

MATE

1.1

Generated by Doxygen 1.7.3

Sun Sep 18 2011 23:49:55



# Contents

<b>1</b>	<b>Deprecated List</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>7</b>
3.1	Class List . . . . .	7
<b>4</b>	<b>Class Documentation</b>	<b>13</b>
4.1	ACProxy Class Reference . . . . .	13
4.1.1	Detailed Description . . . . .	14
4.1.2	Constructor & Destructor Documentation . . . . .	14
4.1.2.1	ACProxy . . . . .	14
4.1.3	Member Function Documentation . . . . .	15
4.1.3.1	AddInstr . . . . .	15
4.1.3.2	FuncParamChange . . . . .	15
4.1.3.3	InsertFunctionCall . . . . .	15
4.1.3.4	LoadLibrary . . . . .	16
4.1.3.5	OneTimeFuncCall . . . . .	16
4.1.3.6	RemoveFuncCall . . . . .	16
4.1.3.7	RemoveInstr . . . . .	16
4.1.3.8	ReplaceFunction . . . . .	17
4.1.3.9	SetVariableValue . . . . .	17
4.1.3.10	StartApplication . . . . .	17
4.2	Common::ActiveObject Class Reference . . . . .	18
4.2.1	Detailed Description . . . . .	19
4.3	Common::AddInstrRequest Class Reference . . . . .	19
4.3.1	Detailed Description . . . . .	20
4.3.2	Constructor & Destructor Documentation . . . . .	21
4.3.2.1	AddInstrRequest . . . . .	21
4.4	Common::Address Class Reference . . . . .	21
4.4.1	Detailed Description . . . . .	22
4.4.2	Constructor & Destructor Documentation . . . . .	22
4.4.2.1	Address . . . . .	22
4.4.2.2	Address . . . . .	22
4.4.2.3	Address . . . . .	23
4.4.3	Member Function Documentation . . . . .	23
4.4.3.1	GetHostName . . . . .	23
4.4.3.2	GetSize . . . . .	23

4.5	AdjustingNWTunlet Class Reference	23
4.5.1	Member Function Documentation	24
4.5.1.1	FindIterData	24
4.5.1.2	HandleEvent	24
4.5.1.3	Initialize	25
4.5.1.4	InsertEvents	25
4.5.1.5	InsertMasterEvents	25
4.5.1.6	InsertWorkerEvents	25
4.5.1.7	TaskStarted	25
4.5.1.8	TaskTerminated	25
4.5.1.9	TryTuning	25
4.6	Analyzer Class Reference	26
4.6.1	Constructor & Destructor Documentation	26
4.6.1.1	Analyzer	26
4.7	Model::Application Class Reference	26
4.7.1	Detailed Description	29
4.7.2	Constructor & Destructor Documentation	29
4.7.2.1	Application	29
4.7.3	Member Function Documentation	30
4.7.3.1	AddEvent	30
4.7.3.2	AddHost	30
4.7.3.3	AddTask	30
4.7.3.4	DispatchEvent	30
4.7.3.5	FuncParamChange	31
4.7.3.6	GetHosts	31
4.7.3.7	GetName	31
4.7.3.8	GetStatus	31
4.7.3.9	GetTasks	32
4.7.3.10	InsertFunctionCall	32
4.7.3.11	LoadLibrary	32
4.7.3.12	NumActiveTasks	32
4.7.3.13	OneTimeFuncCall	33
4.7.3.14	OnEvent	33
4.7.3.15	ProcessEvent	33
4.7.3.16	ProcessEvents	33
4.7.3.17	RemoveEvent	34
4.7.3.18	RemoveFuncCall	34
4.7.3.19	RemoveTask	34
4.7.3.20	ReplaceFunction	34
4.7.3.21	SetHostHandler	35
4.7.3.22	SetTaskHandler	35
4.7.3.23	SetVariableValue	35
4.7.3.24	Start	35
4.8	Common::Attribute Class Reference	36
4.8.1	Detailed Description	37
4.8.2	Constructor & Destructor Documentation	37
4.8.2.1	Attribute	37
4.9	Common::AttributeValue Class Reference	37
4.9.1	Detailed Description	39
4.9.2	Member Function Documentation	39

4.9.2.1	GetCharValue	39
4.9.2.2	GetDoubleValue	40
4.9.2.3	GetFloatValue	40
4.9.2.4	GetIntValue	40
4.9.2.5	GetShortValue	40
4.9.2.6	GetStringValue	40
4.10	auto_iterator< T > Class Template Reference	41
4.11	auto_vector< T > Class Template Reference	41
4.12	Common::BasicLogFilter Class Reference	42
4.12.1	Detailed Description	42
4.12.2	Constructor & Destructor Documentation	42
4.12.2.1	BasicLogFilter	42
4.13	Common::BasicLogFormatter Class Reference	43
4.13.1	Detailed Description	44
4.13.2	Constructor & Destructor Documentation	44
4.13.2.1	BasicLogFormatter	44
4.14	Common::BasicLogger Class Reference	44
4.14.1	Detailed Description	45
4.14.2	Member Function Documentation	46
4.14.2.1	Accept	46
4.15	BatchData Class Reference	46
4.15.1	Detailed Description	47
4.16	Common::Breakpoint Class Reference	47
4.16.1	Detailed Description	48
4.17	Common::ByteStream Class Reference	48
4.17.1	Detailed Description	49
4.17.2	Constructor & Destructor Documentation	49
4.17.2.1	ByteStream	49
4.17.2.2	ByteStream	49
4.17.3	Member Function Documentation	50
4.17.3.1	GetData	50
4.17.3.2	GetDataSize	50
4.17.3.3	Write	50
4.18	CommandLine Class Reference	50
4.18.1	Detailed Description	52
4.18.2	Constructor & Destructor Documentation	52
4.18.2.1	CommandLine	52
4.18.3	Member Function Documentation	53
4.18.3.1	DisplayHelp	53
4.18.3.2	GetAppArgc	53
4.18.3.3	GetAppArgc	53
4.18.3.4	GetAppArgv	53
4.18.3.5	GetAppArgv	53
4.18.3.6	GetAppPath	53
4.18.3.7	GetAppPath	54
4.18.3.8	GetArgc	54
4.18.3.9	GetArgc	54
4.18.3.10	GetArgv	54
4.18.3.11	GetArgv	54
4.18.3.12	GetConfigFile	54

4.18.3.13	GetConfigFileName	55
4.18.3.14	HasConfig	55
4.18.3.15	HasConfig	55
4.18.3.16	IsOk	55
4.18.3.17	IsOk	55
4.19	comptime Struct Reference	56
4.20	Common::Config Class Reference	56
4.20.1	Detailed Description	57
4.20.2	Member Function Documentation	57
4.20.2.1	AddEntry	57
4.20.2.2	Contains	58
4.20.2.3	GetBoolValue	58
4.20.2.4	GetBoolValue	58
4.20.2.5	GetIntValue	59
4.20.2.6	GetIntValue	59
4.20.2.7	GetKeys	59
4.20.2.8	GetStringValue	60
4.21	Common::ConfigException Class Reference	60
4.21.1	Detailed Description	61
4.21.2	Constructor & Destructor Documentation	61
4.21.2.1	ConfigException	61
4.21.3	Member Function Documentation	61
4.21.3.1	Display	61
4.21.3.2	GetReason	61
4.22	Common::ConfigHelper Class Reference	62
4.22.1	Detailed Description	62
4.22.2	Member Function Documentation	62
4.22.2.1	ReadFromFile	62
4.23	Common::ConfigMap Class Reference	63
4.23.1	Detailed Description	63
4.23.2	Member Function Documentation	64
4.23.2.1	Add	64
4.23.2.2	Contains	64
4.23.2.3	GetValue	64
4.24	Common::ConfigReader Class Reference	65
4.24.1	Detailed Description	65
4.25	Controller Class Reference	66
4.25.1	Detailed Description	66
4.25.2	Constructor & Destructor Documentation	67
4.25.2.1	Controller	67
4.25.3	Member Function Documentation	67
4.25.3.1	Run	67
4.25.3.2	Run	67
4.26	Common::CountingSerializer Class Reference	68
4.26.1	Detailed Description	69
4.27	curState Struct Reference	69
4.28	Common::DateTime Class Reference	69
4.28.1	Detailed Description	70
4.28.2	Member Function Documentation	70
4.28.2.1	GetStringValue	70

4.29	Common::DeSerializer Class Reference	71
4.29.1	Detailed Description	71
4.30	DiEx Class Reference	72
4.31	DiFunction Class Reference	72
4.32	DiImage Class Reference	73
4.33	DiIntType Class Reference	73
4.34	DiIntVariable Class Reference	73
4.35	DiPoint Class Reference	74
4.36	DiProcess Class Reference	74
4.37	DiSnippetHandle Class Reference	75
4.38	DiType Class Reference	76
4.39	DiVariable Class Reference	76
4.40	DTLibrary Class Reference	77
4.40.1	Detailed Description	77
4.40.2	Member Function Documentation	77
4.40.2.1	CreateApplication	77
4.40.2.2	GetApplication	77
4.41	DTLibraryFactory Class Reference	78
4.41.1	Detailed Description	78
4.41.2	Member Function Documentation	78
4.41.2.1	CreateLibrary	78
4.41.2.2	DestroyLibrary	79
4.42	DynInst Class Reference	79
4.43	ECPAcceptor Class Reference	79
4.43.1	Detailed Description	80
4.43.2	Constructor & Destructor Documentation	80
4.43.2.1	ECPAcceptor	80
4.43.3	Member Function Documentation	80
4.43.3.1	GetHandle	80
4.43.3.2	SetEventCollector	80
4.44	ECPHandler Class Reference	80
4.44.1	Detailed Description	81
4.44.2	Member Function Documentation	81
4.44.2.1	GetHandle	81
4.44.2.2	SetService	81
4.45	Common::ECPMessage Class Reference	82
4.45.1	Detailed Description	82
4.45.2	Constructor & Destructor Documentation	83
4.45.2.1	ECPMessage	83
4.46	Common::ECPMsgHeader Class Reference	83
4.46.1	Detailed Description	84
4.47	ECPProtocol Class Reference	85
4.47.1	Detailed Description	85
4.47.2	Member Function Documentation	85
4.47.2.1	ReadMessageEx	85
4.47.2.2	ReadMessageHeader	85
4.48	Model::Event Class Reference	86
4.48.1	Detailed Description	87
4.48.2	Constructor & Destructor Documentation	87
4.48.2.1	Event	87

4.48.2.2	Event	87
4.48.3	Member Function Documentation	87
4.48.3.1	GetAttributes	87
4.48.3.2	GetEventHandler	88
4.48.3.3	GetFunctionName	88
4.48.3.4	GetId	88
4.48.3.5	GetInstrPlace	88
4.48.3.6	GetNumAttributes	88
4.48.3.7	SetAttribute	89
4.48.3.8	SetEventHandler	89
4.49	Common::Event Class Reference	89
4.49.1	Detailed Description	90
4.49.2	Constructor & Destructor Documentation	90
4.49.2.1	Event	90
4.50	EventCollector Class Reference	91
4.50.1	Detailed Description	92
4.50.2	Constructor & Destructor Documentation	92
4.50.2.1	EventCollector	92
4.50.2.2	~EventCollector	92
4.50.3	Member Function Documentation	92
4.50.3.1	GetListener	92
4.50.3.2	IsAborted	92
4.50.3.3	SetListener	92
4.51	DMLib::EventCollectorProxy Class Reference	93
4.51.1	Detailed Description	93
4.52	Common::EventDemultiplexer Class Reference	94
4.52.1	Detailed Description	94
4.52.2	Member Function Documentation	95
4.52.2.1	Select	95
4.53	Common::EventException Class Reference	95
4.53.1	Detailed Description	96
4.53.2	Constructor & Destructor Documentation	96
4.53.2.1	EventException	96
4.53.3	Member Function Documentation	96
4.53.3.1	Display	96
4.53.3.2	GetReason	96
4.54	Common::EventHandler Class Reference	97
4.54.1	Detailed Description	97
4.55	Model::EventHandler Class Reference	97
4.55.1	Detailed Description	98
4.55.2	Member Function Documentation	98
4.55.2.1	HandleEvent	98
4.56	EventListener Class Reference	98
4.56.1	Detailed Description	99
4.56.2	Member Function Documentation	99
4.56.2.1	OnEvent	99
4.57	Common::EventMap Class Reference	99
4.57.1	Detailed Description	100
4.57.2	Member Function Documentation	100
4.57.2.1	Add	100



4.57.2.2	GetId . . . . .	100
4.58	Common::EventMsg Class Reference . . . . .	101
4.58.1	Detailed Description . . . . .	102
4.58.2	Member Function Documentation . . . . .	103
4.58.2.1	Reset . . . . .	103
4.59	EventMsgReader Class Reference . . . . .	103
4.59.1	Member Function Documentation . . . . .	104
4.59.1.1	GetAttrType . . . . .	104
4.59.1.2	GetCharValue . . . . .	104
4.59.1.3	GetDoubleValue . . . . .	104
4.59.1.4	GetFloatValue . . . . .	104
4.59.1.5	GetIntValue . . . . .	104
4.59.1.6	GetParamCount . . . . .	105
4.59.1.7	GetShortValue . . . . .	105
4.59.1.8	GetStringValue . . . . .	105
4.60	DMLib::EventMsgWriter Class Reference . . . . .	105
4.60.1	Detailed Description . . . . .	106
4.60.2	Member Function Documentation . . . . .	106
4.60.2.1	OpenEvent . . . . .	106
4.61	Model::EventRecord Class Reference . . . . .	107
4.61.1	Detailed Description . . . . .	108
4.61.2	Constructor & Destructor Documentation . . . . .	108
4.61.2.1	EventRecord . . . . .	108
4.61.3	Member Function Documentation . . . . .	108
4.61.3.1	GetAttributeValue . . . . .	108
4.61.3.2	GetAttributeValues . . . . .	108
4.61.3.3	GetEvent . . . . .	109
4.61.3.4	GetEventId . . . . .	109
4.61.3.5	GetTask . . . . .	109
4.61.3.6	GetTimestamp . . . . .	109
4.61.3.7	ParseAttrs . . . . .	109
4.62	Model::Events Class Reference . . . . .	110
4.62.1	Detailed Description . . . . .	110
4.62.2	Member Function Documentation . . . . .	110
4.62.2.1	Add . . . . .	110
4.62.2.2	Find . . . . .	110
4.62.2.3	Remove . . . . .	111
4.62.2.4	Size . . . . .	111
4.63	Common::Exception Class Reference . . . . .	111
4.63.1	Detailed Description . . . . .	112
4.63.2	Constructor & Destructor Documentation . . . . .	113
4.63.2.1	Exception . . . . .	113
4.63.3	Member Function Documentation . . . . .	113
4.63.3.1	Display . . . . .	113
4.64	Common::ExecProcess Class Reference . . . . .	113
4.64.1	Detailed Description . . . . .	114
4.64.2	Constructor & Destructor Documentation . . . . .	115
4.64.2.1	ExecProcess . . . . .	115
4.64.3	Member Function Documentation . . . . .	115
4.64.3.1	Start . . . . .	115

4.64.3.2	WaitForEvent	115
4.65	FactoringTunlet Class Reference	116
4.65.1	Detailed Description	116
4.65.2	Member Function Documentation	116
4.65.2.1	HandleEvent	116
4.65.2.2	Initialize	116
4.65.2.3	TaskStarted	117
4.65.2.4	TaskTerminated	117
4.66	Common::FileConfigReader Class Reference	117
4.66.1	Detailed Description	118
4.66.2	Constructor & Destructor Documentation	118
4.66.2.1	FileConfigReader	118
4.66.3	Member Function Documentation	118
4.66.3.1	Read	118
4.67	Common::FileLogger Class Reference	119
4.67.1	Detailed Description	119
4.67.2	Constructor & Destructor Documentation	119
4.67.2.1	FileLogger	119
4.68	Common::FuncDef Class Reference	120
4.68.1	Detailed Description	120
4.68.2	Constructor & Destructor Documentation	121
4.68.2.1	FuncDef	121
4.69	Common::FuncDefException Class Reference	121
4.69.1	Detailed Description	122
4.69.2	Constructor & Destructor Documentation	122
4.69.2.1	FuncDefException	122
4.69.3	Member Function Documentation	122
4.69.3.1	Display	122
4.69.3.2	GetReason	123
4.70	Common::FuncDefs Class Reference	123
4.70.1	Detailed Description	123
4.70.2	Constructor & Destructor Documentation	124
4.70.2.1	FuncDefs	124
4.70.3	Member Function Documentation	124
4.70.3.1	Add	124
4.70.3.2	Find	124
4.71	Common::FunctionParamChangeRequest Class Reference	124
4.71.1	Detailed Description	125
4.71.2	Constructor & Destructor Documentation	126
4.71.2.1	FunctionParamChangeRequest	126
4.72	Common::HandlerMap Class Reference	126
4.72.1	Detailed Description	127
4.73	Model::Host Class Reference	127
4.73.1	Detailed Description	128
4.73.2	Member Function Documentation	128
4.73.2.1	GetName	128
4.74	Model::HostHandler Class Reference	128
4.74.1	Detailed Description	128
4.74.2	Member Function Documentation	129
4.74.2.1	HostAdded	129

4.74.2.2	HostRemoved	129
4.75	Common::InsertFunctionCallRequest Class Reference	129
4.75.1	Detailed Description	130
4.75.2	Constructor & Destructor Documentation	131
4.75.2.1	InsertFunctionCallRequest	131
4.76	InstrGroup Class Reference	131
4.76.1	Detailed Description	132
4.76.2	Member Function Documentation	132
4.76.2.1	AddHandler	132
4.76.2.2	begin	133
4.76.2.3	end	133
4.76.2.4	GetEventId	133
4.76.2.5	GetFuncName	133
4.76.2.6	GetSize	133
4.76.2.7	IsEmpty	133
4.76.2.8	RemoveHandler	134
4.77	Common::ConfigMap::Iterator Class Reference	134
4.77.1	Detailed Description	134
4.78	IterData Class Reference	135
4.78.1	Detailed Description	135
4.79	Common::Config::KeyIterator Class Reference	136
4.79.1	Detailed Description	136
4.80	Common::LoadLibraryRequest Class Reference	137
4.80.1	Detailed Description	137
4.80.2	Constructor & Destructor Documentation	138
4.80.2.1	LoadLibraryRequest	138
4.81	Common::LogEntry Class Reference	138
4.81.1	Detailed Description	139
4.81.2	Constructor & Destructor Documentation	139
4.81.2.1	LogEntry	139
4.81.3	Member Function Documentation	139
4.81.3.1	GetSeverity	139
4.82	Common::LogFilter Class Reference	139
4.82.1	Detailed Description	140
4.82.2	Member Function Documentation	140
4.82.2.1	Accept	140
4.83	Common::LogFormatter Class Reference	141
4.83.1	Detailed Description	141
4.84	Common::Logger Class Reference	141
4.84.1	Detailed Description	142
4.85	ModelParam Struct Reference	143
4.86	ModuleList Class Reference	143
4.87	Monitor Class Reference	143
4.87.1	Detailed Description	144
4.87.2	Constructor & Destructor Documentation	144
4.87.2.1	Monitor	144
4.87.3	Member Function Documentation	144
4.87.3.1	AddInstr	144
4.87.3.2	RemoveInstr	144
4.88	Common::Mutex Class Reference	145

4.88.1 Detailed Description . . . . .	145
4.88.2 Constructor & Destructor Documentation . . . . .	146
4.88.2.1 Mutex . . . . .	146
4.88.3 Member Function Documentation . . . . .	146
4.88.3.1 Enter . . . . .	146
4.88.3.2 Leave . . . . .	146
4.89 Common::MutexLock Class Reference . . . . .	146
4.89.1 Detailed Description . . . . .	147
4.90 myauto_ptr< X > Class Template Reference . . . . .	147
4.91 Common::NetworkDeSerializer Class Reference . . . . .	147
4.91.1 Detailed Description . . . . .	149
4.92 Common::NetworkSerializer Class Reference . . . . .	149
4.92.1 Detailed Description . . . . .	150
4.92.2 Constructor & Destructor Documentation . . . . .	150
4.92.2.1 NetworkSerializer . . . . .	150
4.93 Common::OneTimeFunctionCallRequest Class Reference . . . . .	151
4.93.1 Detailed Description . . . . .	152
4.93.2 Constructor & Destructor Documentation . . . . .	152
4.93.2.1 OneTimeFunctionCallRequest . . . . .	152
4.94 Common::OutputStream Class Reference . . . . .	152
4.94.1 Detailed Description . . . . .	153
4.95 Common::Pipe Class Reference . . . . .	153
4.95.1 Detailed Description . . . . .	154
4.95.2 Constructor & Destructor Documentation . . . . .	155
4.95.2.1 Pipe . . . . .	155
4.95.3 Member Function Documentation . . . . .	155
4.95.3.1 Read . . . . .	155
4.95.3.2 Write . . . . .	155
4.96 PointList Class Reference . . . . .	155
4.97 ProcedureList Class Reference . . . . .	156
4.98 Common::Process Class Reference . . . . .	156
4.98.1 Detailed Description . . . . .	157
4.99 PTPAcceptor Class Reference . . . . .	157
4.99.1 Detailed Description . . . . .	158
4.99.2 Constructor & Destructor Documentation . . . . .	158
4.99.2.1 PTPAcceptor . . . . .	158
4.99.3 Member Function Documentation . . . . .	158
4.99.3.1 GetHandle . . . . .	158
4.100 PTPHandler Class Reference . . . . .	159
4.100.1 Detailed Description . . . . .	159
4.100.2 Constructor & Destructor Documentation . . . . .	159
4.100.2.1 PTPHandler . . . . .	159
4.100.3 Member Function Documentation . . . . .	160
4.100.3.1 GetHandle . . . . .	160
4.101 Common::PTPMessage Class Reference . . . . .	160
4.101.1 Detailed Description . . . . .	161
4.102 Common::PTPMsgHeader Class Reference . . . . .	161
4.102.1 Detailed Description . . . . .	162
4.103 Common::PTPProtocol Class Reference . . . . .	163
4.103.1 Detailed Description . . . . .	163

4.104Common::Queue< T > Class Template Reference . . . . .	163
4.104.1 Detailed Description . . . . .	164
4.104.2 Member Function Documentation . . . . .	165
4.104.2.1 Get . . . . .	165
4.104.2.2 GetB . . . . .	165
4.104.2.3 Put . . . . .	165
4.105Common::Reactor Class Reference . . . . .	165
4.105.1 Detailed Description . . . . .	166
4.106Common::RegisterMsg Class Reference . . . . .	166
4.106.1 Detailed Description . . . . .	167
4.106.2 Constructor & Destructor Documentation . . . . .	167
4.106.2.1 RegisterMsg . . . . .	167
4.107Common::RemoteProcess Class Reference . . . . .	168
4.107.1 Detailed Description . . . . .	168
4.107.2 Constructor & Destructor Documentation . . . . .	169
4.107.2.1 RemoteProcess . . . . .	169
4.107.2.2 RemoteProcess . . . . .	169
4.108Common::RemoveFunctionCallRequest Class Reference . . . . .	169
4.108.1 Detailed Description . . . . .	170
4.108.2 Constructor & Destructor Documentation . . . . .	171
4.108.2.1 RemoveFunctionCallRequest . . . . .	171
4.109Common::RemoveInstrRequest Class Reference . . . . .	171
4.109.1 Detailed Description . . . . .	172
4.109.2 Constructor & Destructor Documentation . . . . .	172
4.109.2.1 RemoveInstrRequest . . . . .	172
4.110Common::ReplaceFunctionRequest Class Reference . . . . .	173
4.110.1 Detailed Description . . . . .	174
4.110.2 Constructor & Destructor Documentation . . . . .	174
4.110.2.1 ReplaceFunctionRequest . . . . .	174
4.111Common::Semaphore Class Reference . . . . .	174
4.111.1 Detailed Description . . . . .	175
4.111.2 Constructor & Destructor Documentation . . . . .	176
4.111.2.1 Semaphore . . . . .	176
4.111.3 Member Function Documentation . . . . .	176
4.111.3.1 Post . . . . .	176
4.111.3.2 TryWait . . . . .	176
4.111.3.3 Wait . . . . .	176
4.112Common::Serializable Class Reference . . . . .	176
4.112.1 Detailed Description . . . . .	177
4.113Common::Serializer Class Reference . . . . .	177
4.113.1 Detailed Description . . . . .	178
4.114Common::ServerSocket Class Reference . . . . .	178
4.114.1 Detailed Description . . . . .	179
4.114.2 Constructor & Destructor Documentation . . . . .	179
4.114.2.1 ServerSocket . . . . .	179
4.115Service Class Reference . . . . .	180
4.115.1 Detailed Description . . . . .	180
4.115.2 Constructor & Destructor Documentation . . . . .	180
4.115.2.1 Service . . . . .	180
4.115.3 Member Function Documentation . . . . .	181

4.115.3.1 Add	181
4.115.3.2 Remove	181
4.116Common::SetVariableValueRequest Class Reference	181
4.116.1 Detailed Description	182
4.116.2 Constructor & Destructor Documentation	183
4.116.2.1 SetVariableValueRequest	183
4.116.3 Member Function Documentation	183
4.116.3.1 GetValueBuffer	183
4.117ShutDownManager Class Reference	183
4.117.1 Detailed Description	184
4.117.2 Member Function Documentation	184
4.117.2.1 isFinished	184
4.117.2.2 Run	185
4.117.2.3 setApp	185
4.118ShutDownSlave Class Reference	185
4.118.1 Detailed Description	186
4.119SnippetHandler Class Reference	186
4.119.1 Detailed Description	187
4.119.2 Constructor & Destructor Documentation	187
4.119.2.1 SnippetHandler	187
4.119.3 Member Function Documentation	187
4.119.3.1 GetEventId	187
4.119.3.2 GetFuncName	187
4.119.3.3 GetHandle	188
4.119.3.4 GetInstrPlace	188
4.120SnippetMaker Class Reference	188
4.120.1 Detailed Description	188
4.120.2 Constructor & Destructor Documentation	189
4.120.2.1 SnippetMaker	189
4.120.3 Member Function Documentation	189
4.120.3.1 MakeEventSnippet	189
4.121Common::Socket Class Reference	189
4.121.1 Detailed Description	191
4.121.2 Constructor & Destructor Documentation	191
4.121.2.1 Socket	191
4.121.2.2 Socket	192
4.121.3 Member Function Documentation	192
4.121.3.1 GetReceiveTimeout	192
4.121.3.2 GetSendTimeout	192
4.121.3.3 Receive	192
4.121.3.4 ReceiveN	192
4.121.3.5 Send	193
4.121.3.6 Send	193
4.121.3.7 Send	193
4.121.3.8 SetKeepAlive	193
4.121.3.9 SetReceiveTimeout	193
4.121.3.10SetReuseAddress	194
4.121.3.11SetSendTimeout	194
4.121.3.12SetTCPNoDelay	194
4.122Common::SocketBase Class Reference	194

4.122.1 Detailed Description	196
4.122.2 Constructor & Destructor Documentation	197
4.122.2.1 SocketBase	197
4.122.2.2 SocketBase	197
4.122.3 Member Function Documentation	197
4.122.3.1 Bind	197
4.122.3.2 DoSend	198
4.122.3.3 GetOption	198
4.122.3.4 GetReceiveTimeout	198
4.122.3.5 GetSendTimeout	198
4.122.3.6 Listen	198
4.122.3.7 Receive	199
4.122.3.8 ReceiveN	199
4.122.3.9 Send	199
4.122.3.10Send	199
4.122.3.11Send	200
4.122.3.12SetKeepAlive	200
4.122.3.13SetOption	200
4.122.3.14SetReceiveTimeout	200
4.122.3.15SetReuseAddress	200
4.122.3.16SetSendTimeout	201
4.122.3.17SetTCPNoDelay	201
4.123Common::StartAppRequest Class Reference	201
4.123.1 Detailed Description	202
4.123.2 Constructor & Destructor Documentation	202
4.123.2.1 StartAppRequest	202
4.124Stats Struct Reference	203
4.125Common::StreamLogger Class Reference	203
4.125.1 Detailed Description	204
4.126Common::StringArray Class Reference	204
4.126.1 Detailed Description	205
4.126.2 Constructor & Destructor Documentation	205
4.126.2.1 StringArray	205
4.127Common::SysException Class Reference	206
4.127.1 Detailed Description	206
4.127.2 Constructor & Destructor Documentation	207
4.127.2.1 SysException	207
4.127.2.2 SysException	207
4.127.3 Member Function Documentation	207
4.127.3.1 Display	207
4.128Common::Syslog Class Reference	207
4.128.1 Detailed Description	209
4.128.2 Member Function Documentation	209
4.128.2.1 Configure	209
4.128.2.2 Debug	210
4.128.2.3 Debug	210
4.128.2.4 Error	210
4.128.2.5 Error	210
4.128.2.6 Fatal	210
4.128.2.7 Fatal	210

4.128.2.8 Info	210
4.128.2.9 Info	210
4.128.2.10Warn	211
4.128.2.11Warn	211
4.129Task Class Reference	211
4.129.1 Detailed Description	212
4.129.2 Constructor & Destructor Documentation	213
4.129.2.1 Task	213
4.129.3 Member Function Documentation	213
4.129.3.1 AddDelayedTuning	213
4.129.3.2 GetImage	213
4.129.3.3 GetInstr	213
4.129.3.4 GetPid	214
4.129.3.5 GetProcess	214
4.129.3.6 IsStopped	214
4.129.3.7 IsStoppedOnBreakpoint	214
4.129.3.8 IsTerminated	214
4.129.3.9 ProcessBreakpoint	214
4.129.3.10Terminate	215
4.130Model::Task Class Reference	215
4.130.1 Detailed Description	217
4.130.2 Constructor & Destructor Documentation	217
4.130.2.1 Task	217
4.130.3 Member Function Documentation	218
4.130.3.1 AddEvent	218
4.130.3.2 FuncParamChange	218
4.130.3.3 GetACProxy	218
4.130.3.4 GetHost	218
4.130.3.5 GetMpiRank	219
4.130.3.6 GetName	219
4.130.3.7 GetPid	219
4.130.3.8 GetStatus	219
4.130.3.9 InsertFunctionCall	219
4.130.3.10IsMaster	220
4.130.3.11IsRunning	220
4.130.3.12LoadLibrary	220
4.130.3.13OneTimeFuncCall	220
4.130.3.14RemoveEvent	221
4.130.3.15RemoveFuncCall	221
4.130.3.16ReplaceFunction	221
4.130.3.17SetMaster	222
4.130.3.18SetVariableValue	222
4.131TaskCollection Class Reference	222
4.131.1 Detailed Description	223
4.131.2 Member Function Documentation	223
4.131.2.1 Add	223
4.131.2.2 Delete	223
4.131.2.3 operator[]	223
4.131.2.4 operator[]	224
4.132TaskExitHandler Class Reference	224



4.132.1 Detailed Description	224
4.133Model::TaskHandler Class Reference	225
4.133.1 Detailed Description	225
4.133.2 Member Function Documentation	225
4.133.2.1 TaskStarted	225
4.133.2.2 TaskTerminated	226
4.134TaskInstr Class Reference	226
4.134.1 Detailed Description	227
4.134.2 Member Function Documentation	227
4.134.2.1 Add	227
4.134.2.2 FindGroup	227
4.134.2.3 GetBreakpoint	228
4.134.2.4 GetSize	228
4.134.2.5 Remove	228
4.135TaskManager Class Reference	228
4.135.1 Detailed Description	228
4.136Model::Tasks Class Reference	229
4.136.1 Detailed Description	230
4.136.2 Member Function Documentation	230
4.136.2.1 Add	230
4.136.2.2 Delete	230
4.136.2.3 FindById	230
4.136.2.4 GetById	231
4.136.2.5 operator[]	231
4.136.2.6 operator[]	231
4.136.2.7 Remove	231
4.136.2.8 Size	232
4.137TaskStats Class Reference	232
4.138Common::Thread Class Reference	232
4.138.1 Detailed Description	233
4.138.2 Constructor & Destructor Documentation	233
4.138.2.1 Thread	233
4.138.3 Member Function Documentation	233
4.138.3.1 WaitForDeath	233
4.139Common::TimeValue Class Reference	234
4.139.1 Detailed Description	235
4.139.2 Constructor & Destructor Documentation	235
4.139.2.1 TimeValue	235
4.139.2.2 TimeValue	236
4.139.2.3 TimeValue	236
4.139.2.4 TimeValue	236
4.139.2.5 TimeValue	236
4.140Tuner Class Reference	236
4.141Common::TuningRequest Class Reference	237
4.141.1 Detailed Description	238
4.142Tunlet Class Reference	238
4.142.1 Member Function Documentation	238
4.142.1.1 Initialize	238
4.143TunletContainer Class Reference	239
4.143.1 Detailed Description	239

4.144 Common::UnRegisterMsg Class Reference . . . . .	239
4.144.1 Detailed Description . . . . .	240
4.145 Ventana Struct Reference . . . . .	240
4.146 WorkerData Class Reference . . . . .	240
4.146.1 Detailed Description . . . . .	241

## Chapter 1

# Deprecated List

Member `Model::Application::Start()`



## Chapter 2

# Class Index

### 2.1 Class Hierarchy

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

ACProxy . . . . .	13
Common::ActiveObject . . . . .	18
ShutDownManager . . . . .	183
ShutDownSlave . . . . .	185
Common::Address . . . . .	21
Analyzer . . . . .	26
auto_iterator< T > . . . . .	41
auto_vector< T > . . . . .	41
BatchData . . . . .	46
CommandLine . . . . .	50
comptime . . . . .	56
Common::Config . . . . .	56
Common::ConfigHelper . . . . .	62
Common::ConfigMap . . . . .	63
Common::ConfigReader . . . . .	65
Common::FileConfigReader . . . . .	117
Controller . . . . .	66
curState . . . . .	69
Common::DateTime . . . . .	69
Common::DeSerializer . . . . .	71
Common::NetworkDeSerializer . . . . .	147
DiEx . . . . .	72
DiFunction . . . . .	72
DiImage . . . . .	73
DiPoint . . . . .	74
DiProcess . . . . .	74
DiSnippetHandle . . . . .	75
DiType . . . . .	76

DiIntType . . . . .	73
DiVariable . . . . .	76
DiIntVariable . . . . .	73
DTLibrary . . . . .	77
DTLibraryFactory . . . . .	78
DynInst . . . . .	79
ECPAcceptor . . . . .	79
ECPHandler . . . . .	80
Common::ECPMessage . . . . .	82
Common::EventMsg . . . . .	101
Common::RegisterMsg . . . . .	166
Common::UnRegisterMsg . . . . .	239
ECPProtocol . . . . .	85
Model::Event . . . . .	86
Common::Event . . . . .	89
EventCollector . . . . .	91
DMLib::EventCollectorProxy . . . . .	93
Common::EventDemultiplexer . . . . .	94
Common::EventHandler . . . . .	97
Model::EventHandler . . . . .	97
AdjustingNWTunlet . . . . .	23
FactoringTunlet . . . . .	116
EventListener . . . . .	98
Model::Application . . . . .	26
Common::EventMap . . . . .	99
EventMsgReader . . . . .	103
DMLib::EventMsgWriter . . . . .	105
Model::EventRecord . . . . .	107
Model::Events . . . . .	110
Common::Exception . . . . .	111
Common::ConfigException . . . . .	60
Common::EventException . . . . .	95
Common::FuncDefException . . . . .	121
Common::SysException . . . . .	206
Common::FuncDef . . . . .	120
Common::FuncDefs . . . . .	123
Common::HandlerMap . . . . .	126
Model::Host . . . . .	127
Model::HostHandler . . . . .	128
InstrGroup . . . . .	131
Common::ConfigMap::Iterator . . . . .	134
IterData . . . . .	135
Common::Config::KeyIterator . . . . .	136
Common::LogEntry . . . . .	138
Common::LogFilter . . . . .	139
Common::BasicLogFilter . . . . .	42
Common::LogFormatter . . . . .	141
Common::BasicLogFormatter . . . . .	43

Common::Logger . . . . .	141
Common::BasicLogger . . . . .	44
Common::FileLogger . . . . .	119
Common::StreamLogger . . . . .	203
ModelParam . . . . .	143
ModuleList . . . . .	143
Monitor . . . . .	143
Common::Mutex . . . . .	145
Common::MutexLock . . . . .	146
myauto_ptr< X > . . . . .	147
Common::OutputStream . . . . .	152
Common::ByteStream . . . . .	48
Common::Pipe . . . . .	153
PointList . . . . .	155
ProcedureList . . . . .	156
Common::Process . . . . .	156
Common::ExecProcess . . . . .	113
Common::RemoteProcess . . . . .	168
PTPAcceptor . . . . .	157
PTPHandler . . . . .	159
Common::PTPProtocol . . . . .	163
Common::Queue< T > . . . . .	163
Common::Reactor . . . . .	165
Common::Semaphore . . . . .	174
Common::Serializable . . . . .	176
Common::Attribute . . . . .	36
Common::AttributeValue . . . . .	37
Common::Breakpoint . . . . .	47
Common::ECPMsgHeader . . . . .	83
Common::PTPMessage . . . . .	160
Common::AddInstrRequest . . . . .	19
Common::RemoveInstrRequest . . . . .	171
Common::StartAppRequest . . . . .	201
Common::TuningRequest . . . . .	237
Common::FunctionParamChangeRequest . . . . .	124
Common::InsertFunctionCallRequest . . . . .	129
Common::LoadLibraryRequest . . . . .	137
Common::OneTimeFunctionCallRequest . . . . .	151
Common::RemoveFunctionCallRequest . . . . .	169
Common::ReplaceFunctionRequest . . . . .	173
Common::SetVariableValueRequest . . . . .	181
Common::PTPMsgHeader . . . . .	161
Common::Serializer . . . . .	177
Common::CountingSerializer . . . . .	68
Common::NetworkSerializer . . . . .	149
Common::ServerSocket . . . . .	178
Service . . . . .	180
SnippetHandler . . . . .	186

SnippetMaker . . . . .	188
Common::Socket . . . . .	189
Common::SocketBase . . . . .	194
Stats . . . . .	203
Common::StringArray . . . . .	204
Common::Syslog . . . . .	207
Task . . . . .	211
Model::Task . . . . .	215
TaskCollection . . . . .	222
TaskExitHandler . . . . .	224
Model::TaskHandler . . . . .	225
AdjustingNWTunlet . . . . .	23
FactoringTunlet . . . . .	116
TaskInstr . . . . .	226
TaskManager . . . . .	228
Model::Tasks . . . . .	229
TaskStats . . . . .	232
Common::Thread . . . . .	232
Common::TimeValue . . . . .	234
Tuner . . . . .	236
Tunlet . . . . .	238
AdjustingNWTunlet . . . . .	23
FactoringTunlet . . . . .	116
TunletContainer . . . . .	239
Ventana . . . . .	240
WorkerData . . . . .	240



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">ACProxy</a> (Creates a connection with AC and acts as an interface with it. This class have a socket object to represent the connection and a PTPProtocol object for making request in a common language ) . . .	13
<a href="#">Common::ActiveObject</a> (Abstract class, encapsulates OS thread (pthreads) POSIX compatible ) . . . . .	18
<a href="#">Common::AddInstrRequest</a> (Represents message sent when analyzer requests to add instrumentation ) . . . . .	19
<a href="#">Common::Address</a> (Encapsulates a socket address of the AF_INET family ) .	21
<a href="#">AdjustingNWTunlet</a> . . . . .	23
<a href="#">Analyzer</a> . . . . .	26
<a href="#">Model::Application</a> (Tuned application in the analyzer. Holds identificative information of the application, the tasks that form it (and which one is the master), the host where they are running, the places from where we are getting events and hanlers to both, tasks and hosts. Provides methods to: ) . . . . .	26
<a href="#">Common::Attribute</a> (Contains the necessary information of an attribute to be inserted in a program ) . . . . .	36
<a href="#">Common::AttributeValue</a> (Contains the vale of an attribute ) . . . . .	37
<a href="#">auto_iterator&lt; T &gt;</a> . . . . .	41
<a href="#">auto_vector&lt; T &gt;</a> . . . . .	41
<a href="#">Common::BasicLogFilter</a> (Filters <a href="#">LogEntry</a> objects to be inserted in a log ) .	42
<a href="#">Common::BasicLogFormatter</a> (Formats <a href="#">LogEntry</a> objects to be inserted in a log ) . . . . .	43
<a href="#">Common::BasicLogger</a> (Stores information of events in a system ) . . . . .	44
<a href="#">BatchData</a> (Statistics of a single batch ) . . . . .	46
<a href="#">Common::Breakpoint</a> (Denotes place in a function (on the entry or at the end) )	47
<a href="#">Common::ByteStream</a> (Stores stream of bytes ) . . . . .	48

<a href="#">CommandLine</a> (Encapsulates methods to interact with the user of analyzer. Basically reads the arguments of the analyzer and parses them to get the configuration file and the objective application with its parameters. Once read, encapsulates the information and provides accessors to it. There are two formats <a href="#">Analyzer</a> can be called: ) . . .	50
<a href="#">comptime</a> . . . . .	56
<a href="#">Common::Config</a> (Manages a configuration of the system ) . . . . .	56
<a href="#">Common::ConfigException</a> ( <a href="#">Config</a> , <a href="#">ConfigReader</a> and <a href="#">ConfigMap</a> exceptions ) . . . . .	60
<a href="#">Common::ConfigHelper</a> (Static class that contains methods to manage <a href="#">Config</a> objects ) . . . . .	62
<a href="#">Common::ConfigMap</a> (Contains and manages a collection of <a href="#">Config</a> objects )	63
<a href="#">Common::ConfigReader</a> (Abstract class, generates <a href="#">Config</a> objects from reading sources ) . . . . .	65
<a href="#">Controller</a> (Provides the logic and controls the execution flow of the application ) . . . . .	66
<a href="#">Common::CountingSerializer</a> (Stores the size of serialized data ) . . . . .	68
<a href="#">curState</a> . . . . .	69
<a href="#">Common::DateTime</a> (Holds a timestamp ) . . . . .	69
<a href="#">Common::DeSerializer</a> (Abstract class, recovers serialized data from a stream ) . . . . .	71
<a href="#">DiEx</a> . . . . .	72
<a href="#">DiFunction</a> . . . . .	72
<a href="#">DiImage</a> . . . . .	73
<a href="#">DiIntType</a> . . . . .	73
<a href="#">DiIntVariable</a> . . . . .	73
<a href="#">DiPoint</a> . . . . .	74
<a href="#">DiProcess</a> . . . . .	74
<a href="#">DiSnippetHandle</a> . . . . .	75
<a href="#">DiType</a> . . . . .	76
<a href="#">DiVariable</a> . . . . .	76
<a href="#">DTLibrary</a> (Dynamic Tuning Library that offers DT API. Encapsulates information about the application model and the event collector. Provides methods to create application models ) . . . . .	77
<a href="#">DTLibraryFactory</a> (Handles the creation and destruction of DT Libraries ) . .	78
<a href="#">DynInst</a> . . . . .	79
<a href="#">ECPAcceptor</a> . . . . .	79
<a href="#">ECPHandler</a> (Encapsulates data structures and methods to handle incoming event collector inputs ) . . . . .	80
<a href="#">Common::ECPMessage</a> (Abstract class, EventCollectorProtocol, represents message interchanged between DMLib and analyzer ) . . . . .	82
<a href="#">Common::ECPMsgHeader</a> (Represents header of an <a href="#">ECPMessage</a> object ) . .	83
<a href="#">ECPProtocol</a> (Encapsulates methods to read and handle incoming network messages ) . . . . .	85
<a href="#">Model::Event</a> (Encapsulates information about the events that the target application generates. For each event holds identificative information (id, name), the place where it is produced, its attributes (for example the parameters of a function) and a reference to a handler. As this is a model class the methods provided are for accessing and setting the members of the data structure ) . . . . .	86

<a href="#">Common::Event</a> (Encapsulates information to record an event ) . . . . .	89
<a href="#">EventCollector</a> (Processes the incoming event records from the DMLibs. It is based on an active object (thread) that collects incoming ECP events It stores a moving window of events incoming from different processes using a pool of buffers. The maximum size of this event window can be configured by the tunlets ) . . . . .	91
<a href="#">DMLib::EventCollectorProxy</a> (Connects to the analyzer host and sends requests ) . . . . .	93
<a href="#">Common::EventDemultiplexer</a> (Part of the reactor design pattern, takes requests coming from the reactor and passes them to different handlers ) . . . . .	94
<a href="#">Common::EventException</a> ( <a href="#">Event</a> , <a href="#">EventMap</a> and <a href="#">EventHandler</a> exceptions )	95
<a href="#">Common::EventHandler</a> (Abstract class, processes the requests sent to the reactor ) . . . . .	97
<a href="#">Model::EventHandler</a> (Abstract class that holds a method to manage event records ) . . . . .	97
<a href="#">EventListener</a> (Provides an interface for event listeners, which consist in methods to respond to events and errors ) . . . . .	98
<a href="#">Common::EventMap</a> (Contains and manages a collection of <a href="#">Event</a> objects, TODO ) . . . . .	99
<a href="#">Common::EventMsg</a> (Encapsulates a message generated by DMLib to trace events ) . . . . .	101
<a href="#">EventMsgReader</a> . . . . .	103
<a href="#">DMLib::EventMsgWriter</a> (Creates EventMsg objects ) . . . . .	105
<a href="#">Model::EventRecord</a> (Particular instance of the event abstraction. Holds information about the kind of event, the task that produced, the message sent and the values it contained. On the one hand it provides methods to get/set the information above, on the other hand, it provides methods to parse messages and get the information that they contain ) . . . . .	107
<a href="#">Model::Events</a> (Encapsulates information to create and manage events lists. Uses a data structure based on a vector to keep data and a map to retrieve it. Provides methods to add, remove and find elements in the list ) . . . . .	110
<a href="#">Common::Exception</a> (Abstract class, stores information of errors on determined situations ) . . . . .	111
<a href="#">Common::ExecProcess</a> (Executes a program as a child of the current process )	113
<a href="#">FactoringTunlet</a> (Factoring optimization tunlet for m/w apps ) . . . . .	116
<a href="#">Common::FileConfigReader</a> (Parses the content of a file into a <a href="#">Config</a> object )	117
<a href="#">Common::FileLogger</a> (Stores information of interest into a file ) . . . . .	119
<a href="#">Common::FuncDef</a> (Represents definition of the function to be traced ) . . . .	120
<a href="#">Common::FuncDefException</a> ( <a href="#">FuncDef</a> exceptions ) . . . . .	121
<a href="#">Common::FuncDefs</a> (Creates and stores objects of the <a href="#">FuncDef</a> class ) . . . .	123
<a href="#">Common::FunctionParamChangeRequest</a> (Encapsulates a tuning request to set the value of an input parameter of a given function in a given application process ) . . . . .	124
<a href="#">Common::HandlerMap</a> (Contains and manages a collection of <a href="#">EventHandler</a> objects ) . . . . .	126
<a href="#">Model::Host</a> (Encapsulates host information. Basically consists in a string with the name of the host and a method to access it ) . . . . .	127

<a href="#">Model::HostHandler</a> (Provides mechanisms to handle the addition and the remove of hosts ) . . . . .	128
<a href="#">Common::InsertFunctionCallRequest</a> (Encapsulates a tuning request to insert a new function invocation code with a specified attributes at a given location in an application process ) . . . . .	129
<a href="#">InstrGroup</a> (Contains a group of snippets to be inserted in a function ) . . . .	131
<a href="#">Common::ConfigMap::Iterator</a> (Iterates over a <a href="#">ConfigMap</a> object ) . . . . .	134
<a href="#">IterData</a> (Statistics for a single iteration ) . . . . .	135
<a href="#">Common::Config::KeyIterator</a> (Iterates over the keys of a <a href="#">Config</a> object ) . . .	136
<a href="#">Common::LoadLibraryRequest</a> (Encapsulates a tuning request to load a specified shared library to a given application process ) . . . . .	137
<a href="#">Common::LogEntry</a> (Entry on a log ) . . . . .	138
<a href="#">Common::LogFilter</a> (Abstract class, validates logs ) . . . . .	139
<a href="#">Common::LogFormatter</a> (Abstract class, Gives logs the correct format ) . . .	141
<a href="#">Common::Logger</a> (Abstract class, tracks and stores information about events of interest happening in a system ) . . . . .	141
<a href="#">ModelParam</a> . . . . .	143
<a href="#">ModuleList</a> . . . . .	143
<a href="#">Monitor</a> (Adds request to add or remove instrumentation in/from the tasks it is monitoring ) . . . . .	143
<a href="#">Common::Mutex</a> (Guarantees non concurrent access to a resource ) . . . . .	145
<a href="#">Common::MutexLock</a> (System to manage access to a resource with a mutex )	146
<a href="#">myauto_ptr&lt; X &gt;</a> . . . . .	147
<a href="#">Common::NetworkDeSerializer</a> (Extracts serialized data from an istream object ) . . . . .	147
<a href="#">Common::NetworkSerializer</a> (Puts serialized data into an <a href="#">OutputStream</a> object ) . . . . .	149
<a href="#">Common::OneTimeFunctionCallRequest</a> (Encapsulates a tuning request to invoke one time a given function in a given application process ) . .	151
<a href="#">Common::OutputStream</a> (Abstract class, represents an output stream of bytes )	152
<a href="#">Common::Pipe</a> (Element used to join output and input from two processes ) .	153
<a href="#">PointList</a> . . . . .	155
<a href="#">ProcedureList</a> . . . . .	156
<a href="#">Common::Process</a> (Abstract class, creates a new process to perform different operations on the overridden method Run() ) . . . . .	156
<a href="#">PTPAcceptor</a> (Manages socket connection and handles data input through them ) . . . . .	157
<a href="#">PTPHandler</a> (Manages the requests from the <a href="#">PTPAcceptor</a> ) . . . . .	159
<a href="#">Common::PTPMessage</a> (Performance tuning protocol, represents message interchanged between analyzer and tuner/tracer ) . . . . .	160
<a href="#">Common::PTPMsgHeader</a> (Represents header of a <a href="#">PTPMessage</a> object ) . . .	161
<a href="#">Common::PTPProtocol</a> (Communicates analyzer and tuner ) . . . . .	163
<a href="#">Common::Queue&lt; T &gt;</a> (Data structure that stores objects of any class ) . . .	163
<a href="#">Common::Reactor</a> (Registers, removes and dispatches <a href="#">EventHandler</a> objects )	165
<a href="#">Common::RegisterMsg</a> (Represents message that is sent when DMLib is registered with analyzer to send event messages ) . . . . .	166
<a href="#">Common::RemoteProcess</a> (Remotely executes a command in another machine ) . . . . .	168
<a href="#">Common::RemoveFunctionCallRequest</a> (Encapsulates a tuning request to remove all calls to a given function from the given caller function ) . .	169

<a href="#">Common::RemoveInstrRequest</a> (Represents message sent when analyzer requests to remove instrumentation ) . . . . .	171
<a href="#">Common::ReplaceFunctionRequest</a> (Encapsulates a tuning request to replace all calls to a function inside a process with calls to another function )	173
<a href="#">Common::Semaphore</a> (Synchronizes access to a resource ) . . . . .	174
<a href="#">Common::Serializable</a> (Abstract class, makes an object able to be passed through a stream using <a href="#">Serializer</a> and <a href="#">DeSerializer</a> objects ) . . . . .	176
<a href="#">Common::Serializer</a> (Abstract class, prepares objects to be passed on a stream ) . . . . .	177
<a href="#">Common::ServerSocket</a> (Holds a <a href="#">SocketBase</a> object and represents a TCP/IP server socket ) . . . . .	178
<a href="#">Service</a> (Methods to work with EventCollectorHandlers lists. Holds a list of EventCollHandler and a reference to the reactor. Provides methods to add and remove handlers form the list ) . . . . .	180
<a href="#">Common::SetVariableValueRequest</a> (Encapsulates a tuning request to modify a value of a specified variable in a given application process ) . .	181
<a href="#">ShutDownManager</a> (Handles the shut down of MATE ( <a href="#">Analyzer</a> and AC's) The data structure consists basically in a reference to the application model (to know the hosts where the AC's are running in real time) and a boolean to determine if MATE is finished (to let the main process know, and make it stop) Provides a method to set the application model from outside (when it is ready, the main process of <a href="#">Analyzer</a> will set it). On the other hand, this class inherits from ActiveObject, so its objects are execution threads, this is done to wait for the user to stop MATE without stopping its own execution )	183
<a href="#">ShutDownSlave</a> (Receives terminating message from <a href="#">Analyzer</a> ) . . . . .	185
<a href="#">SnippetHandler</a> (Contains he necessary fields to manage snippets ) . . . . .	186
<a href="#">SnippetMaker</a> (Prepares the snippets to be inserted into the processes ) . . . .	188
<a href="#">Common::Socket</a> (Holds a <a href="#">SocketBase</a> object and represents a client socket )	189
<a href="#">Common::SocketBase</a> (Represents an endpoint for communication between two machines ) . . . . .	194
<a href="#">Common::StartAppRequest</a> (Represents a request to start the application ) . .	201
<a href="#">Stats</a> . . . . .	203
<a href="#">Common::StreamLogger</a> (Stores the logged information into a stream ) . . . .	203
<a href="#">Common::StringArray</a> (Container of strings ) . . . . .	204
<a href="#">Common::SysException</a> (System exception ) . . . . .	206
<a href="#">Common::Syslog</a> (Holds and manages a loggers on the system ) . . . . .	207
<a href="#">Task</a> (Represents each of the processes that we can modify using dyninst ) . .	211
<a href="#">Model::Task</a> (Encapsulates information to define the tasks that form the application. The data structure of a task consist of identification data (pid, mpiRank, name),status data, where is it running (host), which events are being collected from it and if it is either a master task or not. Provides methods to: ) . . . . .	215
<a href="#">TaskCollection</a> (Groups task in a single, easy to handle, collection ) . . . . .	222
<a href="#">TaskExitHandler</a> (Contains a virtual function to handle the exit of a task ) . .	224
<a href="#">Model::TaskHandler</a> (Abstract class that provides methods to determine if a task is started or terminated ) . . . . .	225
<a href="#">TaskInstr</a> (Adds and remove instrumentation from the process in execution ) .	226
<a href="#">TaskManager</a> (Single class that starts and handles all the tasks ) . . . . .	228

<a href="#">Model::Tasks</a> ( <a href="#">Tasks</a> encapsulates methods to work with lists of <a href="#">Task</a> objects. The data structure to hold the information is an <a href="#">auto_vector</a> . This class provides methods to add, remove, access <a href="#">Task</a> objects in an array. Also it provides methods to find <a href="#">Tasks</a> and for measure the array ) . . . . .	229
<a href="#">TaskStats</a> . . . . .	232
<a href="#">Common::Thread</a> (Posix thread ) . . . . .	232
<a href="#">Common::TimeValue</a> (Stores a time value up to microseconds ) . . . . .	234
<a href="#">Tuner</a> . . . . .	236
<a href="#">Common::TuningRequest</a> (Encapsulates a tuning request from the analyzer ) .	237
<a href="#">Tunlet</a> . . . . .	238
<a href="#">TunletContainer</a> (TO BE IMPLEMENTED ) . . . . .	239
<a href="#">Common::UnRegisterMsg</a> (Represents message that is sent when DMLib is unregistered with analyzer ) . . . . .	239
<a href="#">Ventana</a> . . . . .	240
<a href="#">WorkerData</a> (Worker task statistics for a single batch ) . . . . .	240

## Chapter 4

# Class Documentation

### 4.1 ACProxy Class Reference

Creates a connection with AC and acts as an interface with it. This class have a socket object to represent the connection and a PTPProtocol object for making request in a common language.

```
#include <ACProxy.h>
```

#### Public Member Functions

- [ACProxy](#) (std::string const &host, int port=8888)  
*constructor, creates a connection with the given host:port.*
- void [StartApplication](#) (char const \*appPath, int argc, char const \*\*argv, char const \*analyzerHost)  
*Starts the execution of the application in the AC host.*
- void [AddInstr](#) (int tid, int eventId, std::string const &fName, InstrPlace place, int nAttrs, Attribute \*attrs)  
*Requests for adding an instruction in the target application.*
- void [RemoveInstr](#) (int tid, int eventId, InstrPlace place)  
*Requests for removing an instruction from the target application.*
- void [LoadLibrary](#) (int tid, std::string const &libPath)  
*Requests for loading a library in the target application.*
- void [SetVariableValue](#) (int tid, std::string const &varName, AttributeValue const &varValue, Breakpoint \*brkpt)  
*Requests for changing the value of a variable in the target application.*

- void [ReplaceFunction](#) (int tid, std::string const &oldFunc, std::string const &newFunc, Breakpoint \*brkpt)  
*Requests for changin all the instances of a function from the target application.*
- void [InsertFunctionCall](#) (int tid, std::string const &funcName, int nAttrs, Attribute \*attrs, std::string const &destFunc, InstrPlace destPlace, Breakpoint \*brkpt)  
*Requests for the insertion of a function call in a point of the target application.*
- void [OneTimeFuncCall](#) (int tid, std::string const &funcName, int nAttrs, Attribute \*attrs, Breakpoint \*brkpt)  
*Requests for the call of a function in this point of the target application executon.*
- void [RemoveFuncCall](#) (int tid, std::string const &funcName, std::string const &callerFunc, Breakpoint \*brkpt)  
*Requests for removing all the calls to a function in the target application.*
- void [FuncParamChange](#) (int tid, std::string const &funcName, int paramIdx, int newValue, int \*requiredOldValue, Breakpoint \*brkpt)  
*Requests for the changing of the value of a certain parameter in one function of the target application.*

#### 4.1.1 Detailed Description

Creates a connection with AC and acts as an interface with it. This class have a socket object to represent the connection and a PTPProtocol object for making request in a common language. Its methods encapsulate the requests in the adequated kind of request object and use the protocol to serialize and write them in the socket.

##### Version

1.0b

##### Author

Ania Morajko, 2002

##### Since

1.0b

#### 4.1.2 Constructor & Destructor Documentation

4.1.2.1 [ACProxy::ACProxy](#) ( std::string const & *host*, int *port* = 8888 ) [inline]

constructor, creates a connection with the given host:port.

##### Parameters



<i>host</i>	host where the target AC is running
<i>port</i>	port for the AC process in the host

### 4.1.3 Member Function Documentation

#### 4.1.3.1 void ACProxy::AddInstr ( int *tid*, int *eventId*, std::string const & *fName*, InstrPlace *place*, int *nAttrs*, Attribute \* *attrs* )

Requests for adding an instruction in the target application.

##### Parameters

<i>tid</i>	identifier of the thread in which we will add the instruction
<i>eventId</i>	identifier of the event
<i>fName</i>	name of the function in which we will add the instruction
<i>place</i>	place in the function where we will add the instruction values are: instrUnknown, ipFuncEntry & ipFuncExit
<i>nAttrs</i>	number of Attributes
<i>attrs</i>	Attributes array

#### 4.1.3.2 void ACProxy::FuncParamChange ( int *tid*, std::string const & *funcName*, int *paramIdx*, int *newValue*, int \* *requiredOldValue*, Breakpoint \* *brkpt* )

Requests for the changing of the value of a certain parameter in one function of the target application.

##### Parameters

<i>tid</i>	identifier of the thread in which the function is placed
<i>funcName</i>	name of the function
<i>paramIdx</i>	position of the parameter in the parameter list
<i>newValue</i>	new value for the argument
<i>requiredOldValue</i>	old value required to change for the new one
<i>brkpt</i>	---

#### 4.1.3.3 void ACProxy::InsertFunctionCall ( int *tid*, std::string const & *funcName*, int *nAttrs*, Attribute \* *attrs*, std::string const & *destFunc*, InstrPlace *destPlace*, Breakpoint \* *brkpt* )

Requests for the insertion of a function call in a point of the target application.

##### Parameters

<i>tid</i>	identifier of the thread in which the call will be placed
------------	---

<i>funcName</i>	name of the function
<i>nAttrs</i>	number of attributes
<i>attrs</i>	attributes vector
<i>destFunc</i>	name of the destination function
<i>destPlace</i>	point where the call will be placed
<i>brkpt</i>	---

#### 4.1.3.4 void ACProxy::LoadLibrary ( int *tid*, std::string const & *libPath* )

Requests for loading a library in the target application.

##### Parameters

<i>tid</i>	identifier of the thread in which we will load the library library path
------------	---

#### 4.1.3.5 void ACProxy::OneTimeFuncCall ( int *tid*, std::string const & *funcName*, int *nAttrs*, Attribute \* *attrs*, Breakpoint \* *brkpt* )

Requests for the call of a function in this point of the target application execution.

##### Parameters

<i>tid</i>	identifier of the thread in which the call will be placed
<i>funcName</i>	name of the function
<i>nAttrs</i>	number of attributes
<i>attrs</i>	attributes vector
<i>brkpt</i>	---

#### 4.1.3.6 void ACProxy::RemoveFuncCall ( int *tid*, std::string const & *funcName*, std::string const & *callerFunc*, Breakpoint \* *brkpt* )

Requests for removing all the calls to a function in the target application.

##### Parameters

<i>tid</i>	identifier of the thread in which the call will be removed
<i>funcName</i>	name of the function to be removed
<i>callerFunc</i>	function which makes the call
<i>brkpt</i>	---

#### 4.1.3.7 void ACProxy::RemoveInstr ( int *tid*, int *eventId*, InstrPlace *place* )

Requests for removing an instruction from the target application.

**Parameters**

<i>tid</i>	identifier of the thread in which we will remove the instruction
<i>eventId</i>	identifier of the associated event
<i>place</i>	place in the function where the instruction will be removed

**4.1.3.8 void ACProxy::ReplaceFunction ( int *tid*, std::string const & *oldFunc*, std::string const & *newFunc*, Breakpoint \* *brkpt* )**

Requests for changin all the instances of a function from the target application.

**Parameters**

<i>tid</i>	identifier of the thread in which the function is placed
<i>oldFunc</i>	name of the function to be changed
<i>newFunc</i>	name of the new function
<i>brkpt</i>	---

**4.1.3.9 void ACProxy::SetVariableValue ( int *tid*, std::string const & *varName*, AttributeValue const & *varValue*, Breakpoint \* *brkpt* )**

Requests for changing the value of a variable in the target application.

**Parameters**

<i>tid</i>	identifier of the thread in which the variable is placed
<i>varName</i>	name of the variable to change
<i>varValue</i>	new value for the variable
<i>brkpt</i>	---

**4.1.3.10 void ACProxy::StartApplication ( char const \* *appPath*, int *argc*, char const \*\* *argv*, char const \* *analyzerHost* )**

Starts the execution of the application in the AC host.

**Parameters**

<i>appPath</i>	path to the application executable
<i>argc</i>	number of arguments to the application main
<i>argv</i>	argument vector to the application main
<i>analyzer-Host</i>	node where the analyzer is executed

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

- Analyzer/ACProxy.h

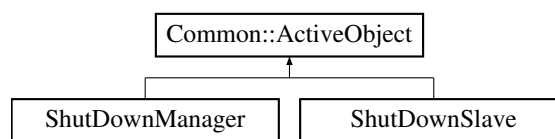
- Analyzer/ACProxy.cpp

## 4.2 Common::ActiveObject Class Reference

Abstract class, encapsulates OS thread (pthreads) POSIX compatible.

```
#include <ActiveObject.h>
```

Inheritance diagram for Common::ActiveObject:



### Public Member Functions

- [ActiveObject](#) ()  
*Constructor.*
- virtual [~ActiveObject](#) ()  
*Destructor.*
- void [Kill](#) ()  
*Stops the thread execution.*

### Protected Member Functions

- virtual void **InitThread** ()=0
- virtual void **Run** ()=0
- virtual void **FlushThread** ()=0
- void [Resume](#) ()  
*Continues with the execution of the thread.*

### Protected Attributes

- int **\_isDying**

### 4.2.1 Detailed Description

Abstract class, encapsulates OS thread (pthread) POSIX compatible. Last thing in the constructor of a class derived from [ActiveObject](#) must be a call to `_thread.Resume()`;

Inside the loop the Run method must keep checking `_isDying`:

```
if (_isDying)
    return;
```

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

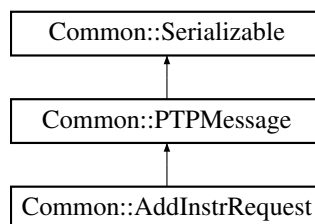
- Common/ActiveObject.h
- Common/ActiveObject.cpp

## 4.3 Common::AddInstrRequest Class Reference

Represents message sent when analyzer requests to add instrumentation.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::AddInstrRequest:



### Public Member Functions

- [AddInstrRequest](#) (int pid=0, int eventId=0, std::string const &funcName=std::string(), InstrPlace place=ipFuncEntry, int nAttrs=0, [Attribute](#) \*attrs=0)

*Constructor.*

- [~AddInstrRequest \(\)](#)  
*Destructor.*
- PTPMsgType [GetType \(\)](#) const  
*Returns type of message (PTPAddInstr).*
- int [GetPid \(\)](#) const  
*Returns the process id.*
- InstrPlace [GetInstrPlace \(\)](#) const  
*Returns the place where the instruction should be added.*
- std::string const & [GetFunctionName \(\)](#) const  
*Returns function name.*
- int [GetEventId \(\)](#) const  
*Returns the event id.*
- Attribute \* [GetAttributes \(\)](#) const  
*Returns array of attributes.*
- int [GetAttrsCount \(\)](#) const  
*Returns number of attributes the function has.*
- void [Serialize \(Serializer &out\)](#) const  
*Sends the message.*
- void [DeSerialize \(DeSerializer &in\)](#)  
*Receives the message.*

#### 4.3.1 Detailed Description

Represents message sent when analyzer requests to add instrumentation.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003

### 4.3.2 Constructor & Destructor Documentation

**4.3.2.1** `Common::AddInstrRequest::AddInstrRequest ( int pid = 0, int eventId = 0, std::string const & funcName = std::string(), InstrPlace place = ipFuncEntry, int nAttrs = 0, Attribute * attrs = 0 ) [inline]`

Constructor.

#### Parameters

<i>pid</i>	Id of the process where the instrumentation will be added, default 0.
<i>eventId</i>	<a href="#">Event</a> id, default 0.
<i>funcName</i>	Name of the function to modify, default "".
<i>place</i>	Place where the instrumentation will be added, default ipFuncEntry.
<i>nAttrs</i>	Number of attributes the function has, default 0.
<i>attrs</i>	<a href="#">Attribute</a> array, default 0.

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.4 Common::Address Class Reference

Encapsulates a socket address of the AF\_INET family.

```
#include <Address.h>
```

### Public Member Functions

- [Address](#) (std::string const &host, int port)  
*Constructor.*
- [Address](#) (int port)  
*Constructor.*
- [Address](#) ()  
*Constructor.*
- [operator struct sockaddr \\* \(\)](#)  
*Returns a pointer to the sockaddr intern structure.*
- [operator struct sockaddr\\_in \\* \(\)](#)  
*Returns a pointer to the sockaddr\_in intern structure.*
- socklen\_t [GetSize](#) () const

*Returns size of current address.*

- `std::string GetHostName () const`

*Returns name of host.*

#### 4.4.1 Detailed Description

Encapsulates a socket address of the AF\_INET family. This class contains methods to initialize the address of a socket.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.4.2 Constructor & Destructor Documentation

##### 4.4.2.1 `Common::Address::Address ( std::string const & host, int port )`

Constructor.

##### Parameters

<i>host</i>	Host where the socket will be located.
<i>port</i>	Port used.

##### Exceptions

<a href="#">SysException</a>
------------------------------

##### 4.4.2.2 `Address::Address ( int port )`

Constructor.

Uses the INADDR\_ANY address.

##### Parameters

<i>port</i>	Port used.
-------------	------------



**4.4.2.3 Address::Address ( )**

Constructor.

Initializes an empty address, setting the memory of the object to 0.

**4.4.3 Member Function Documentation****4.4.3.1 string Address::GetHostName ( ) const**

Returns name of host.

**Exceptions**

<i><a href="#">SysException</a></i>
-------------------------------------

**4.4.3.2 socklen\_t Common::Address::GetSize ( ) const [inline]**

Returns size of current address.

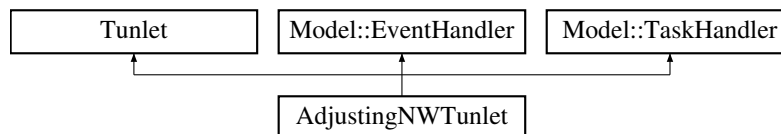
Number of bytes the address occupies on memory.

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

- Common/Address.h
- Common/Address.cpp

**4.5 AdjustingNWTunlet Class Reference**

Inheritance diagram for AdjustingNWTunlet:

**Public Member Functions**

- [AdjustingNWTunlet \( \)](#)  
*Constructor.*
- [~AdjustingNWTunlet \( \)](#)  
*Destructor.*

- void [Initialize](#) ([Model::Application](#) &app)
- void **BeforeAppStart** ()
- void **AppStarted** ()
- void **Destroy** ()
- void [HandleEvent](#) ([Model::EventRecord](#) const &r)
- void [TaskStarted](#) ([Model::Task](#) &t)
- void [TaskTerminated](#) ([Model::Task](#) &t)
- void [InsertWorkerEvents](#) ([Model::Task](#) &t)
- void [InsertMasterEvents](#) ([Model::Task](#) &t)
- void [InsertEvents](#) ([Task](#) &t)
- [IterData](#) \* [FindIterData](#) (int iterIdx)
- void [TryTuning](#) (int iterIdx)
- void **Calculate\_Vt** ()
- void **Calculate\_Ct** ()
- void **Calculate\_Vt\_Ct** ()
- void **Calculate\_Tt** ()
- int **Calculate\_Nopt** ()
- void **TuningParam** ()
- void **Update\_n** (int nw)
- void **Update\_tl** (double t)
- void **Update\_lambda** (double l)
- void **Update\_vi** (int v)
- void **Update\_vm** (int v)
- void **Update\_cti** (struct [comptime](#) \*c)
- void **Initialize\_cti** ()

#### 4.5.1 Member Function Documentation

##### 4.5.1.1 [IterData](#) \* [AdjustingNWTunlet::FindIterData](#) ( int *iterIdx* )

###### Parameters

<i>iterIdx</i>	
----------------	--

##### 4.5.1.2 void [AdjustingNWTunlet::HandleEvent](#) ( [Model::EventRecord](#) const & *r* ) [virtual]

###### Parameters

<i>r</i>	
----------	--

Implements [Model::EventHandler](#).

**4.5.1.3** void AdjustingNWTunlet::Initialize ( [Model::Application](#) & *app* ) [virtual]

**Parameters**

<i>app</i>	
------------	--

Implements [Tunlet](#).

**4.5.1.4** void AdjustingNWTunlet::InsertEvents ( [Task](#) & *t* )

**Parameters**

<i>t</i>	
----------	--

**4.5.1.5** void AdjustingNWTunlet::InsertMasterEvents ( [Model::Task](#) & *t* )

**Parameters**

<i>t</i>	
----------	--

**4.5.1.6** void AdjustingNWTunlet::InsertWorkerEvents ( [Model::Task](#) & *t* )

**Parameters**

<i>t</i>	
----------	--

**4.5.1.7** void AdjustingNWTunlet::TaskStarted ( [Model::Task](#) & *t* ) [virtual]

**Parameters**

<i>t</i>	
----------	--

Implements [Model::TaskHandler](#).

**4.5.1.8** void AdjustingNWTunlet::TaskTerminated ( [Model::Task](#) & *t* ) [virtual]

**Parameters**

<i>t</i>	
----------	--

Implements [Model::TaskHandler](#).

**4.5.1.9** void AdjustingNWTunlet::TryTuning ( int *iterIdx* )

**Parameters**

<i>iterIdx</i>	
----------------	--

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

- Analyzer/AdjustingNWTunlet.h
- Analyzer/AdjustingNWTunlet.cpp

## 4.6 Analyzer Class Reference

### Public Member Functions

- [Analyzer](#) (EventList &list)  
*Constructor.*
- void [AnalyzeEvent](#) ()  
*Analyzes an event and, if it finds a problem, makes tuning actions.*
- void [Instrument](#) ()  
*Requests to add instrumentation in the application to get information from it.*
- void [RemoveInstr](#) ()  
*Requests to remove instrumentation.*
- void [Tune](#) ()  
*Requests to modify the application in order to improve its behavior.*

### 4.6.1 Constructor & Destructor Documentation

#### 4.6.1.1 Analyzer::Analyzer ( EventList & list ) [inline]

Constructor.

#### Parameters

<i>list</i>	of events to be analyzed
-------------	--------------------------

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

- Analyzer/Analyzer.h
- Analyzer/Analyzer.cpp

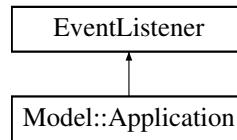
## 4.7 Model::Application Class Reference

represents tuned application in the analyzer. Holds identificative information of the application, the tasks that form it (and which one is the master), the host where they

are running, the places from where we are getting events and hanlers to both, tasks and hosts. Provides methods to:

```
#include <AppModel.h>
```

Inheritance diagram for Model::Application:



### Public Member Functions

- [Application](#) (char const \*appPath, int argc, char const \*\*argv)  
*Constructor.*
- string [GetName](#) () const  
*name getter.*
- int [NumActiveTasks](#) () const  
*number of tasks getter.*
- [Tasks](#) & [GetTasks](#) ()  
*tasks getter.*
- [Hosts](#) & [GetHosts](#) ()  
*hosts getter.*
- [Task](#) \* [GetMasterTask](#) ()  
*master task getter. return a reference to the master task of the application*
- Status [GetStatus](#) () const
- void [Start](#) ()  
*starts the application.*
- int [AddEvent](#) ([Event](#) const &e)  
*Adds a definition of new event to be traced in all running tasks of the application.*
- int [RemoveEvent](#) (int eventId, InstrPlace place)  
*Removes previously added event from all running tasks.*
- int [LoadLibrary](#) (string const &libPath)  
*Loads a shared library to all running tasks. This enables the [Analyzer](#) to load any additional code required for the tuning.*

- `int SetVariableValue` (string const &varName, AttributeValue const &varValue, Breakpoint \*brkpt)  
*Modifies a value of a specified variable in a given set of tasks.*
- `int ReplaceFunction` (string const &oldFunc, string const &newFunc, Breakpoint \*brkpt)  
*Replaces all calls to a function with calls to another one in a given set of tasks.*
- `int InsertFunctionCall` (string const &funcName, int nAttrs, Attribute \*attrs, string const &destFunc, InstrPlace destPlace, Breakpoint \*brkpt)  
*Inserts a new function invocation code at a given location in a given set of tasks.*
- `int OneTimeFuncCall` (string const &funcName, int nAttrs, Attribute \*attrs, Breakpoint \*brkpt)  
*inserts a new function invocation code in a given set of tasks and calls it once*
- `int RemoveFuncCall` (string const &funcName, string const &callerFunc, Breakpoint \*brkpt)  
*removes all calls to a given function from the given caller function in a given set of tasks. For example this method can be used to remove all flush() function calls from a debug() function.*
- `int FuncParamChange` (string const &funcName, int paramIdx, int newValue, int \*requiredOldValue, Breakpoint \*brkpt)  
*Sets the value of an input parameter of a given function in a given set of tasks. This parameter value is modified before the function body is invoked. There is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.*
- `void SetTaskHandler` (TaskHandler &h)  
*Installs a callback function that is called when a new task is started or existing one is terminated.*
- `void SetHostHandler` (HostHandler &h)  
*Installs a callback function that is called when a new host is added to the virtual machine or an existing one is removed.*
- `int ProcessEvents` (bool block=true)  
*processes application events (ECP).*
- `void OnEvent` (ECPMessage \*msg)  
*This method is called in the context of [Event](#) Collector thread.*
- `void OnFatalError` ()  
*This method is called when fatal [EventCollector](#) error occurs. [Application](#) changes its status to stAborted.*

### Protected Member Functions

- void [ProcessEvent](#) (ECPMessage \*msg)  
*takes the properly actions depending on the kind of message received.*
- void [DispatchEvent](#) (EventMsg const &msg)  
*finds the sender-task corresponding object and dispatchs the event for it (see task).*
- [Host](#) & [AddHost](#) (string const &name)  
*creates & adds a new host to the host list of the application.*
- void [AddTask](#) (int pid, int mpiRank, string const &name, [Host](#) &h)  
*adds a task to the application list.*
- void [RemoveTask](#) (int tid)  
*removes a task when an unregister message is received (see processEvent).*

#### 4.7.1 Detailed Description

represents tuned application in the analyzer. Holds identificative information of the application, the tasks that form it (and which one is the master), the host where they are running, the places from where we are getting events and hanlers to both, tasks and hosts. Provides methods to:

- Retrieve application information
- Monitoring: add/remove events to trace.
- Tuning: loading libraries, changing variables & parameter values, adding/removing function calls and calling them explicitly.

Basically the monitoring a tuning methods call to the corresponding methods in App-Task for all the tasks that conform the application.

#### 4.7.2 Constructor & Destructor Documentation

##### 4.7.2.1 Application::Application ( char const \* appPath, int argc, char const \*\* argv )

Constructor.

##### Parameters

<i>appPath</i>	path to the executable of the application.
<i>argc</i>	Number of arguments of the application.
<i>argv</i>	Arguments of the application.

### 4.7.3 Member Function Documentation

#### 4.7.3.1 `int Application::AddEvent ( Event const & e )`

Adds a definition of new event to be traced in all running tasks of the application.

##### Parameters

<i>e</i>	event to be traced.
----------	---------------------

##### Returns

number of tasks where the event tracing was added.

#### 4.7.3.2 `Host & Application::AddHost ( string const & name )` [protected]

creates & adds a new host to the host list of the application.

##### Parameters

<i>name</i>	name of the host
-------------	------------------

##### Returns

reference to the created host

#### 4.7.3.3 `void Application::AddTask ( int pid, int mpiRank, string const & name, Host & h )` [protected]

adds a task to the application list.

##### Parameters

<i>pid</i>	process identifier of the task.
<i>mpiRank</i>	MPI identifier of the task
<i>name</i>	process name
<i>host</i>	host where the task is running

#### 4.7.3.4 `void Application::DispatchEvent ( EventMsg const & msg )` [protected]

finds the sender-task corresponding object and dispatchs the event for it (see task).

##### Parameters

<i>msg</i>	message that contains an event request from an AC.
------------	--



**4.7.3.5** `int Model::Application::FuncParamChange ( string const & funcName, int paramIdx, int newValue, int * requiredOldValue, Breakpoint * brkpt )`

Sets the value of an input parameter of a given function in a given set of tasks. This parameter value is modified before the function body is invoked. There is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

#### Parameters

<i>funcName</i>	name of the function
<i>paramIdx</i>	id of the parameter to change
<i>newValue</i>	new value for the parameter
<i>re- quiredOld- Value</i>	required old value of the parameter to change it
<i>brkpt</i>	---

#### Returns

number of tasks where the parameter was changed.

**4.7.3.6** `Hosts& Model::Application::GetHosts ( ) [inline]`

hosts getter.

#### Returns

a collection of [Host](#) objects that form the virtual machines

**4.7.3.7** `string Model::Application::GetName ( ) const [inline]`

name getter.

#### Returns

name of the running program

**4.7.3.8** `Status Model::Application::GetStatus ( ) const [inline]`

#### Returns

the application status information

**4.7.3.9** `Tasks& Model::Application::GetTasks ( ) [inline]`

tasks getter.

**Returns**

a collection of [Task](#) objects

**4.7.3.10** `int Model::Application::InsertFunctionCall ( string const & funcName, int nAttrs, Attribute * attrs, string const & destFunc, InstrPlace destPlace, Breakpoint * brkpt )`

Inserts a new function invocation code at a given location in a given set of tasks.

**Parameters**

<i>funcName</i>	name of the function to call.
<i>nAttrs</i>	number of parameters of the function.
<i>attrs</i>	values for each parameter.
<i>destFunc</i>	function where the calls will be placed.
<i>destPlace</i>	point of the function where the calls will be placed.
<i>brkpt</i>	---

**Returns**

number of tasks where the function calls were added.

**4.7.3.11** `int Model::Application::LoadLibrary ( string const & libPath )`

Loads a shared library to all running tasks. This enables the [Analyzer](#) to load any additional code required for the tuning.

**Parameters**

<i>libPath</i>	path to the library.
----------------	----------------------

**Returns**

number of tasks where the library is loaded.

**4.7.3.12** `int Model::Application::NumActiveTasks ( ) const [inline]`

number of tasks getter.

**Returns**

number of tasks actually running

**4.7.3.13** `int Model::Application::OneTimeFuncCall ( string const & funcName, int nAttrs, Attribute * attrs, Breakpoint * brkpt )`

inserts a new function invocation code in a given set of tasks and calls it once

#### Parameters

<i>funcName</i>	name of the function to call
<i>nAttrs</i>	number of arguments of the function
<i>attrs</i>	values for each argument of the function
<i>brkpt</i>	---

#### Returns

number of tasks where the function was call.

**4.7.3.14** `void Application::OnEvent ( ECPMessage * msg )` [virtual]

This method is called in the context of [Event](#) Collector thread.

#### Parameters

<i>msg</i>	pointer to a message object that must be deleted by a receiver.
------------	---

Implements [EventListener](#).

**4.7.3.15** `void Application::ProcessEvent ( ECPMessage * msg )` [protected]

takes the properly actions depending on the kind of message received.

- register: adds the host where the new task was created and creates a task object to represent it.
- unregister: removes the task from the list of task.
- event: calls `DispatchEvent` to handle it.

#### Parameters

<i>msg</i>	message that contains a request from an AC.
------------	---

**4.7.3.16** `int Application::ProcessEvents ( bool block = true )`

processes application events (ECP).

#### Parameters

<i>block</i>	indicates if the function blocks and waits for next event.
--------------	--

**Returns**

number of processed events

**4.7.3.17 int Application::RemoveEvent ( int *eventId*, InstrPlace *place* )**

Removes previously added event from all running tasks.

**Parameters**

<i>eventId</i>	id of the event
<i>place</i>	place of the function where the event is recorded?

**Returns**

number of tasks where the event was removed.

**4.7.3.18 int Model::Application::RemoveFuncCall ( string const & *funcName*, string const & *callerFunc*, Breakpoint \* *brkpt* )**

removes all calls to a given function from the given caller function in a given set of tasks. For example this method can be used to remove all flush() function calls from a debug() function.

**Parameters**

<i>funcName</i>	name of the function
<i>callerFunc</i>	function that calls the function that is will be removed
<i>brkpt</i>	---

**Returns**

number of tasks where the function call is removed.

**4.7.3.19 void Application::RemoveTask ( int *tid* ) [protected]**

removes a task when an unregister message is received (see processEvent).

**Parameters**

<i>tid</i>	process (thread) identificator of the task.
------------	---

**4.7.3.20 int Model::Application::ReplaceFunction ( string const & *oldFunc*, string const & *newFunc*, Breakpoint \* *brkpt* )**

Replaces all calls to a function with calls to another one in a given set of tasks.

**Parameters**

<i>oldFunc</i>	name of the function to replace.
<i>newFunc</i>	name of the new function.
<i>brkpt</i>	---

**Returns**

number of tasks where the function calls were changed.

**4.7.3.21 void Application::SetHostHandler ( HostHandler & h )**

Installs a callback function that is called when a new host is added to the virtual machine or an existing one is removed.

**Parameters**

<i>h</i>	handler for the new hosts.
----------	----------------------------

**4.7.3.22 void Application::SetTaskHandler ( TaskHandler & h )**

Installs a callback function that is called when a new task is started or existing one is terminated.

**Parameters**

<i>h</i>	handler for the new tasks.
----------	----------------------------

**4.7.3.23 int Model::Application::SetVariableValue ( string const & varName, AttributeValue const & varValue, Breakpoint \* brkpt )**

Modifies a value of a specified variable in a given set of tasks.  
application process.

**Parameters**

<i>varName</i>	name of the variable.
<i>varValue</i>	new value for the variable.
<i>brkpt</i>	---

**Returns**

number of tasks where the values were changed.

**4.7.3.24 void Application::Start ( )**

starts the application.

## Deprecated

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

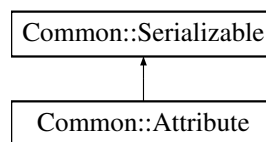
- Analyzer/AppModel.h
- Analyzer/AppModel.cpp

## 4.8 Common::Attribute Class Reference

Contains the necessary information of an attribute to be inserted in a program.

```
#include <Utils.h>
```

Inheritance diagram for Common::Attribute:



### Public Member Functions

- [Attribute](#) ([Attribute](#) const &a)  
*Copy constructor.*
- [Attribute](#) ()  
*Constructor.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the data serialized.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Gets the data deserialized.*
- string [GetSourceString](#) () const  
*Returns the string of the source.*
- string [GetTypeString](#) () const  
*Returns the type of the attribute.*
- void [Dump](#) () const  
*Logs the information of the attribute on the System Log.*

### Static Public Member Functions

- static string [GetTypeString](#) (AttrValueType type)

*Given a value of the enumerator AttrValueType returns the type in a string.*

### Public Attributes

- AttrSource **source**
- AttrValueType **type**
- string **id**

#### 4.8.1 Detailed Description

Contains the necessary information of an attribute to be inserted in a program.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.8.2 Constructor & Destructor Documentation

##### 4.8.2.1 Common::Attribute::Attribute ( ) [inline]

Constructor.

Creates a default [Attribute](#) object of the integer type.

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

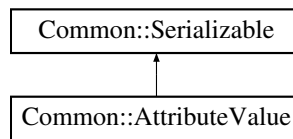
- Common/Utils.h
- Common/Utils.cpp

### 4.9 Common::AttributeValue Class Reference

Contains the value of an attribute.

```
#include <Utils.h>
```

Inheritance diagram for Common::AttributeValue:



## Public Member Functions

- [AttributeValue](#) ()  
*Constructor.*
- [AttributeValue](#) ([AttributeValue](#) const &av)  
*Copy constructor.*
- void [operator=](#) ([AttributeValue](#) const &av)  
*Assignment operator, copies content of the given object.*
- AttrValueType [GetType](#) () const  
*Returns type of the attribute.*
- void [SetType](#) (AttrValueType attrType)  
*Sets the type of the attribute.*
- int [GetIntValue](#) () const  
*Gets the integer value.*
- std::string [GetStringValue](#) () const  
*Gets the string value.*
- short [GetShortValue](#) () const  
*Gets the short value.*
- float [GetFloatValue](#) () const  
*Gets the float value.*
- double [GetDoubleValue](#) () const  
*Gets the double value.*
- char [GetCharValue](#) () const  
*Gets the char value.*
- void \* [GetValueBuffer](#) ()  
*Gets the pointer to the buffer.*
- int [GetSize](#) () const



*Gets the size of the value in memory.*

- string [ToString](#) () const  
*Returns the value in a string.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the data serialized.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Gets the data deserialized.*

### Public Attributes

- union {  
    int **intValue**  
    short **shortValue**  
    float **floatValue**  
    double **doubleValue**  
    char **charValue**  
};

### 4.9.1 Detailed Description

Contains the vale of an attribute.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

### 4.9.2 Member Function Documentation

#### 4.9.2.1 char AttributeValue::GetCharValue ( ) const [inline]

Gets the char value.

#### Exceptions

<a href="#">Exception</a>	
---------------------------	--

**4.9.2.2 double AttributeValue::GetDoubleValue ( ) const** [inline]

Gets the double value.

**Exceptions**

<a href="#"><i>Exception</i></a>
----------------------------------

**4.9.2.3 float AttributeValue::GetFloatValue ( ) const** [inline]

Gets the float value.

**Exceptions**

<a href="#"><i>Exception</i></a>
----------------------------------

**4.9.2.4 int AttributeValue::GetIntValue ( ) const** [inline]

Gets the integer value.

**Exceptions**

<a href="#"><i>Exception</i></a>
----------------------------------

**4.9.2.5 short AttributeValue::GetShortValue ( ) const** [inline]

Gets the short value.

**Exceptions**

<a href="#"><i>Exception</i></a>
----------------------------------

**4.9.2.6 std::string AttributeValue::GetStringValue ( ) const** [inline]

Gets the string value.

**Exceptions**

<a href="#"><i>Exception</i></a>
----------------------------------

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

- Common/Utils.h
- Common/Utils.cpp

## 4.10 `auto_iterator< T >` Class Template Reference

### Public Member Functions

- `auto_iterator` (`auto_ptr< T > *pp`)
- `bool operator!=` (`auto_iterator< T > const &it`) `const`
- `auto_iterator` `const & operator++` (`int`)
- `auto_iterator operator++` (`()`)
- `T * operator*` (`()`)
- `T const * operator*` (`() const`)
- `T * operator->` (`()`)

`template<class T> class auto_iterator< T >`

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

- `Common/auto_vector.h`

## 4.11 `auto_vector< T >` Class Template Reference

### Public Types

- `typedef auto_iterator< T > iterator`

### Public Member Functions

- `auto_vector` (`size_t capacity=0`)
- `T const * operator[]` (`size_t i`) `const`
- `T * operator[]` (`size_t i`)
- `void assign` (`size_t i`, `auto_ptr< T > &p`)
- `void assign_direct` (`size_t i`, `T *p`)
- `void Dump` (`()`)
- `void clear` (`()`)
- `void push_back` (`auto_ptr< T > &p`)
- `auto_ptr< T > pop_back` (`()`)
- `auto_ptr< T > acquire` (`size_t i`)
- `iterator begin` (`() const`)
- `iterator end` (`() const`)
- `int size` (`() const`)

`template<class T> class auto_vector< T >`

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

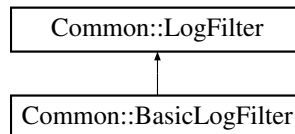
- `Common/auto_vector.h`

## 4.12 Common::BasicLogFilter Class Reference

Filters [LogEntry](#) objects to be inserted in a log.

```
#include <Syslog.h>
```

Inheritance diagram for Common::BasicLogFilter:



### Public Member Functions

- [BasicLogFilter](#) ()  
*Constructor.*
- [BasicLogFilter](#) (int mask)  
*Constructor.*
- bool [Accept](#) ([LogEntry](#) const &entry) const  
*Returns true if the log is accepted, false otherwise.*

### 4.12.1 Detailed Description

Filters [LogEntry](#) objects to be inserted in a log.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 Common::BasicLogFilter::BasicLogFilter ( int mask ) [inline]

Constructor.

#### Parameters

<i>mask</i>	Severity mask.
-------------	----------------

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

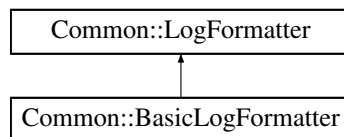
- Common/Syslog.h
- Common/Syslog.cpp

## 4.13 Common::BasicLogFormatter Class Reference

Formats [LogEntry](#) objects to be inserted in a log.

```
#include <Syslog.h>
```

Inheritance diagram for Common::BasicLogFormatter:



### Public Member Functions

- [BasicLogFormatter](#) ()  
*Constructor.*
- [BasicLogFormatter](#) (Config &cfg)  
*Constructor.*
- std::string [GetLogHeader](#) () const  
*Returns a string containing the log header.*
- std::string [GetLogFooter](#) () const  
*Returns a string containing the log footer.*
- std::string [Format](#) ([LogEntry](#) const &entry) const  
*Returns a string containing the [LogEntry](#) object formatted.*
- void [ShowTimestamp](#) (bool value)  
*Enables the timestamp view.*
- void [ShowSeverity](#) (bool value)  
*Enables the severity view.*

- void [ShowChannel](#) (bool value)

*Enables the channel view.*

#### 4.13.1 Detailed Description

Formats [LogEntry](#) objects to be inserted in a log.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.13.2 Constructor & Destructor Documentation

##### 4.13.2.1 BasicLogFormatter::BasicLogFormatter ( Config & *cfg* )

Constructor.

##### Parameters

<i>cfg</i>	A <a href="#">Config</a> object containing initial settings of the log formatter.
------------	---

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

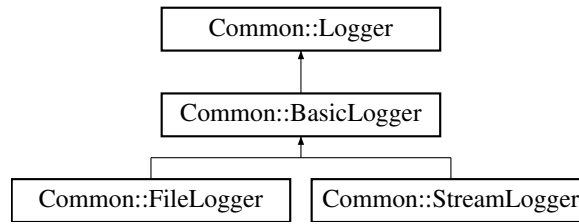
- Common/Syslog.h
- Common/Syslog.cpp

#### 4.14 Common::BasicLogger Class Reference

Stores information of events in a system.

```
#include <Syslog.h>
```

Inheritance diagram for Common::BasicLogger:



### Public Member Functions

- [BasicLogger](#) ()  
*Constructor.*
- void [SetFilter](#) (LogFilterPtr &filter)  
*Sets the [LogFilter](#) to be used by the logger.*
- [LogFilter](#) const \* [GetFilter](#) () const  
*Returns the [LogFilter](#) the logger uses.*
- void [SetFormatter](#) (LogFormatterPtr &formatter)  
*Sets the [LogFormatter](#) to be used by the logger.*
- [LogFormatter](#) const \* [GetFormatter](#) () const  
*Returns the [LogFormatter](#) the logger uses.*
- bool [Accept](#) (LogEntry const &entry) const  
*Inserts an entry to the log.*

### Protected Attributes

- LogFilterPtr **\_filter**
- LogFormatterPtr **\_formatter**

#### 4.14.1 Detailed Description

Stores information of events in a system.

##### Version

1.0b

##### Since

1.0b

**Author**

Ania Morajko, 2002

**4.14.2 Member Function Documentation****4.14.2.1 bool BasicLogger::Accept ( LogEntry const & entry ) const**

Inserts an entry to the log.

**Returns**

True if the insert was successful, false otherwise.

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

- Common/Syslog.h
- Common/Syslog.cpp

**4.15 BatchData Class Reference**

statistics of a single batch

```
#include <FactoringStats_nw.h>
```

**Public Member Functions**

- **BatchData** (int batchIdx)
- void **OnNewBatch** (int numChunks)
- [WorkerData](#) & **GetWorkerData** (int workerTid)
- [WorkerData](#) & **NewWorkerData** (int workerTid)
- bool **IsComplete** () const
- bool **IsActualize** () const
- void **SetActualize** ()
- bool **AreWorkersComplete** () const
- [WorkerData](#) \*\* **AllocWorkersArray** ()
- double **MeanComputingTime** ()
- double **DeviationComputingTime** ()
- int **GetNumChunks** () const
- double **GetMeanStats** ()
- double **GetStdStats** ()
- void **SizeTaskReceived** (int sizeTasks)
- int **GetSizeTaskReceived** () const
- [ModelParam](#) **GetModelParam** ()



### 4.15.1 Detailed Description

statistics of a single batch

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

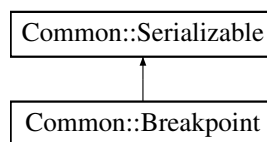
- Analyzer/FactoringStats\_nw.h
- Analyzer/FactoringStats\_nw.cpp

## 4.16 Common::Breakpoint Class Reference

Denotes place in a function (on the entry or at the end).

```
#include <Utils.h>
```

Inheritance diagram for Common::Breakpoint:



### Public Member Functions

- [Breakpoint](#) ()  
*Constructor.*
- [Breakpoint](#) ([Breakpoint](#) const &b)  
*Copy Constructor.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Serializes the breakpoint through the given [Serializer](#).*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Deserializes the breakpoint from the given [DeSerializer](#).*

### Public Attributes

- std::string **funcName**
- InstrPlace **place**

### 4.16.1 Detailed Description

Denotes place in a function (on the entry or at the end).

**Version**

1.0

**Since**

1.0

**Author**

Ania Morajko, 2002

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

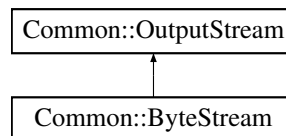
- Common/Utils.h

## 4.17 Common::ByteStream Class Reference

Stores stream of bytes.

```
#include <ByteStream.h>
```

Inheritance diagram for Common::ByteStream:



### Public Member Functions

- [ByteStream](#) (char \*buf, size\_t bufSize)  
*Constructor.*
- [ByteStream](#) (size\_t bufSize)  
*Constructor.*
- void [Write](#) (char const \*buf, size\_t bufSize)  
*Adds the content of the buffer to the stream.*
- char const \* [GetData](#) () const  
*Returns pointer to the intern buffer.*

- `size_t GetDataSize () const`  
*Returns size of the stream.*
- `void Reset ()`  
*Clears the stream.*

### 4.17.1 Detailed Description

Stores stream of bytes.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 Common::ByteStream::ByteStream ( char \* buf, size\_t bufSize ) [inline]

Constructor.

##### Parameters

<i>buf</i>	Intern buffer to be used.
<i>bufSize</i>	Size of the intern buffer.

#### 4.17.2.2 Common::ByteStream::ByteStream ( size\_t bufSize ) [inline]

Constructor.

Creates an intern buffer.

##### Parameters

<i>bufSize</i>	Size of the intern buffer.
----------------	----------------------------

### 4.17.3 Member Function Documentation

#### 4.17.3.1 `char const* Common::ByteStream::GetData ( ) const` `[inline]`

Returns pointer to the intern buffer.

When used the returned pointer should always use the [GetDataSize\(\)](#) method to iterate over the buffer.

#### Returns

Read-only pointer to the intern buffer.

#### 4.17.3.2 `size_t Common::ByteStream::GetDataSize ( ) const` `[inline]`

Returns size of the stream.

Buffer size.

#### 4.17.3.3 `void ByteStream::Write ( char const * buf, size_t bufSize )` `[virtual]`

Adds the content of the buffer to the stream.

#### Parameters

<i>buf</i>	Buffer to read.
<i>bufSize</i>	Size of the given buffer.

Implements [Common::OutputStream](#).

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

- Common/ByteStream.h
- Common/ByteStream.cpp

## 4.18 CommandLine Class Reference

Encapsulates methods to interact with the user of analyzer. Basically reads the arguments of the analyzer and parses them to get the configuration file and the objective application with its parameters. Once read, encapsulates the information and provides accessors to it. There are two formats [Analyzer](#) can be called:

```
#include <cmdline.h>
```

### Public Member Functions

- [CommandLine](#) (int argc, char \*\*argv)  
*Constructor.*

- bool [IsOk](#) () const  
*Returns the value of \_isOk.*
- int [GetArgc](#) () const  
*Getter for the \_argc variable.*
- char \*\* [GetArgv](#) () const  
*Getter for the \_argv variable.*
- int [GetAppArgc](#) () const  
*Getter for the \_appArgc variable.*
- char \* [GetAppPath](#) () const  
*Getter for the \_appPath variable.*
- char \*\* [GetAppArgv](#) () const  
*Getter for the \_appArgv variable.*
- bool [HasConfig](#) () const  
*Checks if there's a path for the configuration file, if not returns 0.*
- char \* [GetConfigFileName](#) () const  
*Getter for the \_configFile variable.*
- void [DisplayHelp](#) () const  
*Prints help message on the terminal.*
- [CommandLine](#) (int argc, char \*\*argv)  
*Constructor, parses the arguments provided to analyzer.*
- bool [IsOk](#) () const  
*status of the arguments getter.*
- int [GetArgc](#) () const  
*number of arguments getter.*
- char \*\* [GetArgv](#) () const  
*arguments getter.*
- bool [HasConfig](#) () const
- char \* [GetConfigFile](#) () const  
*configuration file getter.*
- char const \* [GetAppPath](#) () const  
*application path getter.*

- int [GetAppArgc](#) () const  
*application number of arguments getter.*
- char const \*\* [GetAppArgv](#) () const  
*application arguments getter.*
- void [DisplayHelp](#) () const  
*Explains the user which arguments can be provided to analyzer.*

#### 4.18.1 Detailed Description

Encapsulates methods to interact with the user of analyzer. Basically reads the arguments of the analyzer and parses them to get the configuration file and the objective application with its parameters. Once read, encapsulates the information and provides accessors to it. There are two formats [Analyzer](#) can be called: Checks for the necessary data in the arguments passed to main and parses them.

- [Analyzer](#) <AppPath> [<AppArgs>]
- [Analyzer](#) -config file.ini <App> [<AppArgs>]

Notes

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

#### 4.18.2 Constructor & Destructor Documentation

##### 4.18.2.1 `CommandLine::CommandLine ( int argc, char ** argv )` `[inline]`

Constructor, parses the arguments provided to analyzer.

#### Parameters

<i>argc</i>	number of arguments for analyzer
<i>argv</i>	arguments for analyzer

### 4.18.3 Member Function Documentation

#### 4.18.3.1 void CommandLine::DisplayHelp ( ) const [inline]

Prints help message on the terminal.

Tells the user how to introduce the necessary arguments.

#### 4.18.3.2 int CommandLine::GetAppArgc ( ) const [inline]

Getter for the \_appArgc variable.

##### Returns

Size of the arguments vector for the app.

#### 4.18.3.3 int CommandLine::GetAppArgc ( ) const [inline]

application number of arguments getter.

##### Returns

the number of arguments of the target application.

#### 4.18.3.4 char const\*\* CommandLine::GetAppArgv ( ) const [inline]

application arguments getter.

##### Returns

the arguments of the target application.

#### 4.18.3.5 char\*\* CommandLine::GetAppArgv ( ) const [inline]

Getter for the \_appArgv variable.

##### Returns

Vector that contains the app's arguments.

#### 4.18.3.6 char\* CommandLine::GetAppPath ( ) const [inline]

Getter for the \_appPath variable.

##### Returns

Path to the executable of the app.

**4.18.3.7** `char const* CommandLine::GetAppPath ( ) const` `[inline]`

application path getter.

**Returns**

The path of the target application.

**4.18.3.8** `int CommandLine::GetArgc ( ) const` `[inline]`

Getter for the `_argc` variable.

**Returns**

Size of the vector of arguments.

**4.18.3.9** `int CommandLine::GetArgc ( ) const` `[inline]`

number of arguments getter.

**Returns**

the number of arguments provided to analyzer.

**4.18.3.10** `char** CommandLine::GetArgv ( ) const` `[inline]`

arguments getter.

**Returns**

the arguments provided to analyzer.

**4.18.3.11** `char** CommandLine::GetArgv ( ) const` `[inline]`

Getter for the `_argv` variable.

**Returns**

Vector of arguments.

**4.18.3.12** `char* CommandLine::GetConfigFile ( ) const` `[inline]`

configuration file getter.

**Returns**

the configuration file of analyzer.



**4.18.3.13** `char* CommandLine::GetConfigFileName ( ) const [inline]`

Getter for the \_configFile variable.

**Returns**

Path for the configuration file.

**4.18.3.14** `bool CommandLine::HasConfig ( ) const [inline]`

if the user has chosen his own configuration file.

**Returns**

if the user provided a specific configuration file.

**4.18.3.15** `bool CommandLine::HasConfig ( ) const [inline]`

Checks if there's a path for the configuration file, if not returns 0.

**Returns**

Path for the configuration file.

**4.18.3.16** `bool CommandLine::IsOk ( ) const [inline]`

Returns the value of \_isOk.

**Returns**

Boolean variable thats true if the configuration has been parsed correctly.

**4.18.3.17** `bool CommandLine::IsOk ( ) const [inline]`

status of the arguments getter.

**Returns**

if the arguments provided to analyzer are correct or not.

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

- AC/cmdline.h
- Analyzer/cmdline.h

## 4.19 comptime Struct Reference

### Public Attributes

- long\_t **iniTime**
- long\_t **finTime**
- long\_t **ct**

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

- Analyzer/AdjustingNWUnlet.h

## 4.20 Common::Config Class Reference

Manages a configuration of the system.

```
#include <Config.h>
```

### Classes

- class [KeyIterator](#)  
*Iterates over the keys of a [Config](#) object.*

### Public Member Functions

- [Config](#) ()  
*Constructor.*
- std::string const & [GetStringValue](#) (std::string const &section, std::string const &key) const  
*Returns string value of the entry specified by the parameters.*
- int [GetIntValue](#) (std::string const &section, std::string const &key) const  
*Returns integer value of the entry specified by the parameters.*
- int [GetIntValue](#) (std::string const &section, std::string const &key, int default-Value) const  
*Returns integer value of the entry specified by the parameters.*
- bool [GetBoolValue](#) (std::string const &section, std::string const &key) const  
*Returns boolean value of the entry specified by the parameters.*
- bool [GetBoolValue](#) (std::string const &section, std::string const &key, bool defaultValue) const

*Returns boolean value of the entry specified by the parameters.*

- bool [Contains](#) (std::string const &section, std::string const &key) const  
*Finds an entry on the configuration.*
- [KeyIterator GetKeys](#) (std::string const &section) const  
*Returns an iterator of the keys inside the requested section.*
- void [AddEntry](#) (std::string const &section, std::string const &key, std::string const &value)  
*Adds a new entry to the configuration.*

## Friends

- class **KeyIterator**
- class **ConfigReader**

### 4.20.1 Detailed Description

Manages a configuration of the system. The configuration is based on section and keys, and the format is the following:

```
[section]
key = value
key = value
...

[newsection]
key = value
...
```

## Version

1.0b

## Since

1.0b

## Author

Ania Morajko, 2000

### 4.20.2 Member Function Documentation

**4.20.2.1** void **Common::Config::AddEntry** ( std::string const & *section*, std::string const & *key*, std::string const & *value* ) `[inline]`

Adds a new entry to the configuration.

**Parameters**

<i>section</i>	Section of the new entry.
<i>key</i>	Key of the new entry.
<i>value</i>	Value of the new entry.

**4.20.2.2** `bool Common::Config::Contains ( std::string const & section, std::string const & key ) const [inline]`

Finds an entry on the configuration.

**Parameters**

<i>section</i>	Section to find the entry.
<i>key</i>	Key to find the entry.

**Returns**

True if the entry was found, false otherwise.

**4.20.2.3** `bool Common::Config::GetBoolValue ( std::string const & section, std::string const & key, bool defaultValue ) const [inline]`

Returns boolean value of the entry specified by the parameters.

If the configuration doesn't contain the specified entry, returns the default value.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.
<i>defaultValue</i>	Value returned if the requested entry is not inside the configuration.

**Returns**

Boolean containing the requested value.

**4.20.2.4** `bool Common::Config::GetBoolValue ( std::string const & section, std::string const & key ) const`

Returns boolean value of the entry specified by the parameters.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.

**Returns**

Boolean containing the requested value.

**Exceptions**

<a href="#"><i>ConfigException</i></a>	
--	--

**4.20.2.5 int Common::Config::GetIntValue ( std::string const & *section*, std::string const & *key* ) const**

Returns integer value of the entry specified by the parameters.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.

**Exceptions**

<a href="#"><i>ConfigException</i></a>	
--	--

**4.20.2.6 int Common::Config::GetIntValue ( std::string const & *section*, std::string const & *key*, int *defaultValue* ) const [inline]**

Returns integer value of the entry specified by the parameters.

If the configuration doesn't contain the specified entry, returns the default value.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.
<i>defaultValue</i>	Value returned if the requested entry is not inside the configuration.

**Returns**

Integer containing the requested value.

**4.20.2.7 KeyIterator Common::Config::GetKeys ( std::string const & *section* ) const [inline]**

Returns an iterator of the keys inside the requested section.

**Parameters**

<i>section</i>	Section requested.
----------------	--------------------

**Returns**

Iterator to the keys inside the section.

**4.20.2.8** `std::string const& Common::Config::GetStringValue ( std::string const & section,  
std::string const & key ) const` `[inline]`

Returns string value of the entry specified by the parameters.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.

**Exceptions**

<a href="#"><i>ConfigException</i></a>	
--	--

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

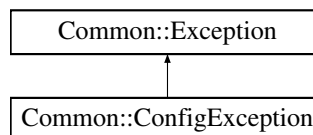
- Common/Config.h

**4.21 Common::ConfigException Class Reference**

[Config](#), [ConfigReader](#) and [ConfigMap](#) exceptions.

```
#include <ConfigException.h>
```

Inheritance diagram for Common::ConfigException:

**Public Member Functions**

- [ConfigException](#) (std::string const &msg, std::string const &objName=std::string())

*Constructor.*

- void [Display](#) (std::ostream &os) const

*Displays exception message on the given output stream.*

- void [Display](#) () const

*Displays exception message on the standard error output.*

- `std::string GetReason () const`  
*Returns a string containing the error message.*

### 4.21.1 Detailed Description

[Config](#), [ConfigReader](#) and [ConfigMap](#) exceptions.

#### Version

1.0b

#### Since

1.0b

#### Author

Noel De Martin, 2011

### 4.21.2 Constructor & Destructor Documentation

**4.21.2.1** `Common::ConfigException::ConfigException ( std::string const & msg, std::string const & objName = std::string () ) [inline]`

Constructor.

#### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>objName</i>	Name of the object causing the exception, "" by default.

### 4.21.3 Member Function Documentation

**4.21.3.1** `void Common::ConfigException::Display ( std::ostream & os ) const [virtual]`

Displays exception message on the given output stream.

#### Parameters

<i>os</i>	Output stream to display the message.
-----------	---------------------------------------

Reimplemented from [Common::Exception](#).

**4.21.3.2** `string ConfigException::GetReason ( ) const`

Returns a string containing the error message.

**Returns**

String with the error.

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

- Common/ConfigException.h
- Common/ConfigException.cpp

## 4.22 Common::ConfigHelper Class Reference

Static class that contains methods to manage [Config](#) objects.

```
#include <Config.h>
```

**Static Public Member Functions**

- static [Config ReadFromFile](#) (std::string const &fileName)  
*Returns a [Config](#) object loaded from the given file.*

### 4.22.1 Detailed Description

Static class that contains methods to manage [Config](#) objects.

**Version**

1.0b

**Since**

1.0b

**Author**

Noel De Martin, 2011

### 4.22.2 Member Function Documentation

**4.22.2.1 static Config Common::ConfigHelper::ReadFromFile ( std::string const & fileName )**  
[inline, static]

Returns a [Config](#) object loaded from the given file.

**Exceptions**

<a href="#">ConfigException</a>
---------------------------------



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

- Common/Config.h

## 4.23 Common::ConfigMap Class Reference

Contains and manages a collection of [Config](#) objects.

```
#include <ConfigMap.h>
```

### Classes

- class [Iterator](#)  
*Iterates over a [ConfigMap](#) object.*

### Public Member Functions

- [ConfigMap](#) ()  
*Constructor.*
- bool [Add](#) (std::string const &section, std::string const &key, std::string const &value)  
*Adds a new value to the map.*
- std::string const & [GetValue](#) (std::string const &section, std::string const &key) const  
*Returns a requested value on the map.*
- bool [Contains](#) (std::string const &section, std::string const &key) const  
*Looks for the entry specified by the parameters of the function.*
- int [GetSize](#) () const  
*Returns size of the map.*

### Friends

- class **Iterator**

#### 4.23.1 Detailed Description

Contains and manages a collection of [Config](#) objects.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2000

**4.23.2 Member Function Documentation****4.23.2.1** `bool Common::ConfigMap::Add ( std::string const & section, std::string const & key, std::string const & value )`

Adds a new value to the map.

If the entry already exists on the map returns false.

**Parameters**

<i>section</i>	Section of the new entry.
<i>key</i>	Key of the new entry.
<i>value</i>	Value of the new entry.

**Returns**

True if the insertion was successful, false otherwise.

**4.23.2.2** `bool ConfigMap::Contains ( std::string const & section, std::string const & key ) const`

Looks for the entry specified by the parameters of the function.

**Parameters**

<i>section</i>	Section to find the entry.
<i>key</i>	Key to find the entry.

**Returns**

True if the entry was found, false otherwise.

**4.23.2.3** `std::string const& Common::ConfigMap::GetValue ( std::string const & section, std::string const & key ) const`

Returns a requested value on the map.

**Parameters**

<i>section</i>	Section to find the value.
<i>key</i>	Key to find the value.

**Exceptions**

<a href="#"><i>ConfigException</i></a>	
--	--

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

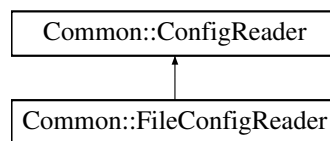
- Common/ConfigMap.h
- Common/ConfigMap.cpp

**4.24 Common::ConfigReader Class Reference**

Abstract class, generates [Config](#) objects from reading sources.

```
#include <ConfigReader.h>
```

Inheritance diagram for Common::ConfigReader:

**Public Member Functions**

- [ConfigReader](#) ()  
*Constructor.*
- virtual [Config](#) Read ()=0

**Protected Member Functions**

- void [AnalyzeLine](#) ([Config](#) &config, std::string const &line)  
*Loads the information of the line into the [Config](#) object.*

**4.24.1 Detailed Description**

Abstract class, generates [Config](#) objects from reading sources.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2000

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

- Common/ConfigReader.h
- Common/ConfigReader.cpp

## 4.25 Controller Class Reference

Provides the logic and controls the execution flow of the application.

```
#include <Ctrl.h>
```

**Public Member Functions**

- [Controller](#) ([CommandLine](#) &cmdLine)  
*Constructor.*
- [~Controller](#) ()  
*Destructor.*
- void [Run](#) ()  
*Initializes all the necessary fields and starts the main loop of the AC.*
- void [Interrupt](#) ()  
*Sets the `_fInterrupted` variable to 1.*
- [Controller](#) ([CommandLine](#) &cmdLine, std::string const &cfgFile)  
*Constructor, sets the command line for the user, determines the configuration for the application and prepares the system log.*
- void [Run](#) ([ShutDownManager](#) \*sdm)  
*Manages the execution flow of the application. The execution flow of analyzer is:*

### 4.25.1 Detailed Description

Provides the logic and controls the execution flow of the application. Contains the main functionality of the AC, including its main loop which runs until all tuning operations have been finished.

**Version**

1.0

**Since**

1.0

**Author**

Ania Morajko, 2002

**4.25.2 Constructor & Destructor Documentation****4.25.2.1 Controller::Controller ( CommandLine & *cmdLine* )**

Constructor.

**Parameters**

<i>cmdLine</i>	Class that provides commandline communications with the user.
----------------	---

**4.25.3 Member Function Documentation****4.25.3.1 void Controller::Run ( ShutDownManager \* *sdm* )**

Manages the execution flow of the application. The execution flow of analyzer is:

- create DTAPI, initialize collector, etc.
- create application model
- initialize all tunlets
- start application
- handle events
- destroy tunlets
- destroy app model

**4.25.3.2 void Controller::Run ( )**

Initializes all the necessary fields and starts the main loop of the AC.

Creates a [TaskManager](#) objects and a Reactor and [PTPAcceptor](#) which will provide event handling and tuning capabilities.

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

- AC/Ctrl.h

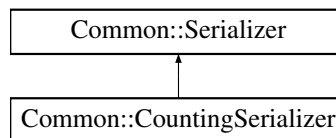
- Analyzer/Ctrl.h
- AC/Ctrl.cpp
- Analyzer/Ctrl.cpp

## 4.26 Common::CountingSerializer Class Reference

Stores the size of serialized data.

```
#include <NetSer.h>
```

Inheritance diagram for Common::CountingSerializer:



### Public Member Functions

- [CountingSerializer](#) ()  
*Constructor.*
- int\_t [GetSize](#) () const  
*Returns size of the serialized data.*
- void [PutLong](#) (long\_t l)  
*Adds the size of a serialized long.*
- void [PutDouble](#) (double\_t d)  
*Adds the size of a serialized double.*
- void [PutBool](#) (bool\_t b)  
*Adds the size of a serialized boolean.*
- void [PutShort](#) (short\_t s)  
*Adds the size of a serialized short.*
- void [PutByte](#) (byte\_t b)  
*Adds the size of a serialized byte.*
- void [PutChar](#) (char\_t c)  
*Adds the size of a serialized char.*
- void [PutString](#) (std::string const &str)

*Adds the size of a serialized string.*

- void **PutInt** (int\_t i)  
*Adds the size of a serialized integer.*
- void **PutBuffer** (char const \*buffer, int bufferSize)  
*Adds the size of a serialized buffer.*

#### 4.26.1 Detailed Description

Stores the size of serialized data.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/NetSer.h

## 4.27 curState Struct Reference

### Public Attributes

- int **iter**
- int **batch**
- int **numTuples**

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

- Analyzer/FactoringTunlet\_nw.cpp

## 4.28 Common::DateTime Class Reference

Holds a timestamp.

```
#include <DateTime.h>
```

## Public Member Functions

- [DateTime](#) ()  
*Constructor, sets the current date and time.*
- int [GetYear](#) () const  
*Returns year represented by this date.*
- int [GetMonth](#) () const  
*Returns month represented by this date.*
- int [GetDay](#) () const  
*Returns day represented by this date.*
- int [GetHour](#) () const  
*Returns hour represented by this date.*
- int [GetMinute](#) () const  
*Returns minute represented by this date.*
- int [GetSecond](#) () const  
*Returns second represented by this date.*
- std::string [GetStringValue](#) () const  
*Returns a string with the date.*

### 4.28.1 Detailed Description

Holds a timestamp.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2001

### 4.28.2 Member Function Documentation

#### 4.28.2.1 string DateTime::GetStringValue ( ) const

Returns a string with the date.



The format of the returned string is "dd.mm.yyyy hh:MM:ss".

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

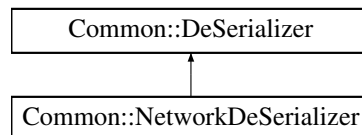
- Common/DateTime.h
- Common/DateTime.cpp

## 4.29 Common::DeSerializer Class Reference

Abstract class, recovers serialized data from a stream.

```
#include <Serial.h>
```

Inheritance diagram for Common::DeSerializer:



### Public Member Functions

- virtual byte\_t **GetByte** ()=0
- virtual char\_t **GetChar** ()=0
- virtual bool\_t **GetBool** ()=0
- virtual short\_t **GetShort** ()=0
- virtual int\_t **GetInt** ()=0
- virtual long\_t **GetLong** ()=0
- virtual double\_t **GetDouble** ()=0
- virtual std::string **GetString** ()=0
- virtual void **GetBuffer** (char \*buffer, int bufferSize)=0

### 4.29.1 Detailed Description

Abstract class, recovers serialized data from a stream.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

- Common/Serial.h

## 4.30 DiEx Class Reference

### Public Member Functions

- **DiEx** (string const &msg, string const &objName=string())
- string const & **GetMessage** () const
- string const & **GetObjectName** () const

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

- Common/di.h

## 4.31 DiFunction Class Reference

### Public Member Functions

- **DiFunction** (BPatch\_image &bpImage, string const &funcName)
- void **GetLineNumber** (unsigned int &start, unsigned int &end, char \*fileName, unsigned int &max)
- unsigned long **GetAddress** ()
- unsigned int **GetSize** ()
- char const \* **GetParams** ()
- PointVector \* **FindPoint** (BPatch\_procedureLocation loc=BPatch\_subroutine)
- void **GetName** (char \*fileName, int len)
- **operator BPatch\_function &** ()

### Static Public Member Functions

- static void **Dump** (FuncVector &fv)

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

- Common/di.h
- Common/di.cpp

## 4.32 DiImage Class Reference

### Public Member Functions

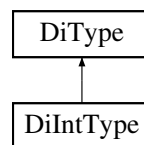
- **DiImage** (BPatch\_process &bpProcess)
- BPatch\_variableExpr \* **FindVariable** (const char \*name)
- **operator BPatch\_image & ()**

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

- Common/di.h
- Common/di.cpp

## 4.33 DiIntType Class Reference

Inheritance diagram for DiIntType:



### Public Member Functions

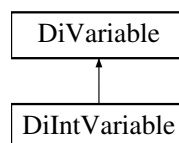
- **DiIntType** (BPatch\_image &bpImage)

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

- Common/di.h

## 4.34 DiIntVariable Class Reference

Inheritance diagram for DiIntVariable:



### Public Member Functions

- **DiIntVariable** (BPatch\_process &bpProcess)

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

- Common/di.h

## 4.35 DiPoint Class Reference

### Public Member Functions

- **DiPoint** (BPatch\_image &bpImage, string const &procName, BPatch\_procedureLocation loc=BPatch\_entry)
- void **GetCalledFuncName** (char \*buf, int size)
- unsigned long **GetAddress** ()
- **operator PointVector &** ()
- PointVector & **getPoints** ()

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

- Common/di.h
- Common/di.cpp

## 4.36 DiProcess Class Reference

### Public Member Functions

- **DiProcess** (char \*mutateeName, int pid)
- **DiProcess** (char \*mutateeName, char \*argv[], char \*envp[] =0)
- **DiProcess** (char \*mutateeName)
- **operator BPatch\_process &** ()
- int **GetPid** ()
- bool **IsStopped** ()
- BPatchSnippetHandle \* **InsertSnippet** (BPatch\_snippet const &expr, BPatch\_point &point)
- BPatchSnippetHandle \* **InsertSnippet** (BPatch\_snippet const &expr, BPatch\_point &point, BPatch\_callWhen when, BPatch\_snippetOrder order)
- BPatchSnippetHandle \* **InsertSnippetBefore** (BPatch\_snippet const &expr, BPatch\_point &point)
- BPatchSnippetHandle \* **InsertSnippetAfter** (BPatch\_snippet const &expr, BPatch\_point &point)
- BPatchSnippetHandle \* **InsertSnippet** (BPatch\_snippet const &expr, PointVector &points)

- BPatchSnippetHandle \* **InsertSnippet** (BPatch\_snippet const &expr, PointVector &points, BPatch\_callWhen when, BPatch\_snippetOrder order)
- BPatchSnippetHandle \* **InsertSnippetBefore** (BPatch\_snippet const &expr, PointVector &points)
- BPatchSnippetHandle \* **InsertSnippetAfter** (BPatch\_snippet const &expr, PointVector &points)
- void **DeleteSnippet** (BPatchSnippetHandle \*handle)
- void **OneTimeCode** (BPatch\_snippet const &expr)
- void **ReplaceFunction** (BPatch\_function &oldFunc, BPatch\_function &newFunc)
- void **ContinueExecution** ()
- bool **StopExecution** ()
- void **WaitFor** ()
- void **Test** ()
- bool **Terminate** ()
- bool **IsTerminated** ()
- void **WaitForStop** ()
- void **loadLibrary** (char \*libName)
- void **GetLineNumber** (unsigned long addr, unsigned short &line, char \*fileName, int length)
- BPatch\_variableExpr \* **Malloc** (BPatch\_type &type)

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

- Common/di.h
- Analyzer/ACProxy.cpp
- Analyzer/AppModel.cpp
- Analyzer/AppTask.cpp
- Common/di.cpp

## 4.37 DiSnippetHandle Class Reference

### Public Member Functions

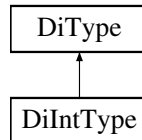
- **DiSnippetHandle** (BPatch\_process &bpProcess, BPatch\_snippet &snippet, BPatch\_point &point, bool needDelete=false)

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

- Common/di.h

### 4.38 DiType Class Reference

Inheritance diagram for DiType:



#### Public Member Functions

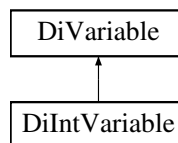
- **DiType** (BPatch\_image &bpImage, char const \*typeName)
- **operator BPatch\_type & ()**

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

- Common/di.h

### 4.39 DiVariable Class Reference

Inheritance diagram for DiVariable:



#### Public Member Functions

- **DiVariable** (BPatch\_process &bpProcess, BPatch\_type const &type)
- **DiVariable** (BPatch\_process &bpProcess, char const \*typeName)
- **DiVariable** (BPatch\_process &bpProcess, int size)
- **operator BPatch\_variableExpr & ()**
- **operator BPatch\_variableExpr \* ()**
- void **GetValue** (void \*dst) const
- long int **GetAddress** () const

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

- Common/di.h
- Common/ConfigMap.cpp

## 4.40 DTLibrary Class Reference

Dynamic Tuning Library that offers DT API. Encapsulates information about the application model and the event collector. Provides methods to create application models.

```
#include <DTAPI.h>
```

### Public Member Functions

- [Model::Application](#) & [CreateApplication](#) (char const \*appPath, int argc, char const \*\*argv)  
*creates a new application model, a new event collector and associates them.*
- [Model::Application](#) & [GetApplication](#) ()  
*application getter.*

### Friends

- class [DTLibraryFactory](#)

#### 4.40.1 Detailed Description

Dynamic Tuning Library that offers DT API. Encapsulates information about the application model and the event collector. Provides methods to create application models.

#### 4.40.2 Member Function Documentation

##### 4.40.2.1 [Model::Application](#) & [DTLibrary::CreateApplication](#) ( char const \* *appPath*, int *argc*, char const \*\* *argv* )

creates a new application model, a new event collector and associates them.

#### Parameters

<i>appPath</i>	path to the target application.
<i>argc</i>	number of arguments of the target application.
<i>argv</i>	list of arguments of the target application.

#### Returns

reference to the application model object.

##### 4.40.2.2 [Model::Application](#) & [DTLibrary::GetApplication](#) ( )

application getter.

**Returns**

a reference to the application model.

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

- Analyzer/DTAPI.h
- Analyzer/DTAPI.cpp

**4.41 DTLibraryFactory Class Reference**

Handles the creation and destruction of DT Libraries.

```
#include <DTAPI.h>
```

**Static Public Member Functions**

- static [DTLibrary](#) \* [CreateLibrary](#) (Config const &cfg)  
*Creates and initializes the DT Library. Implements the singleton design pattern, so if the library is already created it returns a reference to it.*
- static void [DestroyLibrary](#) ([DTLibrary](#) \*lib)  
*destroys the library if there is only one reference last to the object.*

**4.41.1 Detailed Description**

Handles the creation and destruction of DT Libraries.

**4.41.2 Member Function Documentation****4.41.2.1 [DTLibrary](#) \* [DTLibraryFactory::CreateLibrary](#) ( [Config](#) const & *cfg* ) [static]**

Creates and initializes the DT Library. Implements the singleton design pattern, so if the library is already created it returns a reference to it.

**Parameters**

<i>cfg</i>	reference to the configuration object.
------------	--

**Returns**

reference to the library.



**4.41.2.2 void DTLibraryFactory::DestroyLibrary ( DTLibrary \* *lib* ) [static]**

destroys the library if there is only one reference last to the object.

**Parameters**

<i>lib</i>	reference to the library.
------------	---------------------------

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

- Analyzer/DTAPI.h
- Analyzer/DTAPI.cpp

**4.42 DynInst Class Reference****Static Public Member Functions**

- static BPatch & **Instance** ()

**Static Protected Member Functions**

- static void **OnError** (BPatchErrorLevel severity, int number, const char \*const \*params)

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

- Common/di.h
- Common/di.cpp

**4.43 ECPAcceptor Class Reference**

```
#include <EventCollector.h>
```

**Public Member Functions**

- [ECPAcceptor](#) (Reactor &reactor, int port=5555)  
*Constructor, starts listening to the socket and registers itself in the reactor.*
- [~ECPAcceptor](#) ()  
*Destructor, unregister the object from the reactor.*
- void [HandleInput](#) ()  
*When a new connection is accepted, prepares a handler for it, which is registered in the reactor and added to the service.*

- int [GetHandle](#) ()
- void [SetEventCollector](#) ([EventCollector](#) \*collector)

*Setter for the event collector.*

#### 4.43.1 Detailed Description

@brief ---

#### 4.43.2 Constructor & Destructor Documentation

##### 4.43.2.1 [ECPAceptor::ECPAceptor](#) ( [Reactor](#) & *reactor*, int *port* = 5555 )

Constructor, starts listening to the socket and registers itself in the reactor.

##### Parameters

<i>reactor</i>	reactor of the application???
<i>port</i>	socket port.

#### 4.43.3 Member Function Documentation

##### 4.43.3.1 int [ECPAceptor::GetHandle](#) ( ) [inline]

##### Returns

a reference to the handler object

##### 4.43.3.2 void [ECPAceptor::SetEventCollector](#) ( [EventCollector](#) \* *collector* )

Setter for the event collector.

##### Parameters

<i>collector</i>	event collector to be set.
------------------	----------------------------

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

- Analyzer/EventCollector.h
- Analyzer/EventCollector.cpp

## 4.44 ECPHandler Class Reference

Encapsulates data structures and methods to handle incoming event collector inputs.

```
#include <ECPHandler.h>
```

### Public Member Functions

- [ECPHandler](#) (SocketPtr &socket, [EventCollector](#) \*collector)  
*Constructor.*
- void [Remove](#) ()  
*not implemented (here for compatibility reasons)*
- void [HandleInput](#) ()  
*reads an incoming message and handles it depending on its type. First reads a message from the socket, then creates the proper type of message and then calls the onEvent method of the listener of the events (if any).*
- int [GetHandle](#) ()  
*handler getter.*
- void [SetService](#) ([Service](#) \*service)  
*service setter.*

#### 4.44.1 Detailed Description

Encapsulates data structures and methods to handle incoming event collector inputs.

#### 4.44.2 Member Function Documentation

##### 4.44.2.1 int ECPHandler::GetHandle ( ) [inline]

handler getter.

#### Returns

a reference to the handler object

##### 4.44.2.2 void ECPHandler::SetService ( Service \* service ) [inline]

service setter.

#### Parameters

<i>service</i>	reference to the service.
----------------	---------------------------

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

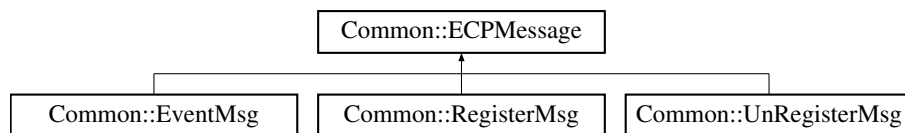
- Analyzer/ECPHandler.h
- Analyzer/ECPHandler.cpp

## 4.45 Common::ECPMessage Class Reference

Abstract class, EventCollectorProtocol, represents message interchanged between DM-Lib and analyzer.

```
#include <ECPMsg.h>
```

Inheritance diagram for Common::ECPMessage:



### Public Member Functions

- virtual ECPMsgType [GetType](#) () const =0  
*To be implemented by subclasses.*
- virtual int [GetDataSize](#) () const  
*Returns size of the data once serialized.*
- virtual void [Serialize](#) (Serializer &out) const =0  
*To be implemented by subclasses.*
- virtual void [DeSerialize](#) (DeSerializer &in)=0  
*To be implemented by subclasses.*

### Protected Member Functions

- [ECPMessage](#) ()  
*Constructor.*

#### 4.45.1 Detailed Description

Abstract class, EventCollectorProtocol, represents message interchanged between DM-Lib and analyzer.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2002

**4.45.2 Constructor & Destructor Documentation****4.45.2.1 Common::ECPMessage::ECPMessage ( ) [inline, protected]**

Constructor.

Protected so that this base class cannot be explicitly instantiated.

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

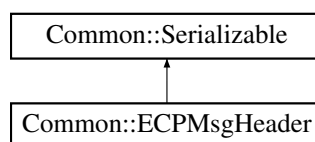
- Common/ECPMsg.h
- Common/ECPMsg.cpp

**4.46 Common::ECPMsgHeader Class Reference**

Represents header of an [ECPMessage](#) object.

```
#include <ECPMsgHeader.h>
```

Inheritance diagram for Common::ECPMsgHeader:

**Public Member Functions**

- [ECPMsgHeader](#) ()  
*Constructor.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the message header.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Receives the message header.*

- int [GetMagic](#) () const  
*Returns magic attribute.*
- int [GetVersion](#) () const  
*Returns version attribute.*
- ECPMsgType [GetType](#) () const  
*Returns type of the message.*
- int [GetDataSize](#) () const  
*Returns data size.*
- int [GetHeaderSize](#) () const  
*Returns header size.*
- void [SetMagic](#) (int magic)  
*Sets magic attribute.*
- void [SetVersion](#) (int version)  
*Sets version attribute.*
- void [SetMsgType](#) (ECPMsgType type)  
*Sets type attribute.*
- void [SetDataSize](#) (int size)  
*data size attribute.*
- void [SetHeaderSize](#) ()  
*Updates header size.*

#### 4.46.1 Detailed Description

Represents header of an [ECPMessage](#) object.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/ECPMsgHeader.h
- Common/ECPMsgHeader.cpp

## 4.47 ECPProtocol Class Reference

encapsulates methods to read and handle incoming network messages.

```
#include <ECPProtocol.h>
```

### Static Public Member Functions

- static ECPMsgHeader [ReadMessageHeader](#) (Socket &sock)  
*Reads a message header from the socket, deserializes it and creates a message header object.*
- static ECPMessage \* [ReadMessageEx](#) (Socket &sock)  
*Reads a message from the socket, deserializes it and creates different kind of message objects depending on their type.*

#### 4.47.1 Detailed Description

encapsulates methods to read and handle incoming network messages.

#### 4.47.2 Member Function Documentation

##### 4.47.2.1 ECPMessage \* ECPProtocol::ReadMessageEx ( Socket & sock ) [static]

Reads a message from the socket, deserializes it and creates different kind of message objects depending on their type.

##### Parameters

<i>sock</i>	reference to the socket.
-------------	--------------------------

##### Returns

reference to the message object created.

##### 4.47.2.2 ECPMsgHeader ECPProtocol::ReadMessageHeader ( Socket & sock ) [static]

Reads a message header from the socket, deserializes it and creates a message header object.

**Parameters**

<i>sock</i>	reference to the socket.
-------------	--------------------------

**Returns**

reference to the message header object created

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

- Analyzer/ECPProtocol.h
- Analyzer/ECPProtocol.cpp

**4.48 Model::Event Class Reference**

Encapsulates information about the events that the target application generates. For each event holds identificative information (id, name), the place where it is produced, its attributes (for example the parameters of a function) and a reference to a handler. As this is a model class the methods provided are for accessing and setting the members of the data structure.

```
#include <AppEvent.h>
```

**Public Member Functions**

- [Event](#) ([Event](#) const &e)  
*Copy constructor.*
- [Event](#) (int id, std::string const &funcName, InstrPlace place)  
*Constructor.*
- [~Event](#) ()  
*Destructor.*
- int [GetId](#) () const  
*globally unique event id getter.*
- string [GetFunctionName](#) () const  
*name getter.*
- InstrPlace [GetInstrPlace](#) () const  
*instruction place getter.*
- int [GetNumAttributes](#) () const  
*number of attributes getter.*
- Attribute \* [GetAttributes](#) () const



*attributes getter.*

- void [SetAttribute](#) (int nAttrs, Attribute \*attrs)

*attributes setter.*

- void [SetEventHandler](#) (EventHandler &h)

*installs a callback function that is called each time a record of this event is delivered.*

- [EventHandler](#) \* [GetEventHandler](#) ()

*event handler getter*

#### 4.48.1 Detailed Description

Encapsulates information about the events that the target application generates. For each event holds identificative information (id, name), the place where it is produced, its attributes (for example the parameters of a function) and a reference to a handler. As this is a model class the methods provided are for accessing and setting the members of the data structure.

#### 4.48.2 Constructor & Destructor Documentation

##### 4.48.2.1 Event::Event ( Event const & e )

Copy constructor.

##### Parameters

<i>e</i>	the event to copy
----------	-------------------

##### 4.48.2.2 Event::Event ( int *id*, std::string const & *funcName*, InstrPlace *place* )

Constructor.

##### Parameters

<i>id</i>	unique identification number for the event
<i>funcName</i>	name of the function which the event is associated to
<i>place</i>	place of the function

#### 4.48.3 Member Function Documentation

##### 4.48.3.1 Attribute\* Model::Event::GetAttributes ( ) const `[inline]`

attributes getter.

**Returns**

a collection of attributes to be recorded with this event

**4.48.3.2   EventHandler\* Model::Event::GetEventHandler (   )   [inline]**

event handler getter

**Returns**

event handler

**4.48.3.3   string Model::Event::GetFunctionName (   ) const   [inline]**

name getter.

**Returns**

name of the function this event is associated to

**4.48.3.4   int Model::Event::GetId (   ) const   [inline]**

globally unique event id getter.

**Returns**

event id

**4.48.3.5   InstrPlace Model::Event::GetInstrPlace (   ) const   [inline]**

instruction place getter.

**Returns**

either the function entry or exit

**4.48.3.6   int Model::Event::GetNumAttributes (   ) const   [inline]**

number of attributes getter.

**Returns**

number of event attributes

**4.48.3.7 void Event::SetAttribute ( int *nAttrs*, Attribute \* *attrs* )**

attributes setter.

**Parameters**

<i>nAttrs</i>	number of attributes
<i>attrs</i>	collection of attributes to be recorded with this event

**4.48.3.8 void Event::SetEventHandler ( EventHandler & *h* )**

installs a callback function that is called each time a record of this event is delivered.

**Parameters**

<i>h</i>	event handler
----------	---------------

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

- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

**4.49 Common::Event Class Reference**

Encapsulates information to record an event.

```
#include <Event.h>
```

**Public Member Functions**

- [Event](#) (long64\_t timestamp, int eventId, EventPlace &place, int tid, int param-Count, std::string const &machine)

*Constructor.*

- [~Event](#) ()

*Destructor.*

- long64\_t [GetTimestamp](#) () const

*Returns timestamp.*

- int [GetPlace](#) ()

*Returns place {EventEntry, EventExit}.*

- int [GetEventId](#) () const

*Returns event id.*

- int [GetTid](#) ()  
*Returns tid attribute.*
- int [GetParamCount](#) ()  
*Returns count of the parameters.*
- std::string const & [GetMachine](#) ()  
*Returns name of the machine.*

#### 4.49.1 Detailed Description

Encapsulates information to record an event. This information will be sent to the [Analyzer](#), who will do the actual recording of the event attributes.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2004

#### 4.49.2 Constructor & Destructor Documentation

##### 4.49.2.1 Common::Event::Event ( long64\_t *timestamp*, int *eventId*, EventPlace & *place*, int *tid*, int *paramCount*, std::string const & *machine* ) [inline]

Constructor.

##### Parameters

<i>timestamp</i>	Time stamp when the event was initialized.
<i>eventId</i>	Id of the event.
<i>place</i>	Part on the program where it'll take place. {EventEntry, EventExit}
<i>tid</i>	<a href="#">Task</a> id.
<i>paramCount</i>	Number of parameters.
<i>machine</i>	String representing the machine where the event takes place.

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

- Common/Event.h

## 4.50 EventCollector Class Reference

Processes the incoming event records from the DMLibs. It is based on an active object (thread) that collects incoming ECP events. It stores a moving window of events incoming from different processes using a pool of buffers. The maximum size of this event window can be configured by the tunlets.

```
#include <EventCollector.h>
```

### Public Types

- enum { **DefaultPort** = 5555 }

### Public Member Functions

- [EventCollector](#) (int port=DefaultPort)
- [~EventCollector](#) ()
- void [SetListener](#) ([EventListener](#) \*listener)
- [EventListener](#) \* [GetListener](#) ()
- bool [IsAborted](#) () const

*Determines if the collector is aborted.*

### Protected Member Functions

- void [InitThread](#) ()  
*Not implemented (here for compatibility reasons).*
- void [Run](#) ()  
*Runner of the execution thread, handles events until it dies.*
- void [FlushThread](#) ()  
*Not implemented (here for compatibility reasons).*
- void [Fatal](#) ()  
*Called when an exception is caught in the execution thread.*

### Protected Attributes

- [EventListener](#) \* **\_listener**
- Reactor **\_reactor**
- [ECPAcceptor](#) **\_acceptor**
- bool **\_aborted**

### 4.50.1 Detailed Description

Processes the incoming event records from the DMLibs. It is based on an active object (thread) that collects incoming ECP events. It stores a moving window of events incoming from different processes using a pool of buffers. The maximum size of this event window can be configured by the tunlets.

### 4.50.2 Constructor & Destructor Documentation

#### 4.50.2.1 `EventCollector::EventCollector ( int port = DefaultPort )`

Constructor, starts a execution thread.

##### Parameters

<i>port</i>	acceptor port.
-------------	----------------

#### 4.50.2.2 `EventCollector::~~EventCollector ( )`

Destructor, stops the execution thread.

### 4.50.3 Member Function Documentation

#### 4.50.3.1 `EventListener* EventCollector::GetListener ( ) [inline]`

Getter for the listener.

##### Returns

listener of the event collector.

#### 4.50.3.2 `bool EventCollector::IsAborted ( ) const [inline]`

Determines if the collector is aborted.

##### Returns

the status of the collector.

#### 4.50.3.3 `void EventCollector::SetListener ( EventListener * listener )`

Setter for the listener.

##### Parameters

<i>listener</i>	listener to be set.
-----------------	---------------------

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

- Analyzer/EventCollector.h
- Analyzer/EventCollector.cpp

## 4.51 DMLib::EventCollectorProxy Class Reference

Connects to the analyzer host and sends requests.

```
#include <ECPPProxy.h>
```

### Public Member Functions

- [EventCollectorProxy](#) (std::string const &host, int port)  
*Constructor.*
- [~EventCollectorProxy](#) ()  
*Destructor.*
- void [RegisterLib](#) (int pid, int mpiRank, std::string host, std::string taskName)  
*Sends a request to the [Analyzer](#) to register a new worker.*
- void [SendEvent](#) (EventMsg const &event)  
*Sends a message to the analyzer.*
- void [UnregisterLibrary](#) (int pid)  
*Sends a request to the [Analyzer](#) to unregister a worker.*

### 4.51.1 Detailed Description

Connects to the analyzer host and sends requests.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

- DMLib/ECPPProxy.h
- DMLib/ECPPProxy.cpp

## 4.52 Common::EventDemultiplexer Class Reference

Part of the reactor design pattern, takes requests coming from the reactor and passes them to different handlers.

```
#include <Reactor.h>
```

### Public Member Functions

- [EventDemultiplexer](#) ()  
*Constructor.*
- void [AddHandle](#) (int handle)  
*Adds a new handle.*
- void [RemoveHandle](#) (int handle)  
*Removes selected handle.*
- int [Select](#) ([TimeValue](#) \*timeout=0)  
*Returns number of socket handles ready or 0 if the time limit expired.*
- bool [IsHandleActivated](#) (int handle) const  
*Returns true if the given handle is activated, false otherwise.*
- int [GetMaxHandle](#) () const  
*Returns value of the max handle.*

### 4.52.1 Detailed Description

Part of the reactor design pattern, takes requests coming from the reactor and passes them to different handlers.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002



### 4.52.2 Member Function Documentation

#### 4.52.2.1 int EventDemultiplexer::Select ( TimeValue \* timeout = 0 )

Returns number of socket handles ready or 0 if the time limit expired.

#### Parameters

<i>timeout</i>	If the parameter is a <a href="#">TimeValue</a> object, it will wait the object value for events. In case that the value is 0 it will check without blocking. If the parameter is a 0 (not a <a href="#">TimeValue</a> object, default value) it will check and block in forever loop.
----------------	--

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

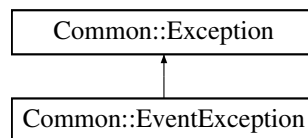
- Common/Reactor.h
- Common/Reactor.cpp

## 4.53 Common::EventException Class Reference

[Event](#), [EventMap](#) and [EventHandler](#) exceptions.

```
#include <EventException.h>
```

Inheritance diagram for Common::EventException:



### Public Member Functions

- [EventException](#) (std::string const &msg, std::string const &objName=std::string())

*Constructor.*

- void [Display](#) (std::ostream &os) const  
*Displays exception message on the given output stream.*
- void [Display](#) () const  
*Displays exception message on the standard error output.*
- std::string [GetReason](#) () const  
*Returns a string containing the error message.*

### 4.53.1 Detailed Description

[Event](#), [EventMap](#) and [EventHandler](#) exceptions.

#### Version

1.0b

#### Since

1.0b

#### Author

Noel De Martin, 2011

### 4.53.2 Constructor & Destructor Documentation

**4.53.2.1** `Common::EventException::EventException ( std::string const & msg, std::string const & objName = std::string () ) [inline]`

Constructor.

#### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>objName</i>	Name of the object causing the exception, "" by default.

### 4.53.3 Member Function Documentation

**4.53.3.1** `void Common::EventException::Display ( std::ostream & os ) const [virtual]`

Displays exception message on the given output stream.

#### Parameters

<i>os</i>	Output stream to display the message.
-----------	---------------------------------------

Reimplemented from [Common::Exception](#).

**4.53.3.2** `string EventException::GetReason ( ) const`

Returns a string containing the error message.

#### Returns

String with the error.

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

- Common/EventException.h
- Common/EventException.cpp

## 4.54 Common::EventHandler Class Reference

Abstract class, processes the requests sent to the reactor.

```
#include <EventHandler.h>
```

### Public Member Functions

- virtual void [HandleInput](#) ()=0  
*Reads the data from the socket and treats it.*
- virtual int [GetHandle](#) ()=0  
*Returns socket descriptor.*

### 4.54.1 Detailed Description

Abstract class, processes the requests sent to the reactor.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

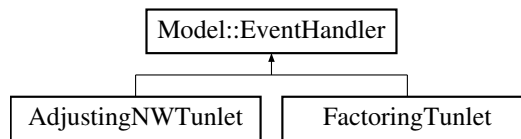
- Common/EventHandler.h

## 4.55 Model::EventHandler Class Reference

Abstract class that holds a method to manage event records.

```
#include <AppEvent.h>
```

Inheritance diagram for Model::EventHandler:



### Public Member Functions

- virtual void [HandleEvent](#) ([EventRecord](#) const &r)=0  
*handles an event record (virtual).*

#### 4.55.1 Detailed Description

Abstract class that holds a method to manage event records.

#### 4.55.2 Member Function Documentation

**4.55.2.1** virtual void [Model::EventHandler::HandleEvent](#) ( [EventRecord](#) const & *r* ) [pure virtual]

handles an event record (virtual).

#### Parameters

<i>r</i>	event record to be handled
----------	----------------------------

Implemented in [AdjustingNWTunlet](#), and [FactoringTunlet](#).

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

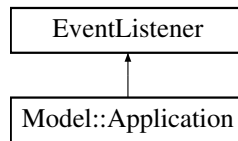
- Analyzer/AppEvent.h

## 4.56 EventListener Class Reference

Provides an interface for event listeners, which consist in methods to respond to events and errors.

```
#include <EventCollector.h>
```

Inheritance diagram for EventListener:



### Public Member Functions

- virtual void [OnEvent](#) (ECPMessage \*msg)=0  
*function which is triggered when an event happens.*
- virtual void [OnFatalError](#) ()=0  
*function which is triggered when a fatal error happens.*

#### 4.56.1 Detailed Description

Provides an interface for event listeners, which consist in methods to respond to events and errors.

#### 4.56.2 Member Function Documentation

4.56.2.1 virtual void EventListener::OnEvent ( ECPMessage \* *msg* ) [pure virtual]

function which is triggered when an event happens.

#### Parameters

<i>msg</i>	message that contains the event data.
------------	---------------------------------------

Implemented in [Model::Application](#).

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

- Analyzer/EventManager.h

## 4.57 Common::EventManager Class Reference

Contains and manages a collection of [Event](#) objects, TODO.

```
#include <EventManager.h>
```

### Public Member Functions

- [EventManager](#) ()

*Constructor:*

- void [Add](#) (std::string const &name, int id)  
*Adds a new event into the map.*
- int [GetId](#) (std::string const &name) const  
*Returns id of the given event.*
- int [GetSize](#) () const  
*Returns map size.*

#### 4.57.1 Detailed Description

Contains and manages a collection of [Event](#) objects, TODO.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2001

#### 4.57.2 Member Function Documentation

##### 4.57.2.1 void Common::EventMap::Add ( std::string const & name, int id )

Adds a new event into the map.

##### Exceptions

<a href="#">EventException</a>
--------------------------------

##### 4.57.2.2 int EventMap::GetId ( std::string const & name ) const

Returns id of the given event.

##### Exceptions

<a href="#">EventException</a>
--------------------------------

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

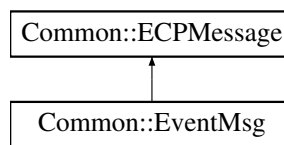
- Common/EventMap.h
- Common/EventMap.cpp

## 4.58 Common::EventMsg Class Reference

Encapsulates a message generated by DMLib to trace events.

```
#include <ECPMsg.h>
```

Inheritance diagram for Common::EventMsg:



### Public Member Functions

- [EventMsg](#) ()  
*Constructor.*
- [~EventMsg](#) ()  
*Destructor.*
- void [Reset](#) (long\_t timestamp, int eventId, InstrPlace place, int paramCount)  
*Sets the message to the indicated state.*
- void [SetTid](#) (int tid)  
*Sets the task id.*
- ECPMsgType [GetType](#) () const  
*Returns the type of event.*
- void [SetParams](#) (char const \*buffer, int size)  
*Sets the buffer to be used and indicates its size.*
- void [SetBuffer](#) (char \*buffer)  
*sets the parameters buffer.*
- int [GetParamBufSize](#) () const  
*Returns buffer size.*
- const char \* [GetParamBuffer](#) () const  
*Returns a pointer to the content of the buffer.*

- `long_t GetTimestamp () const`  
*Returns timestamp.*
- `int GetPlace () const`  
*Returns place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.*
- `int GetEventId () const`  
*Returns event ID.*
- `int GetParamCount () const`  
*Returns parameters count.*
- `int GetDataSize () const`  
*Returns size of the data serialized.*
- `void Serialize (Serializer &out) const`  
*Serializes the message with the given [Serializer](#).*
- `void DeSerialize (DeSerializer &in)`  
*Deserializes the message with the given [DeSerializer](#).*
- `int GetTid () const`  
*Returns the task id.*

#### 4.58.1 Detailed Description

Encapsulates a message generated by DMLib to trace events. The message indicates what information should be gathered of certain event, this messages are created with an EventMsgWriter object.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002



### 4.58.2 Member Function Documentation

4.58.2.1 **void EventMsg::Reset ( long\_t *timestamp*, int *eventId*, InstrPlace *place*, int *paramCount* )**

Sets the message to the indicated state.

#### Parameters

<i>timestamp</i>	Timestamp when the event occurs.
<i>eventId</i>	Id of the event.
<i>place</i>	Place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.
<i>paramCount</i>	Number of parameters.

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

- Common/ECPMsg.h
- Common/ECPMsg.cpp

## 4.59 EventMsgReader Class Reference

### Public Member Functions

- **EventMsgReader** (EventMsg const &msg)
- int [GetParamCount](#) () const  
*getter of ParamCount.*
- AttrValueType [GetAttrType](#) ()  
*getter of AttrType.*
- int [GetIntValue](#) ()  
*get an integer from the stream.*
- float [GetFloatValue](#) ()  
*get a float from the stream.*
- double [GetDoubleValue](#) ()  
*get a double from the stream.*
- char [GetCharValue](#) ()  
*get a character from the stream.*
- short [GetShortValue](#) ()  
*get a short from the stream.*
- std::string [GetStringValue](#) ()

*get a string from the stream.*

- void [DumpValues](#) ()

*Gets the value of each ECP event parameter. For each parameter checks the type and use the proper getter.*

#### 4.59.1 Member Function Documentation

##### 4.59.1.1 AttrValueType EventMsgReader::GetAttrType ( ) [inline]

getter of AttrType.

##### Returns

type of the attribute.

##### 4.59.1.2 char EventMsgReader::GetCharValue ( ) [inline]

get a character from the stream.

##### Returns

character value.

##### 4.59.1.3 double EventMsgReader::GetDoubleValue ( ) [inline]

get a double from the stream.

##### Returns

double value.

##### 4.59.1.4 float EventMsgReader::GetFloatValue ( ) [inline]

get a float from the stream.

##### Returns

float value.

##### 4.59.1.5 int EventMsgReader::GetIntValue ( ) [inline]

get an integer from the stream.

##### Returns

integer value.

**4.59.1.6** `int EventMsgReader::GetParamCount ( ) const [inline]`

getter of ParamCount.

#### Returns

number of parameters.

**4.59.1.7** `short EventMsgReader::GetShortValue ( ) [inline]`

get a short from the stream.

#### Returns

short value.

**4.59.1.8** `std::string EventMsgReader::GetStringValue ( ) [inline]`

get a string from the stream.

#### Returns

string value.

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

- Analyzer/EventMsgReader.h
- Analyzer/EventMsgReader.cpp

## 4.60 DMLib::EventMsgWriter Class Reference

Creates EventMsg objects.

```
#include <EventMsgWriter.h>
```

### Public Member Functions

- [EventMsgWriter \(\)](#)  
*Constructor.*
- [~EventMsgWriter \(\)](#)  
*Destructor.*
- void [OpenEvent](#) (long\_t timestamp, int eventId, InstrPlace place, int param-Count)

*Open the event and sets its specifications.*

- void [AddIntParam](#) (int value)  
*Adds an integer parameter to the event.*
- void [AddFloatParam](#) (float value)  
*Adds a float parameter to the event.*
- void [AddDoubleParam](#) (double value)  
*Adds a double parameter to the event.*
- void [AddCharParam](#) (char c)  
*Adds a char parameter to the event.*
- void [AddStringParam](#) (std::string const &s)  
*Adds a string parameter to the event.*
- EventMsg const & [CloseEvent](#) ()  
*Closes the event and returns the object.*

#### 4.60.1 Detailed Description

Creates EventMsg objects. Loads the specifications of an EventMsg object and prepares it. Once it's been prepared it returns the object using the CloseEvent method.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2001

#### 4.60.2 Member Function Documentation

**4.60.2.1** void EventMsgWriter::OpenEvent ( long\_t *timestamp*, int *eventId*, InstrPlace *place*, int *paramCount* )

Open the event and sets its specifications.

##### Parameters

<i>timestamp</i>	Timestamp when the event occurs.
<i>eventId</i>	Id of the event.
<i>place</i>	Place where the event is located {instrUnknown, ipFuncEntry, ipFuncExit}.
<i>paramCount</i>	Number of parameters. Generated on Sun Sep 18 2011 23:49:55 for MATE by Doxygen

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

- DMLib/EventMsgWriter.h
- DMLib/EventMsgWriter.cpp

## 4.61 Model::EventRecord Class Reference

particular instance of the event abstraction. Holds information about the kind of event, the task that produced, the message sent and the values it contained. On the one hand it provides methods to get/set the information above, on the other hand, it provides methods to parse messages and get the information that they contain.

```
#include <AppEvent.h>
```

### Public Member Functions

- `int GetEventId () const`  
*id getter.*
- `Event const & GetEvent () const`  
*associated event getter.*
- `long_t GetTimestamp () const`  
*time stamp getter.*
- `Task & GetTask () const`  
*task getter.*
- `AttributeValue * GetAttributeValues () const`  
*values getter.*
- `AttributeValue const & GetAttributeValue (int index) const`  
*Gets the i-th attribute from the list of values.*

### Protected Member Functions

- `EventRecord (Event const &e, Task &t, EventMsg const &msg)`  
*Constructor.*
- `void ParseAttrs (EventMsg const &msg)`  
*Reads from the message and sets the value of the attributes depending on their type.*

## Friends

- class [Task](#)

### 4.61.1 Detailed Description

particular instance of the event abstraction. Holds information about the kind of event, the task that produced, the message sent and the values it contained. On the one hand it provides methods to get/set the information above, on the other hand, it provides methods to parse messages and get the information that they contain.

### 4.61.2 Constructor & Destructor Documentation

#### 4.61.2.1 `EventRecord::EventRecord ( Event const & e, Task & t, EventMsg const & msg )` [protected]

Constructor.

#### Parameters

<i>e</i>	event object this record is associated to
<i>t</i>	task object which produces the event
<i>msg</i>	message produced by the event

### 4.61.3 Member Function Documentation

#### 4.61.3.1 `AttributeValue const& Model::EventRecord::GetAttributeValue ( int index ) const` [inline]

Gets the i-th attribute from the list of values.

#### Parameters

<i>index</i>	position of the attribute from which we want the value
--------------	--

#### Returns

the recorded value for the i-th attribute

#### 4.61.3.2 `AttributeValue* Model::EventRecord::GetAttributeValues ( ) const` [inline]

values getter.

#### Returns

a collection of recorded attribute values

**4.61.3.3** Event const& Model::EventRecord::GetEvent ( ) const [inline]

associated event getter.

**Returns**

event object this record is associated to

**4.61.3.4** int Model::EventRecord::GetEventId ( ) const [inline]

id getter.

**Returns**

globally unique event id

**4.61.3.5** Task& Model::EventRecord::GetTask ( ) const [inline]

task getter.

**Returns**

the task that generated this event

**4.61.3.6** long\_t Model::EventRecord::GetTimestamp ( ) const [inline]

time stamp getter.

**Returns**

time stamp that indicates when the event happened

**4.61.3.7** void EventRecord::ParseAttrs ( EventMsg const & msg ) [protected]

Reads from the message and sets the value of the attributes depending on their type.

**Parameters**

<i>msg</i>	reference to the msg to be read.
------------	----------------------------------

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

- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

## 4.62 Model::Events Class Reference

encapsulates information to create and manage events lists. Uses a data structure based on a vector to keep data and a map to retrieve it. Provides methods to add, remove and find elements in the list.

```
#include <AppEvent.h>
```

### Public Member Functions

- [Events](#) ()  
*Constructor.*
- void [Add](#) ([Event](#) const &e)  
*Maps and adds an event to the events list.*
- bool [Remove](#) (int eventId, InstrPlace place)  
*Removes an event from the events list.*
- [Event](#) \* [Find](#) (int eventId, InstrPlace place)  
*Searches for an event in the event list.*
- int [Size](#) () const  
*size getter.*

### 4.62.1 Detailed Description

encapsulates information to create and manage events lists. Uses a data structure based on a vector to keep data and a map to retrieve it. Provides methods to add, remove and find elements in the list.

### 4.62.2 Member Function Documentation

#### 4.62.2.1 void Events::Add ( Event const & e )

Maps and adds an event to the events list.

#### Parameters

<i>e</i>	the event to be added
----------	-----------------------

#### 4.62.2.2 Event \* Events::Find ( int eventId, InstrPlace place )

Searches for an event in the event list.



**Parameters**

<i>eventId</i>	unique Id of the event.
<i>place</i>	instruction where the event is placed

**Returns**

a reference to the found event or NULL if not found

**4.62.2.3 bool Events::Remove ( int eventId, InstrPlace place )**

Removes an event from the events list.

**Parameters**

<i>eventId</i>	unique Id of the event
<i>place</i>	instruction where the event is placed

**Returns**

true if found & removed, false otherwise

**4.62.2.4 int Events::Size ( ) const**

size getter.

**Returns**

number of events

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

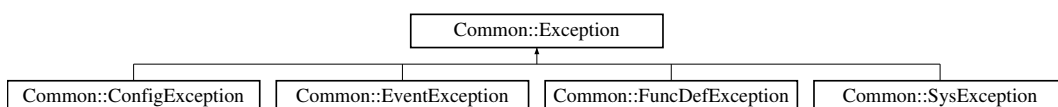
- Analyzer/AppEvent.h
- Analyzer/AppEvent.cpp

## 4.63 Common::Exception Class Reference

Abstract class, stores information of errors on determined situations.

```
#include <Exception.h>
```

Inheritance diagram for Common::Exception:



### Public Member Functions

- [Exception](#) (std::string const &msg, std::string const &objName=std::string(), long err=0)  
*Constructor.*
- [Exception](#) ()  
*Constructor.*
- virtual [~Exception](#) ()  
*Destructor.*
- long [GetError](#) () const  
*Returns error code.*
- std::string const & [GetErrorMessage](#) () const  
*Returns error message.*
- std::string const & [GetObjectName](#) () const  
*Returns the name of the object.*
- virtual void [Display](#) () const  
*Displays exception message on the standard error output.*
- virtual void [Display](#) (std::ostream &os) const  
*Displays exception message on the given output stream.*

### Protected Attributes

- long **\_err**
- std::string **\_msg**
- std::string **\_objName**

#### 4.63.1 Detailed Description

Abstract class, stores information of errors on determined situations.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

### 4.63.2 Constructor & Destructor Documentation

**4.63.2.1** `Common::Exception::Exception ( std::string const & msg, std::string const & objName = std::string (), long err = 0 ) [inline]`

Constructor.

#### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>objName</i>	Name of the object causing the exception, "" by default.

### 4.63.3 Member Function Documentation

**4.63.3.1** `virtual void Common::Exception::Display ( std::ostream & os ) const [virtual]`

Displays exception message on the given output stream.

#### Parameters

<i>os</i>	Output stream to display the message.
-----------	---------------------------------------

Reimplemented in [Common::ConfigException](#), [Common::EventException](#), [Common::FuncDefException](#), and [Common::SysException](#).

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

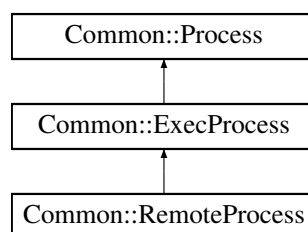
- Common/Exception.h
- Common/Exception.cpp

## 4.64 Common::ExecProcess Class Reference

Executes a program as a child of the current process.

```
#include <Process.h>
```

Inheritance diagram for Common::ExecProcess:



### Public Types

- enum **Status** {  
    **stOutReady**, **stOutEof**, **stErrReady**, **stErrEof**,  
    **stTimeout** }

### Public Member Functions

- [ExecProcess](#) (std::string const &programPath, char \*const argv[])  
*Constructor.*
- void [Start](#) ()  
*Executes the program.*
- Status [WaitForEvent](#) (char \*buffer, int bufSize, int &bytesRead, [TimeValue](#) \*timeout=0)

*Waits until an event is placed on any of the outputs of the process or the time limit is reached.*

### Protected Member Functions

- [ExecProcess](#) ()  
*Constructor.*
- int [Run](#) ()  
*Executes de process.*

#### 4.64.1 Detailed Description

Executes a program as a child of the current process.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2001

### 4.64.2 Constructor & Destructor Documentation

#### 4.64.2.1 Common::ExecProcess::ExecProcess ( std::string const & *programPath*, char \*const *argv*[] ) [inline]

Constructor.

Example usage:

```
char * argv [] = { "/usr/bin/vi", "param1", "param2", 0 };
ExecProcess p ("/usr/bin/vi", argv);
```

Notes:

- first element of argv must be program path
- last element of argv table must be 0

#### Parameters

<i>program-Path</i>	Path of the program to execute.
<i>argv</i>	Arguments to pass to the execution of the program.

### 4.64.3 Member Function Documentation

#### 4.64.3.1 void ExecProcess::Start ( ) [virtual]

Executes the program.

The standard outputs and inputs of the program will be redirected to internal pipes, to be handled on the [WaitForEvent\(\)](#) method.

Reimplemented from [Common::Process](#).

#### 4.64.3.2 ExecProcess::Status ExecProcess::WaitForEvent ( char \* *buffer*, int *bufSize*, int & *bytesRead*, TimeValue \* *timeout* = 0 )

Waits until an event is placed on any of the outputs of the process or the time limit is reached.

If any event was sent by the process, it is placed on the buffer.

#### Returns

Information about how the function ended and what was placed on the buffer.

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

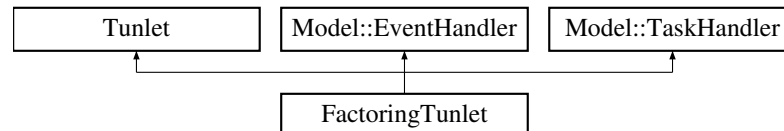
- Common/Process.h
- Common/Process.cpp

## 4.65 FactoringTunlet Class Reference

Factoring optimization tunlet for m/w apps.

```
#include <FactoringTunlet_nw.h>
```

Inheritance diagram for FactoringTunlet:



### Public Member Functions

- void [Initialize](#) ([Model::Application](#) &app)
- void **BeforeAppStart** ()
- void **Destroy** ()
- void [HandleEvent](#) ([Model::EventRecord](#) const &r)  
*handles all incoming events*
- void [TaskStarted](#) ([Model::Task](#) &t)
- void [TaskTerminated](#) ([Model::Task](#) &t)

### 4.65.1 Detailed Description

Factoring optimization tunlet for m/w apps.

### 4.65.2 Member Function Documentation

**4.65.2.1** void [FactoringTunlet::HandleEvent](#) ( [Model::EventRecord](#) const & *r* )  
[virtual]

handles all incoming events

#### Parameters

<i>r</i>	
----------	--

Implements [Model::EventHandler](#).

**4.65.2.2** void [FactoringTunlet::Initialize](#) ( [Model::Application](#) & *app* ) [virtual]

#### Parameters

<i>app</i>	
------------	--

Implements [Tunlet](#).

**4.65.2.3** void FactoringTunlet::TaskStarted ( Model::Task & *t* ) [virtual]

#### Parameters

<i>t</i>	
----------	--

Implements [Model::TaskHandler](#).

**4.65.2.4** void FactoringTunlet::TaskTerminated ( Model::Task & *t* ) [virtual]

#### Parameters

<i>t</i>	
----------	--

Implements [Model::TaskHandler](#).

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

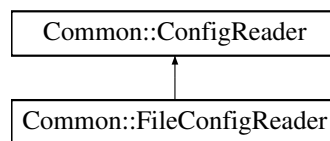
- Analyzer/FactoringTunlet\_nw.h
- Analyzer/FactoringTunlet\_nw.cpp

## 4.66 Common::FileConfigReader Class Reference

Parses the content of a file into a [Config](#) object.

```
#include <ConfigReader.h>
```

Inheritance diagram for Common::FileConfigReader:



### Public Member Functions

- [FileConfigReader](#) (std::string const &fileName)  
*Constructor.*
- [Config Read](#) ()  
*Parses the configuration of the file into a [Config](#) object.*

### 4.66.1 Detailed Description

Parses the content of a file into a [Config](#) object. Extends [ConfigReader](#)

#### Version

1.0b

#### Since

1.0b

#### Author

Noel De Martin, 2011

### 4.66.2 Constructor & Destructor Documentation

#### 4.66.2.1 `Common::FileConfigReader::FileConfigReader ( std::string const & fileName )` [inline]

Constructor.

#### Parameters

<i>fileName</i>	Path of the file to read.
-----------------	---------------------------

#### Exceptions

<a href="#">ConfigException</a>	
---------------------------------	--

### 4.66.3 Member Function Documentation

#### 4.66.3.1 `Config FileConfigReader::Read ( )` [virtual]

Parses the configuration of the file into a [Config](#) object.

#### Exceptions

<a href="#">ConfigException</a>	
---------------------------------	--

Implements [Common::ConfigReader](#).

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

- Common/ConfigReader.h
- Common/ConfigReader.cpp

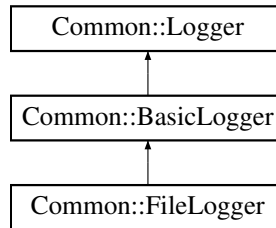


## 4.67 Common::FileLogger Class Reference

Stores information of interest into a file.

```
#include <Syslog.h>
```

Inheritance diagram for Common::FileLogger:



### Public Member Functions

- [FileLogger](#) (std::string const &filepath, bool append=false)  
*Constructor.*
- [~FileLogger](#) ()  
*Destructor.*
- void [Log](#) ([LogEntry](#) const &entry)  
*Inserts a new entry to the log.*

#### 4.67.1 Detailed Description

Stores information of interest into a file.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.67.2 Constructor & Destructor Documentation

##### 4.67.2.1 FileLogger::FileLogger ( std::string const & filepath, bool append = false )

Constructor.

**Parameters**

<i>filepath</i>	Path of the file where the log will be stored. append Flag that determines if the file will be overwritten or the logs will be appended, default false.
-----------------	---

**Exceptions**

<a href="#"><i>SysException</i></a>
-------------------------------------

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

- Common/Syslog.h
- Common/Syslog.cpp

**4.68 Common::FuncDef Class Reference**

Represents definition of the function to be traced.

```
#include <FuncDefs.h>
```

**Public Member Functions**

- [FuncDef](#) (std::string const &name, std::string const &paramFormat, int paramCount, int funcId)

*Constructor.*

- std::string const & [GetName](#) () const

*Returns name of the function.*

- std::string const & [GetParamFormat](#) () const

*Returns format of the parameters.*

- int [GetParamCount](#) () const

*Returns number of parameters used by the function.*

- int [GetFuncId](#) () const

*Returns Id of the function.*

**4.68.1 Detailed Description**

Represents definition of the function to be traced.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2003

**4.68.2 Constructor & Destructor Documentation****4.68.2.1 Common::FuncDef::FuncDef ( std::string const & *name*, std::string const & *paramFormat*, int *paramCount*, int *funcId* ) [inline]**

Constructor.

**Parameters**

<i>name</i>	Name of the function
<i>paramFormat</i>	String denoting types of the parameters. S: String, I: Integer, P: Pointer.
<i>paramCount</i>	Number of parameters.
<i>funcId</i>	Function Id.

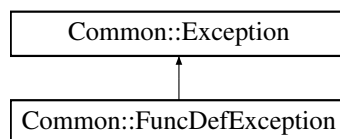
The documentation for this class was generated from the following file:

- Common/FuncDefs.h

**4.69 Common::FuncDefException Class Reference**[FuncDef](#) exceptions.

#include &lt;FuncDefException.h&gt;

Inheritance diagram for Common::FuncDefException:

**Public Member Functions**

- [FuncDefException](#) (std::string const &msg, std::string const &objName=std::string())

*Constructor.*

- void [Display](#) (std::ostream &os) const

*Displays exception message on the given output stream.*

- void [Display](#) () const

*Displays exception message on the standard error output.*

- std::string [GetReason](#) () const

*Returns a string containing the error message.*

#### 4.69.1 Detailed Description

[FuncDef](#) exceptions.

##### Version

1.0b

##### Since

1.0b

##### Author

Noel De Martin, 2011

#### 4.69.2 Constructor & Destructor Documentation

**4.69.2.1** `Common::FuncDefException::FuncDefException ( std::string const & msg, std::string const & objName = std::string () ) [inline]`

Constructor.

##### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>objName</i>	Name of the object causing the exception, "" by default.

#### 4.69.3 Member Function Documentation

**4.69.3.1** `void FuncDefException::Display ( std::ostream & os ) const [virtual]`

Displays exception message on the given output stream.

##### Parameters

<i>os</i>	Output stream to display the message.
-----------	---------------------------------------

Reimplemented from [Common::Exception](#).

#### 4.69.3.2 string FuncDefException::GetReason ( ) const

Returns a string containing the error message.

##### Returns

String with the error.

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

- Common/FuncDefException.h
- Common/FuncDefException.cpp

## 4.70 Common::FuncDefs Class Reference

Creates and stores objects of the [FuncDef](#) class.

```
#include <FuncDefs.h>
```

### Public Member Functions

- [FuncDefs](#) ()  
*Constructor.*
- void [Add](#) (std::string const &funcName, std::string const &paramFormat, int paramCount, int funcId)  
*Adds a [FuncDef](#) object.*
- [FuncDef](#) const & [Find](#) (std::string const &name)  
*Returns a [FuncDef](#) object with the given name.*
- int [GetSize](#) () const  
*Returns number of [FuncDef](#) objects stored.*

### 4.70.1 Detailed Description

Creates and stores objects of the [FuncDef](#) class.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2003

## 4.70.2 Constructor & Destructor Documentation

### 4.70.2.1 FuncDefs::FuncDefs ( )

Constructor.

#### Exceptions

<a href="#"><i>FuncDefException</i></a>
---

## 4.70.3 Member Function Documentation

### 4.70.3.1 void Common::FuncDefs::Add ( std::string const & *funcName*, std::string const & *paramFormat*, int *paramCount*, int *funcId* )

Adds a [\*FuncDef\*](#) object.

Uses the default constructor of [\*FuncDef\*](#) with the given parameters.

#### Exceptions

<a href="#"><i>FuncDefException</i></a>
---

### 4.70.3.2 FuncDef const & FuncDefs::Find ( std::string const & *name* )

Returns a [\*FuncDef\*](#) object with the given name.

#### Exceptions

<a href="#"><i>FuncDefException</i></a>
---

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

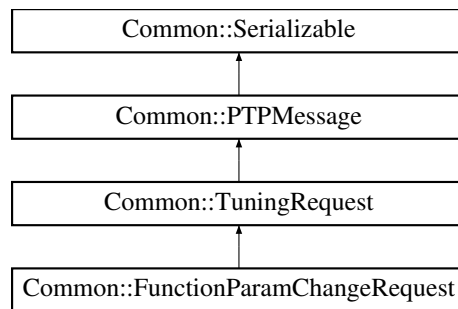
- Common/FuncDefs.h
- Common/FuncDefs.cpp

## 4.71 Common::FunctionParamChangeRequest Class Reference

Encapsulates a tuning request to set the value of an input parameter of a given function in a given application process.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::FunctionParamChangeRequest:



### Public Member Functions

- [FunctionParamChangeRequest](#) (int pid=0, std::string const &funcName=std::string(), int paramIdx=0, int newValue=0, int \*requiredOldValue=0, [Breakpoint](#) \*brkpt=0)

*Constructor.*

- PTPMsgType [GetType](#) () const  
*Returns type of message (PTPFuncParamChange).*
- std::string const & [GetFuncName](#) () const  
*Returns name of the function.*
- int [GetParamIdx](#) () const  
*Returns index of the parameter to change on the attributes array.*
- int [GetNewValue](#) () const  
*Returns the new value to replace on the function call.*
- int const \* [GetReqOldValue](#) () const  
*Returns the value the parameter should have for the tuning to be performed.*
- void [Serialize](#) (Serializer &out) const  
*Sends the message.*
- void [DeSerialize](#) (DeSerializer &in)  
*Receives the message.*

#### 4.71.1 Detailed Description

Encapsulates a tuning request to set the value of an input parameter of a given function in a given application process. This parameter value is modified before the function body is invoked. It's also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is

changed to a new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

### Version

1.0b

### Since

1.0b

### Author

Ania Morajko, 2003

## 4.71.2 Constructor & Destructor Documentation

**4.71.2.1** `Common::FunctionParamChangeRequest::FunctionParamChangeRequest ( int pid = 0, std::string const & funcName = std::string(), int paramIdx = 0, int newValue = 0, int * requiredOldValue = 0, Breakpoint * brkpt = 0 ) [inline]`

Constructor.

### Parameters

<i>pid</i>	Id of the process where the parameter will be changed, default 0.
<i>funcName</i>	Name of the function call to modify, default "".
<i>paramIdx</i>	Parameter index inside the attributes array, default 0.
<i>newValue</i>	New Value to set, default 0.
<i>requiredOldValue</i>	Current value the parameter should have to perform the tuning. If the value doesn't match this one the tuning won't be performed. If this value is 0, the tuning will be performed without checking the old value, default 0.
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.72 Common::HandlerMap Class Reference

Contains and manages a collection of [EventHandler](#) objects.

```
#include <Reactor.h>
```

### Public Member Functions

- [HandlerMap](#) ()



*Constructor.*

- void [Add](#) (int handle, [EventHandler](#) \*handler)  
*Adds the handler to the map.*
- [EventHandler](#) \* [Get](#) (int handle)  
*Returns the [EventHandler](#) object stored with the given handle.*
- int [GetSize](#) () const  
*Returns map size.*

#### 4.72.1 Detailed Description

Contains and manages a collection of [EventHandler](#) objects.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/Reactor.h
- Common/ConfigMap.cpp
- Common/EventMap.cpp
- Common/FuncDefs.cpp
- Common/Reactor.cpp

## 4.73 Model::Host Class Reference

Encapsulates host information. Basically consists in a string with the name of the host and a method to access it.

```
#include <Host.h>
```

### Public Member Functions

- string [GetName](#) () const

### Protected Member Functions

- [Host](#) (string const &name)  
*Constructor.*

### Friends

- class **Application**

#### 4.73.1 Detailed Description

Encapsulates host information. Basically consists in a string with the name of the host and a method to access it.

#### 4.73.2 Member Function Documentation

4.73.2.1 `string Model::Host::GetName ( ) const [inline]`

##### Returns

name of the host

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

- Analyzer/Host.h

### 4.74 Model::HostHandler Class Reference

Provides mechanisms to handle the addition and the remove of hosts.

```
#include <Host.h>
```

#### Public Member Functions

- virtual void [HostAdded](#) ([Host](#) &h)=0  
*called when a new host is added to the virtual machine.*
- virtual void [HostRemoved](#) ([Host](#) &h)=0  
*called when a host is removed from the virtual machine.*

#### 4.74.1 Detailed Description

Provides mechanisms to handle the addition and the remove of hosts.

### 4.74.2 Member Function Documentation

**4.74.2.1** `virtual void Model::HostHandler::HostAdded ( Host & h )` [pure virtual]

called when a new host is added to the virtual machine.

#### Parameters

<i>h</i>	added host.
----------	-------------

**4.74.2.2** `virtual void Model::HostHandler::HostRemoved ( Host & h )` [pure virtual]

called when a host is removed from the virtual machine.

#### Parameters

<i>h</i>	removed host.
----------	---------------

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

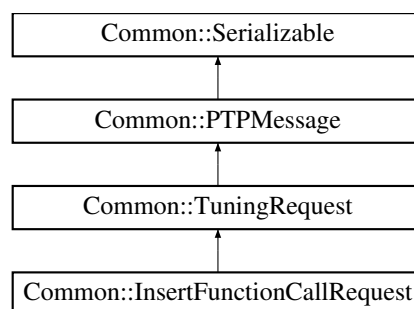
- Analyzer/Host.h

## 4.75 Common::InsertFunctionCallRequest Class Reference

Encapsulates a tuning request to insert a new function invocation code with a specified attributes at a given location in an application process.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::InsertFunctionCallRequest:



### Public Member Functions

- [InsertFunctionCallRequest](#) (int pid=0, std::string const &funcName=std::string(), int nAttrs=0, [Attribute](#) \*attrs=0, std::string const &destFunc=std::string(), InstrPlace place=ipFuncEntry, [Breakpoint](#) \*brkpt=0)

*Constructor.*

- [~InsertFunctionCallRequest \(\)](#)

*Destructor.*

- PTPMsgType [GetType \(\)](#) const

*Returns type of message (PTPInsertFuncCall).*

- std::string const & [GetFuncName \(\)](#) const

*Returns name of the function to add.*

- int [GetAttrCount \(\)](#) const

*Returns number of attributes the function has.*

- Attribute \* [GetAttributes \(\)](#) const

*Returns array of attributes.*

- std::string const & [GetDestFunc \(\)](#) const

*Returns name of the function where the call will be added.*

- InstrPlace [GetInstrPlace \(\)](#) const

*Returns the place where the call will be added.*

- void [Serialize \(Serializer &out\)](#) const

*Sends the message.*

- void [DeSerialize \(DeSerializer &in\)](#)

*Receives the message.*

#### 4.75.1 Detailed Description

Encapsulates a tuning request to insert a new function invocation code with a specified attributes at a given location in an application process.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003

### 4.75.2 Constructor & Destructor Documentation

**4.75.2.1** `Common::InsertFunctionCallRequest::InsertFunctionCallRequest ( int pid = 0, std::string const & funcName = std::string(), int nAttrs = 0, Attribute * attrs = 0, std::string const & destFunc = std::string(), InstrPlace place = ipFuncEntry, Breakpoint * brkpt = 0 ) [inline]`

Constructor.

#### Parameters

<i>pid</i>	Id of the process where the call will be inserted, default 0.
<i>funcName</i>	Name of the function to call, default "".
<i>nAttrs</i>	Number of attributes the function has, default 0.
<i>attrs</i>	<a href="#">Attribute</a> array, default 0.
<i>destFunc</i>	Function where the call will be inserted, default "".
<i>place</i>	Place where the call will be added, default ipFuncEntry.
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.76 InstrGroup Class Reference

Contains a group of snippets to be inserted in a function.

```
#include <InstrSet.h>
```

### Public Types

- typedef vector< [SnippetHandler](#) \* >::iterator **Iterator**

### Public Member Functions

- [InstrGroup](#) (int eventId, std::string const &funcName)  
*Constructor.*
- [~InstrGroup](#) ()  
*Destructor.*
- int [GetEventId](#) () const  
*Getter for the variable \_eventId.*

- int [GetSize](#) () const  
*Getter of the size of `_vector`.*
- bool [IsEmpty](#) () const  
*Checks if `_vector` is empty.*
- std::string const & [GetFuncName](#) () const  
*Getter of the name of the function.*
- void [AddHandler](#) (InstrPlace place, BPatchSnippetHandle \*handle)  
*Add the handler passed as a parameter to the `_vector`.*
- void [RemoveHandler](#) (InstrPlace place)  
*Eliminates the handlers from the vector to be inserted in the place passed as a parameter.*
- Iterator [begin](#) ()  
*Getter for an iterator pointing to the first element in the [InstrGroup](#).*
- Iterator [end](#) ()  
*Getter for an iterator pointing to the last instruction (handler) on the group.*

#### 4.76.1 Detailed Description

Contains a group of snippets to be inserted in a function.

##### Version

1.0

##### Since

1.0

##### Author

Ania Morajko, 2002

#### 4.76.2 Member Function Documentation

##### 4.76.2.1 void InstrGroup::AddHandler ( InstrPlace *place*, BPatchSnippetHandle \* *handle* )

Add the handler passed as a parameter to the `_vector`.

##### Parameters

<i>place</i>	Object that represents the place in the program in which the snippet will be inserted.
<i>handle</i>	Object of the class BPatchSnippetHandle that handles a dyninst snippet.

**4.76.2.2** Iterator InstrGroup::begin ( ) [inline]

Getter for an iterator pointing to the first element in the [InstrGroup](#).

**Returns**

Iterator for the variable `_vector` that points to its beginning.

**4.76.2.3** Iterator InstrGroup::end ( ) [inline]

Getter for an iterator pointing to the last instruction (handler) on the group.

**Returns**

Iterator for the variable `_vector` that points to its final element.

**4.76.2.4** int InstrGroup::GetEventId ( ) const [inline]

Getter for the variable `_eventId`.

**Returns**

Id of the event.

**4.76.2.5** std::string const& InstrGroup::GetFuncName ( ) const [inline]

Getter of the name of the function.

**Returns**

String that contains the name of the function.

**4.76.2.6** int InstrGroup::GetSize ( ) const [inline]

Getter of the size of `_vector`.

**Returns**

Size of the vector `_vector`.

**4.76.2.7** bool InstrGroup::IsEmpty ( ) const [inline]

Checks if `_vector` is empty.

**Returns**

0 if not empty, 1 if empty.

#### 4.76.2.8 void InstrGroup::RemoveHandler ( InstrPlace *place* )

Eliminates the handlers from the vector to be inserted in the place passed as a parameter.

##### Parameters

<i>place</i>	Object that represents the place in the program in which the snippet will be inserted..
--------------	---

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

- AC/InstrSet.h
- AC/InstrSet.cpp

## 4.77 Common::ConfigMap::Iterator Class Reference

Iterates over a [ConfigMap](#) object.

```
#include <ConfigMap.h>
```

### Public Member Functions

- [Iterator](#) ([ConfigMap](#) const &map)  
*Constructor.*
- bool [AtEnd](#) () const  
*Indicates whether the iterator is pointing to the end of the map or not.*
- void [Next](#) ()  
*The pointer increases a position on the map.*
- std::string [GetSection](#) () const  
*Returns section of the current position.*
- std::string [GetKey](#) () const  
*Returns key of the current position.*
- std::string const & [GetValue](#) () const  
*Returns value of the current position.*

#### 4.77.1 Detailed Description

Iterates over a [ConfigMap](#) object.



**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2000

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

- Common/ConfigMap.h
- Common/ConfigMap.cpp

## 4.78 IterData Class Reference

statistics for a single iteration

```
#include <FactoringStats_nw.h>
```

**Public Member Functions**

- **IterData** (int iterIdx)
- void **OnIterStart** (long\_t time, int numTuples, int sizeBytes, int nw)
- void **OnIterEnd** (long\_t time)
- void **OnNewBatch** ()
- **BatchData** & **GetBatchData** (int IdxBatch)
- bool **IsComplete** () const
- bool **AreBatchsComplete** () const
- int **GetTupleSizeInBytes** () const
- **BatchData** \*\* **AllocBatchsArray** ()
- int **GetNumWorkers** ()
- int **GetTotalTasks** () const
- int **GetNumBatchs** () const

### 4.78.1 Detailed Description

statistics for a single iteration

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

- Analyzer/FactoringStats\_nw.h
- Analyzer/FactoringStats\_nw.cpp

## 4.79 Common::Config::KeyIterator Class Reference

Iterates over the keys of a [Config](#) object.

```
#include <Config.h>
```

### Public Member Functions

- [KeyIterator](#) ([Config](#) const &config, std::string const &section)  
*Constructor.*
- bool [AtEnd](#) () const  
*Indicates whether the iterator is pointing to the end of the map or not.*
- void [Next](#) ()  
*The pointer increases a position on the config.*
- std::string [GetKey](#) () const  
*Returns key of the current position.*
- std::string const & [GetValue](#) () const  
*Returns value of the current position.*
- int [GetIntValue](#) () const  
*Returns integer value of the current position.*

### 4.79.1 Detailed Description

Iterates over the keys of a [Config](#) object.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2000

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

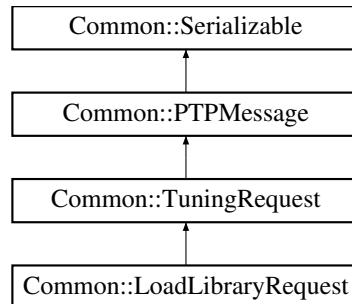
- Common/Config.h
- Common/Config.cpp

## 4.80 Common::LoadLibraryRequest Class Reference

Encapsulates a tuning request to load a specified shared library to a given application process.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::LoadLibraryRequest:



### Public Member Functions

- [LoadLibraryRequest](#) (int pid=0, std::string const &libPath=std::string())  
*Constructor.*
- PTPMsgType [GetType](#) () const  
*Returns type of message (PTPLoadLibrary).*
- std::string const & [GetLibraryPath](#) () const  
*Returns the path of the library to be loaded.*
- void [Serialize](#) (Serializer &out) const  
*Sends the message.*
- void [DeSerialize](#) (DeSerializer &in)  
*Receives the message.*

#### 4.80.1 Detailed Description

Encapsulates a tuning request to load a specified shared library to a given application process.

#### Version

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2003

**4.80.2 Constructor & Destructor Documentation****4.80.2.1 Common::LoadLibraryRequest::LoadLibraryRequest ( int *pid* = 0, std::string const & *libPath* = std::string() ) [inline]**

Constructor.

**Parameters**

<i>pid</i>	Id of the process where the library will be included, default 0.
<i>libPath</i>	Path of the library, default "".

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

**4.81 Common::LogEntry Class Reference**

Entry on a log.

```
#include <Syslog.h>
```

**Public Member Functions**

- [LogEntry](#) (LogSeverity s, std::string const &message)  
*Constructor.*
- [DateTime](#) const & [GetTimestamp](#) () const  
*Returns the date when the entry was performed.*
- LogSeverity [GetSeverity](#) () const  
*Returns log severity.*
- std::string const & [GetMessage](#) () const  
*Returns a string containing the log message.*

### 4.81.1 Detailed Description

Entry on a log.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

### 4.81.2 Constructor & Destructor Documentation

**4.81.2.1 Common::LogEntry::LogEntry ( LogSeverity *s*, std::string const & *message* )**  
[inline]

Constructor.

#### Parameters

<i>s</i>	Log severity, can be DEBUG, INFO, WARNING, ERROR or FATAL.
<i>message</i>	Message.

### 4.81.3 Member Function Documentation

**4.81.3.1 LogSeverity Common::LogEntry::GetSeverity ( ) const** [inline]

Returns log severity.

DEBUG A log generated during debugging of the software. INFO An informational message. WARNING A warning message that the system administrator might want to know about ERROR One of the software components caused an error or exception. FATAL One of the software components is no longer functional.

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

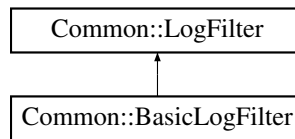
- Common/Syslog.h

## 4.82 Common::LogFilter Class Reference

Abstract class, validates logs.

```
#include <Syslog.h>
```

Inheritance diagram for Common::LogFilter:



### Public Member Functions

- virtual [~LogFilter](#) ()  
*Constructor.*
- virtual bool [Accept](#) ([LogEntry](#) const &entry) const =0  
*Filters log entry.*

#### 4.82.1 Detailed Description

Abstract class, validates logs.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.82.2 Member Function Documentation

**4.82.2.1** virtual bool [Common::LogFilter::Accept](#) ( [LogEntry](#) const & *entry* ) const [pure virtual]

Filters log entry.

##### Returns

True if entry is accepted, false otherwise.

Implemented in [Common::BasicLogFilter](#).

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

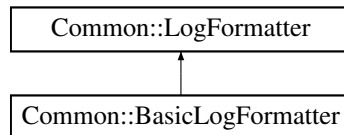
- Common/Syslog.h

## 4.83 Common::LogFormatter Class Reference

Abstract class, Gives logs the correct format.

```
#include <Syslog.h>
```

Inheritance diagram for Common::LogFormatter:



### Public Member Functions

- virtual [~LogFormatter](#) ()  
*Destructor.*
- virtual std::string **GetLogHeader** () const =0
- virtual std::string **GetLogFooter** () const =0
- virtual std::string **Format** ([LogEntry](#) const &entry) const =0

#### 4.83.1 Detailed Description

Abstract class, Gives logs the correct format.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

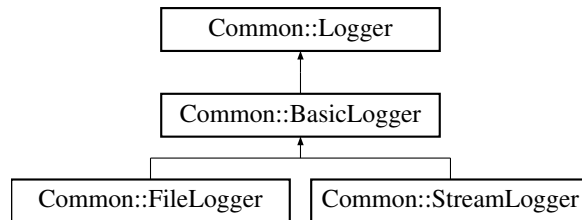
- Common/Syslog.h

## 4.84 Common::Logger Class Reference

Abstract class, tracks and stores information about events of interest happening in a system.

```
#include <Syslog.h>
```

Inheritance diagram for Common::Logger:



### Public Member Functions

- virtual `~Logger ()`  
*Destructor.*
- virtual void **Log** (`LogEntry` const &entry)=0
- void **SetName** (std::string const &name)  
*Sets logger name.*
- std::string const & **GetName** () const  
*Returns a string containing the logger name.*
- virtual void **SetFilter** (LogFilterPtr &filter)=0
- virtual `LogFilter` const \* **GetFilter** () const =0
- virtual void **SetFormatter** (LogFormatterPtr &formatter)=0
- virtual `LogFormatter` const \* **GetFormatter** () const =0

#### 4.84.1 Detailed Description

Abstract class, tracks and stores information about events of interest happening in a system.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/Syslog.h



## 4.85 ModelParam Struct Reference

### Public Attributes

- int **TotalDataVolume**
- int **TotalDataSendW**
- double **TotalCompTime**

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

- Analyzer/FactoringStats\_nw.h

## 4.86 ModuleList Class Reference

### Public Member Functions

- **ModuleList** (BPatch\_image &bpImage)
- int **GetSize** () const
- BPatch\_module & **operator[]** (int i) const

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

- Common/di.h

## 4.87 Monitor Class Reference

Adds request to add or remove instrumentation in/from the tasks it is monitoring.

```
#include <Monitor.h>
```

### Public Member Functions

- [Monitor](#) ([TaskCollection](#) &tasks)  
*Constructor.*
- void [AddInstr](#) (AddInstrRequest &instrReq)  
*Adds the instructions requested to the task they belong to.*
- void [RemoveInstr](#) (RemoveInstrRequest &instrReq)  
*Removes the instructions requested from the selected task.*

### 4.87.1 Detailed Description

Adds request to add or remove instrumentation in/from the tasks it is monitoring.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

### 4.87.2 Constructor & Destructor Documentation

#### 4.87.2.1 Monitor::Monitor ( TaskCollection & *tasks* ) [inline]

Constructor.

#### Parameters

<i>tasks</i>	Collection of tasks susceptible to be modified.
--------------	---

### 4.87.3 Member Function Documentation

#### 4.87.3.1 void Monitor::AddInstr ( AddInstrRequest & *instrReq* )

Adds the instructions requested to the task they belong to.

#### Parameters

<i>instrReq</i>	Object that represents the request for instrumentation to be added to a task.
-----------------	---

#### 4.87.3.2 void Monitor::RemoveInstr ( RemoveInstrRequest & *instrReq* )

Removes the instructions requested from the selected task.

#### Parameters

<i>instrReq</i>	Object that represents the request for instrumentation to be removed from a task.
-----------------	---

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

- AC/Monitor.h
- AC/Monitor.cpp

## 4.88 Common::Mutex Class Reference

Guarantees non concurrent access to a resource.

```
#include <sync.h>
```

### Public Member Functions

- [Mutex](#) ()  
*Constructor.*
- [~Mutex](#) ()  
*Destructor.*
- `operator pthread_mutex_t *` ()

### Protected Member Functions

- void [Enter](#) ()  
*Denotes that someone starts using the resource.*
- bool [CanEnter](#) ()  
*Returns true if the resource is not being used, false otherwise.*
- void [Leave](#) ()  
*Denotes that someone stops using the resource.*

### Friends

- class **MutexLock**

#### 4.88.1 Detailed Description

Guarantees non concurrent access to a resource. Mutual-Exclusion Object

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

## 4.88.2 Constructor & Destructor Documentation

### 4.88.2.1 Common::Mutex::Mutex ( ) [inline]

Constructor.

The mutex is always initialized as a recursive entity.

## 4.88.3 Member Function Documentation

### 4.88.3.1 void Mutex::Enter ( ) [protected]

Denotes that someone starts using the resource.

#### Exceptions

<a href="#">SysException</a>
------------------------------

### 4.88.3.2 void Mutex::Leave ( ) [protected]

Denotes that someone stops using the resource.

#### Exceptions

<a href="#">SysException</a>
------------------------------

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

- Common/sync.h
- Common/sync.cpp

## 4.89 Common::MutexLock Class Reference

System to manage access to a resource with a mutex.

```
#include <sync.h>
```

### Public Member Functions

- [MutexLock](#) ([Mutex](#) &mutex)

*Constructor.*

- [~MutexLock](#) ()

*Destructor.*

### 4.89.1 Detailed Description

System to manage access to a resource with a mutex.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

- Common/sync.h

## 4.90 myauto\_ptr< X > Class Template Reference

### Public Types

- typedef X **element\_type**

### Public Member Functions

- **myauto\_ptr** (X \*p=0)
- **myauto\_ptr** (const [myauto\\_ptr](#) &a)
- [myauto\\_ptr](#) & **operator=** (const [myauto\\_ptr](#) &a)
- X & **operator\*** () const
- X \* **operator->** () const
- X \* **get** () const
- X \* **release** () const

**template<class X> class myauto\_ptr< X >**

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

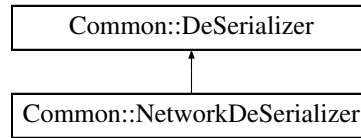
- Common/auto\_ptr.h

## 4.91 Common::NetworkDeSerializer Class Reference

Extracts serialized data from an istream object.

```
#include <NetSer.h>
```

Inheritance diagram for Common::NetworkDeSerializer:



### Public Member Functions

- [NetworkDeSerializer](#) (std::istream &stream)  
*constructor.*
- std::istream & [GetStream](#) ()  
*Returns istream object where the data is serialized.*
- long\_t [GetLong](#) ()  
*Reads long value from the stream.*
- double\_t [GetDouble](#) ()  
*Reads double value from the stream.*
- bool\_t [GetBool](#) ()  
*Reads bool value from the stream.*
- short\_t [GetShort](#) ()  
*Reads short value from the stream.*
- byte\_t [GetByte](#) ()  
*Reads byte value from the stream.*
- char\_t [GetChar](#) ()  
*Reads char value from the stream.*
- std::string [GetString](#) ()  
*Reads string value from the stream.*
- int\_t [GetInt](#) ()  
*Reads int value from the stream.*
- void [GetBuffer](#) (char \*buffer, int bufferSize)  
*Reads data directly from the stream.*

### 4.91.1 Detailed Description

Extracts serialized data from an istream object.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2002

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

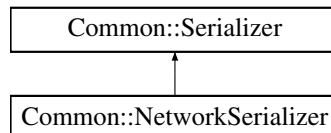
- Common/NetSer.h
- Common/NetSer.cpp

## 4.92 Common::NetworkSerializer Class Reference

Puts serialized data into an [OutputStream](#) object.

```
#include <NetSer.h>
```

Inheritance diagram for Common::NetworkSerializer:



### Public Member Functions

- [NetworkSerializer](#) ([OutputStream](#) &stream)

*Constructor.*

- void [PutLong](#) (long\_t l)

*Puts a long into the stream.*

- void [PutDouble](#) (double\_t d)

*Puts a double into the stream.*

- void [PutBool](#) (bool\_t b)

*Puts a boolean into the stream.*

- void [PutShort](#) (short\_t s)  
*Puts a short into the stream.*
- void [PutByte](#) (byte\_t b)  
*Puts a byte into the stream.*
- void [PutChar](#) (char\_t c)  
*Puts a char into the stream.*
- void [PutString](#) (std::string const &str)  
*Puts a string into the stream.*
- void [PutInt](#) (int\_t i)  
*Puts an integer long into the stream.*
- void [PutBuffer](#) (char const \*buffer, int bufferSize)  
*Puts a buffer into the stream.*

#### 4.92.1 Detailed Description

Puts serialized data into an [OutputStream](#) object.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.92.2 Constructor & Destructor Documentation

##### 4.92.2.1 [Common::NetworkSerializer::NetworkSerializer](#) ( [OutputStream](#) & *stream* ) [inline]

Constructor.

##### Parameters

<i>stream</i>	Stream where the serialized data will be written.
---------------	---

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



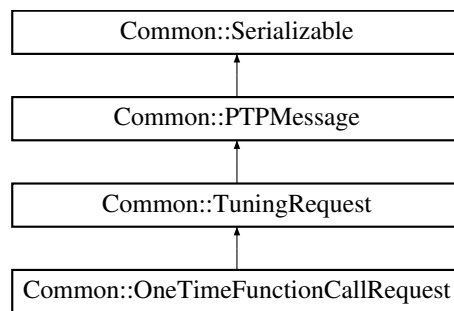
- Common/NetSer.h
- Common/NetSer.cpp

## 4.93 Common::OneTimeFunctionCallRequest Class Reference

Encapsulates a tuning request to invoke one time a given function in a given application process.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::OneTimeFunctionCallRequest:



### Public Member Functions

- **OneTimeFunctionCallRequest** (int pid=0, std::string const &funcName=std::string(), int nAttrs=0, **Attribute** const \*attrs=0, **Breakpoint** const \*brkpt=0)  
*Constructor.*
- **~OneTimeFunctionCallRequest** ()  
*Destructor.*
- PTPMsgType **GetType** () const  
*Returns type of message (PTPOneTimeFuncCall).*
- std::string const & **GetFuncName** () const  
*Returns name of the function to be added.*
- int **GetAttrCount** () const  
*Returns number of attributes the function has.*
- **Attribute** \* **GetAttributes** () const  
*Returns array of attributes.*
- void **Serialize** (**Serializer** &out) const  
*Sends the message.*

- void [DeSerialize](#) ([DeSerializer](#) &in)

*Gets the message.*

#### 4.93.1 Detailed Description

Encapsulates a tuning request to invoke one time a given function in a given application process.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003

#### 4.93.2 Constructor & Destructor Documentation

**4.93.2.1** `Common::OneTimeFunctionCallRequest::OneTimeFunctionCallRequest ( int pid = 0, std::string const & funcName = std::string(), int nAttrs = 0, Attribute const * attrs = 0, Breakpoint const * brkpt = 0 ) [inline]`

Constructor.

##### Parameters

<i>pid</i>	Id of the process where the call will be inserted, default 0.
<i>funcName</i>	Name of the function to call, default "".
<i>nAttrs</i>	Number of attributes the function has, default 0.
<i>attrs</i>	<a href="#">Attribute</a> array, default 0.
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

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

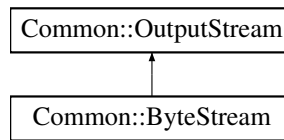
- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.94 Common::OutputStream Class Reference

Abstract class, represents an output stream of bytes.

```
#include <OutputStream.h>
```

Inheritance diagram for Common::OutputStream:



### Public Member Functions

- virtual void **Write** (char const \*buf, size\_t size)=0

#### 4.94.1 Detailed Description

Abstract class, represents an output stream of bytes. This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of [OutputStream](#) must always provide at least a method that writes one byte of output.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2003

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

- Common/OutputStream.h

## 4.95 Common::Pipe Class Reference

Element used to join output and input from two processes.

```
#include <Pipe.h>
```

### Public Member Functions

- [Pipe](#) ()

*Constructor.*

- [~Pipe](#) ()

*Destructor.*

- bool [IsReadOpen](#) () const

*Returns whether the read end is open or not.*

- bool [IsWriteOpen](#) () const

*Returns whether the write end is open or not.*

- int [GetRead](#) () const

*Returns read file descriptor.*

- int [GetWrite](#) () const

*Returns write file descriptor.*

- void [CloseRead](#) ()

*Closes the read end.*

- void [CloseWrite](#) ()

*Closes the write end.*

- int [Read](#) (char \*buf, int bufSize)

*Reads from the read end and stores the content on the buffer.*

- int [Write](#) (char const \*buf, int bufSize)

*Writes the content of the buffer on the write end.*

#### 4.95.1 Detailed Description

Element used to join output and input from two processes. A pair of channels that implements a unidirectional pipe.

A pipe consists of a pair of channels: A writable sink channel and a readable source channel. Once some bytes are written to the sink channel they can be read from source channel in exactly the order in which they were written.

Whether or not a thread writing bytes to a pipe will block until another thread reads those bytes, or some previously-written bytes, from the pipe is system-dependent and therefore unspecified. Many pipe implementations will buffer up to a certain number of bytes between the sink and source channels, but such buffering should not be assumed.

#### Version

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2001

**4.95.2 Constructor & Destructor Documentation****4.95.2.1 Pipe::Pipe ( )**

Constructor.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.95.3 Member Function Documentation****4.95.3.1 int Pipe::Read ( char \* *buf*, int *bufSize* )**

Reads from the read end and stores the content on the buffer.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.95.3.2 int Pipe::Write ( char const \* *buf*, int *bufSize* )**

Writes the content of the buffer on the write end.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

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

- Common/Pipe.h
- Common/Pipe.cpp

**4.96 PointList Class Reference****Public Member Functions**

- **PointList** ([\*DiFunction\*](#) &func)
- int **GetSize** () const

- void **GetCalledFuncName** (BPatch\_point &point, char \*name, int length)
- unsigned long **GetAddress** (BPatch\_point &point)
- BPatch\_point & **operator[]** (int i) const

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

- Common/di.h

## 4.97 ProcedureList Class Reference

### Public Member Functions

- **ProcedureList** (BPatch\_image &bpImage)
- int **GetSize** () const
- BPatch\_function & **operator[]** (int i) const

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

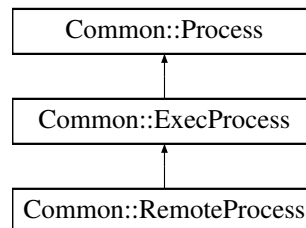
- Common/di.h

## 4.98 Common::Process Class Reference

Abstract class, creates a new process to perform different operations on the overridden method Run().

```
#include <Process.h>
```

Inheritance diagram for Common::Process:



### Public Member Functions

- virtual void **Start** ()  
*Executes the process.*
- int **GetPid** () const  
*Returns process id.*

### Protected Member Functions

- [Process \(\)](#)  
*Constructor.*
- virtual int **Run** ()=0

### Protected Attributes

- int **\_pid**

#### 4.98.1 Detailed Description

Abstract class, creates a new process to perform different operations on the overridden method Run().

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2001

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

- Common/Process.h
- AC/Tuner.cpp
- Common/Process.cpp

## 4.99 PTPAcceptor Class Reference

Manages socket connection and handles data input through them.

```
#include <PTPAcceptor.h>
```

### Public Member Functions

- [PTPAcceptor](#) (Reactor &reactor, [TaskManager](#) &tm)  
*Constructor.*
- [~PTPAcceptor](#) ()  
*Destructor.*

- void [HandleInput](#) ()  
*Gets the socket for the client and binds it with the task manager.*
- int [GetHandle](#) ()  
*Getter of a handler for the variable `_socket`.*

#### 4.99.1 Detailed Description

Manages socket connection and handles data input through them.

##### Version

1.0

##### Since

1.0

##### Author

Ania Morajko, 2002

#### 4.99.2 Constructor & Destructor Documentation

##### 4.99.2.1 PTPAcceptor::PTPAcceptor ( Reactor & *reactor*, TaskManager & *tm* )

Constructor.

##### Parameters

<i>reactor</i>	Object of class reactor that manages event handlers.
<i>tm</i>	<a href="#">Task</a> manager.

#### 4.99.3 Member Function Documentation

##### 4.99.3.1 int PTPAcceptor::GetHandle ( ) [inline]

Getter of a handler for the variable `_socket`.

##### Returns

Handle of the server socket.

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

- AC/PTPAcceptor.h
- AC/PTPAcceptor.cpp



## 4.100 PTPHandler Class Reference

Manages the requests from the [PTPAccepter](#).

```
#include <PTPHandler.h>
```

### Public Member Functions

- [PTPHandler](#) (SocketPtr &socket, [TaskManager](#) &tm)  
*Constructor.*
- void **Remove** ()
- void [HandleInput](#) ()  
*Reads message from socket and handles the different kinds of requests that are received.*
- int [GetHandle](#) ()  
*Getter of a handler for the variable \_socket.*

### 4.100.1 Detailed Description

Manages the requests from the [PTPAccepter](#).

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

### 4.100.2 Constructor & Destructor Documentation

#### 4.100.2.1 PTPHandler::PTPHandler ( SocketPtr & socket, TaskManager & tm ) [inline]

Constructor.

#### Parameters

<i>socket</i>	Pinter to the socket used to get the input (request).
<i>tm</i>	<a href="#">Task</a> manager that handles the task to which the request affects.

### 4.100.3 Member Function Documentation

#### 4.100.3.1 int PTPHandler::GetHandle ( ) [inline]

Getter of a handler for the variable `_socket`.

#### Returns

Handle of the socket.

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

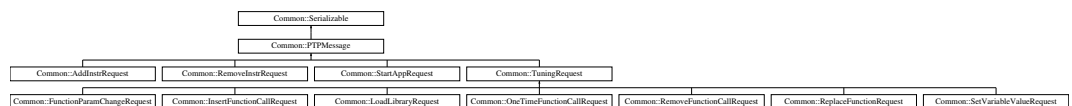
- AC/PTPHandler.h
- AC/PTPHandler.cpp

## 4.101 Common::PTPMessage Class Reference

Performance tunning protocol, represents message interchanged between analyzer and tuner/tracer.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::PTPMessage:



### Public Member Functions

- [PTPMessage](#) ()  
*Constructor.*
- virtual PTPMsgType [GetType](#) () const  
*To be implemented by subclasses.*
- int [GetDataSize](#) () const  
*Returns size of the data once serialized.*
- virtual [~PTPMessage](#) ()  
*Destructor.*

### 4.101.1 Detailed Description

Performance tuning protocol, represents message interchanged between analyzer and tuner/tracer.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2003

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

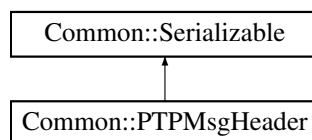
- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.102 Common::PTPMsgHeader Class Reference

Represents header of a [PTPMessage](#) object.

```
#include <PTPMsgHeader.h>
```

Inheritance diagram for Common::PTPMsgHeader:

**Public Member Functions**

- [PTPMsgHeader](#) ()  
*Constructor.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the message header.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Receives the message header.*
- int [GetMagic](#) () const

*Returns magic attribute.*

- `int GetVersion () const`  
*Returns version attribute.*
- `PTPMsgType GetType () const`  
*Returns the type of the message.*
- `int GetDataSize () const`  
*Returns data size.*
- `int GetHeaderSize () const`  
*Returns header size.*
- `void SetMagic (int magic)`  
*Sets the magic attribute.*
- `void SetVersion (int version)`  
*Sets the version attribute.*
- `void SetMsgType (PTPMsgType type)`  
*Sets the type of the message.*
- `void SetDataSize (int size)`  
*Sets data size.*
- `void SetHeaderSize ()`  
*Updates header size.*

#### 4.102.1 Detailed Description

Represents header of a [PTPMessage](#) object.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- `Common/PTPMsgHeader.h`
- `Common/PTPMsgHeader.cpp`

## 4.103 Common::PTPProtocol Class Reference

Communicates analyzer and tuner.

```
#include <PTPProtocol.h>
```

### Static Public Member Functions

- static void [WriteMessage](#) (PTPMessage &msg, [OutputStream](#) &stream)  
*Sends a message through a stream to the tuner.*
- static [PTPMessage](#) \* [ReadMessage](#) (std::istream &stream)  
*Receives a message through a stream from the analyzer.*
- static void [WriteMessageEx](#) (PTPMessage &msg, [Socket](#) &sock)  
*Sends a message through a socket to the tuner.*
- static [PTPMessage](#) \* [ReadMessageEx](#) ([Socket](#) &sock)  
*Receives a message through a socket from the analyzer.*

### 4.103.1 Detailed Description

Communicates analyzer and tuner.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

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

- Common/PTPProtocol.h
- Common/PTPProtocol.cpp

## 4.104 Common::Queue< T > Class Template Reference

Data structure that stores objects of any class.

```
#include <Queue.h>
```

### Public Member Functions

- [Queue](#) (int maxSize)  
*Constructor.*
- [~Queue](#) ()  
*Destructor.*
- bool [IsEmpty](#) () const  
*Returns true if the queue is empty, false otherwise.*
- bool [IsFull](#) () const  
*Returns true if the queue is full, false otherwise.*
- int [GetMaxSize](#) () const  
*Returns the maximum quantity of objects the queue can store.*
- int [GetCount](#) () const  
*Returns current size of the queue.*
- bool [Get](#) (T &item)  
*Returns first object of the queue and removes it.*
- void [GetB](#) (T &item)  
*Returns first object of the queue and removes it.*
- void [Put](#) (T &item)  
*Puts an object at the end of the queue.*

#### 4.104.1 Detailed Description

`template<class T> class Common::Queue< T >`

Data structure that stores objects of any class. This data structure manages the objects using a FIFO priority.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

### 4.104.2 Member Function Documentation

#### 4.104.2.1 `template<class T> bool Queue::Get ( T & item )`

Returns first object of the queue and removes it.

Returns false if the queue is empty.

#### 4.104.2.2 `template<class T> void Queue::GetB ( T & item )`

Returns first object of the queue and removes it.

If the queue is empty, waits until it has any object.

#### 4.104.2.3 `template<class T> void Queue::Put ( T & item )`

Puts an object at the end of the queue.

If the queue is full, it waits until there's space.

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

- Common/Queue.h

## 4.105 Common::Reactor Class Reference

Registers, removes and dispatches [EventHandler](#) objects.

```
#include <Reactor.h>
```

### Public Member Functions

- [Reactor](#) ()  
*Constructor.*
- void [Register](#) ([EventHandler](#) &handler)  
*Registers a new [EventHandler](#).*
- void [UnRegister](#) ([EventHandler](#) &handler)  
*Removes given [EventHandler](#).*
- void [HandleEvents](#) ([TimeValue](#) \*timeout=0)  
*Runs event loop.*
- [EventHandler](#) & [GetHandler](#) (int handle)  
*Returns selected handler.*

### 4.105.1 Detailed Description

Registers, removes and dispatches [EventHandler](#) objects. Uses reactor design pattern.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

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

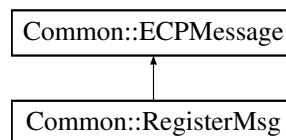
- Common/Reactor.h
- Common/Reactor.cpp

## 4.106 Common::RegisterMsg Class Reference

Represents message that is sent when DMLib is registered with analyzer to send event messages.

```
#include <ECPMsg.h>
```

Inheritance diagram for Common::RegisterMsg:



### Public Member Functions

- [RegisterMsg](#) (int pid, int mpiRank, std::string host, std::string taskName)  
*Constructor.*
- ECPMsgType [GetType](#) () const  
*Returns the type of event.*
- int [GetPid](#) () const  
*Returns Id of the process where the library will be loaded.*
- int [GetMpiRank](#) () const



*Returns mpi rank.*

- string const & [GetHost](#) () const

*Returns host name where the process is located.*

- string const & [GetTaskName](#) () const

*Returns task name.*

- void [Serialize](#) (Serializer &out) const

*Sends the message.*

- void [DeSerialize](#) (DeSerializer &in)

*Receives the message.*

#### 4.106.1 Detailed Description

Represents message that is sent when DMLib is registered with analyzer to send event messages.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.106.2 Constructor & Destructor Documentation

**4.106.2.1** `Common::RegisterMsg::RegisterMsg ( int pid, int mpiRank, std::string host, std::string taskName ) [inline]`

Constructor.

##### Parameters

<i>pid</i>	Id of the process where the library will be registered.
<i>mpiRank</i>	Mpi rank.
<i>host</i>	Host where the process is located.
<i>taskName</i>	Name of the task.

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

- Common/ECPMsg.h

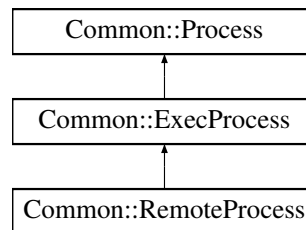
- Common/ECPMsg.cpp

## 4.107 Common::RemoteProcess Class Reference

Remotely executes a command in another machine.

```
#include <Process.h>
```

Inheritance diagram for Common::RemoteProcess:



### Public Member Functions

- [RemoteProcess](#) (std::string hostName, std::string command, std::string rshPath=std::string("/usr/bin/rsh"))

*Constructor.*

- [RemoteProcess](#) (std::string hostName, std::string userName, std::string command, std::string rshPath=std::string("/usr/bin/rsh"))

*Constructor.*

- void [SetOutputToNull](#) ()

*Enables output to null property.*

### Protected Member Functions

- int [Run](#) ()

*Executes de process.*

- std::string [GetHostName](#) () const

#### 4.107.1 Detailed Description

Remotely executes a command in another machine. Uses the rsh program to perform the execution of the process.

**Version**

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2001

**4.107.2 Constructor & Destructor Documentation**

**4.107.2.1** `Common::RemoteProcess::RemoteProcess ( std::string hostName, std::string command, std::string rshPath = std::string("/usr/bin/rsh") )`  
`[inline]`

Constructor.

**Parameters**

<i>hostName</i>	Host where the command will be executed remotely.
<i>command</i>	Command to execute.
<i>rshPath</i>	Local path of the rsh program, default = "/usr/bin/rsh".

**4.107.2.2** `Common::RemoteProcess::RemoteProcess ( std::string hostName, std::string userName, std::string command, std::string rshPath = std::string("/usr/bin/rsh") )`  
`[inline]`

Constructor.

**Parameters**

<i>hostName</i>	Host where the command will be executed remotely.
<i>command</i>	Command to execute.
<i>userName</i>	Name of the user on the host to execute the command.
<i>rshPath</i>	Local path of the rsh program, default = "/usr/bin/rsh".

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

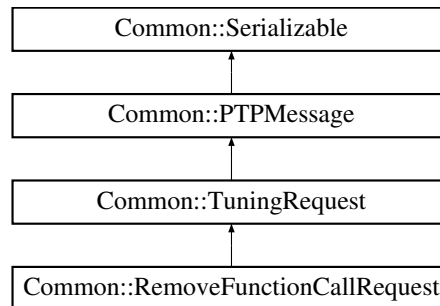
- Common/Process.h
- Common/Process.cpp

**4.108 Common::RemoveFunctionCallRequest Class Reference**

Encapsulates a tuning request to remove all calls to a given function from the given caller function.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::RemoveFunctionCallRequest:



### Public Member Functions

- [RemoveFunctionCallRequest](#) (int pid=0, std::string const &funcName=std::string(), std::string const &callerFunc=string(), [Breakpoint](#) const \*brkpt=0)

*Constructor.*

- PTPMsgType [GetType](#) () const

*Returns type of message (PTPRemoveFuncCall).*

- std::string const & [GetFuncName](#) () const

*Returns name of the function to be added.*

- std::string const & [GetCallerFunc](#) () const

*Returns function caller name.*

- void [Serialize](#) ([Serializer](#) &out) const

*Sends the message.*

- void [DeSerialize](#) ([DeSerializer](#) &in)

*Receives the message.*

### 4.108.1 Detailed Description

Encapsulates a tuning request to remove all calls to a given function from the given caller function.

#### Version

1.0

#### Since

1.0

**Author**

Ania Morajko, 2003

**4.108.2 Constructor & Destructor Documentation**

**4.108.2.1** `Common::RemoveFunctionCallRequest::RemoveFunctionCallRequest ( int pid = 0, std::string const & funcName = std::string(), std::string const & callerFunc = string(), Breakpoint const * brkpt = 0 ) [inline]`

Constructor.

**Parameters**

<i>pid</i>	Id of the process where the call will be removed, default 0.
<i>funcName</i>	Name of the function to remove, default "".
<i>callerFunc</i>	Name of the caller function, default "".
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

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

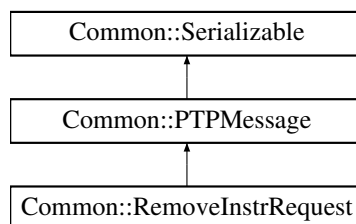
- Common/PTPMsg.h
- Common/PTPMsg.cpp

**4.109 Common::RemoveInstrRequest Class Reference**

Represents message sent when analyzer requests to remove instrumentation.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::RemoveInstrRequest:

**Public Member Functions**

- [RemoveInstrRequest](#) (int *pid*=0, int *eventId*=0, InstrPlace *place*=ipFuncEntry)

*Constructor.*

- PTPMsgType [GetType](#) () const

*Returns type of message (PTPRemoveInstr).*

- int [GetPid](#) () const  
*Returns the process id.*
- int [GetEventId](#) () const  
*Returns the event id.*
- InstrPlace [GetInstrPlace](#) () const  
*Returns the place where the instruction should be removed.*
- void [Serialize](#) (Serializer &out) const  
*Sends the message.*
- void [DeSerialize](#) (DeSerializer &in)  
*Receives the message.*

#### 4.109.1 Detailed Description

Represents message sent when analyzer requests to remove instrumentation.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003

#### 4.109.2 Constructor & Destructor Documentation

4.109.2.1 `Common::RemoveInstrRequest::RemoveInstrRequest ( int pid = 0, int eventId = 0, InstrPlace place = ipFuncEntry ) [inline]`

Constructor.

##### Parameters

<i>pid</i>	Id of the process where the instrumentation will be removed, default 0.
<i>eventId</i>	<a href="#">Event</a> id, default 0.
<i>place</i>	Place where the instrumentation will be removed, default ipFuncEntry.

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

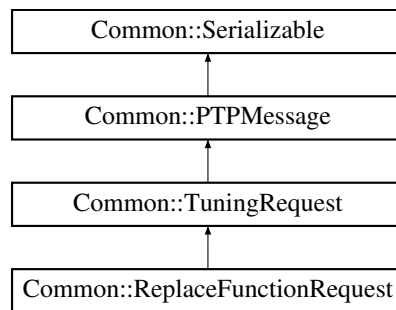
- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.110 Common::ReplaceFunctionRequest Class Reference

Encapsulates a tuning request to replace all calls to a function inside a process with calls to another function.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::ReplaceFunctionRequest:



### Public Member Functions

- [ReplaceFunctionRequest](#) (int pid=0, std::string const &oldFunc=std::string(), std::string const &newFunc=std::string(), [Breakpoint](#) \*brkpt=0)

*Constructor.*

- PTPMsgType [GetType](#) () const

*Returns type of message (PTPReplaceFunction).*

- std::string const & [GetOldFunction](#) () const

*Returns a string containing the name of the function to replace.*

- std::string const & [GetNewFunction](#) () const

*Returns a string containing the name of the function added.*

- void [Serialize](#) ([Serializer](#) &out) const

*Sends the message.*

- void [DeSerialize](#) ([DeSerializer](#) &in)

*Receives the message.*

### 4.110.1 Detailed Description

Encapsulates a tuning request to replace all calls to a function inside a process with calls to another function.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2003

### 4.110.2 Constructor & Destructor Documentation

**4.110.2.1** `Common::ReplaceFunctionRequest::ReplaceFunctionRequest ( int pid = 0, std::string const & oldFunc = std::string(), std::string const & newFunc = std::string(), Breakpoint * brkpt = 0 ) [inline]`

Constructor.

#### Parameters

<i>pid</i>	Id of the process where the function will be replaced, default 0.
<i>oldFunc</i>	Name of the function to replace, default "".
<i>newFunc</i>	Name of the function to add, default "".
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.111 Common::Semaphore Class Reference

Synchronizes access to a resource.

```
#include <sync.h>
```

### Public Member Functions

- [Semaphore](#) (int initialValue=0, bool crossProcess=false)

*Constructor.*



- [~Semaphore \(\)](#)

*Destructor.*

- void [Wait \(\)](#)

*Stops the current thread until the acces to the resource is open.*

- bool [TryWait \(\)](#)

*Returns true if the access to the resource is open, false otherwise.*

- void [Post \(\)](#)

*Gives a signal to the semaphore indicating that a client has stop using the resource.*

### Protected Member Functions

- **Semaphore** (sem\_t s)

### Protected Attributes

- sem\_t **\_semaphore**

#### 4.111.1 Detailed Description

Synchronizes access to a resource. A semaphore is generally used as a synchronization object between multiple threads or to protect a limited and finite resource such as a memory or thread pool. The semaphore has a counter which only permits access by one or more threads when the value of the semaphore is non-zero. Each access reduces the current value of the semaphore by 1. One or more threads can wait on a semaphore until it is no longer 0, and hence the semaphore can be used as a simple thread synchronization object to enable one thread to pause others until the thread is ready or has provided data for them.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

### 4.111.2 Constructor & Destructor Documentation

**4.111.2.1** `Common::Semaphore::Semaphore ( int initialValue = 0, bool crossProcess = false ) [inline]`

Constructor.

#### Parameters

<i>initialValue</i>	Initial value of the semaphore, default 0.
<i>crossProcess</i>	Flag indicating whether or not the semaphore should be shared with forker processes. default false.

### 4.111.3 Member Function Documentation

**4.111.3.1** `void Semaphore::Post ( )`

Gives a signal to the semaphore indicating that a client has stop using the resource.

Posting to a semaphore increments its current value and releases the first thread waiting for the semaphore if it is currently at 0.

**4.111.3.2** `bool Semaphore::TryWait ( )`

Returns true if the access to the resource is open, false otherwise.

TryWait is a non-blocking variant of Wait. If the semaphore counter is greater than 0, then the thread is accepted and the semaphore counter is decreased. If the semaphore counter is 0 TryWait returns immediately with false.

**4.111.3.3** `void Semaphore::Wait ( )`

Stops the current thread until the acces to the resource is open.

Wait is used to keep a thread held until the semaphore counter is greater than 0. If the current thread is held, then another thread must increment the semaphore. Once the thread is accepted, the semaphore is automatically decremented, and the thread continues execution.

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

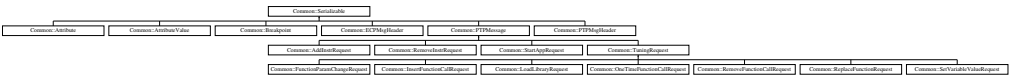
- Common/sync.h
- Common/sync.cpp

## 4.112 Common::Serializable Class Reference

Abstract class, makes an object able to be passed through a stream using [Serializer](#) and [DeSerializer](#) objects.

```
#include <Serial.h>
```

Inheritance diagram for Common::Serializable:



Public Member Functions

- virtual void **Serialize** ([Serializer](#) &out) const =0
- virtual void **DeSerialize** ([DeSerializer](#) &in)=0

4.112.1 Detailed Description

Abstract class, makes an object able to be passed through a stream using [Serializer](#) and [DeSerializer](#) objects.

Version

1.0b

Since

1.0b

Author

Ania Morajko, 2002

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

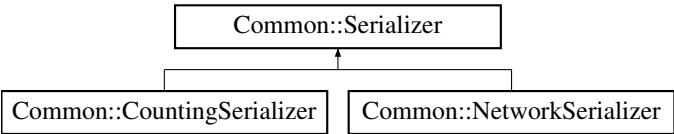
- Common/Serial.h

4.113 Common::Serializer Class Reference

Abstract class, prepares objects to be passed on a stream.

```
#include <Serial.h>
```

Inheritance diagram for Common::Serializer:



### Public Member Functions

- virtual void **PutByte** (byte\_t b)=0
- virtual void **PutChar** (char\_t c)=0
- virtual void **PutBool** (bool\_t b)=0
- virtual void **PutShort** (short\_t s)=0
- virtual void **PutInt** (int\_t i)=0
- virtual void **PutLong** (long\_t l)=0
- virtual void **PutDouble** (double\_t d)=0
- virtual void **PutString** (std::string const &str)=0
- virtual void **PutBuffer** (char const \*buffer, int bufferSize)=0

#### 4.113.1 Detailed Description

Abstract class, prepares objects to be passed on a stream.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/Serial.h
- Common/NetSer.cpp

## 4.114 Common::ServerSocket Class Reference

Holds a [SocketBase](#) object and represents a TCP/IP server socket.

```
#include <Socket.h>
```

### Public Member Functions

- [ServerSocket](#) (int port, int backLog=5)  
*Constructor.*
- void [Listen](#) ()  
*Sets the socket to a listening state.*

- SocketPtr [Accept](#) ()  
*Accepts a connection and creates a socket.*
- SocketPtr [Accept](#) (int timeoutMs)  
*Accepts a connection and creates a socket, waits the given timeout.*
- [Address](#) const & [GetAddress](#) () const  
*Returns local server address.*
- int [GetLocalPort](#) () const  
*Returns the port the socket is listening.*
- int [GetHandle](#) () const  
*Returns socket handle.*

#### 4.114.1 Detailed Description

Holds a [SocketBase](#) object and represents a TCP/IP server socket. All the functions on this class are present on the [SocketBase](#) class and have the same functionality.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.114.2 Constructor & Destructor Documentation

##### 4.114.2.1 ServerSocket::ServerSocket ( int port, int backlog = 5 )

Constructor.

##### Parameters

<i>backLog</i>	Maximum length of the queue for pending connections.
----------------	--

##### Exceptions

<i>SysException.</i>	
----------------------	--

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

- Common/Socket.h
- Common/Socket.cpp

## 4.115 Service Class Reference

provides methods to work with EventCollectorHandlers lists. Holds a list of EventColl-Handler and a reference to the reactor. Provides methods to add and remove handlers form the list.

```
#include <Service.h>
```

### Public Member Functions

- [Service](#) (Reactor &reactor)  
*Constructor.*
- [~Service](#) ()  
*Destructor, deletes the handlers and the references to them.*
- void [Add](#) (ECPHandler \*handler)  
*adds a handler to the list and sets its service to this.*
- void [Remove](#) (ECPHandler \*handler)  
*Unregisters the handler from the reactor and removes it from the list.*

### 4.115.1 Detailed Description

provides methods to work with EventCollectorHandlers lists. Holds a list of EventColl-Handler and a reference to the reactor. Provides methods to add and remove handlers form the list.

### 4.115.2 Constructor & Destructor Documentation

#### 4.115.2.1 Service::Service ( Reactor & reactor ) [inline]

Constructor.

#### Parameters

<i>reactor</i>	reactor of the application??
----------------	------------------------------

### 4.115.3 Member Function Documentation

#### 4.115.3.1 void Service::Add ( ECPHandler \* *handler* )

adds a handler to the list and sets its service to this.

##### Parameters

<i>handler</i>	ECP Handler.
----------------	--------------

#### 4.115.3.2 void Service::Remove ( ECPHandler \* *handler* )

Unregisters the handler from the reactor and removes it from the list.

##### Parameters

<i>handler</i>	ECP Handler
----------------	-------------

##### Exceptions

<i>exception</i>	when the handler does not exist in the list.
------------------	--

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

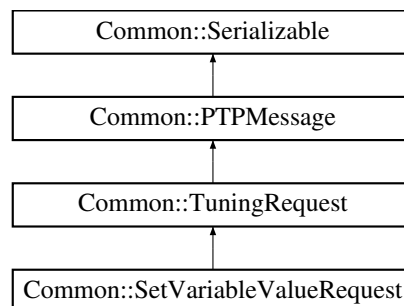
- Analyzer/Service.h
- Analyzer/Service.cpp

## 4.116 Common::SetVariableValueRequest Class Reference

Encapsulates a tuning request to modify a value of a specified variable in a given application process.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::SetVariableValueRequest:



### Public Member Functions

- [SetVariableValueRequest](#) (int pid=0, std::string const &varName=std::string(), [AttributeValue](#) const &varValue=[AttributeValue](#)(), [Breakpoint](#) \*brkpt=0)

*Constructor.*

- void \* [GetValueBuffer](#) ()

*Returns the value to set in a buffer format.*

- int [GetValueSize](#) () const

*Returns size of the variable new value.*

- std::string [GetValueString](#) () const

*Returns a string containing the value of the variable.*

- PTPMsgType [GetType](#) () const

*Returns type of message (PTPSetVariableValue).*

- std::string const & [GetVariableName](#) () const

*Returns a string containing the variable name.*

- void [Serialize](#) ([Serializer](#) &out) const

*Sends the message.*

- void [DeSerialize](#) ([DeSerializer](#) &in)

*Receives the message.*

#### 4.116.1 Detailed Description

Encapsulates a tuning request to modify a value of a specified variable in a given application process.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003



### 4.116.2 Constructor & Destructor Documentation

**4.116.2.1** `Common::SetVariableValueRequest::SetVariableValueRequest ( int pid = 0, std::string const & varName = std::string(), AttributeValue const & varValue = AttributeValue (), Breakpoint * brkpt = 0 ) [inline]`

Constructor.

#### Parameters

<i>pid</i>	Id of the process where the variable will be modified, default 0.
<i>varName</i>	Name of the variable, default "".
<i>varValue</i>	New value to set, default empty <a href="#">AttributeValue</a> object.
<i>brkpt</i>	Used for synchronization purposes, the actual tuning will be executed when the execution reaches the breakpoint, default 0.

### 4.116.3 Member Function Documentation

**4.116.3.1** `void* Common::SetVariableValueRequest::GetValueBuffer ( ) [inline]`

Returns the value to set in a buffer format.

If the type is known, it can be used using a cast, for example:

```
int foo = (int) VarRequest.GetValueBuffer();
```

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

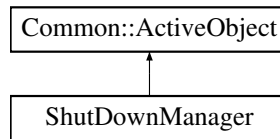
- Common/PTPMsg.h
- Common/PTPMsg.cpp

## 4.117 ShutDownManager Class Reference

Handles the shut down of MATE ([Analyzer](#) and AC's) The data structure consists basically in a reference to the application model (to know the hosts where the AC's are running in real time) and a boolean to determine if MATE is finished (to let the main process know, and make it stop) Provides a method to set the application model from outside (when it is ready, the main process of [Analyzer](#) will set it). On the other hand, this class inherits from `ActiveObject`, so its objects are execution threads, this is done to wait for the user to stop MATE without stopping its own execution.

```
#include <ShutDownManager.h>
```

Inheritance diagram for `ShutDownManager`:



## Public Member Functions

- [ShutDownManager](#) ()  
*Constructor. Sets the finished member to false and starts the thread.*
- virtual [~ShutDownManager](#) ()  
*Destructor.*
- void [Run](#) ()
- void [InitThread](#) ()  
*Not implemented (Here for compatibility reasons).*
- void [FlushThread](#) ()  
*Not implemented (Here for compatibility reasons).*
- bool [isFinished](#) ()  
*getter of finished boolean.*
- void [setApp](#) ([Model::Application](#) &app)  
*Application model reference setter.*

### 4.117.1 Detailed Description

Handles the shut down of MATE ([Analyzer](#) and AC's) The data structure consists basically in a reference to the application model (to know the hosts where the AC's are running in real time) and a boolean to determine if MATE is finished (to let the main process know, and make it stop) Provides a method to set the application model from outside (when it is ready, the main process of [Analyzer](#) will set it). On the other hand, this class inherits from [ActiveObject](#), so its objects are execution threads, this is done to wait for the user to stop MATE without stopping its own execution.

### 4.117.2 Member Function Documentation

#### 4.117.2.1 bool ShutDownManager::isFinished ( ) [inline]

getter of finished boolean.

**Returns**

true if the user stopped MATE, false otherwise.

**4.117.2.2 void ShutDownManager::Run ( ) [virtual]**

Function which is executed by the thread, waits for the user to stop MATE, when receives the commandment sends a stop signal to AC's and sets the variable finished to true in order to stop the [Analyzer](#) itself.

Implements [Common::ActiveObject](#).

**4.117.2.3 void ShutDownManager::setApp ( Model::Application & app ) [inline]**

Application model reference setter.

**Parameters**

<i>app</i>	reference to the application model
------------	------------------------------------

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

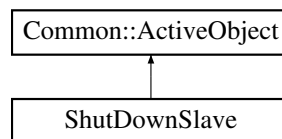
- Analyzer/ShutDownManager.h
- Analyzer/ShutDownManager.cpp

**4.118 ShutDownSlave Class Reference**

Receives terminating message from [Analyzer](#).

```
#include <ShutDownSlave.h>
```

Inheritance diagram for ShutDownSlave:

**Public Member Functions**

- **ShutDownSlave** (string analyzerHost, int analyzerPort, [Controller](#) ctrl)
- void **Run** ()

**Protected Member Functions**

- void **InitThread** ()

- void **FlushThread** ()

#### 4.118.1 Detailed Description

Receives terminating message from [Analyzer](#). Runs a thread that waits blocked for a message from the analyzer. When it receives this message the main loop in the controller is stopped, and subsequently the AC is terminated.

note: the tasks should be deleted at this point.

#### Version

1.1

#### Since

1.1

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

- AC/ShutDownSlave.h
- AC/ShutDownSlave.cpp

### 4.119 SnippetHandler Class Reference

Contains he necessary fields to manage snippets.

```
#include <InstrSet.h>
```

#### Public Member Functions

- [SnippetHandler](#) (int eventId, std::string const &funcName, InstrPlace place, BPatch-SnippetHandle \*handle)

*Constructor.*

- int [GetEventId](#) () const

*Getter for the variable \_eventId.*

- std::string const & [GetFuncName](#) () const

*Getter for the variable \_funcName.*

- InstrPlace [GetInstrPlace](#) () const

*Getter for the variable \_place.*

- BPatchSnippetHandle \* [GetHandle](#) () const

*Getter for the variable \_handle.*

### 4.119.1 Detailed Description

Contains he necessary fields to manage snippets.

#### Version

1.0

#### Since

1.0

#### Author

Ania Morajko, 2002

### 4.119.2 Constructor & Destructor Documentation

**4.119.2.1** `SnippetHandler::SnippetHandler ( int eventId, std::string const & funcName, InstrPlace place, BPatchSnippetHandle * handle )` `[inline]`

Constructor.

#### Parameters

<i>eventId</i>	Unique identifier for the event.
<i>funcName</i>	Name of the function in which the snippet will be inserted.
<i>place</i>	Position in which the instrumentation will be added.
<i>handle</i>	Handle for the dyninst snippet.

### 4.119.3 Member Function Documentation

**4.119.3.1** `int SnippetHandler::GetEventId ( )` `const` `[inline]`

Getter for the variable `_eventId`.

#### Returns

Id of the event.

**4.119.3.2** `std::string const& SnippetHandler::GetFuncName ( )` `const` `[inline]`

Getter for the variable `_funcName`.

#### Returns

Name of the function in which the snippet will be inserted.

#### 4.119.3.3 BPatchSnippetHandle\* SnippetHandler::GetHandle ( ) const [inline]

Getter for the variable `_handle`.

##### Returns

Handle of the snippet to be inserted.

#### 4.119.3.4 InstrPlace SnippetHandler::GetInstrPlace ( ) const [inline]

Getter for the variable `_place`.

##### Returns

Position of the function in which the snippet will be inserted.

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

- AC/InstrSet.h

## 4.120 SnippetMaker Class Reference

Prepares the snippets to be inserted into the processes.

```
#include <SnippetMaker.h>
```

### Public Member Functions

- [SnippetMaker](#) ([DiProcess](#) &process, [DiImage](#) &image)  
*Constructor.*
- BPatchSnippetHandle \* [MakeEventSnippet](#) (int eventId, std::string const &func-Name, InstrPlace instrPlace, int nAttrs, Attribute \*attrs)  
*Creates and inserts a snippet into the running process.*

#### 4.120.1 Detailed Description

Prepares the snippets to be inserted into the processes.

##### Version

1.0

##### Since

1.0

**Author**

Ania Morajko, 2002

**4.120.2 Constructor & Destructor Documentation****4.120.2.1 SnippetMaker::SnippetMaker ( DiProcess & *process*, DiImage & *image* )**  
[inline]

Constructor.

**Parameters**

<i>process</i>	Dyninst process to be modified.
<i>image</i>	Dyninst image of the process to be modified.

**4.120.3 Member Function Documentation****4.120.3.1 BPatchSnippetHandle \* SnippetMaker::MakeEventSnippet ( int *eventId*, std::string const & *funcName*, InstrPlace *instrPlace*, int *nAttrs*, Attribute \* *attrs* )**

Creates and inserts a snippet into the running process.

**Parameters**

<i>eventId</i>	Identifier of the event.
<i>funcName</i>	Name of the function to be modified.
<i>instrPlace</i>	Place in the function where the snippet will be inserted.
<i>nAttrs</i>	Number of attributes.
<i>attrs</i>	Array of attributes.

**Returns**

Handle for the prepared snippet.

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

- AC/SnippetMaker.h
- AC/SnippetMaker.cpp

**4.121 Common::Socket Class Reference**

Holds a [SocketBase](#) object and represents a client socket.

```
#include <Socket.h>
```

**Public Member Functions**

- [Socket](#) ([Address](#) &address)

*Constructor.*

- [Socket](#) (std::string const &host, int port)

*Constructor.*

- [Address](#) const & [GetRemoteAddress](#) () const

*Returns the address to which the socket is connected.*

- [Address](#) [GetLocalAddress](#) () const

*Returns the local address the socket is listening to.*

- int [GetLocalPort](#) () const

*Returns the local port the socket is listening.*

- void [Send](#) (char const \*buf, int bufSize, int flags=0)

*Sends the data through the socket converted to network byte order.*

- void [Send](#) (std::string const &str, int flags=0)

*Sends the data through the socket converted to network byte order.*

- void [Send](#) ([ByteStream](#) &stream, int flags=0)

*Sends the data through the socket converted to network byte order.*

- int [Receive](#) (char \*buf, int bufSize, int flags=0)

*Gets data from the socket.*

- int [ReceiveN](#) (char \*buf, int bufSize, int flags=0)

*Gets data from the socket, performs multiple Recive() calls until the buffer is full.*

- [operator int](#) ()

*Cast to int returns the socket handle.*

- int [GetHandle](#) () const

*Returns socket handle.*

- void [SetTCPNoDelay](#) (bool value)

*Sets the TCPNoDelay property.*

- void [SetReuseAddress](#) (bool value)

*Sets the ReuseAddress property.*

- void [SetKeepAlive](#) (bool value)

*Sets the KeepAlive property.*

- void [SetReceiveTimeout](#) (int timeoutMs)

*Sets the ReceiveTimeout property.*



- void [SetSendTimeout](#) (int timeoutMs)  
*Sets the SendTimeout property.*
- int [GetReceiveTimeout](#) ()  
*Gets the ReceiveTimeout property.*
- int [GetSendTimeout](#) ()  
*Gets the SendTimeout property.*

### Protected Member Functions

- [Socket](#) (int hSocket, [Address](#) &addr)  
*Constructor.*

### Friends

- class [SocketBase](#)

#### 4.121.1 Detailed Description

Holds a [SocketBase](#) object and represents a client socket. All the functions on this class are present on the [SocketBase](#) class and have the same functionality.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002

#### 4.121.2 Constructor & Destructor Documentation

##### 4.121.2.1 [Socket::Socket](#) ( [Address](#) & *address* ) [[inline](#)]

Constructor.

#### Exceptions

<a href="#">SysException</a>	
------------------------------	--

#### 4.121.2.2 Socket::Socket ( std::string const & *host*, int *port* ) [inline]

Constructor.

##### Exceptions

<a href="#">SysException</a>
------------------------------

#### 4.121.3 Member Function Documentation

##### 4.121.3.1 int Socket::GetReceiveTimeout ( ) [inline]

Gets the ReceiveTimeout property.

##### Exceptions

<a href="#">SysException</a>
------------------------------

##### 4.121.3.2 int Socket::GetSendTimeout ( ) [inline]

Gets the SendTimeout property.

##### Exceptions

<a href="#">SysException</a>
------------------------------

##### 4.121.3.3 int Socket::Receive ( char \* *buf*, int *bufSize*, int *flags* = 0 ) [inline]

Gets data from the socket.

If no incoming data is available at the socket, the call blocks and waits for data to arrive.

##### Returns

Number of bytes received.

##### Exceptions

<a href="#">SysException</a>
------------------------------

##### 4.121.3.4 int Socket::ReceiveN ( char \* *buf*, int *bufSize*, int *flags* = 0 ) [inline]

Gets data from the socket, performs multiple Recive() calls until the buffer is full.

##### Returns

Number of bytes received.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.5** void Socket::Send ( std::string const & *str*, int *flags* = 0 ) [inline]

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.6** void Socket::Send ( ByteStream & *stream*, int *flags* = 0 )

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.7** void Socket::Send ( char const \* *buf*, int *bufSize*, int *flags* = 0 ) [inline]

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.8** void Socket::SetKeepAlive ( bool *value* ) [inline]

Sets the KeepAlive property.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.9** void Socket::SetReceiveTimeout ( int *timeoutMs* ) [inline]

Sets the ReceiveTimeout property.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.121.3.10 void Socket::SetReuseAddress ( bool *value* ) [inline]**

Sets the ReuseAddress property.

**Exceptions**

<a href="#">SysException</a>
------------------------------

**4.121.3.11 void Socket::SetSendTimeout ( int *timeoutMs* ) [inline]**

Sets the SendTimeout property.

**Exceptions**

<a href="#">SysException</a>
------------------------------

**4.121.3.12 void Socket::SetTCPNoDelay ( bool *value* ) [inline]**

Sets the TCPNoDelay property.

**Exceptions**

<a href="#">SysException</a>
------------------------------

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

- Common/Socket.h
- Common/Socket.cpp

## 4.122 Common::SocketBase Class Reference

Represents an endpoint for communication between two machines.

```
#include <Socket.h>
```

**Public Member Functions**

- [SocketBase](#) (int family=AF\_INET, int type=SOCK\_STREAM, int protocol=0)  
*Constructor.*
- [SocketBase](#) (int hSocket, [Address](#) &addr)  
*Constructor.*
- virtual [~SocketBase](#) ()

*Destructor:*

- SocketPtr [Accept](#) ()  
*Accepts a connection and creates a socket.*
- void [Bind](#) (int port)  
*Associates the socket with a local endpoint.*
- void [Listen](#) (int backLog)  
*Sets the socket to a listening state.*
- void [Connect](#) ([Address](#) &address)  
*Connects a socket on the given address.*
- void [Connect](#) (std::string const &host, int port)  
*Connects a socket on the given address.*
- void [Send](#) (char const \*buf, int bufSize, int flags=0)  
*Sends the data through the socket converted to network byte order.*
- void [Send](#) (std::string const &str, int flags=0)  
*Sends the data through the socket converted to network byte order.*
- void [Send](#) ([ByteStream](#) &stream, int flags=0)  
*Sends the data through the socket converted to network byte order.*
- int [Receive](#) (char \*buf, int bufSize, int flags=0)  
*Gets data from the socket.*
- int [ReceiveN](#) (char \*buf, int bufSize, int flags=0)  
*Gets data from the socket, performs multiple Recive() calls until the buffer is full.*
- operator int ()  
*Cast to int returns the socket handle.*
- int [GetHandle](#) () const  
*Returns socket handle.*
- void [SetTCPNoDelay](#) (bool value)  
*Sets the TCPNoDelay property.*
- void [SetKeepAlive](#) (bool value)  
*Sets the KeepAlive property.*
- void [SetReuseAddress](#) (bool value)  
*Sets the ReuseAddress property.*

- void [SetReceiveTimeout](#) (int timeoutMs)  
*Sets the ReceiveTimeout property.*
- void [SetSendTimeout](#) (int timeoutMs)  
*Sets the SendTimeout property.*
- int [GetReceiveTimeout](#) ()  
*Gets the ReceiveTimeout property.*
- int [GetSendTimeout](#) ()  
*Gets the SendTimeout property.*
- [Address](#) const & [GetAddress](#) () const  
*Returns the address where the socket is connected.*
- int [GetLocalPort](#) () const  
*Returns the port the socket is listening.*

### Protected Member Functions

- void [SetOption](#) (int level, int option, char const \*value, int valueSize)  
*Sets a socket option.*
- int [GetOption](#) (int level, int option, char \*value, int &valueSize)  
*Gets the value of a socket option.*
- int [DoSend](#) (char const \*buf, int bufSize, int flags=0)  
*Sends the data through the socket.*

### Protected Attributes

- int [\\_hSocket](#)
- [Address](#) [\\_addr](#)
- int [\\_localPort](#)

#### 4.122.1 Detailed Description

Represents an endpoint for communication between two machines. This class works as an adapter for the socket functions included on the sys/socket library.

#### Version

1.0b

**Since**

1.0b

**Author**

Ania Morajko, 2002

**4.122.2 Constructor & Destructor Documentation****4.122.2.1 SocketBase::SocketBase ( int *family* = AF\_INET, int *type* = SOCK\_STREAM, int *protocol* = 0 )**

Constructor.

**Parameters**

<i>family</i>	<a href="#">Socket</a> family, default AF_INET.
<i>type</i>	<a href="#">Socket</a> type, default SOCK_STREAM.
<i>protocol</i>	<a href="#">Socket</a> protocol, default 0.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.122.2.2 SocketBase::SocketBase ( int *hSocket*, Address & *addr* )**

Constructor.

Sets the socket to the given handle with the given address.

**Parameters**

<i>hSocket</i>	<a href="#">Socket</a> handle.
<i>addr</i>	<a href="#">Socket</a> address.

**4.122.3 Member Function Documentation****4.122.3.1 void SocketBase::Bind ( int *port* )**

Associates the socket with a local endpoint.

**Parameters**

<i>port</i>	Port where the socket will perform the connection.
-------------	--

**4.122.3.2** `int SocketBase::DoSend ( char const * buf, int bufSize, int flags = 0 )`  
[protected]

Sends the data through the socket.

#### Exceptions

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.3** `int SocketBase::GetOption ( int level, int option, char * value, int & valueSize )`  
[protected]

Gets the value of a socket option.

#### Parameters

<i>level</i>	Protocol level at which the option resides.
<i>option</i>	Option name.
<i>value</i>	Buffer to save the value.
<i>valueSize</i>	Integer to save the size of the value.

#### Exceptions

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.4** `int SocketBase::GetReceiveTimeout ( )`

Gets the ReceiveTimeout property.

#### Exceptions

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.5** `int SocketBase::GetSendTimeout ( )`

Gets the SendTimeout property.

#### Exceptions

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.6** `void SocketBase::Listen ( int backLog )`

Sets the socket to a listening state.



**Parameters**

<i>backLog</i>	Maximum length of the queue for pending connections.
----------------	--

**4.122.3.7 int SocketBase::Receive ( char \* *buf*, int *bufSize*, int *flags* = 0 )**

Gets data from the socket.

If no incoming data is available at the socket, the call blocks and waits for data to arrive.

**Returns**

Number of bytes received.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.122.3.8 int SocketBase::ReceiveN ( char \* *buf*, int *bufSize*, int *flags* = 0 )**

Gets data from the socket, performs multiple Recive() calls until the buffer is full.

**Returns**

Number of bytes received.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.122.3.9 void SocketBase::Send ( ByteStream & *stream*, int *flags* = 0 )**

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.122.3.10 void SocketBase::Send ( char const \* *buf*, int *bufSize*, int *flags* = 0 )**

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#"><i>SysException</i></a>	
-------------------------------------	--

**4.122.3.11 void SocketBase::Send ( std::string const & *str*, int *flags* = 0 )**

Sends the data through the socket converted to network byte order.

**Exceptions**

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.12 void SocketBase::SetKeepAlive ( bool *value* )**

Sets the KeepAlive property.

**Exceptions**

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.13 void SocketBase::SetOption ( int *level*, int *option*, char const \* *value*, int *valueSize* ) [protected]**

Sets a socket option.

**Parameters**

<i>level</i>	Protocol level at which the option resides.
<i>option</i>	Option name.
<i>value</i>	Option value to set.
<i>valueSize</i>	Size of the value.

**Exceptions**

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.14 void SocketBase::SetReceiveTimeout ( int *timeoutMs* )**

Sets the ReceiveTimeout property.

**Exceptions**

<a href="#"><i>SysException</i></a>
-------------------------------------

**4.122.3.15 void SocketBase::SetReuseAddress ( bool *value* )**

Sets the ReuseAddress property.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.122.3.16 void SocketBase::SetSendTimeout ( int *timeoutMs* )**

Sets the SendTimeout property.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

**4.122.3.17 void SocketBase::SetTCPNoDelay ( bool *value* )**

Sets the TCPNoDelay property.

**Exceptions**

<a href="#">SysException</a>	
------------------------------	--

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

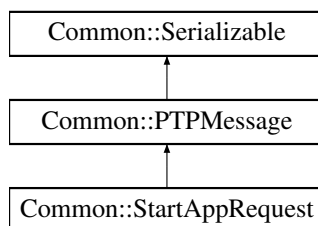
- Common/Socket.h
- Common/Socket.cpp

## 4.123 Common::StartAppRequest Class Reference

Represents a request to start the application.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::StartAppRequest:

**Public Member Functions**

- [StartAppRequest](#) (std::string const &appPath=std::string(), int argc=0, char const \*\*argv=0, std::string const &analyzerHost=std::string())

*Constructor.*

- [~StartAppRequest \(\)](#)

*Destructor.*

- PTPMsgType [GetType \(\)](#) const  
*Returns type of message (PTPStartApp).*
- std::string const & [GetAppPath \(\)](#) const  
*Returns the application path.*
- std::string const & [GetAnalyzerHost \(\)](#) const  
*Returns the analyzer host name.*
- char \*\* [GetArgs \(\)](#) const  
*Returns a pointer to the array of arguments.*
- int [GetArgCount \(\)](#) const  
*Returns number of arguments given.*
- void [Serialize \(Serializer &out\)](#) const  
*Sends the message.*
- void [DeSerialize \(DeSerializer &in\)](#)  
*Receives the message.*

#### 4.123.1 Detailed Description

Represents a request to start the application.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2003

#### 4.123.2 Constructor & Destructor Documentation

- ##### 4.123.2.1 StartAppRequest::StartAppRequest ( std::string const & *appPath* = std::string(), int *argc* = 0, char const \*\* *argv* = 0, std::string const & *analyzerHost* = std::string() )

Constructor.

**Parameters**

<i>appPath</i>	Application path, default 0.
<i>argc</i>	Argument count, default 0.
<i>argv</i>	Arguments array, default 0.
<i>analyzer-Host</i>	Host of the analyzer, default 0.

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

**4.124 Stats Struct Reference****Public Attributes**

- double **desv**
- double **mean**

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

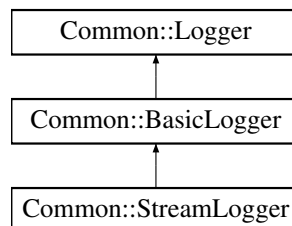
- Analyzer/FactoringTunlet\_nw.h

**4.125 Common::StreamLogger Class Reference**

Stores the logged information into a stream.

```
#include <Syslog.h>
```

Inheritance diagram for Common::StreamLogger:

**Public Member Functions**

- [StreamLogger](#) (std::ostream &stream)  
*Constructor.*

- [~StreamLogger](#) ()  
*Destructor.*
- void [Log](#) ([LogEntry](#) const &entry)  
*Inserts an entry to the log.*

#### 4.125.1 Detailed Description

Stores the logged information into a stream.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/Syslog.h
- Common/Syslog.cpp

### 4.126 Common::StringArray Class Reference

Container of strings.

```
#include <StringArray.h>
```

#### Public Member Functions

- [StringArray](#) (int size=0)  
*Constructor.*
- [~StringArray](#) ()  
*Destructor.*
- void [AddString](#) (char const \*s)  
*Adds a string to the array.*
- void [Grow](#) (int newSize)  
*Increments max size of the array.*

- int [GetCount](#) () const  
*Returns number of strings currently stored.*
- int [GetSize](#) () const  
*Returns max size of the array.*
- char const \* [GetString](#) (int idx) const  
*Returns string stored on the given position.*
- char \*\* [GetAccess](#) () const  
*Returns a pointer to the actual array.*
- void [Dump](#) () const  
*Writes current state of the array on the standard output.*

#### 4.126.1 Detailed Description

Container of strings.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.126.2 Constructor & Destructor Documentation

##### 4.126.2.1 StringArray::StringArray ( int size = 0 )

Constructor.

##### Parameters

<i>size</i>	Size of the array, default 0.
-------------	-------------------------------

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

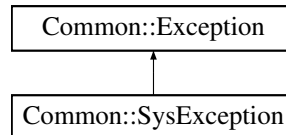
- Common/StringArray.h
- Common/StringArray.cpp

## 4.127 Common::SysException Class Reference

System exception.

```
#include <SysException.h>
```

Inheritance diagram for Common::SysException:



### Public Member Functions

- [SysException](#) (std::string const &msg, std::string const &objName=std::string())

*Constructor.*

- [SysException](#) (std::string const &msg, long errorCode)

*Constructor.*

- void [Display](#) (std::ostream &os) const

*Displays exception message on the given output stream.*

- void [Display](#) () const

*Displays exception message on the standard error output.*

- std::string [GetReason](#) () const

*Returns a string containing the error message.*

### 4.127.1 Detailed Description

System exception.

#### Version

1.0b

#### Since

1.0b

#### Author

Ania Morajko, 2002



### 4.127.2 Constructor & Destructor Documentation

**4.127.2.1** `Common::SysException::SysException ( std::string const & msg, std::string const & objName = std::string () ) [inline]`

Constructor.

#### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>objName</i>	Name of the object causing the exception, "" by default.

**4.127.2.2** `Common::SysException::SysException ( std::string const & msg, long errorCode ) [inline]`

Constructor.

#### Parameters

<i>msg</i>	<a href="#">Exception</a> message.
<i>errorCode</i>	<a href="#">Exception</a> error code.

### 4.127.3 Member Function Documentation

**4.127.3.1** `void Common::SysException::Display ( std::ostream & os ) const [virtual]`

Displays exception message on the given output stream.

#### Parameters

<i>os</i>	Output stream to display the message.
-----------	---------------------------------------

Reimplemented from [Common::Exception](#).

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

- Common/SysException.h
- Common/SysException.cpp

## 4.128 Common::Syslog Class Reference

Holds and manages a loggers on the system.

```
#include <Syslog.h>
```

## Public Types

- typedef [auto\\_vector](#)< [Logger](#) > **LoggerVector**
- typedef [auto\\_iterator](#)< [Logger](#) > **LoggerIterator**

## Static Public Member Functions

- static void [Configure](#) ()  
*Configures the system with a default configuration.*
- static void [Configure](#) ([Config](#) &cfg, string loggerName="")  
*Configures the logger with a given [Config](#).*
- static void [LogEvent](#) (LogSeverity s, std::string const &message)  
*Adds an entry with the given event to all the loggers.*
- static void [Debug](#) (std::string const &message)  
*Logs an event with **DEBUG** level of severity.*
- static void [Info](#) (std::string const &message)  
*Logs an event with **INFO** level of severity.*
- static void [Warn](#) (std::string const &message)  
*Logs an event with **WARNING** level of severity.*
- static void [Error](#) (std::string const &message)  
*Logs an event with **ERROR** level of severity.*
- static void [Fatal](#) (std::string const &message)  
*Logs an event with **FATAL** level of severity.*
- static void [Debug](#) (char \*formatStr,...)  
*Logs an event with **DEBUG** level of severity.*
- static void [Info](#) (char \*formatStr,...)  
*Logs an event with **INFO** level of severity.*
- static void [Warn](#) (char \*formatStr,...)  
*Logs an event with **WARNING** level of severity.*
- static void [Error](#) (char \*formatStr,...)  
*Logs an event with **ERROR** level of severity.*
- static void [Fatal](#) (char \*formatStr,...)  
*Logs an event with **FATAL** level of severity.*

- static bool [CanWrite](#) ()  
*Returns true if the system log is enabled, false otherwise.*
- static [Logger](#) const \* [GetLogger](#) (std::string const &name)  
*Returns logger with given name.*
- static void [AddLogger](#) (LoggerPtr &logger)  
*Adds a new logger to the system log.*
- static void [RemoveLogger](#) (std::string const &name)  
*Not implemented.*
- static [LoggerIterator](#) [GetLoggers](#) ()  
*Returns an iterator pointing to the first logger.*

#### 4.128.1 Detailed Description

Holds and manages a loggers on the system. This is a static class that can't be instantiated.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.128.2 Member Function Documentation

##### 4.128.2.1 void Syslog::Configure ( ) [static]

Configures the system with a default configuration.

This configuration uses an only [Logger](#) that outputs it's logs on the standard error output.

##### Exceptions

<a href="#">SysException</a>	
------------------------------	--

**4.128.2.2 void Syslog::Debug ( std::string const & *message* ) [inline, static]**

Logs an event with DEBUG level of severity.

Uses a string as a parameter.

**4.128.2.3 void Syslog::Debug ( char \* *formatStr*, ... ) [static]**

Logs an event with DEBUG level of severity.

Uses a char \* as a parameter.

**4.128.2.4 void Syslog::Error ( char \* *formatStr*, ... ) [static]**

Logs an event with ERROR level of severity.

Uses a char \* as a parameter.

**4.128.2.5 void Syslog::Error ( std::string const & *message* ) [inline, static]**

Logs an event with ERROR level of severity.

Uses a string as a parameter.

**4.128.2.6 void Syslog::Fatal ( std::string const & *message* ) [inline, static]**

Logs an event with FATAL level of severity.

Uses a string as a parameter.

**4.128.2.7 void Syslog::Fatal ( char \* *formatStr*, ... ) [static]**

Logs an event with FATAL level of severity.

Uses a char \* as a parameter.

**4.128.2.8 void Syslog::Info ( std::string const & *message* ) [inline, static]**

Logs an event with INFO level of severity.

Uses a string as a parameter.

**4.128.2.9 void Syslog::Info ( char \* *formatStr*, ... ) [static]**

Logs an event with INFO level of severity.

Uses a char \* as a parameter.

**4.128.2.10 void Syslog::Warn ( std::string const & message ) [inline, static]**

Logs an event with WARNING level of severity.

Uses a string as a parameter.

**4.128.2.11 void Syslog::Warn ( char \* formatStr, ... ) [static]**

Logs an event with WARNING level of severity.

Uses a char \* as a parameter.

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

- Common/Syslog.h
- Common/Syslog.cpp

**4.129 Task Class Reference**

Represents each of the processes that we can modify using dyninst.

```
#include <Task.h>
```

**Public Member Functions**

- [Task](#) (const std::string &path, char \*args[], TimeValue const &clockDiff, string const &analyzerHost, int analyzerPort, int debugLevel, int debugStdErr, string const &DMLibName)

*Constructor.*

- [~Task](#) ()

*Destructor.*

- int [GetPid](#) ()

*Getter of the identifier of the process being modified.*

- int [GetMpiRank](#) ()

*Getter of the MPIRank attribute.*

- void [Continue](#) ()

*Restarts the execution of the process after breakpoint.*

- void [WaitFor](#) ()

*Waits for the process to be terminated.*

- [DiProcess](#) & [GetProcess](#) ()

*Getter of the variable \_process.*

- [DiImage](#) & [GetImage](#) ()  
*Getter of the variable `_image`.*
- [TaskInstr](#) & [GetInstr](#) ()  
*Getter of the variable `_instr`.*
- bool [IsStopped](#) ()  
*Checks if the process is running.*
- bool [Terminate](#) ()  
*Terminates a running process and invokes the callback function if exists.*
- bool [IsTerminated](#) ()  
*Checks if the process is terminated.*
- void [AddDelayedTuning](#) ([Common::TuningRequest](#) \*req)  
*Adds a tuning request to the pending list.*
- bool [IsStoppedOnBreakpoint](#) ()  
*Checks if the process is stopped in a breakpoint.*
- void [ProcessBreakpoint](#) ([Tuner](#) &t)  
*Is called when the process hits a breakpoint to handle it.*
- void [UnloadLibrary](#) ()  
*Unload the DMLib from a process which has terminated.*

#### 4.129.1 Detailed Description

Represents each of the processes that we can modify using dyninst. Provides the necessary function to manage the process to be modified and controlling its execution during the modifications.

##### Version

1.0

##### Since

1.0

##### Author

Ania Morajko, 2002

## 4.129.2 Constructor & Destructor Documentation

**4.129.2.1** `Task::Task ( const std::string & path, char * args[], TimeValue const & clockDiff, string const & analyzerHost, int analyzerPort, int debugLevel, int debugStdErr, string const & DMLibName )`

Constructor.

### Parameters

<i>path</i>	Path to the executable of the application.
<i>args</i> [ ]	Array that contains the arguments with which the application should be executed.
<i>clockDiff</i>	Correction of the clock difference.
<i>analyzerHost</i>	Address of the host in which the analyzer is running.
<i>analyzerPort</i>	Port through which the analyzer communicates.
<i>debugLevel</i>	Selected level of debugging.
<i>debugStdErr</i>	Debug messages output.
<i>DMLib-Name</i>	Name of the dynamic library to be loaded into the process.

## 4.129.3 Member Function Documentation

**4.129.3.1** `void Task::AddDelayedTuning ( Common::TuningRequest * req )`

Adds a tuning request to the pending list.

### Parameters

<i>req</i>	Request for tuning procedure.
------------	-------------------------------

**4.129.3.2** `DiImage& Task::GetImage ( ) [inline]`

Getter of the variable `_image`.

### Returns

Image of the process thats being modified.

**4.129.3.3** `TaskInstr& Task::GetInstr ( ) [inline]`

Getter of the variable `_instr`.

### Returns

Instrumentation to be inserted.

**4.129.3.4** `int Task::GetPid ( ) [inline]`

Getter of the identifier of the process being modified.

**Returns**

Pid of the process being modified.

**4.129.3.5** `DiProcess& Task::GetProcess ( ) [inline]`

Getter of the variable `_process`.

**Returns**

Process thats being modified.

**4.129.3.6** `bool Task::IsStopped ( ) [inline]`

Checks if the process is running.

**Returns**

false if its running true if stopped.

**4.129.3.7** `bool Task::IsStoppedOnBreakpoint ( ) [inline]`

Checks if the process is stopped in a breakpoint.

**Returns**

True if process is currently stopped and situated in a breakpoint .

**4.129.3.8** `bool Task::IsTerminated ( ) [inline]`

Checks if the process is terminated.

**Returns**

True if the process has exited.

**4.129.3.9** `void Task::ProcessBreakpoint ( Tuner & t )`

Is called when the process hits a breakpoint to handle it.

**Parameters**



<i>t</i>	<a href="#">Tuner</a> that will apply the changes specified in the request for each task.
----------	---

#### 4.129.3.10 bool Task::Terminate ( ) [inline]

Terminates a running process and invokes the callback function if exists.

#### Returns

true for success, false for failure.

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

- AC/Task.h
- AC/Task.cpp

## 4.130 Model::Task Class Reference

Encapsulates information to define the tasks that form the application. The data structure of a task consist of identification data (pid, mpiRank, name), status data, where is it running (host), which events are being collected from it and if it is either a master task or not. Provides methods to:

```
#include <AppTask.h>
```

#### Public Member Functions

- int [GetPid](#) () const  
*pid getter.*
- int [GetMpiRank](#) () const  
*MPI Rank getter.*
- string [GetName](#) () const  
*name getter.*
- [Host](#) & [GetHost](#) () const  
*host getter.*
- bool [IsRunning](#) () const  
*indicates if the task is still running.*
- bool [IsMaster](#) () const  
*indicates if the task is the master task.*

- Status [GetStatus](#) () const  
*status getter.*
- void [AddEvent](#) ([Event](#) const &e)  
*Adds a definition of new event to be traced in this task.*
- bool [RemoveEvent](#) (int eventId, InstrPlace place)  
*Removes previously added event from this task.*
- void [LoadLibrary](#) (string const &libPath)  
*Loads a shared library to this task. This enables the [Analyzer](#) to load any additional code required for the tuning.*
- void [SetVariableValue](#) (string const &varName, AttributeValue const &varValue, Breakpoint \*brkpt)  
*Modifies a value of a specified variable in the running task application process.*
- void [ReplaceFunction](#) (string const &oldFunc, string const &newFunc, Breakpoint \*brkpt)  
*Replaces all calls to a function with calls to another one in this task.*
- void [InsertFunctionCall](#) (string const &funcName, int nAttrs, Attribute \*attrs, string const &destFunc, InstrPlace destPlace, Breakpoint \*brkpt)  
*Inserts a new function invocation code at a given location in this task.*
- void [OneTimeFuncCall](#) (string const &funcName, int nAttrs, Attribute \*attrs, Breakpoint \*brkpt)  
*inserts a new function invocation code in this task and invokes it once.*
- void [RemoveFuncCall](#) (string const &funcName, string const &callerFunc, Breakpoint \*brkpt)  
*removes all calls to a given function from the given caller function in this task. For example this method can be used to remove all flush() function calls from a debug() function.*
- void [FuncParamChange](#) (string const &funcName, int paramIdx, int newValue, int \*requiredOldValue, Breakpoint \*brkpt)  
*sets the value of an input parameter of a given function in this task. This parameter value is modified before the function body is invoked. There is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.*
- void [SetTaskExitHandler](#) ([TaskHandler](#) &h)  
*installs a callback function that is called when this task terminates.*

## Protected Member Functions

- **Task** (int pid, int mpiRank, string const &name, [Host](#) &h)  
*Constructor.*
- void **SetMaster** (bool value)  
*Sets if this task is Master or not.*
- [ACProxy](#) \* **GetACProxy** ()
- void **DispatchEvent** (EventMsg const &msg)

## Friends

- class **Application**

### 4.130.1 Detailed Description

Encapsulates information to define the tasks that form the application. The data structure of a task consist of identification data (pid, mpiRank, name), status data, where is it running (host), which events are being collected from it and if it is either a master task or not. Provides methods to:

- Retrieve application information
- Monitoring: add/remove events to trace.
- Tuning: loading libraries, changing variables & parameter values, adding/removing function calls and calling them explicitly.

### 4.130.2 Constructor & Destructor Documentation

4.130.2.1 **Task::Task** ( int *pid*, int *mpiRank*, string const & *name*, [Host](#) & *h* )  
[protected]

Constructor.

#### Parameters

<i>pid</i>	globally unique task id.
<i>mpiRank</i>	id associated to MPI
<i>name</i>	process name.
<i>h</i>	reference to the host object this task is running on.

### 4.130.3 Member Function Documentation

#### 4.130.3.1 void Task::AddEvent ( Event const & e )

Adds a definition of new event to be traced in this task.

##### Parameters

<i>e</i>	event to be traced.
----------	---------------------

##### Returns

number of tasks where the event tracing was added.

#### 4.130.3.2 void Task::FuncParamChange ( string const & funcName, int paramIdx, int newValue, int \* requiredOldValue, Breakpoint \* brkpt )

sets the value of an input parameter of a given function in this task. This parameter value is modified before the function body is invoked. There is also possible to change the parameter value under condition, namely if the parameter has a value equal to requiredOldValue, only then its value is changed to new one. If the requiredOldValue is zero, then the value of the parameter is changed unconditionally.

##### Parameters

<i>funcName</i>	name of the function
<i>paramIdx</i>	id of the parameter to change
<i>newValue</i>	new value for the parameter
<i>re- quiredOld- Value</i>	required old value of the parameter to change it
<i>brkpt</i>	---

##### Returns

number of tasks where the parameter was changed.

#### 4.130.3.3 ACPProxy \* Task::GetACPProxy ( ) [protected]

##### Returns

[ACPProxy](#) object of this task.

#### 4.130.3.4 Host& Model::Task::GetHost ( ) const [inline]

host getter.

Returns

reference to the host object this task is running on.

4.130.3.5 int Model::Task::GetMpiRank ( ) const [inline]

MPI Rank getter.

Returns

MPI rank of the task.

4.130.3.6 string Model::Task::GetName ( ) const [inline]

name getter.

Returns

process name

4.130.3.7 int Model::Task::GetPid ( ) const [inline]

pid getter.

Returns

globally unique process id

4.130.3.8 Status Model::Task::GetStatus ( ) const [inline]

status getter.

Returns

task status information

4.130.3.9 void Task::InsertFunctionCall ( string const & funcName, int nAttrs, Attribute \* attrs, string const & destFunc, InstrPlace destPlace, Breakpoint \* brkpt )

Inserts a new function invocation code at a given location in this task.

Parameters

<i>funcName</i>	name of the function to call.
<i>nAttrs</i>	number of parameters of the function.
<i>attrs</i>	values for each parameter.
<i>destFunc</i>	function where the calls will be placed.
<i>destPlace</i>	point of the function where the calls will be placed.
<i>brkpt</i>	---

**Returns**

number of tasks where the function calls were added.

**4.130.3.10 bool Model::Task::IsMaster ( ) const [inline]**

indicates if the task is the master task.

**Returns**

true if master false otherwise.

**4.130.3.11 bool Model::Task::IsRunning ( ) const [inline]**

indicates if the task is still running.

**Returns**

true if still running false otherwise.

**4.130.3.12 void Task::LoadLibrary ( string const & libPath )**

Loads a shared library to this task. This enables the [Analyzer](#) to load any additional code required for the tuning.

**Parameters**

<i>libPath</i>	path to the library.
----------------	----------------------

**Returns**

number of tasks where the library is loaded.

**4.130.3.13 void Task::OneTimeFuncCall ( string const & funcName, int nAttrs, Attribute \* attrs, Breakpoint \* brkpt )**

inserts a new function invocation code in this task and invokes it once.

**Parameters**

<i>funcName</i>	name of the function to call
<i>nAttrs</i>	number of arguments of the function
<i>attrs</i>	values for each argument of the function
<i>brkpt</i>	---

**Returns**

number of tasks where the function was call.

**4.130.3.14 bool Task::RemoveEvent ( int *eventId*, InstrPlace *place* )**

Removes previously added event from this task.

**Parameters**

<i>eventId</i>	id of the event
<i>place</i>	place of the function where the event is recorded?

**Returns**

number of tasks where the event was removed.

**4.130.3.15 void Task::RemoveFuncCall ( string const & *funcName*, string const & *callerFunc*, Breakpoint \* *brkpt* )**

removes all calls to a given function from the given caller function in this task. For example this method can be used to remove all flush() function calls from a debug() function.

**Parameters**

<i>funcName</i>	name of the function
<i>callerFunc</i>	function that calls the function that is will be removed
<i>brkpt</i>	---

**Returns**

number of tasks where the function call is removed.

**4.130.3.16 void Model::Task::ReplaceFunction ( string const & *oldFunc*, string const & *newFunc*, Breakpoint \* *brkpt* )**

Replaces all calls to a function with calls to another one in this task.

**Parameters**

<i>oldFunc</i>	name of the function to replace.
<i>newFunc</i>	name of the new function.
<i>brkpt</i>	---

**Returns**

number of tasks where the function calls were changed.

#### 4.130.3.17 void Model::Task::SetMaster ( bool *value* ) [inline, protected]

Sets if this task is Master or not.

##### Parameters

<i>value</i>	determines if its Master or not.
--------------	----------------------------------

#### 4.130.3.18 void Task::SetVariableValue ( string const & *varName*, AttributeValue const & *varValue*, Breakpoint \* *brkpt* )

Modifies a value of a specified variable in the running task application process.

##### Parameters

<i>varName</i>	name of the variable.
<i>varValue</i>	new value for the variable.
<i>brkpt</i>	---

##### Returns

number of tasks where the values were changed.

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

- Analyzer/AppTask.h
- Analyzer/ACProxy.cpp
- Analyzer/AppModel.cpp
- Analyzer/AppTask.cpp

## 4.131 TaskCollection Class Reference

Groups task in a single, easy to handle, collection.

```
#include <Tasks.h>
```

### Public Types

- enum { **NotFound** = -1 }

### Public Member Functions

- [TaskCollection](#) ()  
*Constructor.*
- void [Add](#) (auto\_ptr< [Task](#) > &task)



*Adds a new task to the collection.*

- void **Delete** (int index)  
*Removes a task from the collection.*
- void **Clear** ()  
*Erase all the elements of the array.*
- **Task** const \* **operator[]** (int index) const  
*Enables the use of the [] to select an element from the collection.*
- **Task** \* **operator[]** (int index)  
*Enables the use of the [] to select an element from the collection.*
- int **FindByPid** (int pid)
- int **GetCount** () const
- **Task** & **GetByPid** (int pid)

#### 4.131.1 Detailed Description

Groups task in a single, easy to handle, collection. Provides collection methods to a group of tasks to facilitate the handling of many tasks at once.

#### 4.131.2 Member Function Documentation

##### 4.131.2.1 void TaskCollection::Add ( auto\_ptr< Task > & task ) [inline]

Adds a new task to the collection.

##### Parameters

<i>task</i>	Pointer to the task to be added.
-------------	----------------------------------

##### 4.131.2.2 void TaskCollection::Delete ( int index ) [inline]

Removes a task from the collection.

##### Parameters

<i>index</i>	Position in the array of the task to be removed.
--------------	--

##### 4.131.2.3 Task const\* TaskCollection::operator[] ( int index ) const [inline]

Enables the use of the [] to select an element from the collection.

**Parameters**

<i>index</i>	Position in the array.
--------------	------------------------

**Returns**

Constant value of a pointer to the selected task.

**4.131.2.4 Task\* TaskCollection::operator[] ( int *index* ) [inline]**

Enables the use of the [] to select an element from the collection.

**Parameters**

<i>index</i>	Position in the array.
--------------	------------------------

**Returns**

Pointer to the selected task.

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

- AC/Tasks.h

**4.132 TaskExitHandler Class Reference**

Contains a virtual function to handle the exit of a task.

```
#include <TaskManager.h>
```

**Public Member Functions**

- virtual void [HandleTaskExit](#) (Task const &task, int exitCode)=0  
*installs a callback function that is called when the task terminates.*

**4.132.1 Detailed Description**

Contains a virtual function to handle the exit of a task.

**Version**

1.0

**Since**

1.0

**Author**

Ania Morajko, 2002

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

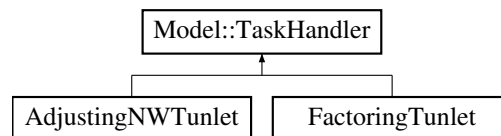
- AC/TaskManager.h

**4.133 Model::TaskHandler Class Reference**

Abstract class that provides methods to determine if a task is started or terminated.

```
#include <AppTask.h>
```

Inheritance diagram for Model::TaskHandler:

**Public Member Functions**

- virtual void [TaskStarted](#) ([Task](#) &t)=0  
*called when a new task is started.*
- virtual void [TaskTerminated](#) ([Task](#) &t)=0  
*called when a task is terminated.*

**4.133.1 Detailed Description**

Abstract class that provides methods to determine if a task is started or terminated.

**4.133.2 Member Function Documentation**

**4.133.2.1** virtual void [Model::TaskHandler::TaskStarted](#) ( [Task](#) & *t* ) [pure virtual]

called when a new task is started.

**Parameters**

<i>t</i>	started task object.
----------	----------------------

Implemented in [AdjustingNWTunlet](#), and [FactoringTunlet](#).

**4.133.2.2** `virtual void Model::TaskHandler::TaskTerminated ( Task & t )` [pure virtual]

called when a task is terminated.

#### Parameters

<code>t</code>	terminated task object.
----------------	-------------------------

Implemented in [AdjustingNWTunlet](#), and [FactoringTunlet](#).

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

- Analyzer/AppTask.h

## 4.134 TaskInstr Class Reference

Adds and remove instrumentation from the process in execution.

```
#include <TaskInstr.h>
```

### Public Member Functions

- [TaskInstr](#) ()  
*Constructor.*
- [~TaskInstr](#) ()  
*Constructor.*
- `int` [GetSize](#) () const  
*Getter of the size of the variable `_map`.*
- `void` [Add](#) (int eventId, string const &functionName, InstrPlace instrPlace, BPatchSnippetHandle \*handler)  
*Add a new snippet (handler) to the group in the map under the eventId key.*
- `void` [Remove](#) (int eventId, InstrPlace instrPlace)  
*Eliminates all the snippets to be inserted on the same event and place.*
- `InstrGroup *` [FindGroup](#) (int eventId)  
*Finds an instrumentation group for a given eventId.*
- `void` [SetBreakpoint](#) (BPatchSnippetHandle \*h)  
*Setter of the variable `_brkptHandle` which represents a place in the process where it should be stopped.*
- `BPatchSnippetHandle *` [GetBreakpoint](#) ()

Getter of the variable `_brkptHandle` which represents a place in the process where it should be stopped.

#### 4.134.1 Detailed Description

Adds and remove instrumentation from the process in execution.

##### Version

1.0

##### Since

1.0

##### Author

Ania Morajko, 2002

#### 4.134.2 Member Function Documentation

##### 4.134.2.1 void TaskInstr::Add ( int *eventId*, string const & *functionName*, InstrPlace *instrPlace*, BPatchSnippetHandle \* *handler* )

Add a new snippet (handler) to the group in the map under the `eventId` key.

If a group doesn't exist with that key one is created and added to the map.

##### Parameters

<i>eventId</i>	Identifier of the event used as key to store the instrumentation groups in the map.
<i>function-Name</i>	Name of the function to be modified.
<i>instrPlace</i>	Place of the function where the snippet will be inserted.
<i>handler</i>	Handle of the snippet to be inserted.

##### 4.134.2.2 InstrGroup \* TaskInstr::FindGroup ( int *eventId* )

Finds an instrumentation group for a given `eventId`.

##### Parameters

<i>eventId</i>	Key to find the instrumentation group in the map.
----------------	---

##### Returns

IntrGroup object that contains all the snippets to be added on a given event.

**4.134.2.3** `BPatchSnippethHandle* TaskInstr::GetBreakpoint ( )` `[inline]`

Getter of the variable `_brkptHandle` which represents a place in the process where it should be stopped.

**Returns**

Handle of the breakpoint.

**4.134.2.4** `int TaskInstr::GetSize ( ) const` `[inline]`

Getter of the size of the variable `_map`.

**Returns**

size of the map `_map`.

**4.134.2.5** `void TaskInstr::Remove ( int eventId, InstrPlace instrPlace )`

Eliminates all the snippets to be inserted on the same event and place.

**Parameters**

<i>eventId</i>	Identifier of the event used as key to find and remove the instrumentation groups in the map.
<i>instrPlace</i>	Place of the function where the snippet would have been.

**Exceptions**

<i>Remove</i>	cannot find instrumentation group.
---------------	------------------------------------

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

- AC/TaskInstr.h
- AC/TaskInstr.cpp

**4.135 TaskManager Class Reference**

Single class that starts and handles all the tasks.

```
#include <TaskManager.h>
```

**4.135.1 Detailed Description**

Single class that starts and handles all the tasks. Originally this class started the applications in each node using MPI, now this is done externally.

**Version**

1.0

**Since**

1.0

**Author**

Ania Morajko, 2002

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

- AC/TaskManager.h

## 4.136 Model::Tasks Class Reference

[Tasks](#) encapsulates methods to work with lists of [Task](#) objects. The data structure to hold the information is an [auto\\_vector](#). This class provides methods to add, remove, access [Task](#) objects in an array. Also it provides methods to find [Tasks](#) and for measure the array.

```
#include <AppTask.h>
```

**Public Types**

- enum { **NotFound** = -1 }

**Public Member Functions**

- [Tasks](#) ()  
*Constructor.*
- void [Add](#) (auto\_ptr< [Task](#) > &task)  
*Adds a task to the list.*
- void [Delete](#) (int index)  
*Deletes a task from the list.*
- [Task](#) const \* [operator\[\]](#) (int index) const  
*Accessor to the array.*
- [Task](#) \* [operator\[\]](#) (int index)  
*Accessor to the array.*
- int [FindById](#) (int id)

*Finds a task by ID and returns its index.*

- `int Size () const`  
*size getter.*
- `Task & GetById (int id)`  
*Finds a task by ID and returns a reference to it.*
- `auto_ptr< Task > Remove (int id)`  
*Removes a task by ID from the list.*

#### 4.136.1 Detailed Description

`Tasks` encapsulates methods to work with lists of `Task` objects. The data structure to hold the information is an `auto_vector`. This class provides methods to add, remove, access `Task` objects in an array. Also it provides methods to find `Tasks` and for measure the array.

#### 4.136.2 Member Function Documentation

##### 4.136.2.1 `void Tasks::Add ( auto_ptr< Task > & task )`

Adds a task to the list.

###### Parameters

<i>task</i>	the task to add.
-------------	------------------

##### 4.136.2.2 `void Tasks::Delete ( int index )`

Deletes a task from the list.

###### Parameters

<i>index</i>	position in the list of the task to delete.
--------------	---

##### 4.136.2.3 `int Tasks::FindByd ( int id )`

Finds a task by ID and returns its index.

###### Parameters

<i>id</i>	identificator of the task.
-----------	----------------------------



**Returns**

index of the task or NotFound if a task with given ID is not stored in the collection.

**4.136.2.4 Task & Tasks::GetById ( int *id* )**

Finds a task by ID and returns a reference to it.

**Parameters**

<i>id</i>	identificator of the task (pid).
-----------	----------------------------------

**Returns**

a reference to the found task.

**Exceptions**

<i>exception</i>	if not found.
------------------	---------------

**4.136.2.5 Task const \* Tasks::operator[] ( int *index* ) const**

Accessor to the array.

**Parameters**

<i>index</i>	position of the array to access.
--------------	----------------------------------

**Returns****4.136.2.6 Task \* Tasks::operator[] ( int *index* )**

Accessor to the array.

**Parameters**

<i>index</i>	position of the array to access.
--------------	----------------------------------

**Returns****4.136.2.7 auto\_ptr< Task > Tasks::Remove ( int *id* )**

Removes a task by ID from the list.

**Parameters**

<i>id</i>	identificator of the task.
-----------	----------------------------

**Returns**

a reference to the removed task or a null pointer if it was not present.

**4.136.2.8 int Tasks::Size ( ) const**

size getter.

**Returns**

number of stored tasks

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

- Analyzer/AppTask.h
- Analyzer/AppTask.cpp

**4.137 TaskStats Class Reference****Public Member Functions**

- **TaskStats** (int tid)
- void **ChangeFragSize** (int size)
- void **Update** (int count, int size)
- double **GetCommCost** () const
- int **GetOptimalFragSize** () const
- int **GetCurrentFragSize** () const
- int **GetNumChanges** () const
- int **GetTid** () const

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

- Analyzer/Analysis.cpp

**4.138 Common::Thread Class Reference**

Posix thread.

```
#include <Thread.h>
```

Public Member Functions

- [Thread](#) (void \*(\*pFun)(void \*arg), void \*pArg)  
*Constructor.*
- void [WaitForDeath](#) ()  
*Waits for termination of the thread.*
- void [Exit](#) ()  
*Stops the thread execution.*

4.138.1 Detailed Description

Posix thread.

Version

1.0b

Since

1.0b

Author

Ania Morajko, 2002

4.138.2 Constructor & Destructor Documentation

4.138.2.1 Thread::Thread ( void (\*)(void \*arg) pFun, void \* pArg )

Constructor.

Parameters

<i>arg</i>	Pointer to the function to run.
<i>pArg</i>	Pointer to the function arguments.

Exceptions

<a href="#">SysException</a>	
------------------------------	--

4.138.3 Member Function Documentation

4.138.3.1 void Thread::WaitForDeath ( )

Waits for termination of the thread.

**Exceptions**

<a href="#">SysException</a>
------------------------------

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

- Common/Thread.h
- Common/Thread.cpp

**4.139 Common::TimeValue Class Reference**

Stores a time value up to microseconds.

```
#include <TimeValue.h>
```

**Public Member Functions**

- [TimeValue](#) ()  
*Constructor.*
- [TimeValue](#) (long secs, long usecs=0)  
*Constructor.*
- [TimeValue](#) ([TimeValue](#) const &t)  
*Constructor.*
- [TimeValue](#) (timeval const &t)  
*Constructor.*
- [TimeValue](#) (struct timeb const &tb)  
*Constructor.*
- [TimeValue](#) & **operator=** ([TimeValue](#) const &t)
- [TimeValue](#) & **operator=** (timeval const &t)
- void **operator+=** ([TimeValue](#) const &t)
- void **operator-=** ([TimeValue](#) const &t)
- long [GetSeconds](#) () const  
*Returns seconds of the object.*
- long [GetMicroseconds](#) () const  
*Returns microseconds of the object without counting on the seconds.*
- long\_t [GetMilliseconds](#) () const  
*Returns milliseconds of the object.*
- long\_t [GetTotalMicroseconds](#) () const

*Returns microseconds of the object.*

- void [Zero](#) ()  
*Sets seconds and microseconds to 0.*
- void [SetCurrentTime](#) ()  
*Sets values to current time.*
- operator struct timeval \* ()

### Static Public Member Functions

- static [TimeValue Now](#) ()  
*Returns current time.*

### Friends

- [TimeValue operator+](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)
- [TimeValue operator-](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)
- [TimeValue operator/](#) ([TimeValue](#) const &t1, int value)
- bool [operator<](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)
- bool [operator>](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)
- bool [operator==](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)
- bool [operator!=](#) ([TimeValue](#) const &t1, [TimeValue](#) const &t2)

#### 4.139.1 Detailed Description

Stores a time value up to microseconds.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

#### 4.139.2 Constructor & Destructor Documentation

##### 4.139.2.1 Common::TimeValue::TimeValue ( ) [inline]

Constructor.

By default the value of the time value is 0 seconds and 0 microseconds.

**4.139.2.2 Common::TimeValue::TimeValue ( long secs, long usecs = 0 ) [inline]**

Constructor.

Sets seconds and microseconds (microseconds are 0 by default).

**Parameters**

<i>secs</i>	Seconds.
<i>usecs</i>	Microseconds.

**4.139.2.3 Common::TimeValue::TimeValue ( TimeValue const & t ) [inline]**

Constructor.

Assigns the values of the [TimeValue](#) object.

**4.139.2.4 Common::TimeValue::TimeValue ( timeval const & t ) [inline]**

Constructor.

Assigns values of the given structure.

**4.139.2.5 Common::TimeValue::TimeValue ( struct timeb const & tb ) [inline]**

Constructor.

Assigns values of the given structure converted to the used format.

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

- Common/TimeValue.h
- Common/TimeValue.cpp

**4.140 Tuner Class Reference****Public Member Functions**

- **Tuner** ([TaskCollection](#) &tasks)
- void **Process** ([Common::TuningRequest](#) \*req)
- void **RemoveLastBreakpoint** ([Task](#) &task)

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

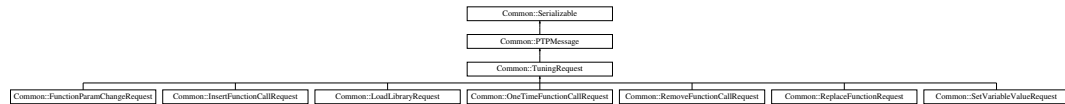
- AC/Tuner.h
- AC/Tuner.cpp

## 4.141 Common::TuningRequest Class Reference

Encapsulates a tuning request from the analyzer.

```
#include <PTPMsg.h>
```

Inheritance diagram for Common::TuningRequest:



### Public Member Functions

- [~TuningRequest](#) ()  
*Destructor.*
- int [GetPid](#) () const  
*Returns the pid of the process associated with the message.*
- [Breakpoint](#) \* [GetBreakpoint](#) () const  
*Returns breakpoint.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the message.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Receives the message.*
- void [ClearBreakpoint](#) ()  
*Clears breakpoint.*

### Protected Member Functions

- **TuningRequest** (int pid, [Breakpoint](#) const \*brkpt=0)

### Protected Attributes

- int **\_pid**
- [Breakpoint](#) \* **\_brkpt**

#### 4.141.1 Detailed Description

Encapsulates a tuning request from the analyzer.

##### Version

1.0b

##### Since

1.0b

##### Author

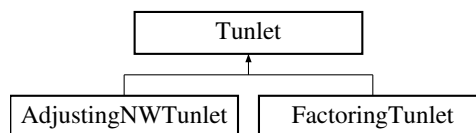
Ania Morajko, 2003

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

- Common/PTPMsg.h
- Common/PTPMsg.cpp

#### 4.142 Tunlet Class Reference

Inheritance diagram for Tunlet:



##### Public Member Functions

- virtual void **Initialize** ([Model::Application](#) &app)=0
- virtual void **BeforeAppStart** ()
- virtual void **AppStarted** ()
- virtual void **Destroy** ()=0
- virtual void [Initialize](#) ([Model::Application](#) &app)=0
- virtual void **BeforeAppStart** ()
- virtual void **AppStarted** ()
- virtual void **Destroy** ()=0

##### 4.142.1 Member Function Documentation

4.142.1.1 virtual void Tunlet::Initialize ( [Model::Application](#) & *app* ) [pure virtual]

##### Parameters



<i>app</i>
------------

Implemented in [AdjustingNWTunlet](#), and [FactoringTunlet](#).

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

- Analyzer/FactoringTunlet\_nw.h
- Analyzer/Tunlet.h

## 4.143 TunletContainer Class Reference

TO BE IMPLEMENTED.

```
#include <TunletsContainer.h>
```

### 4.143.1 Detailed Description

TO BE IMPLEMENTED.

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

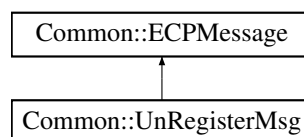
- Analyzer/TunletsContainer.h

## 4.144 Common::UnRegisterMsg Class Reference

Represents message that is sent when DMLib is unregistered with analyzer.

```
#include <ECPMsg.h>
```

Inheritance diagram for Common::UnRegisterMsg:



### Public Member Functions

- [UnRegisterMsg](#) (int pid)  
*Constructor.*
- ECPMsgType [GetType](#) () const  
*Returns the type of event.*

- int [GetPid](#) () const  
*Returns Id of the process where the library will be unregistered.*
- void [Serialize](#) ([Serializer](#) &out) const  
*Sends the message.*
- void [DeSerialize](#) ([DeSerializer](#) &in)  
*Receives the message.*

#### 4.144.1 Detailed Description

Represents message that is sent when DMLib is unregistered with analyzer.

##### Version

1.0b

##### Since

1.0b

##### Author

Ania Morajko, 2002

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

- Common/ECPMsg.h

#### 4.145 Ventana Struct Reference

##### Public Attributes

- int **TAM**
- [Stats](#) \* **historico**

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

- Analyzer/FactoringTunlet\_nw.h

#### 4.146 WorkerData Class Reference

Worker task statistics for a single batch.

```
#include <FactoringStats_nw.h>
```

### Public Member Functions

- void **OnCalcStart** (long\_t time)
- void **OnCalcEnd** (long\_t time)
- bool **IsComplete** () const
- int **GetNumProcessedTuples** () const
- int **GetSizeProcessedTuples** () const
- double **GetTotalCalcTime** () const
- void **OnTupleStart** (int nTuples, int sizeBytes)
- bool **IsTaken** ()
- bool **IsInitialized** ()

#### 4.146.1 Detailed Description

Worker task statistics for a single batch.

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

- Analyzer/FactoringStats\_nw.h
- Analyzer/FactoringStats\_nw.cpp

# Index

- ~EventCollector
  - EventCollector, [92](#)
- Accept
  - Common::BasicLogger, [46](#)
  - Common::LogFilter, [140](#)
- ACProxy, [13](#)
  - ACProxy, [14](#)
  - AddInstr, [15](#)
  - FuncParamChange, [15](#)
  - InsertFunctionCall, [15](#)
  - LoadLibrary, [16](#)
  - OneTimeFuncCall, [16](#)
  - RemoveFuncCall, [16](#)
  - RemoveInstr, [16](#)
  - ReplaceFunction, [17](#)
  - SetVariableValue, [17](#)
  - StartApplication, [17](#)
- Add
  - Common::ConfigMap, [64](#)
  - Common::EventMap, [100](#)
  - Common::FuncDefs, [124](#)
  - Model::Events, [110](#)
  - Model::Tasks, [230](#)
  - Service, [181](#)
  - TaskCollection, [223](#)
  - TaskInstr, [227](#)
- AddDelayedTuning
  - Task, [213](#)
- AddEntry
  - Common::Config, [57](#)
- AddEvent
  - Model::Application, [30](#)
  - Model::Task, [218](#)
- AddHandler
  - InstrGroup, [132](#)
- AddHost
  - Model::Application, [30](#)
- AddInstr
  - ACProxy, [15](#)
  - Monitor, [144](#)
- AddInstrRequest
  - Common::AddInstrRequest, [21](#)
- Address
  - Common::Address, [22](#)
- AddTask
  - Model::Application, [30](#)
- AdjustingNWTunlet, [23](#)
  - FindIterData, [24](#)
  - HandleEvent, [24](#)
  - Initialize, [24](#)
  - InsertEvents, [25](#)
  - InsertMasterEvents, [25](#)
  - InsertWorkerEvents, [25](#)
  - TaskStarted, [25](#)
  - TaskTerminated, [25](#)
  - TryTuning, [25](#)
- Analyzer, [26](#)
  - Analyzer, [26](#)
- Application
  - Model::Application, [29](#)
- Attribute
  - Common::Attribute, [37](#)
- auto\_iterator, [41](#)
- auto\_vector, [41](#)
- BasicLogFilter
  - Common::BasicLogFilter, [42](#)
- BasicLogFormatter
  - Common::BasicLogFormatter, [44](#)
- BatchData, [46](#)
- begin
  - InstrGroup, [133](#)
- Bind
  - Common::SocketBase, [197](#)
- ByteStream
  - Common::ByteStream, [49](#)
- CommandLine, [50](#)
  - CommandLine, [52](#)
  - DisplayHelp, [53](#)
  - GetAppArgc, [53](#)

- GetAppArgv, 53
- GetAppPath, 53
- GetArgc, 54
- GetArgv, 54
- GetConfigFile, 54
- GetConfigFileName, 54
- HasConfig, 55
- IsOk, 55
- Common::ActiveObject, 18
- Common::AddInstrRequest, 19
  - AddInstrRequest, 21
- Common::Address, 21
  - Address, 22
  - GetHostName, 23
  - GetSize, 23
- Common::Attribute, 36
  - Attribute, 37
- Common::AttributeValue, 37
  - GetCharValue, 39
  - GetDoubleValue, 39
  - GetFloatValue, 40
  - GetIntValue, 40
  - GetShortValue, 40
  - GetStringValue, 40
- Common::BasicLogFilter, 42
  - BasicLogFilter, 42
- Common::BasicLogFormatter, 43
  - BasicLogFormatter, 44
- Common::BasicLogger, 44
  - Accept, 46
- Common::Breakpoint, 47
- Common::ByteStream, 48
  - ByteStream, 49
  - GetData, 50
  - GetDataSize, 50
  - Write, 50
- Common::Config, 56
  - AddEntry, 57
  - Contains, 58
  - GetBoolValue, 58
  - GetIntValue, 59
  - GetKeys, 59
  - GetStringValue, 60
- Common::Config::KeyIterator, 136
- Common::ConfigException, 60
  - ConfigException, 61
  - Display, 61
  - GetReason, 61
- Common::ConfigHelper, 62
  - ReadFromFile, 62
- Common::ConfigMap, 63
  - Add, 64
  - Contains, 64
  - GetValue, 64
- Common::ConfigMap::Iterator, 134
- Common::ConfigReader, 65
- Common::CountingSerializer, 68
- Common::DateTime, 69
  - GetStringValue, 70
- Common::DeSerializer, 71
- Common::ECPMessage, 82
  - ECPMessage, 83
- Common::ECPMsgHeader, 83
- Common::Event, 89
  - Event, 90
- Common::EventDemultiplexer, 94
  - Select, 95
- Common::EventException, 95
  - Display, 96
  - EventException, 96
  - GetReason, 96
- Common::EventHandler, 97
- Common::EventMap, 99
  - Add, 100
  - GetId, 100
- Common::EventMsg, 101
  - Reset, 103
- Common::Exception, 111
  - Display, 113
  - Exception, 113
- Common::ExecProcess, 113
  - ExecProcess, 115
  - Start, 115
  - WaitForEvent, 115
- Common::FileConfigReader, 117
  - FileConfigReader, 118
  - Read, 118
- Common::FileLogger, 119
  - FileLogger, 119
- Common::FuncDef, 120
  - FuncDef, 121
- Common::FuncDefException, 121
  - Display, 122
  - FuncDefException, 122
  - GetReason, 122
- Common::FuncDefs, 123
  - Add, 124
  - Find, 124
  - FuncDefs, 124

- Common::FunctionParamChangeRequest, 124
  - FunctionParamChangeRequest, 126
- Common::HandlerMap, 126
- Common::InsertFunctionCallRequest, 129
  - InsertFunctionCallRequest, 131
- Common::LoadLibraryRequest, 137
  - LoadLibraryRequest, 138
- Common::LogEntry, 138
  - GetSeverity, 139
  - LogEntry, 139
- Common::LogFilter, 139
  - Accept, 140
- Common::LogFormatter, 141
- Common::Logger, 141
- Common::Mutex, 145
  - Enter, 146
  - Leave, 146
  - Mutex, 146
- Common::MutexLock, 146
- Common::NetworkDeSerializer, 147
- Common::NetworkSerializer, 149
  - NetworkSerializer, 150
- Common::OneTimeFunctionCallRequest, 150
  - OneTimeFunctionCallRequest, 152
- Common::OutputStream, 152
- Common::Pipe, 153
  - Pipe, 155
  - Read, 155
  - Write, 155
- Common::Process, 156
- Common::PTPMessage, 160
- Common::PTPMsgHeader, 161
- Common::PTPProtocol, 163
- Common::Queue, 163
  - Get, 165
  - GetB, 165
  - Put, 165
- Common::Reactor, 165
- Common::RegisterMsg, 166
  - RegisterMsg, 167
- Common::RemoteProcess, 168
  - RemoteProcess, 169
- Common::RemoveFunctionCallRequest, 169
  - RemoveFunctionCallRequest, 171
- Common::RemoveInstrRequest, 171
  - RemoveInstrRequest, 172
- Common::ReplaceFunctionRequest, 173
  - ReplaceFunctionRequest, 174
- Common::Semaphore, 174
  - Post, 176
  - Semaphore, 176
  - TryWait, 176
  - Wait, 176
- Common::Serializable, 176
- Common::Serializer, 177
- Common::ServerSocket, 178
  - ServerSocket, 179
- Common::SetVariableValueRequest, 181
  - GetValueBuffer, 183
  - SetVariableValueRequest, 183
- Common::Socket, 189
  - GetReceiveTimeout, 192
  - GetSendTimeout, 192
  - Receive, 192
  - ReceiveN, 192
  - Send, 193
  - SetKeepAlive, 193
  - SetReceiveTimeout, 193
  - SetReuseAddress, 193
  - SetSendTimeout, 194
  - SetTCPNoDelay, 194
  - Socket, 191
- Common::SocketBase, 194
  - Bind, 197
  - DoSend, 197
  - GetOption, 198
  - GetReceiveTimeout, 198
  - GetSendTimeout, 198
  - Listen, 198
  - Receive, 199
  - ReceiveN, 199
  - Send, 199
  - SetKeepAlive, 200
  - SetOption, 200
  - SetReceiveTimeout, 200
  - SetReuseAddress, 200
  - SetSendTimeout, 201
  - SetTCPNoDelay, 201
  - SocketBase, 197
- Common::StartAppRequest, 201
  - StartAppRequest, 202
- Common::StreamLogger, 203
- Common::StringArray, 204
  - StringArray, 205
- Common::SysException, 206
  - Display, 207
  - SysException, 207
- Common::Syslog, 207
  - Configure, 209

- Debug, [209](#), [210](#)
- Error, [210](#)
- Fatal, [210](#)
- Info, [210](#)
- Warn, [210](#), [211](#)
- Common::Thread, [232](#)
  - Thread, [233](#)
  - WaitForDeath, [233](#)
- Common::TimeValue, [234](#)
  - TimeValue, [235](#), [236](#)
- Common::TuningRequest, [237](#)
- Common::UnRegisterMsg, [239](#)
- comptime, [56](#)
- ConfigException
  - Common::ConfigException, [61](#)
- Configure
  - Common::Syslog, [209](#)
- Contains
  - Common::Config, [58](#)
  - Common::ConfigMap, [64](#)
- Controller, [66](#)
  - Controller, [67](#)
  - Run, [67](#)
- CreateApplication
  - DTLibrary, [77](#)
- CreateLibrary
  - DTLibraryFactory, [78](#)
- curState, [69](#)
- Debug
  - Common::Syslog, [209](#), [210](#)
- Delete
  - Model::Tasks, [230](#)
  - TaskCollection, [223](#)
- DestroyLibrary
  - DTLibraryFactory, [78](#)
- DiEx, [72](#)
- DiFunction, [72](#)
- DiImage, [73](#)
- DiIntType, [73](#)
- DiIntVariable, [73](#)
- DiPoint, [74](#)
- DiProcess, [74](#)
- DiSnippetHandle, [75](#)
- DispatchEvent
  - Model::Application, [30](#)
- Display
  - Common::ConfigException, [61](#)
  - Common::EventException, [96](#)
  - Common::Exception, [113](#)
  - Common::FuncDefException, [122](#)
  - Common::SysException, [207](#)
- DisplayHelp
  - CommandLine, [53](#)
- DiType, [76](#)
- DiVariable, [76](#)
- DMLib::EventCollectorProxy, [93](#)
- DMLib::EventMsgWriter, [105](#)
  - OpenEvent, [106](#)
- DoSend
  - Common::SocketBase, [197](#)
- DTLibrary, [77](#)
  - CreateApplication, [77](#)
  - GetApplication, [77](#)
- DTLibraryFactory, [78](#)
  - CreateLibrary, [78](#)
  - DestroyLibrary, [78](#)
- DynInst, [79](#)
- ECPAcceptor, [79](#)
  - ECPAcceptor, [80](#)
  - GetHandle, [80](#)
  - SetEventCollector, [80](#)
- ECPHandler, [80](#)
  - GetHandle, [81](#)
  - SetService, [81](#)
- ECPMessage
  - Common::ECPMessage, [83](#)
- ECPProtocol, [85](#)
  - ReadMessageEx, [85](#)
  - ReadMessageHeader, [85](#)
- end
  - InstrGroup, [133](#)
- Enter
  - Common::Mutex, [146](#)
- Error
  - Common::Syslog, [210](#)
- Event
  - Common::Event, [90](#)
  - Model::Event, [87](#)
- EventCollector, [91](#)
  - ~EventCollector, [92](#)
  - EventCollector, [92](#)
  - GetListener, [92](#)
  - IsAborted, [92](#)
  - SetListener, [92](#)
- EventException
  - Common::EventException, [96](#)
- EventListener, [98](#)
  - OnEvent, [99](#)

- EventMsgReader, 103
  - GetAttrType, 104
  - GetCharValue, 104
  - GetDoubleValue, 104
  - GetFloatValue, 104
  - GetIntValue, 104
  - GetParamCount, 104
  - GetShortValue, 105
  - GetStringValue, 105
- EventRecord
  - Model::EventRecord, 108
- Exception
  - Common::Exception, 113
- ExecProcess
  - Common::ExecProcess, 115
- FactoringTunlet, 116
  - HandleEvent, 116
  - Initialize, 116
  - TaskStarted, 117
  - TaskTerminated, 117
- Fatal
  - Common::Syslog, 210
- FileConfigReader
  - Common::FileConfigReader, 118
- FileLogger
  - Common::FileLogger, 119
- Find
  - Common::FuncDefs, 124
  - Model::Events, 110
- FindById
  - Model::Tasks, 230
- FindGroup
  - TaskInstr, 227
- FindIterData
  - AdjustingNWTunlet, 24
- FuncDef
  - Common::FuncDef, 121
- FuncDefException
  - Common::FuncDefException, 122
- FuncDefs
  - Common::FuncDefs, 124
- FuncParamChange
  - ACProxy, 15
  - Model::Application, 30
  - Model::Task, 218
- FunctionParamChangeRequest
  - Common::FunctionParamChangeRequest, 126
- Get
  - Common::Queue, 165
- GetACProxy
  - Model::Task, 218
- GetAppArgc
  - CommandLine, 53
- GetAppArgv
  - CommandLine, 53
- GetApplication
  - DTLibrary, 77
- GetAppPath
  - CommandLine, 53
- GetArgc
  - CommandLine, 54
- GetArgv
  - CommandLine, 54
- GetAttributes
  - Model::Event, 87
- GetAttributeValue
  - Model::EventRecord, 108
- GetAttributeValues
  - Model::EventRecord, 108
- GetAttrType
  - EventMsgReader, 104
- GetB
  - Common::Queue, 165
- GetBoolValue
  - Common::Config, 58
- GetBreakpoint
  - TaskInstr, 227
- GetById
  - Model::Tasks, 231
- GetCharValue
  - Common::AttributeValue, 39
  - EventMsgReader, 104
- GetConfigFile
  - CommandLine, 54
- GetConfigFileName
  - CommandLine, 54
- GetData
  - Common::ByteStream, 50
- GetDataSize
  - Common::ByteStream, 50
- GetDoubleValue
  - Common::AttributeValue, 39
  - EventMsgReader, 104
- GetEvent
  - Model::EventRecord, 108
- GetEventHandler
  - Model::Event, 88



- GetEventId
  - InstrGroup, 133
  - Model::EventRecord, 109
  - SnippetHandler, 187
- GetFloatValue
  - Common::AttributeValue, 40
  - EventMsgReader, 104
- GetFuncName
  - InstrGroup, 133
  - SnippetHandler, 187
- GetFunctionName
  - Model::Event, 88
- GetHandle
  - ECPAcceptor, 80
  - ECPHandler, 81
  - PTPAcceptor, 158
  - PTPHandler, 160
  - SnippetHandler, 187
- GetHost
  - Model::Task, 218
- GetHostName
  - Common::Address, 23
- GetHosts
  - Model::Application, 31
- GetId
  - Common::EventMap, 100
  - Model::Event, 88
- GetImage
  - Task, 213
- GetInstr
  - Task, 213
- GetInstrPlace
  - Model::Event, 88
  - SnippetHandler, 188
- GetIntValue
  - Common::AttributeValue, 40
  - Common::Config, 59
  - EventMsgReader, 104
- GetKeys
  - Common::Config, 59
- GetListener
  - EventCollector, 92
- GetMpiRank
  - Model::Task, 219
- GetName
  - Model::Application, 31
  - Model::Host, 128
  - Model::Task, 219
- GetNumAttributes
  - Model::Event, 88
- GetOption
  - Common::SocketBase, 198
- GetParamCount
  - EventMsgReader, 104
- GetPid
  - Model::Task, 219
  - Task, 213
- GetProcess
  - Task, 214
- GetReason
  - Common::ConfigException, 61
  - Common::EventException, 96
  - Common::FuncDefException, 122
- GetReceiveTimeout
  - Common::Socket, 192
  - Common::SocketBase, 198
- GetSendTimeout
  - Common::Socket, 192
  - Common::SocketBase, 198
- GetSeverity
  - Common::LogEntry, 139
- GetShortValue
  - Common::AttributeValue, 40
  - EventMsgReader, 105
- GetSize
  - Common::Address, 23
  - InstrGroup, 133
  - TaskInstr, 228
- GetStatus
  - Model::Application, 31
  - Model::Task, 219
- GetStringValue
  - Common::AttributeValue, 40
  - Common::Config, 60
  - Common::DateTime, 70
  - EventMsgReader, 105
- GetTask
  - Model::EventRecord, 109
- GetTasks
  - Model::Application, 31
- GetTimestamp
  - Model::EventRecord, 109
- GetValue
  - Common::ConfigMap, 64
- GetValueBuffer
  - Common::SetVariableValueRequest, 183
- HandleEvent
  - AdjustingNWTunlet, 24
  - FactoringTunlet, 116

- Model::EventHandler, 98
- HasConfig
  - CommandLine, 55
- HostAdded
  - Model::HostHandler, 129
- HostRemoved
  - Model::HostHandler, 129
- Info
  - Common::Syslog, 210
- Initialize
  - AdjustingNWTunlet, 24
  - FactoringTunlet, 116
  - Tunlet, 238
- InsertEvents
  - AdjustingNWTunlet, 25
- InsertFunctionCall
  - ACProxy, 15
  - Model::Application, 32
  - Model::Task, 219
- InsertFunctionCallRequest
  - Common::InsertFunctionCallRequest, 131
- InsertMasterEvents
  - AdjustingNWTunlet, 25
- InsertWorkerEvents
  - AdjustingNWTunlet, 25
- InstrGroup, 131
  - AddHandler, 132
  - begin, 133
  - end, 133
  - GetEventId, 133
  - GetFuncName, 133
  - GetSize, 133
  - IsEmpty, 133
  - RemoveHandler, 133
- IsAborted
  - EventCollector, 92
- IsEmpty
  - InstrGroup, 133
- isFinished
  - ShutDownManager, 184
- IsMaster
  - Model::Task, 220
- IsOk
  - CommandLine, 55
- IsRunning
  - Model::Task, 220
- IsStopped
  - Task, 214
- IsStoppedOnBreakpoint
  - Task, 214
- IsTerminated
  - Task, 214
- IterData, 135
- Leave
  - Common::Mutex, 146
- Listen
  - Common::SocketBase, 198
- LoadLibrary
  - ACProxy, 16
  - Model::Application, 32
  - Model::Task, 220
- LoadLibraryRequest
  - Common::LoadLibraryRequest, 138
- LogEntry
  - Common::LogEntry, 139
- MakeEventSnippet
  - SnippetMaker, 189
- Model::Application, 26
  - AddEvent, 30
  - AddHost, 30
  - AddTask, 30
  - Application, 29
  - DispatchEvent, 30
  - FuncParamChange, 30
  - GetHosts, 31
  - GetName, 31
  - GetStatus, 31
  - GetTasks, 31
  - InsertFunctionCall, 32
  - LoadLibrary, 32
  - NumActiveTasks, 32
  - OneTimeFuncCall, 32
  - OnEvent, 33
  - ProcessEvent, 33
  - ProcessEvents, 33
  - RemoveEvent, 34
  - RemoveFuncCall, 34
  - RemoveTask, 34
  - ReplaceFunction, 34
  - SetHostHandler, 35
  - SetTaskHandler, 35
  - SetVariableValue, 35
  - Start, 35
- Model::Event, 86
  - Event, 87
  - GetAttributes, 87

- GetEventHandler, 88
- GetFunctionName, 88
- GetId, 88
- GetInstrPlace, 88
- GetNumAttributes, 88
- SetAttribute, 88
- SetEventHandler, 89
- Model::EventHandler, 97
  - HandleEvent, 98
- Model::EventRecord, 107
  - EventRecord, 108
  - GetAttributeValue, 108
  - GetAttributeValues, 108
  - GetEvent, 108
  - GetEventId, 109
  - GetTask, 109
  - GetTimestamp, 109
  - ParseAttrs, 109
- Model::Events, 110
  - Add, 110
  - Find, 110
  - Remove, 111
  - Size, 111
- Model::Host, 127
  - GetName, 128
- Model::HostHandler, 128
  - HostAdded, 129
  - HostRemoved, 129
- Model::Task, 215
  - AddEvent, 218
  - FuncParamChange, 218
  - GetACProxy, 218
  - GetHost, 218
  - GetMpiRank, 219
  - GetName, 219
  - GetPid, 219
  - GetStatus, 219
  - InsertFunctionCall, 219
  - IsMaster, 220
  - IsRunning, 220
  - LoadLibrary, 220
  - OneTimeFuncCall, 220
  - RemoveEvent, 221
  - RemoveFuncCall, 221
  - ReplaceFunction, 221
  - SetMaster, 221
  - SetVariableValue, 222
  - Task, 217
- Model::TaskHandler, 225
  - TaskStarted, 225
  - TaskTerminated, 225
- Model::Tasks, 229
  - Add, 230
  - Delete, 230
  - FindById, 230
  - GetById, 231
  - Remove, 231
  - Size, 232
- ModelParam, 143
- ModuleList, 143
- Monitor, 143
  - AddInstr, 144
  - Monitor, 144
  - RemoveInstr, 144
- Mutex
  - Common::Mutex, 146
- myauto\_ptr, 147
- NetworkSerializer
  - Common::NetworkSerializer, 150
- NumActiveTasks
  - Model::Application, 32
- OneTimeFuncCall
  - ACProxy, 16
  - Model::Application, 32
  - Model::Task, 220
- OneTimeFunctionCallRequest
  - Common::OneTimeFunctionCallRequest, 152
- OnEvent
  - EventListener, 99
  - Model::Application, 33
- OpenEvent
  - DMLib::EventMsgWriter, 106
- ParseAttrs
  - Model::EventRecord, 109
- Pipe
  - Common::Pipe, 155
- PointList, 155
- Post
  - Common::Semaphore, 176
- ProcedureList, 156
- ProcessBreakpoint
  - Task, 214
- ProcessEvent
  - Model::Application, 33
- ProcessEvents
  - Model::Application, 33

- PTPAcceptor, [157](#)
  - GetHandle, [158](#)
  - PTPAcceptor, [158](#)
- PTPHandler, [159](#)
  - GetHandle, [160](#)
  - PTPHandler, [159](#)
- Put
  - Common::Queue, [165](#)
- Read
  - Common::FileConfigReader, [118](#)
  - Common::Pipe, [155](#)
- ReadFromFile
  - Common::ConfigHelper, [62](#)
- ReadMessageEx
  - ECPProtocol, [85](#)
- ReadMessageHeader
  - ECPProtocol, [85](#)
- Receive
  - Common::Socket, [192](#)
  - Common::SocketBase, [199](#)
- ReceiveN
  - Common::Socket, [192](#)
  - Common::SocketBase, [199](#)
- RegisterMsg
  - Common::RegisterMsg, [167](#)
- RemoteProcess
  - Common::RemoteProcess, [169](#)
- Remove
  - Model::Events, [111](#)
  - Model::Tasks, [231](#)
  - Service, [181](#)
  - TaskInstr, [228](#)
- RemoveEvent
  - Model::Application, [34](#)
  - Model::Task, [221](#)
- RemoveFuncCall
  - ACProxy, [16](#)
  - Model::Application, [34](#)
  - Model::Task, [221](#)
- RemoveFunctionCallRequest
  - Common::RemoveFunctionCallRequest, [171](#)
- RemoveHandler
  - InstrGroup, [133](#)
- RemoveInstr
  - ACProxy, [16](#)
  - Monitor, [144](#)
- RemoveInstrRequest
  - Common::RemoveInstrRequest, [172](#)
- RemoveTask
  - Model::Application, [34](#)
- ReplaceFunction
  - ACProxy, [17](#)
  - Model::Application, [34](#)
  - Model::Task, [221](#)
- ReplaceFunctionRequest
  - Common::ReplaceFunctionRequest, [174](#)
- Reset
  - Common::EventMsg, [103](#)
- Run
  - Controller, [67](#)
  - ShutDownManager, [185](#)
- Select
  - Common::EventDemultiplexer, [95](#)
- Semaphore
  - Common::Semaphore, [176](#)
- Send
  - Common::Socket, [193](#)
  - Common::SocketBase, [199](#)
- ServerSocket
  - Common::ServerSocket, [179](#)
- Service, [180](#)
  - Add, [181](#)
  - Remove, [181](#)
  - Service, [180](#)
- setApp
  - ShutDownManager, [185](#)
- SetAttribute
  - Model::Event, [88](#)
- SetEventCollector
  - ECPAcceptor, [80](#)
- SetEventHandler
  - Model::Event, [89](#)
- SetHostHandler
  - Model::Application, [35](#)
- SetKeepAlive
  - Common::Socket, [193](#)
  - Common::SocketBase, [200](#)
- SetListener
  - EventCollector, [92](#)
- SetMaster
  - Model::Task, [221](#)
- SetOption
  - Common::SocketBase, [200](#)
- SetReceiveTimeout
  - Common::Socket, [193](#)
  - Common::SocketBase, [200](#)
- SetReuseAddress

- Common::Socket, 193
- Common::SocketBase, 200
- SetSendTimeout
  - Common::Socket, 194
  - Common::SocketBase, 201
- SetService
  - ECPHandler, 81
- SetTaskHandler
  - Model::Application, 35
- SetTCPNoDelay
  - Common::Socket, 194
  - Common::SocketBase, 201
- SetVariableValue
  - ACProxy, 17
  - Model::Application, 35
  - Model::Task, 222
- SetVariableValueRequest
  - Common::SetVariableValueRequest, 183
- ShutDownManager, 183
  - isFinished, 184
  - Run, 185
  - setApp, 185
- ShutDownSlave, 185
- Size
  - Model::Events, 111
  - Model::Tasks, 232
- SnippetHandler, 186
  - GetEventId, 187
  - GetFuncName, 187
  - GetHandle, 187
  - GetInstrPlace, 188
  - SnippetHandler, 187
- SnippetMaker, 188
  - MakeEventSnippet, 189
  - SnippetMaker, 189
- Socket
  - Common::Socket, 191
- SocketBase
  - Common::SocketBase, 197
- Start
  - Common::ExecProcess, 115
  - Model::Application, 35
- StartApplication
  - ACProxy, 17
- StartAppRequest
  - Common::StartAppRequest, 202
- Stats, 203
- StringArray
  - Common::StringArray, 205
- SysException
  - Common::SysException, 207
- Task, 211
  - AddDelayedTuning, 213
  - GetImage, 213
  - GetInstr, 213
  - GetPid, 213
  - GetProcess, 214
  - IsStopped, 214
  - IsStoppedOnBreakpoint, 214
  - IsTerminated, 214
  - Model::Task, 217
  - ProcessBreakpoint, 214
  - Task, 213
  - Terminate, 215
- TaskCollection, 222
  - Add, 223
  - Delete, 223
- TaskExitHandler, 224
- TaskInstr, 226
  - Add, 227
  - FindGroup, 227
  - GetBreakpoint, 227
  - GetSize, 228
  - Remove, 228
- TaskManager, 228
- TaskStarted
  - AdjustingNWTunlet, 25
  - FactoringTunlet, 117
  - Model::TaskHandler, 225
- TaskStats, 232
- TaskTerminated
  - AdjustingNWTunlet, 25
  - FactoringTunlet, 117
  - Model::TaskHandler, 225
- Terminate
  - Task, 215
- Thread
  - Common::Thread, 233
- TimeValue
  - Common::TimeValue, 235, 236
- TryTuning
  - AdjustingNWTunlet, 25
- TryWait
  - Common::Semaphore, 176
- Tuner, 236
- Tunlet, 238
  - Initialize, 238
- TunletContainer, 239

Ventana, [240](#)

Wait

    Common::Semaphore, [176](#)

WaitForDeath

    Common::Thread, [233](#)

WaitForEvent

    Common::ExecProcess, [115](#)

Warn

    Common::Syslog, [210](#), [211](#)

WorkerData, [240](#)

Write

    Common::ByteStream, [50](#)

    Common::Pipe, [155](#)