which causes regular timer interrupts), Mark3 uses a "tickless" timer implementation, which only triggers interrupts when absolutely required. This is much more efficient in most cases - timer interrupts occur less frequently, allowing the kernel to stay in sleep much longer than it would otherwise.

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


```

00088 typedef void (*TimerCallback_t)(Thread *pclOwner_, void *pvData_);
00089
00090 //-----
00091 class TimerList;
00092 class TimerScheduler;
00093 class Quantum;
00099 class Timer : public LinkListNode
00100 {
00101 public:
00105     Timer() { }
00106
00110     void Init() { ClearNode(); m_ulInterval = 0;
m_ulTimerTolerance = 0; m_ulTimeLeft = 0;
m_ucFlags = 0; }
00111
00121     void Start( bool bRepeat_, K_ULONG ulIntervalMs_, TimerCallback_t pfCallback_, void *pvData_ );
00122
00134     void Start( bool bRepeat_, K_ULONG ulIntervalMs_, K_ULONG ulToleranceMs_, TimerCallback_t
pfCallback_, void *pvData_ );
00135
00140     void Stop();
00141
00151     void SetFlags( K_UCHAR ucFlags_ ) { m_ucFlags = ucFlags_; }
00152
00160     void SetCallback( TimerCallback_t pfCallback_ ) { m_pfCallback = pfCallback_; }
00161
00169     void SetData( void *pvData_ ) { m_pvData = pvData_; }
00170
00179     void SetOwner( Thread *pclOwner_ ) { m_pclOwner = pclOwner_; }
00180
00188     void SetIntervalTicks( K_ULONG ulTicks_ );
00189
00197     void SetIntervalSeconds( K_ULONG ulSeconds_ );
00198
00199
00200     K_ULONG GetInterval() { return m_ulInterval; }
00201
00209     void SetIntervalMSeconds( K_ULONG ulMSeconds_ );
00210
00218     void SetIntervalUSeconds( K_ULONG ulUSeconds_ );
00219
00229     void SetTolerance( K_ULONG ulTicks_ );
00230
00231 private:
00232
00233     friend class TimerList;
00234
00236     K_UCHAR m_ucFlags;
00237
00239     TimerCallback_t m_pfCallback;
00240
00242     K_ULONG m_ulInterval;
00243
00245     K_ULONG m_ulTimeLeft;
00246
00248     K_ULONG m_ulTimerTolerance;
00249
00251     Thread *m_pclOwner;
00252
00254     void *m_pvData;
00255 };
00256
00257 //-----
00261 class TimerList : public DoubleLinkedList
00262 {
00263 public:
00270     void Init();
00271
00279     void Add( Timer *pclListNode_ );
00280
00288     void Remove( Timer *pclListNode_ );
00289
00296     void Process();
00297
00298 private:
00300     K_ULONG m_ulNextWakeup;
00301
00303     K_UCHAR m_bTimerActive;
00304 };
00305
00306 //-----
00311 class TimerScheduler
00312 {
00313 public:
00320     static void Init() { m_clTimerList.Init(); }
00321
00330     static void Add( Timer *pclListNode_ )

```



```

00044 //-----
00045 void TraceBuffer::Write( K_USHORT *pusData_, K_USHORT usSize_ )
00046 {
00047     // Pipe the data directly to the circular buffer
00048     m_clBuffer.WriteData(pusData_, usSize_);
00049 }
00050
00051 #endif
00052

```

15.195 /home/moslevin/Project/R1/stage/src/tracebuffer.h File Reference

[Kernel](#) trace buffer class declaration.

```

#include "kerneltypes.h"
#include "mark3cfg.h"
#include "writebuf16.h"

```

15.195.1 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](#).

15.196 tracebuffer.h

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00024 #ifndef __TRACEBUFFER_H__
00025 #define __TRACEBUFFER_H__
00026
00027 #include "kerneltypes.h"
00028 #include "mark3cfg.h"
00029 #include "writebuf16.h"
00030
00031 #if KERNEL_USE_DEBUG && !KERNEL_AWARE_SIMULATION
00032
00033 #define TRACE_BUFFER_SIZE          (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

```

15.197 /home/moslevin/Project/R1/stage/src/unit_test.cpp File Reference

Unit test class definition.

```

#include "kerneltypes.h"
#include "unit_test.h"

```

15.197.1 Detailed Description

Unit test class definition.

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

15.198 unit_test.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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 }

```

15.199 /home/moslevin/Project/R1/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.

15.199.1 Detailed Description

Unit test class declarations.

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

15.200 unit_test.h

```

00001  /*=====
00002
00003  _____
00004  /         \
00005  |         |
00006  |         |
00007  |         |
00008  |         |
00009  --[Mark3 Realtime Platform]-----
00010
00011  Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012  See license.txt for more information
00013  =====*/
00018  #ifndef __UNIT_TEST_H__
00019  #define __UNIT_TEST_H__
00020
00021
00022  #include "kerneltypes.h"
00023
00024  //-----
00028  class UnitTest
00029  {
00030  public:
00031      UnitTest();
00032
00041      void SetName( const K_CHAR *szName_ ) { m_szName = szName_; }
00042
00048      void Start() { m_bIsActive = 1; }
00049
00056      void Pass();
00057
00064      void Fail();
00065
00066      void ExpectTrue( bool bExpression_ )
00067          { bExpression_ ? Pass() : Fail(); }
00068
00069      void ExpectFalse( bool bExpression_ )
00070          { !bExpression_ ? Pass() : Fail(); }
00071
00072      void ExpectEquals( bool bVal_, bool bExpression_ )
00073          { (bVal_ == bExpression_) ? Pass() : Fail(); }
00074
00075      void ExpectEquals( K_UCHAR ucVal_, K_UCHAR ucExpression_ )
00076          { (ucVal_ == ucExpression_) ? Pass() : Fail(); }
00077
00078      void ExpectEquals( K_USHORT usVal_, K_USHORT usExpression_ )

```

```

00079         { (usVal_ == usExpression_) ? Pass() : Fail(); }
00080
00081 void ExpectEquals( K_ULONG ulVal_, K_ULONG ulExpression_ )
00082 { (ulVal_ == ulExpression_) ? Pass() : Fail(); }
00083
00084 void ExpectEquals( K_CHAR cVal_, K_CHAR cExpression_ )
00085 { (cVal_ == cExpression_) ? Pass() : Fail(); }
00086
00087 void ExpectEquals( K_SHORT sVal_, K_SHORT sExpression_ )
00088 { (sVal_ == sExpression_) ? Pass() : Fail(); }
00089
00090 void ExpectEquals( K_LONG lVal_, K_LONG lExpression_ )
00091 { (lVal_ == lExpression_) ? Pass() : Fail(); }
00092
00093 void ExpectEquals( void* pvVal_, void* pvExpression_ )
00094 { (pvVal_ == pvExpression_) ? Pass() : Fail(); }
00095
00096
00097 void ExpectFailTrue( bool bExpression_ )
00098 { bExpression_ ? Fail() : Pass(); }
00099
00100 void ExpectFailFalse( bool bExpression_ )
00101 { !bExpression_ ? Fail() : Pass(); }
00102
00103 void ExpectFailEquals( bool bVal_, bool bExpression_ )
00104 { (bVal_ == bExpression_) ? Fail() : Pass(); }
00105
00106 void ExpectFailEquals( K_UCHAR ucVal_, K_UCHAR ucExpression_ )
00107 { (ucVal_ == ucExpression_) ? Fail() : Pass(); }
00108
00109 void ExpectFailEquals( K_USHORT usVal_, K_USHORT usExpression_ )
00110 { (usVal_ == usExpression_) ? Fail() : Pass(); }
00111
00112 void ExpectFailEquals( K_ULONG ulVal_, K_ULONG ulExpression_ )
00113 { (ulVal_ == ulExpression_) ? Fail() : Pass(); }
00114
00115 void ExpectFailEquals( K_CHAR cVal_, K_CHAR cExpression_ )
00116 { (cVal_ == cExpression_) ? Fail() : Pass(); }
00117
00118 void ExpectFailEquals( K_SHORT sVal_, K_SHORT sExpression_ )
00119 { (sVal_ == sExpression_) ? Fail() : Pass(); }
00120
00121 void ExpectFailEquals( K_LONG lVal_, K_LONG lExpression_ )
00122 { (lVal_ == lExpression_) ? Fail() : Pass(); }
00123
00124 void ExpectFailEquals( void* pvVal_, void* pvExpression_ )
00125 { (pvVal_ == pvExpression_) ? Fail() : Pass(); }
00126
00127 void ExpectGreaterThan( K_LONG lVal_, K_LONG lExpression_ )
00128 { (lVal_ > lExpression_) ? Pass() : Fail(); }
00129
00130 void ExpectLessThan( K_LONG lVal_, K_LONG lExpression_ )
00131 { (lVal_ < lExpression_) ? Pass() : Fail(); }
00132
00133 void ExpectGreaterThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00134 { (lVal_ >= lExpression_) ? Pass() : Fail(); }
00135
00136 void ExpectLessThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00137 { (lVal_ <= lExpression_) ? Pass() : Fail(); }
00138
00139 void ExpectFailGreaterThan( K_LONG lVal_, K_LONG lExpression_ )
00140 { (lVal_ > lExpression_) ? Fail() : Pass(); }
00141
00142 void ExpectFailLessThan( K_LONG lVal_, K_LONG lExpression_ )
00143 { (lVal_ < lExpression_) ? Fail() : Pass(); }
00144
00145 void ExpectFailGreaterThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00146 { (lVal_ >= lExpression_) ? Fail() : Pass(); }
00147
00148 void ExpectFailLessThanEquals( K_LONG lVal_, K_LONG lExpression_ )
00149 { (lVal_ <= lExpression_) ? Fail() : Pass(); }
00150
00157 void Complete() { m_bComplete = 1; }
00158
00166 const K_CHAR *GetName() { return m_szName; }
00167
00175 K_BOOL GetResult() { return m_bStatus; }
00176
00184 K_USHORT GetPassed() { return m_usPassed; }
00185
00193 K_USHORT GetFailed() { return m_usIterations -
m_usPassed; }
00194
00202 K_USHORT GetTotal() { return m_usIterations; }
00203
00204 private:
00205     const K_CHAR *m_szName;

```



```

00206     K_BOOL m_bIsActive;
00207     K_UCHAR m_bComplete;
00208     K_BOOL m_bStatus;
00209     K_USHORT m_usIterations;
00210     K_USHORT m_usPassed;
00211 };
00212
00213 #endif

```

15.201 /home/moslevin/Project/R1/stage/src/writebuf16.cpp File Reference

16 bit circular buffer implementation with callbacks.

```

#include "kerneltypes.h"
#include "writebuf16.h"
#include "kernel_debug.h"
#include "threadport.h"

```

15.201.1 Detailed Description

16 bit circular buffer implementation with callbacks.

Definition in file [writebuf16.cpp](#).

15.202 writebuf16.cpp

```

00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 Funkenstein Software Consulting, all rights reserved.
00012 See license.txt for more information
00013 =====*/
00020 #include "kerneltypes.h"
00021 #include "writebuf16.h"
00022 #include "kernel_debug.h"
00023 #include "threadport.h"
00024
00025 #if KERNEL_USE_DEBUG && !KERNEL_AWARE_SIMULATION
00026
00027 //-----
00028 void WriteBuffer16::WriteData( K_USHORT *pusBuf_, K_USHORT usLen_ )
00029 {
00030     K_USHORT *apusBuf[1];
00031     K_USHORT ausLen[1];
00032
00033     apusBuf[0] = pusBuf_;
00034     ausLen[0] = usLen_;
00035
00036     WriteVector( apusBuf, ausLen, 1 );
00037 }
00038
00039 //-----
00040 void WriteBuffer16::WriteVector( K_USHORT **ppusBuf_, K_USHORT *pusLen_, K_UCHAR ucCount_ )
00041 {
00042     K_USHORT usTempHead;
00043     K_UCHAR i;
00044     K_UCHAR j;
00045     K_USHORT usTotalLen = 0;
00046     bool bCallback = false;
00047     bool bRollover = false;
00048     // Update the head pointer synchronously, using a small
00049     // critical section in order to provide thread safety without
00050     // compromising on responsiveness by adding lots of extra
00051     // interrupt latency.
00052

```

```

00053     CS_ENTER();
00054
00055     usTempHead = m_usHead;
00056     {
00057         for (i = 0; i < ucCount_; i++)
00058         {
00059             usTotalLen += pusLen_[i];
00060         }
00061         m_usHead = (usTempHead + usTotalLen) % m_usSize;
00062     }
00063     CS_EXIT();
00064
00065     // Call the callback if we cross the 50% mark or rollover
00066     if (m_usHead < usTempHead)
00067     {
00068         if (m_pfCallback)
00069         {
00070             bCallback = true;
00071             bRollover = true;
00072         }
00073     }
00074     else if ((usTempHead < (m_usSize >> 1)) && (m_usHead >= (m_usSize >> 1)))
00075     {
00076         // Only trigger the callback if it's non-null
00077         if (m_pfCallback)
00078         {
00079             bCallback = true;
00080         }
00081     }
00082
00083     // Are we going to roll-over?
00084     for (j = 0; j < ucCount_; j++)
00085     {
00086         K_USHORT usSegmentLength = pusLen_[j];
00087         if (usSegmentLength + usTempHead >= m_usSize)
00088         {
00089             // We need to two-part this... First part: before the rollover
00090             K_USHORT usTempLen;
00091             K_USHORT *pusTmp = &m_pusData[ usTempHead ];
00092             K_USHORT *pusSrc = ppusBuf_[j];
00093             usTempLen = m_usSize - usTempHead;
00094             for (i = 0; i < usTempLen; i++)
00095             {
00096                 *pusTmp++ = *pusSrc++;
00097             }
00098
00099             // Second part: after the rollover
00100             usTempLen = usSegmentLength - usTempLen;
00101             pusTmp = m_pusData;
00102             for (i = 0; i < usTempLen; i++)
00103             {
00104                 *pusTmp++ = *pusSrc++;
00105             }
00106         }
00107         else
00108         {
00109             // No rollover - do the copy all at once.
00110             K_USHORT *pusSrc = ppusBuf_[j];
00111             K_USHORT *pusTmp = &m_pusData[ usTempHead ];
00112             for (K_USHORT i = 0; i < usSegmentLength; i++)
00113             {
00114                 *pusTmp++ = *pusSrc++;
00115             }
00116         }
00117     }
00118
00119     // Call the callback if necessary
00120     if (bCallback)
00121     {
00122         if (bRollover)
00123         {
00124             // Rollover - process the back-half of the buffer
00125             m_pfCallback( &m_pusData[ m_usSize >> 1 ], m_usSize >> 1 );
00126         }
00127         else
00128         {
00129             // 50% point - process the front-half of the buffer
00130             m_pfCallback( m_pusData, m_usSize >> 1 );
00131         }
00132     }
00133 }
00134 }
00135
00136 #endif

```

15.203 /home/moslevin/Project/R1/stage/src/writebuf16.h File Reference

Thread-safe circular buffer implementation with 16-bit elements.

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

15.203.1 Detailed Description

Thread-safe circular buffer implementation with 16-bit elements.

Definition in file [writebuf16.h](#).

15.204 writebuf16.h

```
00001 /*=====
00002
00003
00004
00005
00006
00007
00008
00009 --[Mark3 Realtime Platform]-----
00010
00011 Copyright (c) 2012-2015 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 #include "mark3cfg.h"
00025
00026 #if KERNEL_USE_DEBUG && !KERNEL_AWARE_SIMULATION
00027
00032 typedef void (*WriteBufferCallback)( K_USHORT *pusData_, K_USHORT usSize_ );
00033
00040 class WriteBuffer16
00041 {
00042 public:
00053     void SetBuffers( K_USHORT *pusData_, K_USHORT usSize_ )
00054     {
00055         m_pusData = pusData_;
00056         m_usSize = usSize_;
00057         m_usHead = 0;
00058         m_usTail = 0;
00059     }
00060
00072     void SetCallback( WriteBufferCallback pfCallback_ )
00073     { m_pfCallback = pfCallback_; }
00074
00083     void WriteData( K_USHORT *pusBuf_, K_USHORT usLen_ );
00084
00094     void WriteVector( K_USHORT **ppusBuf_, K_USHORT *pusLen_, K_UCHAR ucCount_);
00095
00096 private:
00097     K_USHORT *m_pusData;
00098
00099     volatile K_USHORT m_usSize;
00100     volatile K_USHORT m_usHead;
00101     volatile K_USHORT m_usTail;
00102
00103     WriteBufferCallback m_pfCallback;
00104 };
00105 #endif
00106
00107 #endif
```

Index

/home/moslevin/Project/R1/stage/src/atomic.cpp, [221](#)
/home/moslevin/Project/R1/stage/src/atomic.h, [223](#)
/home/moslevin/Project/R1/stage/src/blocking.cpp, [224](#)
/home/moslevin/Project/R1/stage/src/blocking.h, [225](#)
/home/moslevin/Project/R1/stage/src/control_button.-
cpp, [226](#)
/home/moslevin/Project/R1/stage/src/control_button.h,
[229](#)
/home/moslevin/Project/R1/stage/src/control_checkbox.-
cpp, [230](#), [231](#)
/home/moslevin/Project/R1/stage/src/control_checkbox.-
h, [234](#)
/home/moslevin/Project/R1/stage/src/control_gamepanel.-
cpp, [235](#)
/home/moslevin/Project/R1/stage/src/control_gamepanel.-
h, [236](#)
/home/moslevin/Project/R1/stage/src/control_groupbox.-
cpp, [236](#), [237](#)
/home/moslevin/Project/R1/stage/src/control_groupbox.-
h, [238](#)
/home/moslevin/Project/R1/stage/src/control_label.h,
[239](#)
/home/moslevin/Project/R1/stage/src/control_notification.-
cpp, [240](#)
/home/moslevin/Project/R1/stage/src/control_notification.-
h, [242](#)
/home/moslevin/Project/R1/stage/src/control_panel.cpp,
[243](#)
/home/moslevin/Project/R1/stage/src/control_panel.h,
[244](#)
/home/moslevin/Project/R1/stage/src/control_progress.-
cpp, [245](#)
/home/moslevin/Project/R1/stage/src/control_progress.-
h, [246](#), [247](#)
/home/moslevin/Project/R1/stage/src/control_slickbutton.-
h, [247](#), [248](#)
/home/moslevin/Project/R1/stage/src/control_slickprogress/
cpp, [248](#), [249](#)
/home/moslevin/Project/R1/stage/src/control_slickprogress/
h, [250](#)
/home/moslevin/Project/R1/stage/src/dcpu.cpp, [251](#),
[252](#)
/home/moslevin/Project/R1/stage/src/dcpu.h, [262](#), [266](#)
/home/moslevin/Project/R1/stage/src/debug_tokens.h,
[270](#), [272](#)
/home/moslevin/Project/R1/stage/src/draw.h, [273](#), [274](#)
/home/moslevin/Project/R1/stage/src/driver.cpp, [276](#),
[277](#)
/home/moslevin/Project/R1/stage/src/driver.h, [278](#), [280](#)
/home/moslevin/Project/R1/stage/src/eventflag.cpp, [281](#)
/home/moslevin/Project/R1/stage/src/eventflag.h, [285](#)
/home/moslevin/Project/R1/stage/src/fixed_heap.cpp,
[286](#)
/home/moslevin/Project/R1/stage/src/fixed_heap.h, [288](#)
/home/moslevin/Project/R1/stage/src/font.h, [289](#), [290](#)
/home/moslevin/Project/R1/stage/src/graphics.cpp, [290](#),
[291](#)
/home/moslevin/Project/R1/stage/src/graphics.h, [303](#),
[304](#)
/home/moslevin/Project/R1/stage/src/gui.cpp, [305](#)
/home/moslevin/Project/R1/stage/src/gui.h, [313](#), [315](#)
/home/moslevin/Project/R1/stage/src/kernel.cpp, [320](#),
[321](#)
/home/moslevin/Project/R1/stage/src/kernel.h, [322](#)
/home/moslevin/Project/R1/stage/src/kernel_debug.h,
[323](#)
/home/moslevin/Project/R1/stage/src/kernelswi.cpp, [325](#)
/home/moslevin/Project/R1/stage/src/kernelswi.h, [326](#),
[327](#)
/home/moslevin/Project/R1/stage/src/kerneltimer.cpp,
[327](#)
/home/moslevin/Project/R1/stage/src/kerneltimer.h, [330](#)
/home/moslevin/Project/R1/stage/src/kerneltypes.h, [331](#)
/home/moslevin/Project/R1/stage/src/keycodes.h, [332](#),
[333](#)
/home/moslevin/Project/R1/stage/src/kprofile.cpp, [335](#)
/home/moslevin/Project/R1/stage/src/kprofile.h, [336](#),
[337](#)
/home/moslevin/Project/R1/stage/src/ksemaphore.cpp,
[337](#), [338](#)
/home/moslevin/Project/R1/stage/src/ksemaphore.h,
[341](#)
/home/moslevin/Project/R1/stage/src/ll.cpp, [342](#)
/home/moslevin/Project/R1/stage/src/ll.h, [344](#), [345](#)
/home/moslevin/Project/R1/stage/src/manual.h, [346](#)
/home/moslevin/Project/R1/stage/src/mark3cfg.h, [347](#),
[350](#)
/home/moslevin/Project/R1/stage/src/memutil.cpp, [351](#),
[352](#)
/home/moslevin/Project/R1/stage/src/memutil.h, [357](#)
/home/moslevin/Project/R1/stage/src/message.cpp,
[358](#), [359](#)
/home/moslevin/Project/R1/stage/src/message.h, [361](#),
[362](#)
/home/moslevin/Project/R1/stage/src/mutex.cpp, [363](#)
/home/moslevin/Project/R1/stage/src/mutex.h, [367](#)
/home/moslevin/Project/R1/stage/src/nlfs.cpp, [368](#), [369](#)
/home/moslevin/Project/R1/stage/src/nlfs.h, [380](#), [382](#)

- /home/moslevin/Project/R1/stage/src/nlfs_config.h, 385, 386
- /home/moslevin/Project/R1/stage/src/nlfs_file.cpp, 386
- /home/moslevin/Project/R1/stage/src/nlfs_file.h, 390, 391
- /home/moslevin/Project/R1/stage/src/nlfs_ram.cpp, 391, 392
- /home/moslevin/Project/R1/stage/src/nlfs_ram.h, 393
- /home/moslevin/Project/R1/stage/src/panic_codes.h, 394
- /home/moslevin/Project/R1/stage/src/profile.cpp, 394, 395
- /home/moslevin/Project/R1/stage/src/profile.h, 396, 397
- /home/moslevin/Project/R1/stage/src/quantum.cpp, 398
- /home/moslevin/Project/R1/stage/src/quantum.h, 400
- /home/moslevin/Project/R1/stage/src/rand_lfsr.h, 401
- /home/moslevin/Project/R1/stage/src/scheduler.cpp, 402
- /home/moslevin/Project/R1/stage/src/scheduler.h, 403, 404
- /home/moslevin/Project/R1/stage/src/screen.cpp, 405
- /home/moslevin/Project/R1/stage/src/screen.h, 406, 407
- /home/moslevin/Project/R1/stage/src/shell_support.cpp, 408
- /home/moslevin/Project/R1/stage/src/shell_support.h, 410, 411
- /home/moslevin/Project/R1/stage/src/slip.cpp, 412
- /home/moslevin/Project/R1/stage/src/slip.h, 416, 417
- /home/moslevin/Project/R1/stage/src/slip_mux.cpp, 417, 418
- /home/moslevin/Project/R1/stage/src/slip_mux.h, 419, 420
- /home/moslevin/Project/R1/stage/src/slipterm.cpp, 421
- /home/moslevin/Project/R1/stage/src/slipterm.h, 422
- /home/moslevin/Project/R1/stage/src/system_heap.cpp, 423
- /home/moslevin/Project/R1/stage/src/system_heap.h, 425, 426
- /home/moslevin/Project/R1/stage/src/system_heap_config.h, 429, 430
- /home/moslevin/Project/R1/stage/src/thread.cpp, 430, 431
- /home/moslevin/Project/R1/stage/src/thread.h, 435, 436
- /home/moslevin/Project/R1/stage/src/threadlist.cpp, 438, 439
- /home/moslevin/Project/R1/stage/src/threadlist.h, 440
- /home/moslevin/Project/R1/stage/src/threadport.cpp, 441
- /home/moslevin/Project/R1/stage/src/threadport.h, 443, 444
- /home/moslevin/Project/R1/stage/src/timerlist.cpp, 446, 447
- /home/moslevin/Project/R1/stage/src/timerlist.h, 451, 452
- /home/moslevin/Project/R1/stage/src/tracebuffer.cpp, 454
- /home/moslevin/Project/R1/stage/src/tracebuffer.h, 455
- /home/moslevin/Project/R1/stage/src/unit_test.cpp, 456
- /home/moslevin/Project/R1/stage/src/unit_test.h, 457
- /home/moslevin/Project/R1/stage/src/writebuf16.cpp, 459
- /home/moslevin/Project/R1/stage/src/writebuf16.h, 461
- Activate
 - ButtonControl, 54
 - CheckBoxControl, 56
 - GamePanelControl, 84
 - GroupBoxControl, 91
 - GuiControl, 95
 - LabelControl, 120
 - NotificationControl, 156
 - PanelControl, 159
 - ProgressControl, 164
 - Screen, 174
 - SlickButtonControl, 181
 - SlickGroupBoxControl, 183
 - SlickProgressControl, 185
 - StubControl, 194
- Add
 - CircularLinkedList, 57
 - DoubleLinkedList, 67
 - DriverList, 78
 - LinkedList, 122
 - Scheduler, 170
 - ThreadList, 205
 - TimerList, 215
 - TimerScheduler, 216
- AddControl
 - GuiWindow, 105
- AddPlugin
 - DCPU, 60
- AddThread
 - Quantum, 168
- AddWindow
 - GuiEventSurface, 101
- AdvanceByte
 - BitStreamer, 49
- Alloc
 - BlockHeap, 51
 - FixedHeap, 82
 - SystemHeap, 195
- Append_Block_To_Node
 - NLFS, 136
- aucBox
 - control_checkbox.cpp, 231
- aucCheck
 - control_checkbox.cpp, 231
- BitStreamer, 49
 - AdvanceByte, 49
 - Init, 49
 - ReadBits, 50
- Bitmap
 - GraphicsDriver, 88
- Block
 - BlockingObject, 52
- BlockHeap, 50

- Alloc, [51](#)
- Create, [51](#)
- Free, [51](#)
- IsFree, [51](#)
- BlockingObject, [52](#)
 - Block, [52](#)
 - UnBlock, [53](#)
- ButtonControl, [53](#)
 - Activate, [54](#)
 - Draw, [54](#)
 - Init, [54](#)
 - ProcessEvent, [54](#)
- CS_ENTER
 - threadport.h, [444](#)
- CS_EXIT
 - threadport.h, [444](#)
- CheckBoxControl, [55](#)
 - Activate, [56](#)
 - Draw, [56](#)
 - Init, [56](#)
 - ProcessEvent, [56](#)
- CheckForOption
 - ShellSupport, [179](#)
- Checksum16
 - MemUtil, [125](#)
- Checksum8
 - MemUtil, [126](#)
- Circle
 - GraphicsDriver, [88](#)
- CircularLinkList, [57](#)
 - Add, [57](#)
 - Remove, [57](#)
- Claim
 - Mutex, [133](#)
- Cleanup_Node_Links
 - NLFS, [137](#)
- Clear
 - EventFlag, [80](#)
- ClearInTimer
 - Quantum, [168](#)
- ClearStale
 - GuiControl, [95](#)
- Close
 - DevNull, [65](#)
 - Driver, [76](#)
 - NLFS_File, [146](#)
- CommandLine_t, [58](#)
- CompareMemory
 - MemUtil, [126](#)
- CompareStrings
 - MemUtil, [126](#)
- Complete
 - UnitTest, [219](#)
- ComputeCurrentTicks
 - ProfileTimer, [162](#)
- ContextSwitchSWI
 - Thread, [199](#)
- Control
 - DevNull, [65](#)
 - Driver, [76](#)
- control_checkbox.cpp
 - aucBox, [231](#)
 - aucCheck, [231](#)
- CopyEvent
 - GuiEventSurface, [101](#)
- CopyMemory
 - MemUtil, [126](#)
- CopyString
 - MemUtil, [127](#)
- Create
 - BlockHeap, [51](#)
 - FixedHeap, [82](#)
- Create_Dir
 - NLFS, [137](#)
- Create_File
 - NLFS, [137](#)
- Create_File_i
 - NLFS, [137](#)
- CycleFocus
 - GuiWindow, [105](#)
- DCPU, [58](#)
 - AddPlugin, [60](#)
 - GetOperand, [60](#)
 - GetRegisters, [61](#)
 - HWN, [61](#)
 - IAQ, [61](#)
 - Init, [61](#)
 - m_clPluginList, [62](#)
 - RFI, [61](#)
 - SendInterrupt, [61](#)
- DCPU_OpBasic
 - dcpu.h, [264](#)
- DCPU_OpExtended
 - dcpu.h, [265](#)
- DCPU_Registers, [62](#)
- DCPUPlugin, [62](#)
 - Enumerate, [63](#)
 - GetDeviceNumber, [63](#)
 - Init, [64](#)
 - Interrupt, [64](#)
- DI
 - KernelSWI, [116](#)
- dcpu.h
 - OP_18, [265](#)
 - OP_19, [265](#)
 - OP_1C, [265](#)
 - OP_1D, [265](#)
 - OP_ADD, [265](#)
 - OP_ADX, [265](#)
 - OP_AND, [265](#)
 - OP_ASR, [265](#)
 - OP_BOR, [265](#)
 - OP_DIV, [265](#)
 - OP_DVI, [265](#)
 - OP_EX_13, [266](#)
 - OP_EX_14, [266](#)

- OP_EX_15, [266](#)
- OP_EX_16, [266](#)
- OP_EX_17, [266](#)
- OP_EX_18, [266](#)
- OP_EX_19, [266](#)
- OP_EX_1A, [266](#)
- OP_EX_1B, [266](#)
- OP_EX_1C, [266](#)
- OP_EX_1D, [266](#)
- OP_EX_1E, [266](#)
- OP_EX_1F, [266](#)
- OP_EX_2, [265](#)
- OP_EX_3, [265](#)
- OP_EX_4, [265](#)
- OP_EX_5, [265](#)
- OP_EX_6, [266](#)
- OP_EX_7, [266](#)
- OP_EX_D, [266](#)
- OP_EX_E, [266](#)
- OP_EX_F, [266](#)
- OP_EX_HWI, [266](#)
- OP_EX_HWN, [266](#)
- OP_EX_HWQ, [266](#)
- OP_EX_IAG, [266](#)
- OP_EX_IAQ, [266](#)
- OP_EX_IAS, [266](#)
- OP_EX_INT, [266](#)
- OP_EX_JSR, [265](#)
- OP_EX_RFI, [266](#)
- OP_IFA, [265](#)
- OP_IFB, [265](#)
- OP_IFC, [265](#)
- OP_IFE, [265](#)
- OP_IFG, [265](#)
- OP_IFL, [265](#)
- OP_IFN, [265](#)
- OP_IFU, [265](#)
- OP_MDI, [265](#)
- OP_MLI, [265](#)
- OP_MOD, [265](#)
- OP_MUL, [265](#)
- OP_NON_BASIC, [265](#)
- OP_SBX, [265](#)
- OP_SET, [265](#)
- OP_SHL, [265](#)
- OP_SHR, [265](#)
- OP_STD, [265](#)
- OP_STI, [265](#)
- OP_SUB, [265](#)
- OP_XOR, [265](#)
- dcpu.h
 - DCPU_OpBasic, [264](#)
 - DCPU_OpExtended, [265](#)
- Deactivate
 - Screen, [174](#)
- DecimalToHex
 - MemUtil, [127](#)
- DecimalToString
- MemUtil, [127](#)
- DecodeByte
 - Slip, [187](#)
- Delete_File
 - NLFS, [138](#)
- Delete_Folder
 - NLFS, [138](#)
- DevNull, [64](#)
 - Close, [65](#)
 - Control, [65](#)
 - Open, [66](#)
 - Read, [66](#)
 - Write, [66](#)
- DoubleLinkedList, [67](#)
 - Add, [67](#)
 - Remove, [68](#)
- Draw
 - ButtonControl, [54](#)
 - CheckBoxControl, [56](#)
 - GamePanelControl, [84](#)
 - GroupBoxControl, [91](#)
 - GuiControl, [95](#)
 - LabelControl, [120](#)
 - NotificationControl, [157](#)
 - PanelControl, [159](#)
 - ProgressControl, [164](#)
 - SlickButtonControl, [181](#)
 - SlickGroupBoxControl, [184](#)
 - SlickProgressControl, [185](#)
 - StubControl, [194](#)
- DrawBitmap_t, [68](#)
- DrawCircle_t, [69](#)
- DrawEllipse_t, [69](#)
- DrawLine_t, [70](#)
- DrawMove_t, [70](#)
- DrawPixel
 - GraphicsDriver, [88](#)
- DrawPoint_t, [71](#)
- DrawPoly_t, [71](#)
- DrawRectangle_t, [72](#)
- DrawStamp_t, [73](#)
- DrawText_t, [73](#)
- DrawVector_t, [74](#)
- DrawWindow_t, [74](#)
- Driver, [75](#)
 - Close, [76](#)
 - Control, [76](#)
 - GetPath, [76](#)
 - Open, [76](#)
 - Read, [77](#)
 - SetName, [77](#)
 - Write, [77](#)
- DriverList, [78](#)
 - Add, [78](#)
 - FindByPath, [78](#)
 - Init, [78](#)
 - Remove, [79](#)
- EVENT_TYPE_COUNT

- gui.h, 315
- EVENT_TYPE_JOYSTICK
 - gui.h, 315
- EVENT_TYPE_KEYBOARD
 - gui.h, 315
- EVENT_TYPE_MOUSE
 - gui.h, 315
- EVENT_TYPE_TIMER
 - gui.h, 315
- EVENT_TYPE_TOUCH
 - gui.h, 315
- Ellipse
 - GraphicsDriver, 88
- EncodeByte
 - Slip, 187
- Enumerate
 - DCPUPlugin, 63
- EventFlag, 79
 - Clear, 80
 - GetMask, 80
 - Set, 80
 - Wait, 80, 81
 - Wait_i, 81
- Exit
 - Thread, 199
- Exit_Simulator
 - Kernel_Aware, 113
- File_Names_Match
 - NLFS, 138
- Find_File
 - NLFS, 138
- Find_Last_Slash
 - NLFS, 139
- Find_Parent_Dir
 - NLFS, 139
- FindByPath
 - DriverList, 78
- FixedHeap, 81
 - Alloc, 82
 - Create, 82
 - Free, 82
- Font_t, 83
- Format
 - NLFS, 139
- fp_internal_command
 - shell_support.h, 411
- Free
 - BlockHeap, 51
 - FixedHeap, 82
 - SystemHeap, 195
- GUI_EVENT_CANCEL
 - gui.h, 315
- GUI_EVENT_CONSUMED
 - gui.h, 315
- GUI_EVENT_OK
 - gui.h, 315
- GUI_EVENT_RETRY
 - gui.h, 315
- GamePanelControl, 83
 - Activate, 84
 - Draw, 84
 - Init, 84
 - ProcessEvent, 84
- GetAverage
 - ProfileTimer, 163
- GetBlockSize
 - NLFS, 140
- GetCode
 - Message, 129
- GetControlIndex
 - GuiControl, 95
- GetControlOffset
 - GuiControl, 96
- GetCount
 - MessageQueue, 131
 - Semaphore, 177
- GetCurPriority
 - Thread, 199
- GetCurrent
 - ProfileTimer, 163
 - Thread, 199
- GetCurrentThread
 - Scheduler, 171
- GetData
 - Message, 129
- GetDeviceNumber
 - DCPUPlugin, 63
- GetDriver
 - GuiWindow, 105
 - Slip, 187
 - SlipMux, 190
- GetEventFlagMask
 - Thread, 199
- GetEventFlagMode
 - Thread, 199
- GetFailed
 - UnitTest, 219
- GetFirstChild
 - NLFS, 140
- GetHead
 - LinkList, 122
- GetHeight
 - GuiControl, 96
 - GuiWindow, 105
- GetID
 - Thread, 200
- GetLeft
 - GuiControl, 96
 - GuiWindow, 105
- GetMask
 - EventFlag, 80
- GetMaxZOrder
 - GuiWindow, 105
- GetName
 - UnitTest, 219

- GetNext
 - LinkListNode, [124](#)
- GetNextPeer
 - NLFS, [140](#)
- GetNextThread
 - Scheduler, [171](#)
- GetNumBlocks
 - NLFS, [140](#)
- GetNumBlocksFree
 - NLFS, [140](#)
- GetNumFiles
 - NLFS, [141](#)
- GetNumFilesFree
 - NLFS, [141](#)
- GetOperand
 - DCPU, [60](#)
- GetOvertime
 - KernelTimer, [118](#)
- GetOwner
 - Thread, [200](#)
- GetParentControl
 - GuiControl, [96](#)
- GetParentWindow
 - GuiControl, [96](#)
- GetPassed
 - UnitTest, [220](#)
- GetPath
 - Driver, [76](#)
- GetPrev
 - LinkListNode, [124](#)
- GetPriority
 - Thread, [200](#)
- GetQuantum
 - Thread, [200](#)
- GetQueue
 - SlipMux, [190](#)
- GetRandom
 - PseudoRandom, [166](#)
- GetRegisters
 - DCPU, [61](#)
- GetResult
 - UnitTest, [220](#)
- GetSlip
 - SlipMux, [190](#)
- GetStackSlack
 - Thread, [200](#)
- GetStat
 - NLFS, [141](#)
- GetStopList
 - Scheduler, [171](#)
- GetTail
 - LinkList, [122](#)
- GetThreadList
 - Scheduler, [171](#)
- GetTop
 - GuiControl, [97](#)
 - GuiWindow, [106](#)
- GetTotal
 - UnitTest, [220](#)
- GetWidth
 - GuiControl, [97](#)
 - GuiWindow, [106](#)
- GetZOrder
 - GuiControl, [97](#)
- GlobalMessagePool, [85](#)
 - Pop, [85](#)
 - Push, [85](#)
- Glyph_t, [86](#)
- GraphicsDriver, [86](#)
 - Bitmap, [88](#)
 - Circle, [88](#)
 - DrawPixel, [88](#)
 - Ellipse, [88](#)
 - Line, [88](#)
 - Move, [88](#)
 - Point, [89](#)
 - ReadPixel, [89](#)
 - Rectangle, [89](#)
 - SetWindow, [89](#)
 - Stamp, [89](#)
 - Text, [89](#)
 - TextFX, [90](#)
 - TriangleFill, [90](#)
 - TriangleWire, [90](#)
- GroupBoxControl, [90](#)
 - Activate, [91](#)
 - Draw, [91](#)
 - Init, [91](#)
 - ProcessEvent, [92](#)
- gui.h
 - EVENT_TYPE_COUNT, [315](#)
 - EVENT_TYPE_JOYSTICK, [315](#)
 - EVENT_TYPE_KEYBOARD, [315](#)
 - EVENT_TYPE_MOUSE, [315](#)
 - EVENT_TYPE_TIMER, [315](#)
 - EVENT_TYPE_TOUCH, [315](#)
 - GUI_EVENT_CANCEL, [315](#)
 - GUI_EVENT_CONSUMED, [315](#)
 - GUI_EVENT_OK, [315](#)
 - GUI_EVENT_RETRY, [315](#)
- gui.h
 - GuiEventType_t, [315](#)
 - GuiReturn_t, [315](#)
- GuiControl, [92](#)
 - Activate, [95](#)
 - ClearStale, [95](#)
 - Draw, [95](#)
 - GetControlIndex, [95](#)
 - GetControlOffset, [96](#)
 - GetHeight, [96](#)
 - GetLeft, [96](#)
 - GetParentControl, [96](#)
 - GetParentWindow, [96](#)
 - GetTop, [97](#)
 - GetWidth, [97](#)
 - GetZOrder, [97](#)

- Init, [97](#)
- IsInFocus, [97](#)
- IsStale, [97](#)
- m_ucControllIndex, [99](#)
- m_ucZOrder, [99](#)
- ProcessEvent, [98](#)
- SetControllIndex, [98](#)
- SetHeight, [98](#)
- SetLeft, [98](#)
- SetParentControl, [98](#)
- SetParentWindow, [99](#)
- SetTop, [99](#)
- SetWidth, [99](#)
- SetZOrder, [99](#)
- GuiEvent_t, [100](#)
- GuiEventSurface, [100](#)
 - AddWindow, [101](#)
 - CopyEvent, [101](#)
 - Init, [102](#)
 - InvalidateRegion, [102](#)
 - ProcessEvent, [102](#)
 - RemoveWindow, [102](#)
 - SendEvent, [102](#)
- GuiEventType_t
 - gui.h, [315](#)
- GuiReturn_t
 - gui.h, [315](#)
- GuiWindow, [103](#)
 - AddControl, [105](#)
 - CycleFocus, [105](#)
 - GetDriver, [105](#)
 - GetHeight, [105](#)
 - GetLeft, [105](#)
 - GetMaxZOrder, [105](#)
 - GetTop, [106](#)
 - GetWidth, [106](#)
 - Init, [106](#)
 - InvalidateRegion, [106](#)
 - IsInFocus, [106](#)
 - m_pclDriver, [109](#)
 - ProcessEvent, [107](#)
 - Redraw, [107](#)
 - RemoveControl, [107](#)
 - SetDriver, [107](#)
 - SetFocus, [107](#)
 - SetHeight, [107](#)
 - SetLeft, [109](#)
 - SetTop, [109](#)
 - SetWidth, [109](#)
- HEAP_BLOCK_SIZE_1
 - system_heap_config.h, [430](#)
- HEAP_RAW_SIZE
 - system_heap.h, [426](#)
- HEAP_RAW_SIZE_1
 - system_heap.h, [426](#)
- HWN
 - DCPU, [61](#)
- HeapConfig, [109](#)
- HighestWaiter
 - ThreadList, [205](#)
- IAQ
 - DCPU, [61](#)
- InheritPriority
 - Thread, [200](#)
- Init
 - BitStreamer, [49](#)
 - ButtonControl, [54](#)
 - CheckBoxControl, [56](#)
 - DCPU, [61](#)
 - DCPUPlugin, [64](#)
 - DriverList, [78](#)
 - GamePanelControl, [84](#)
 - GroupBoxControl, [91](#)
 - GuiControl, [97](#)
 - GuiEventSurface, [102](#)
 - GuiWindow, [106](#)
 - Kernel, [111](#)
 - LabelControl, [121](#)
 - NotificationControl, [157](#)
 - PanelControl, [159](#)
 - Profiler, [160](#)
 - ProfileTimer, [163](#)
 - ProgressControl, [165](#)
 - Semaphore, [177](#)
 - SlickButtonControl, [181](#)
 - SlickGroupBoxControl, [184](#)
 - SlickProgressControl, [185](#)
 - SlipMux, [190](#)
 - SlipTerm, [192](#)
 - StubControl, [194](#)
 - Thread, [201](#)
 - TimerList, [215](#)
 - TimerScheduler, [216](#)
- InitStack
 - ThreadPort, [207](#)
- InstallHandler
 - SlipMux, [190](#)
- Interrupt
 - DCPUPlugin, [64](#)
- InvalidateRegion
 - GuiEventSurface, [102](#)
 - GuiWindow, [106](#)
- IsEnabled
 - Scheduler, [171](#)
- IsFree
 - BlockHeap, [51](#)
- IsInFocus
 - GuiControl, [97](#)
 - GuiWindow, [106](#)
- IsPanic
 - Kernel, [111](#)
- IsStale
 - GuiControl, [97](#)
- IsStarted
 - Kernel, [111](#)

- JoystickEvent_t, 110
- KERNEL_USE_ATOMIC
 - mark3cfg.h, 348
- KERNEL_USE_MESSAGE
 - mark3cfg.h, 349
- KERNEL_USE_PROFILER
 - mark3cfg.h, 349
- KERNEL_USE_QUANTUM
 - mark3cfg.h, 349
- KERNEL_USE_TIMEOUTS
 - mark3cfg.h, 350
- KERNEL_USE_TIMERS
 - mark3cfg.h, 350
- Kernel, 110
 - Init, 111
 - IsPanic, 111
 - IsStarted, 111
 - Panic, 111
 - SetPanic, 112
 - Start, 112
- Kernel_Aware, 112
 - Exit_Simulator, 113
 - Print, 113
 - Profile_Init, 113
 - Profile_Report, 114
 - Profile_Start, 114
 - Profile_Stop, 114
 - Trace, 114, 115
 - Trace_i, 115
- KernelAwareData_t, 115
- KernelSWI, 116
 - DI, 116
 - RI, 116
- KernelTimer, 117
 - GetOvertime, 118
 - RI, 118
 - Read, 118
 - SetExpiry, 118
 - SubtractExpiry, 118
 - TimeToExpiry, 119
- KeyEvent_t, 119
- LabelControl, 119
 - Activate, 120
 - Draw, 120
 - Init, 121
 - ProcessEvent, 121
- Line
 - GraphicsDriver, 88
- LinkedList, 121
 - Add, 122
 - GetHead, 122
 - GetTail, 122
 - Remove, 122
- LinkedListNode, 123
 - GetNext, 124
 - GetPrev, 124
- LoadTile
 - Tile_8x8, 208
- m_clPluginList
 - DCPU, 62
- m_pclDriver
 - GuiWindow, 109
- m_ucControlIndex
 - GuiControl, 99
- m_ucVerbosity
 - SlipTerm, 193
- m_ucZOrder
 - GuiControl, 99
- mark3cfg.h
 - KERNEL_USE_ATOMIC, 348
 - KERNEL_USE_MESSAGE, 349
 - KERNEL_USE_PROFILER, 349
 - KERNEL_USE_QUANTUM, 349
 - KERNEL_USE_TIMEOUTS, 350
 - KERNEL_USE_TIMERS, 350
 - SAFE_UNLINK, 350
- MemUtil, 124
 - Checksum16, 125
 - Checksum8, 126
 - CompareMemory, 126
 - CompareStrings, 126
 - CopyMemory, 126
 - CopyString, 127
 - DecimalToHex, 127
 - DecimalToString, 127
 - SetMemory, 127
 - StringLength, 127
 - StringSearch, 128
 - Tokenize, 128
- Message, 128
 - GetCode, 129
 - GetData, 129
 - SetCode, 129
 - SetData, 130
- MessageQueue, 130
 - GetCount, 131
 - Receive, 131
 - Send, 131
- MessageReceive
 - SlipMux, 191
- Mount
 - NLFS, 141
- MouseEvent_t, 132
- Move
 - GraphicsDriver, 88
- Mutex, 132
 - Claim, 133
 - Release, 133
 - WakeMe, 134
- NLFS_FILE_APPEND
 - nlfs_file.h, 390
- NLFS_FILE_CREATE
 - nlfs_file.h, 390
- NLFS_FILE_READ

- nlfs_file.h, 390
- NLFS_FILE_TRUNCATE
 - nlfs_file.h, 390
- NLFS_FILE_WRITE
 - nlfs_file.h, 390
- NLFS_NODE_DIR
 - nlfs.h, 382
- NLFS_NODE_FILE
 - nlfs.h, 382
- NLFS_NODE_FREE
 - nlfs.h, 382
- NLFS_NODE_ROOT
 - nlfs.h, 382
- NLFS, 134
 - Append_Block_To_Node, 136
 - Cleanup_Node_Links, 137
 - Create_Dir, 137
 - Create_File, 137
 - Create_File_i, 137
 - Delete_File, 138
 - Delete_Folder, 138
 - File_Names_Match, 138
 - Find_File, 138
 - Find_Last_Slash, 139
 - Find_Parent_Dir, 139
 - Format, 139
 - GetBlockSize, 140
 - GetFirstChild, 140
 - GetNextPeer, 140
 - GetNumBlocks, 140
 - GetNumBlocksFree, 140
 - GetNumFiles, 141
 - GetNumFilesFree, 141
 - GetStat, 141
 - Mount, 141
 - Pop_Free_Block, 141
 - Pop_Free_Node, 142
 - Print_Dir_Details, 142
 - Print_File_Details, 142
 - Print_Free_Details, 142
 - Print_Node_Details, 142
 - Push_Free_Block, 143
 - Push_Free_Node, 143
 - Read_Block, 143
 - Read_Block_Header, 143
 - Read_Node, 143
 - RootSync, 144
 - Set_Node_Name, 144
 - Write_Block, 144
 - Write_Block_Header, 144
 - Write_Node, 144
- NLFS_Block_t, 145
- NLFS_File, 145
 - Close, 146
 - Open, 146
 - Read, 146
 - Seek, 147
 - Write, 147
- NLFS_File_Mode
 - nlfs_file.h, 390
- NLFS_File_Node_t, 147
- NLFS_File_Stat_t, 148
- NLFS_Host_t, 149
- NLFS_Node_t, 149
- NLFS_RAM, 150
 - Read_Block, 151
 - Read_Block_Header, 152
 - Read_Node, 152
 - Write_Block, 152
 - Write_Block_Header, 152
 - Write_Node, 154
- NLFS_Root_Node_t, 154
- NLFS_Type_t
 - nlfs.h, 382
- nlfs.h
 - NLFS_NODE_DIR, 382
 - NLFS_NODE_FILE, 382
 - NLFS_NODE_FREE, 382
 - NLFS_NODE_ROOT, 382
- nlfs.h
 - NLFS_Type_t, 382
- nlfs_file.h
 - NLFS_FILE_APPEND, 390
 - NLFS_FILE_CREATE, 390
 - NLFS_FILE_READ, 390
 - NLFS_FILE_TRUNCATE, 390
 - NLFS_FILE_WRITE, 390
- nlfs_file.h
 - NLFS_File_Mode, 390
- NotificationControl, 155
 - Activate, 156
 - Draw, 157
 - Init, 157
 - ProcessEvent, 157
- OP_18
 - dcpu.h, 265
- OP_19
 - dcpu.h, 265
- OP_1C
 - dcpu.h, 265
- OP_1D
 - dcpu.h, 265
- OP_ADD
 - dcpu.h, 265
- OP_ADX
 - dcpu.h, 265
- OP_AND
 - dcpu.h, 265
- OP_ASR
 - dcpu.h, 265
- OP_BOR
 - dcpu.h, 265
- OP_DIV
 - dcpu.h, 265
- OP_DVI
 - dcpu.h, 265

OP_EX_13
 dcpu.h, [266](#)
 OP_EX_14
 dcpu.h, [266](#)
 OP_EX_15
 dcpu.h, [266](#)
 OP_EX_16
 dcpu.h, [266](#)
 OP_EX_17
 dcpu.h, [266](#)
 OP_EX_18
 dcpu.h, [266](#)
 OP_EX_19
 dcpu.h, [266](#)
 OP_EX_1A
 dcpu.h, [266](#)
 OP_EX_1B
 dcpu.h, [266](#)
 OP_EX_1C
 dcpu.h, [266](#)
 OP_EX_1D
 dcpu.h, [266](#)
 OP_EX_1E
 dcpu.h, [266](#)
 OP_EX_1F
 dcpu.h, [266](#)
 OP_EX_2
 dcpu.h, [265](#)
 OP_EX_3
 dcpu.h, [265](#)
 OP_EX_4
 dcpu.h, [265](#)
 OP_EX_5
 dcpu.h, [265](#)
 OP_EX_6
 dcpu.h, [266](#)
 OP_EX_7
 dcpu.h, [266](#)
 OP_EX_D
 dcpu.h, [266](#)
 OP_EX_E
 dcpu.h, [266](#)
 OP_EX_F
 dcpu.h, [266](#)
 OP_EX_HWI
 dcpu.h, [266](#)
 OP_EX_HWN
 dcpu.h, [266](#)
 OP_EX_HWQ
 dcpu.h, [266](#)
 OP_EX_IAG
 dcpu.h, [266](#)
 OP_EX_IAQ
 dcpu.h, [266](#)
 OP_EX_IAS
 dcpu.h, [266](#)
 OP_EX_INT
 dcpu.h, [266](#)
 OP_EX_JSR
 dcpu.h, [265](#)
 OP_EX_RFI
 dcpu.h, [266](#)
 OP_IFA
 dcpu.h, [265](#)
 OP_IFB
 dcpu.h, [265](#)
 OP_IFC
 dcpu.h, [265](#)
 OP_IFE
 dcpu.h, [265](#)
 OP_IFG
 dcpu.h, [265](#)
 OP_IFL
 dcpu.h, [265](#)
 OP_IFN
 dcpu.h, [265](#)
 OP_IFU
 dcpu.h, [265](#)
 OP_MDI
 dcpu.h, [265](#)
 OP_MLI
 dcpu.h, [265](#)
 OP_MOD
 dcpu.h, [265](#)
 OP_MUL
 dcpu.h, [265](#)
 OP_NON_BASIC
 dcpu.h, [265](#)
 OP_SBX
 dcpu.h, [265](#)
 OP_SET
 dcpu.h, [265](#)
 OP_SHL
 dcpu.h, [265](#)
 OP_SHR
 dcpu.h, [265](#)
 OP_STD
 dcpu.h, [265](#)
 OP_STI
 dcpu.h, [265](#)
 OP_SUB
 dcpu.h, [265](#)
 OP_XOR
 dcpu.h, [265](#)
 Open
 DevNull, [66](#)
 Driver, [76](#)
 NLFS_File, [146](#)
 Option_t, [157](#)
 PanelControl, [158](#)
 Activate, [159](#)
 Draw, [159](#)
 Init, [159](#)
 ProcessEvent, [159](#)
 Panic
 Kernel, [111](#)

- Pend
 - Semaphore, 177
- Point
 - GraphicsDriver, 89
- Pop
 - GlobalMessagePool, 85
- Pop_Free_Block
 - NLFS, 141
- Pop_Free_Node
 - NLFS, 142
- Post
 - Semaphore, 177
- Print
 - Kernel_Aware, 113
- Print_Dir_Details
 - NLFS, 142
- Print_File_Details
 - NLFS, 142
- Print_Free_Details
 - NLFS, 142
- Print_Node_Details
 - NLFS, 142
- PrintLn
 - SlipTerm, 192
- Process
 - TimerList, 215
 - TimerScheduler, 216
- ProcessEvent
 - ButtonControl, 54
 - CheckBoxControl, 56
 - GamePanelControl, 84
 - GroupBoxControl, 92
 - GuiControl, 98
 - GuiEventSurface, 102
 - GuiWindow, 107
 - LabelControl, 121
 - NotificationControl, 157
 - PanelControl, 159
 - ProgressControl, 165
 - SlickButtonControl, 181
 - SlickGroupBoxControl, 184
 - SlickProgressControl, 185
 - StubControl, 194
- Profile_Init
 - Kernel_Aware, 113
- Profile_Report
 - Kernel_Aware, 114
- Profile_Start
 - Kernel_Aware, 114
- Profile_Stop
 - Kernel_Aware, 114
- ProfileTimer, 161
 - ComputeCurrentTicks, 162
 - GetAverage, 163
 - GetCurrent, 163
 - Init, 163
 - Start, 163
- Profiler, 160
 - Init, 160
- ProgressControl, 163
 - Activate, 164
 - Draw, 164
 - Init, 165
 - ProcessEvent, 165
- PseudoRandom, 165
 - GetRandom, 166
 - PseudoRandom, 166
 - PseudoRandom, 166
 - Seed, 166
 - Shift, 166
- Push
 - GlobalMessagePool, 85
- Push_Free_Block
 - NLFS, 143
- Push_Free_Node
 - NLFS, 143
- Quantum, 167
 - AddThread, 168
 - ClearInTimer, 168
 - RemoveThread, 168
 - SetInTimer, 168
 - SetTimer, 168
 - UpdateTimer, 168
- RFI
 - DCPU, 61
- RI
 - KernelSWI, 116
 - KernelTimer, 118
- Read
 - DevNull, 66
 - Driver, 77
 - KernelTimer, 118
 - NLFS_File, 146
- Read_Block
 - NLFS, 143
 - NLFS_RAM, 151
- Read_Block_Header
 - NLFS, 143
 - NLFS_RAM, 152
- Read_Node
 - NLFS, 143
 - NLFS_RAM, 152
- ReadBits
 - BitStreamer, 50
- ReadData
 - Slip, 187
- ReadPixel
 - GraphicsDriver, 89
- Receive
 - MessageQueue, 131
- Rectangle
 - GraphicsDriver, 89
- Redraw
 - GuiWindow, 107
- Release

- Mutex, [133](#)
- Remove
 - CircularLinkList, [57](#)
 - DoubleLinkList, [68](#)
 - DriverList, [79](#)
 - LinkList, [122](#)
 - Scheduler, [171](#)
 - ThreadList, [206](#)
 - TimerList, [215](#)
 - TimerScheduler, [216](#)
- RemoveControl
 - GuiWindow, [107](#)
- RemoveThread
 - Quantum, [168](#)
- RemoveWindow
 - GuiEventSurface, [102](#)
- Render
 - Tile_8x8, [208](#)
- RootSync
 - NLFS, [144](#)
- RunCommand
 - ShellSupport, [179](#)
- SLIP_CHANNEL_GRAPHICS
 - slip.h, [416](#)
- SLIP_CHANNEL_HID
 - slip.h, [416](#)
- SLIP_CHANNEL_NVM
 - slip.h, [416](#)
- SLIP_CHANNEL_RESET
 - slip.h, [416](#)
- SLIP_CHANNEL_TERMINAL
 - slip.h, [416](#)
- SLIP_CHANNEL_UNISCOPE
 - slip.h, [416](#)
- SAFE_UNLINK
 - mark3cfg.h, [350](#)
- Schedule
 - Scheduler, [173](#)
- Scheduler, [169](#)
 - Add, [170](#)
 - GetCurrentThread, [171](#)
 - GetNextThread, [171](#)
 - GetStopList, [171](#)
 - GetThreadList, [171](#)
 - IsEnabled, [171](#)
 - Remove, [171](#)
 - Schedule, [173](#)
 - SetScheduler, [173](#)
- Screen, [173](#)
 - Activate, [174](#)
 - Deactivate, [174](#)
- ScreenList, [174](#)
- ScreenManager, [175](#)
- Seed
 - PseudoRandom, [166](#)
- Seek
 - NLFS_File, [147](#)
- Semaphore, [176](#)
- GetCount, [177](#)
- Init, [177](#)
- Pend, [177](#)
- Post, [177](#)
- WakeMe, [177](#)
- Send
 - MessageQueue, [131](#)
- SendEvent
 - GuiEventSurface, [102](#)
- SendInterrupt
 - DCPU, [61](#)
- Set
 - EventFlag, [80](#)
- Set_Node_Name
 - NLFS, [144](#)
- SetCallback
 - Timer, [210](#)
- SetCode
 - Message, [129](#)
- SetControlIndex
 - GuiControl, [98](#)
- SetCurrent
 - Thread, [201](#)
- SetData
 - Message, [130](#)
 - Timer, [210](#)
- SetDriver
 - GuiWindow, [107](#)
 - Slip, [188](#)
- SetEventFlagMask
 - Thread, [201](#)
- SetEventFlagMode
 - Thread, [201](#)
- SetExpiry
 - KernelTimer, [118](#)
- SetFlagPointer
 - ThreadList, [206](#)
- SetFlags
 - Timer, [211](#)
- SetFocus
 - GuiWindow, [107](#)
- SetHeight
 - GuiControl, [98](#)
 - GuiWindow, [107](#)
- SetID
 - Thread, [201](#)
- SetInTimer
 - Quantum, [168](#)
- SetIntervalMSeconds
 - Timer, [211](#)
- SetIntervalSeconds
 - Timer, [211](#)
- SetIntervalTicks
 - Timer, [211](#)
- SetIntervalUSeconds
 - Timer, [211](#)
- SetLeft
 - GuiControl, [98](#)

- GuiWindow, 109
- SetMemory
 - MemUtil, 127
- SetName
 - Driver, 77
 - UnitTest, 220
- SetOwner
 - Thread, 203
 - Timer, 211
- SetPanic
 - Kernel, 112
- SetParentControl
 - GuiControl, 98
- SetParentWindow
 - GuiControl, 99
- SetPriority
 - Thread, 203
 - ThreadList, 206
- SetPriorityBase
 - Thread, 203
- SetQuantum
 - Thread, 203
- SetQueue
 - SlipMux, 191
- SetScheduler
 - Scheduler, 173
- SetTimer
 - Quantum, 168
- SetTolerance
 - Timer, 213
- SetTop
 - GuiControl, 99
 - GuiWindow, 109
- SetVerbosity
 - SlipTerm, 192
- SetWidth
 - GuiControl, 99
 - GuiWindow, 109
- SetWindow
 - GraphicsDriver, 89
- SetZOrder
 - GuiControl, 99
- shell_support.h
 - fp_internal_command, 411
- ShellCommand_t, 178
- ShellSupport, 178
 - CheckForOption, 179
 - RunCommand, 179
 - TokensToCommandLine, 179
 - UnescapeToken, 180
- Shift
 - PseudoRandom, 166
- Sleep
 - Thread, 203
- SlickButtonControl, 180
 - Activate, 181
 - Draw, 181
 - Init, 181
- ProcessEvent, 181
- SlickGroupBoxControl, 182
 - Activate, 183
 - Draw, 184
 - Init, 184
 - ProcessEvent, 184
- SlickProgressControl, 184
 - Activate, 185
 - Draw, 185
 - Init, 185
 - ProcessEvent, 185
- Slip, 186
 - DecodeByte, 187
 - EncodeByte, 187
 - GetDriver, 187
 - ReadData, 187
 - SetDriver, 188
 - WriteData, 188
 - WriteVector, 188
- slip.h
 - SLIP_CHANNEL_GRAPHICS, 416
 - SLIP_CHANNEL_HID, 416
 - SLIP_CHANNEL_NVM, 416
 - SLIP_CHANNEL_RESET, 416
 - SLIP_CHANNEL_TERMINAL, 416
 - SLIP_CHANNEL_UNISCOPE, 416
- slip.h
 - SlipChannel, 416
- slip_mux.cpp
 - SlipMux_CallBack, 418
- SlipChannel
 - slip.h, 416
- SlipDataVector, 188
- SlipMux, 189
 - GetDriver, 190
 - GetQueue, 190
 - GetSlip, 190
 - Init, 190
 - InstallHandler, 190
 - MessageReceive, 191
 - SetQueue, 191
- SlipMux_CallBack
 - slip_mux.cpp, 418
- SlipTerm, 191
 - Init, 192
 - m_ucVerbosity, 193
 - PrintLn, 192
 - SetVerbosity, 192
 - StrLen, 192
- Stamp
 - GraphicsDriver, 89
- Start
 - Kernel, 112
 - ProfileTimer, 163
 - Timer, 213
- Stop
 - Thread, 203
 - Timer, 213

- StrLen
 - SlipTerm, [192](#)
- StringLength
 - MemUtil, [127](#)
- StringSearch
 - MemUtil, [128](#)
- StubControl, [193](#)
 - Activate, [194](#)
 - Draw, [194](#)
 - Init, [194](#)
 - ProcessEvent, [194](#)
- SubtractExpiry
 - KernelTimer, [118](#)
- system_heap.h
 - HEAP_RAW_SIZE, [426](#)
 - HEAP_RAW_SIZE_1, [426](#)
- SystemHeap, [194](#)
 - Alloc, [195](#)
 - Free, [195](#)
- Text
 - GraphicsDriver, [89](#)
- TextFX
 - GraphicsDriver, [90](#)
- TextFX_t, [196](#)
- Thread, [196](#)
 - ContextSwitchSWI, [199](#)
 - Exit, [199](#)
 - GetCurPriority, [199](#)
 - GetCurrent, [199](#)
 - GetEventFlagMask, [199](#)
 - GetEventFlagMode, [199](#)
 - GetID, [200](#)
 - GetOwner, [200](#)
 - GetPriority, [200](#)
 - GetQuantum, [200](#)
 - GetStackSlack, [200](#)
 - InheritPriority, [200](#)
 - Init, [201](#)
 - SetCurrent, [201](#)
 - SetEventFlagMask, [201](#)
 - SetEventFlagMode, [201](#)
 - SetID, [201](#)
 - SetOwner, [203](#)
 - SetPriority, [203](#)
 - SetPriorityBase, [203](#)
 - SetQuantum, [203](#)
 - Sleep, [203](#)
 - Stop, [203](#)
 - USleep, [204](#)
 - Yield, [204](#)
- ThreadList, [204](#)
 - Add, [205](#)
 - HighestWaiter, [205](#)
 - Remove, [206](#)
 - SetFlagPointer, [206](#)
 - SetPriority, [206](#)
- ThreadPort, [206](#)
 - InitStack, [207](#)
- threadport.h
 - CS_ENTER, [444](#)
 - CS_EXIT, [444](#)
- Tile_8x8, [207](#)
 - LoadTile, [208](#)
 - Render, [208](#)
- TileDef_t, [208](#)
- TimeToExpiry
 - KernelTimer, [119](#)
- Timer, [209](#)
 - SetCallback, [210](#)
 - SetData, [210](#)
 - SetFlags, [211](#)
 - SetIntervalMSeconds, [211](#)
 - SetIntervalSeconds, [211](#)
 - SetIntervalTicks, [211](#)
 - SetIntervalUSeconds, [211](#)
 - SetOwner, [211](#)
 - SetTolerance, [213](#)
 - Start, [213](#)
 - Stop, [213](#)
- TimerEvent_t, [214](#)
- TimerList, [214](#)
 - Add, [215](#)
 - Init, [215](#)
 - Process, [215](#)
 - Remove, [215](#)
- TimerScheduler, [215](#)
 - Add, [216](#)
 - Init, [216](#)
 - Process, [216](#)
 - Remove, [216](#)
- Token_t, [217](#)
- Tokenize
 - MemUtil, [128](#)
- TokensToCommandLine
 - ShellSupport, [179](#)
- TouchEvent_t, [217](#)
- Trace
 - Kernel_Aware, [114](#), [115](#)
- Trace_i
 - Kernel_Aware, [115](#)
- TriangleFill
 - GraphicsDriver, [90](#)
- TriangleWire
 - GraphicsDriver, [90](#)
- USleep
 - Thread, [204](#)
- UnBlock
 - BlockingObject, [53](#)
- UnescapeToken
 - ShellSupport, [180](#)
- UnitTest, [218](#)
 - Complete, [219](#)
 - GetFailed, [219](#)
 - GetName, [219](#)
 - GetPassed, [220](#)
 - GetResult, [220](#)

- GetTotal, [220](#)
- SetName, [220](#)
- UpdateTimer
 - Quantum, [168](#)
- Wait
 - EventFlag, [80](#), [81](#)
- Wait_i
 - EventFlag, [81](#)
- WakeMe
 - Mutex, [134](#)
 - Semaphore, [177](#)
- Write
 - DevNull, [66](#)
 - Driver, [77](#)
 - NLFS_File, [147](#)
- Write_Block
 - NLFS, [144](#)
 - NLFS_RAM, [152](#)
- Write_Block_Header
 - NLFS, [144](#)
 - NLFS_RAM, [152](#)
- Write_Node
 - NLFS, [144](#)
 - NLFS_RAM, [154](#)
- WriteData
 - Slip, [188](#)
- WriteVector
 - Slip, [188](#)
- Yield
 - Thread, [204](#)