

# Hosting Environment (Daemon) Reference Manual

Generated by Doxygen 1.4.7

Mon Apr 23 10:06:13 2012



# Contents

<b>1</b>	<b>Hosting Environment (Daemon) Namespace Index</b>	<b>1</b>
1.1	Hosting Environment (Daemon) Namespace List . . . . .	1
<b>2</b>	<b>Hosting Environment (Daemon) Hierarchical Index</b>	<b>3</b>
2.1	Hosting Environment (Daemon) Class Hierarchy . . . . .	3
<b>3</b>	<b>Hosting Environment (Daemon) Data Structure Index</b>	<b>9</b>
3.1	Hosting Environment (Daemon) Data Structures . . . . .	9
<b>4</b>	<b>Hosting Environment (Daemon) Namespace Documentation</b>	<b>15</b>
4.1	Arc Namespace Reference . . . . .	15
4.2	ArcCredential Namespace Reference . . . . .	44
4.3	DataStaging Namespace Reference . . . . .	46
<b>5</b>	<b>Hosting Environment (Daemon) Data Structure Documentation</b>	<b>49</b>
5.1	Arc::Adler32Sum Class Reference . . . . .	49
5.2	ArcSec::AlgFactory Class Reference . . . . .	52
5.3	Arc::ApplicationEnvironment Class Reference . . . . .	53
5.4	Arc::ArcLocation Class Reference . . . . .	54
5.5	Arc::ArcVersion Class Reference . . . . .	55
5.6	ArcSec::Attr Struct Reference . . . . .	56
5.7	ArcSec::AttributeFactory Class Reference . . . . .	57
5.8	Arc::AttributeIterator Class Reference . . . . .	58
5.9	ArcSec::AttributeProxy Class Reference . . . . .	61
5.10	ArcSec::AttributeValue Class Reference . . . . .	62
5.11	ArcSec::Attrs Class Reference . . . . .	64
5.12	ArcSec::AuthzRequestSection Struct Reference . . . . .	65
5.13	Arc::AutoPointer< T > Class Template Reference . . . . .	66
5.14	Arc::BaseConfig Class Reference . . . . .	68
5.15	Arc::ChainContext Class Reference . . . . .	70

5.16 Arc::Checksum Class Reference . . . . .	71
5.17 Arc::ChecksumAny Class Reference . . . . .	74
5.18 Arc::CStringValue Class Reference . . . . .	77
5.19 Arc::ClientHTTP Class Reference . . . . .	79
5.20 Arc::ClientInterface Class Reference . . . . .	80
5.21 Arc::ClientSOAP Class Reference . . . . .	81
5.22 Arc::ClientTCP Class Reference . . . . .	83
5.23 ArcSec::CombiningAlg Class Reference . . . . .	84
5.24 Arc::Config Class Reference . . . . .	86
5.25 Arc::ConfusaCertHandler Class Reference . . . . .	88
5.26 Arc::ConfusaParserUtils Class Reference . . . . .	89
5.27 Arc::CountedPointer< T > Class Template Reference . . . . .	91
5.28 Arc::Counter Class Reference . . . . .	93
5.29 Arc::CounterTicket Class Reference . . . . .	100
5.30 Arc::CRC32Sum Class Reference . . . . .	102
5.31 Arc::Credential Class Reference . . . . .	105
5.32 Arc::CredentialError Class Reference . . . . .	114
5.33 Arc::CredentialStore Class Reference . . . . .	115
5.34 Arc::Database Class Reference . . . . .	116
5.35 DataStaging::DataDelivery Class Reference . . . . .	119
5.36 DataStaging::DataDeliveryComm Class Reference . . . . .	121
5.37 DataStaging::DataDeliveryComm::Status Struct Reference . . . . .	126
5.38 DataStaging::DataDeliveryCommHandler Class Reference . . . . .	128
5.39 DataStaging::DataDeliveryLocalComm Class Reference . . . . .	129
5.40 DataStaging::DataDeliveryRemoteComm Class Reference . . . . .	131
5.41 ArcSec::DateTimeAttribute Class Reference . . . . .	133
5.42 Arc::DelegationConsumer Class Reference . . . . .	135
5.43 Arc::DelegationConsumerSOAP Class Reference . . . . .	137
5.44 Arc::DelegationContainerSOAP Class Reference . . . . .	139
5.45 Arc::DelegationProvider Class Reference . . . . .	142
5.46 Arc::DelegationProviderSOAP Class Reference . . . . .	144
5.47 ArcSec::DenyOverridesCombiningAlg Class Reference . . . . .	146
5.48 DataStaging::DTR Class Reference . . . . .	148
5.49 DataStaging::DTRCacheParameters Class Reference . . . . .	160
5.50 DataStaging::DTRCallback Class Reference . . . . .	162
5.51 DataStaging::DTRErrorStatus Class Reference . . . . .	163

5.52	DataStaging::DTRLList Class Reference	166
5.53	DataStaging::DTRStatus Class Reference	169
5.54	ArcSec::DurationAttribute Class Reference	174
5.55	ArcSec::EqualFunction Class Reference	176
5.56	ArcSec::EvalResult Struct Reference	178
5.57	ArcSec::EvaluationCtx Class Reference	179
5.58	ArcSec::Evaluator Class Reference	180
5.59	ArcSec::EvaluatorContext Class Reference	183
5.60	ArcSec::EvaluatorLoader Class Reference	184
5.61	Arc::ExecutableType Class Reference	186
5.62	Arc::ExecutionTarget Class Reference	187
5.63	Arc::ExpirationReminder Class Reference	189
5.64	Arc::FileAccess Class Reference	191
5.65	Arc::FileLock Class Reference	196
5.66	ArcSec::FnFactory Class Reference	199
5.67	ArcSec::Function Class Reference	200
5.68	DataStaging::Generator Class Reference	201
5.69	Arc::GLUE2 Class Reference	202
5.70	Arc::InfoCache Class Reference	203
5.71	Arc::InfoFilter Class Reference	204
5.72	Arc::InfoRegister Class Reference	205
5.73	Arc::InfoRegisterContainer Class Reference	206
5.74	Arc::InfoRegisters Class Reference	207
5.75	Arc::InfoRegistrar Class Reference	208
5.76	Arc::InformationContainer Class Reference	209
5.77	Arc::InformationInterface Class Reference	211
5.78	Arc::InformationRequest Class Reference	213
5.79	Arc::InformationResponse Class Reference	214
5.80	Arc::initializeCredentialsType Class Reference	215
5.81	Arc::IntraProcessCounter Class Reference	216
5.82	Arc::Job Class Reference	220
5.83	Arc::JobControllerPluginLoader Class Reference	227
5.84	Arc::JobDescription Class Reference	229
5.85	Arc::JobDescriptionParser Class Reference	232
5.86	Arc::JobDescriptionParserLoader Class Reference	233
5.87	Arc::JobIdentificationType Class Reference	235

5.88 Arc::JobState Class Reference . . . . .	237
5.89 Arc::JobSupervisor Class Reference . . . . .	238
5.90 Arc::Loader Class Reference . . . . .	245
5.91 Arc::LogDestination Class Reference . . . . .	246
5.92 Arc::LogFile Class Reference . . . . .	248
5.93 Arc::Logger Class Reference . . . . .	251
5.94 Arc::LoggerContext Class Reference . . . . .	255
5.95 Arc::LogMessage Class Reference . . . . .	256
5.96 Arc::LogStream Class Reference . . . . .	258
5.97 ArcSec::MatchFunction Class Reference . . . . .	260
5.98 Arc::MCC Class Reference . . . . .	262
5.99 Arc::MCC_Status Class Reference . . . . .	265
5.100 Arc::MCCInterface Class Reference . . . . .	268
5.101 Arc::MCCLoader Class Reference . . . . .	270
5.102 Arc::MD5Sum Class Reference . . . . .	272
5.103 Arc::Message Class Reference . . . . .	275
5.104 Arc::MessageAttributes Class Reference . . . . .	278
5.105 Arc::MessageAuth Class Reference . . . . .	281
5.106 Arc::MessageAuthContext Class Reference . . . . .	283
5.107 Arc::MessageContext Class Reference . . . . .	284
5.108 Arc::MessageContextElement Class Reference . . . . .	285
5.109 Arc::MessagePayload Class Reference . . . . .	286
5.110 Arc::ModuleDesc Class Reference . . . . .	287
5.111 Arc::ModuleManager Class Reference . . . . .	288
5.112 Arc::MultiSecAttr Class Reference . . . . .	291
5.113 Arc::MySQLDatabase Class Reference . . . . .	292
5.114 Arc::OAuthConsumer Class Reference . . . . .	294
5.115 Arc::PathIterator Class Reference . . . . .	296
5.116 Arc::PayloadRaw Class Reference . . . . .	298
5.117 Arc::PayloadRawInterface Class Reference . . . . .	300
5.118 Arc::PayloadSOAP Class Reference . . . . .	302
5.119 Arc::PayloadStream Class Reference . . . . .	303
5.120 Arc::PayloadStreamInterface Class Reference . . . . .	306
5.121 Arc::PayloadWSRF Class Reference . . . . .	309
5.122 ArcSec::PDP Class Reference . . . . .	310
5.123 ArcSec::PeriodAttribute Class Reference . . . . .	311

5.124ArcSec::PermitOverridesCombiningAlg Class Reference . . . . .	313
5.125Arc::Plexer Class Reference . . . . .	315
5.126Arc::PlexerEntry Class Reference . . . . .	317
5.127Arc::Plugin Class Reference . . . . .	318
5.128Arc::PluginArgument Class Reference . . . . .	320
5.129Arc::PluginDesc Class Reference . . . . .	321
5.130Arc::PluginDescriptor Struct Reference . . . . .	322
5.131Arc::PluginsFactory Class Reference . . . . .	323
5.132ArcSec::Policy Class Reference . . . . .	325
5.133ArcSec::PolicyParser Class Reference . . . . .	328
5.134ArcSec::PolicyStore Class Reference . . . . .	329
5.135DataStaging::Processor Class Reference . . . . .	330
5.136Arc::RegisteredService Class Reference . . . . .	332
5.137Arc::RegularExpression Class Reference . . . . .	333
5.138Arc::RemoteLoggingType Class Reference . . . . .	335
5.139ArcSec::Request Class Reference . . . . .	336
5.140ArcSec::RequestAttribute Class Reference . . . . .	338
5.141ArcSec::RequestItem Class Reference . . . . .	339
5.142ArcSec::Response Class Reference . . . . .	340
5.143ArcSec::ResponseItem Class Reference . . . . .	341
5.144Arc::Run Class Reference . . . . .	342
5.145Arc::SAMLToken Class Reference . . . . .	346
5.146DataStaging::Scheduler Class Reference . . . . .	349
5.147Arc::SecAttr Class Reference . . . . .	352
5.148Arc::SecAttrFormat Class Reference . . . . .	355
5.149Arc::SecAttrValue Class Reference . . . . .	356
5.150ArcSec::SecHandler Class Reference . . . . .	358
5.151ArcSec::SecHandlerConfig Class Reference . . . . .	359
5.152ArcSec::Security Class Reference . . . . .	360
5.153Arc::Service Class Reference . . . . .	361
5.154Arc::SimpleCondition Class Reference . . . . .	364
5.155Arc::SimpleCounter Class Reference . . . . .	366
5.156Arc::SOAPMessage Class Reference . . . . .	368
5.157Arc::Software Class Reference . . . . .	370
5.158Arc::SoftwareRequirement Class Reference . . . . .	378
5.159ArcSec::Source Class Reference . . . . .	385

5.160ArcSec::SourceFile Class Reference . . . . .	387
5.161ArcSec::SourceURL Class Reference . . . . .	388
5.162Arc::SubmitterPlugin Class Reference . . . . .	389
5.163Arc::SubmitterPluginLoader Class Reference . . . . .	391
5.164Arc::ThreadDataItem Class Reference . . . . .	393
5.165Arc::ThreadedPointer< T > Class Template Reference . . . . .	395
5.166Arc::ThreadedPointerBase Class Reference . . . . .	398
5.167Arc::ThreadRegistry Class Reference . . . . .	399
5.168Arc::Time Class Reference . . . . .	400
5.169ArcSec::TimeAttribute Class Reference . . . . .	403
5.170DataStaging::TransferParameters Class Reference . . . . .	405
5.171DataStaging::TransferShares Class Reference . . . . .	406
5.172DataStaging::TransferSharesConf Class Reference . . . . .	408
5.173Arc::URL Class Reference . . . . .	411
5.174Arc::URLLocation Class Reference . . . . .	422
5.175Arc::UserConfig Class Reference . . . . .	424
5.176Arc::UsernameToken Class Reference . . . . .	450
5.177Arc::UserSwitch Class Reference . . . . .	452
5.178Arc::VOMSTrustList Class Reference . . . . .	453
5.179Arc::WSAEndpointReference Class Reference . . . . .	455
5.180Arc::WSAHeader Class Reference . . . . .	457
5.181Arc::WSRF Class Reference . . . . .	461
5.182Arc::WSRFBaseFault Class Reference . . . . .	463
5.183Arc::WSRP Class Reference . . . . .	465
5.184Arc::WSRPFault Class Reference . . . . .	467
5.185Arc::WSRPResourcePropertyChangeFailure Class Reference . . . . .	468
5.186Arc::X509Token Class Reference . . . . .	469
5.187Arc::XMLNode Class Reference . . . . .	471
5.188Arc::XMLNodeContainer Class Reference . . . . .	482
5.189Arc::XMLSecNode Class Reference . . . . .	484



# Chapter 1

## Hosting Environment (Daemon) Namespace Index

### 1.1 Hosting Environment (Daemon) Namespace List

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

<a href="#">Arc</a> ( <a href="#">Arc</a> namespace contains all core ARC classes ) . . . . .	15
<a href="#">ArcCredential</a> . . . . .	44
<a href="#">DataStaging</a> ( <a href="#">DataStaging</a> contains all components for data transfer scheduling and execution ) .	46



## Chapter 2

# Hosting Environment (Daemon) Hierarchical Index

### 2.1 Hosting Environment (Daemon) Class Hierarchy

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

Arc::ArcLocation . . . . .	54
Arc::ArcVersion . . . . .	55
ArcSec::Attr . . . . .	56
Arc::AttributeIterator . . . . .	58
ArcSec::AttributeProxy . . . . .	61
ArcSec::AttributeValue . . . . .	62
ArcSec::DateTimeAttribute . . . . .	133
ArcSec::DurationAttribute . . . . .	174
ArcSec::PeriodAttribute . . . . .	311
ArcSec::TimeAttribute . . . . .	403
ArcSec::Attrs . . . . .	64
ArcSec::AuthzRequestSection . . . . .	65
Arc::AutoPointer< T > . . . . .	66
Arc::BaseConfig . . . . .	68
Arc::ChainContext . . . . .	70
Arc::Checksum . . . . .	71
Arc::Adler32Sum . . . . .	49
Arc::ChecksumAny . . . . .	74
Arc::CRC32Sum . . . . .	102
Arc::MD5Sum . . . . .	272
Arc::ClientInterface . . . . .	80
Arc::ClientTCP . . . . .	83
Arc::ClientHTTP . . . . .	79
Arc::ClientSOAP . . . . .	81
ArcSec::CombiningAlg . . . . .	84
ArcSec::DenyOverridesCombiningAlg . . . . .	146
ArcSec::PermitOverridesCombiningAlg . . . . .	313
Arc::ConfusaCertHandler . . . . .	88
Arc::ConfusaParserUtils . . . . .	89

Arc::CountedPointer< T > . . . . .	91
Arc::CountedPointer< Arc::Broker > . . . . .	91
Arc::Counter . . . . .	93
Arc::IntraProcessCounter . . . . .	216
Arc::CounterTicket . . . . .	100
Arc::Credential . . . . .	105
Arc::CredentialError . . . . .	114
Arc::CredentialStore . . . . .	115
Arc::Database . . . . .	116
Arc::MySQLDatabase . . . . .	292
DataStaging::DataDeliveryComm . . . . .	121
DataStaging::DataDeliveryLocalComm . . . . .	129
DataStaging::DataDeliveryRemoteComm . . . . .	131
DataStaging::DataDeliveryComm::Status . . . . .	126
DataStaging::DataDeliveryCommHandler . . . . .	128
Arc::DelegationConsumer . . . . .	135
Arc::DelegationConsumerSOAP . . . . .	137
Arc::DelegationContainerSOAP . . . . .	139
Arc::DelegationProvider . . . . .	142
Arc::DelegationProviderSOAP . . . . .	144
DataStaging::DTR . . . . .	148
DataStaging::DTRCacheParameters . . . . .	160
DataStaging::DTRCallback . . . . .	162
DataStaging::DataDelivery . . . . .	119
DataStaging::Generator . . . . .	201
DataStaging::Processor . . . . .	330
DataStaging::Scheduler . . . . .	349
DataStaging::DTRErrorStatus . . . . .	163
DataStaging::DTRLList . . . . .	166
DataStaging::DTRStatus . . . . .	169
ArcSec::EvalResult . . . . .	178
ArcSec::EvaluationCtx . . . . .	179
ArcSec::EvaluatorContext . . . . .	183
ArcSec::EvaluatorLoader . . . . .	184
Arc::ExecutableType . . . . .	186
Arc::ExecutionTarget . . . . .	187
Arc::ExpirationReminder . . . . .	189
Arc::FileAccess . . . . .	191
Arc::FileLock . . . . .	196
ArcSec::Function . . . . .	200
ArcSec::EqualFunction . . . . .	176
ArcSec::MatchFunction . . . . .	260
Arc::GLUE2 . . . . .	202
Arc::InfoCache . . . . .	203
Arc::InfoFilter . . . . .	204
Arc::InfoRegister . . . . .	205
Arc::InfoRegisterContainer . . . . .	206
Arc::InfoRegisters . . . . .	207
Arc::InfoRegistrar . . . . .	208
Arc::InformationInterface . . . . .	211
Arc::InformationContainer . . . . .	209

Arc::InformationRequest . . . . .	213
Arc::InformationResponse . . . . .	214
Arc::initializeCredentialsType . . . . .	215
Arc::Job . . . . .	220
Arc::JobDescription . . . . .	229
Arc::JobIdentificationType . . . . .	235
Arc::JobState . . . . .	237
Arc::JobSupervisor . . . . .	238
Arc::Loader . . . . .	245
Arc::JobControllerPluginLoader . . . . .	227
Arc::JobDescriptionParserLoader . . . . .	233
Arc::MCCLoader . . . . .	270
Arc::SubmitterPluginLoader . . . . .	391
Arc::LogDestination . . . . .	246
Arc::LogFile . . . . .	248
Arc::LogStream . . . . .	258
Arc::Logger . . . . .	251
Arc::LoggerContext . . . . .	255
Arc::LogMessage . . . . .	256
Arc::MCC_Status . . . . .	265
Arc::Message . . . . .	275
Arc::MessageAttributes . . . . .	278
Arc::MessageAuth . . . . .	281
Arc::MessageAuthContext . . . . .	283
Arc::MessageContext . . . . .	284
Arc::MessageContextElement . . . . .	285
Arc::MessagePayload . . . . .	286
Arc::PayloadRawInterface . . . . .	300
Arc::PayloadRaw . . . . .	298
Arc::PayloadSOAP . . . . .	302
Arc::PayloadStreamInterface . . . . .	306
Arc::PayloadStream . . . . .	303
Arc::PayloadWSRF . . . . .	309
Arc::ModuleDesc . . . . .	287
Arc::ModuleManager . . . . .	288
Arc::PluginsFactory . . . . .	323
Arc::OAuthConsumer . . . . .	294
Arc::PathIterator . . . . .	296
Arc::PlexerEntry . . . . .	317
Arc::Plugin . . . . .	318
Arc::JobDescriptionParser . . . . .	232
Arc::MCCInterface . . . . .	268
Arc::MCC . . . . .	262
Arc::Plexer . . . . .	315
Arc::Service . . . . .	361
Arc::RegisteredService . . . . .	332
Arc::SubmitterPlugin . . . . .	389
ArcSec::AlgFactory . . . . .	52
ArcSec::AttributeFactory . . . . .	57
ArcSec::Evaluator . . . . .	180
ArcSec::FnFactory . . . . .	199

ArcSec::PDP . . . . .	310
ArcSec::Policy . . . . .	325
ArcSec::Request . . . . .	336
ArcSec::SecHandler . . . . .	358
Arc::PluginArgument . . . . .	320
Arc::PluginDesc . . . . .	321
Arc::PluginDescriptor . . . . .	322
ArcSec::PolicyParser . . . . .	328
ArcSec::PolicyStore . . . . .	329
Arc::RegularExpression . . . . .	333
Arc::RemoteLoggingType . . . . .	335
ArcSec::RequestAttribute . . . . .	338
ArcSec::RequestItem . . . . .	339
ArcSec::Response . . . . .	340
ArcSec::ResponseItem . . . . .	341
Arc::Run . . . . .	342
Arc::SAMLToken . . . . .	346
Arc::SecAttr . . . . .	352
Arc::MultiSecAttr . . . . .	291
Arc::SecAttrFormat . . . . .	355
Arc::SecAttrValue . . . . .	356
Arc::CStringValue . . . . .	77
ArcSec::Security . . . . .	360
Arc::SimpleCondition . . . . .	364
Arc::SimpleCounter . . . . .	366
Arc::SOAPMessage . . . . .	368
Arc::Software . . . . .	370
Arc::ApplicationEnvironment . . . . .	53
Arc::SoftwareRequirement . . . . .	378
ArcSec::Source . . . . .	385
ArcSec::SourceFile . . . . .	387
ArcSec::SourceURL . . . . .	388
Arc::ThreadDataItem . . . . .	393
Arc::ThreadedPointer< T > . . . . .	395
Arc::ThreadedPointer< Arc::SimpleCounter > . . . . .	395
Arc::ThreadedPointerBase . . . . .	398
Arc::ThreadRegistry . . . . .	399
Arc::Time . . . . .	400
DataStaging::TransferParameters . . . . .	405
DataStaging::TransferShares . . . . .	406
DataStaging::TransferSharesConf . . . . .	408
Arc::URL . . . . .	411
Arc::URLLocation . . . . .	422
Arc::UserConfig . . . . .	424
Arc::UsernameToken . . . . .	450
Arc::UserSwitch . . . . .	452
Arc::VOMSTrustList . . . . .	453
Arc::WSAEndpointReference . . . . .	455
Arc::WSAHeader . . . . .	457
Arc::WSRF . . . . .	461
Arc::WSRFBBaseFault . . . . .	463
Arc::WSRPFault . . . . .	467

---

Arc::WSRPResourcePropertyChangeFailure . . . . .	468
Arc::WSRP . . . . .	465
Arc::X509Token . . . . .	469
Arc::XMLNode . . . . .	471
Arc::Config . . . . .	86
Arc::XMLSecNode . . . . .	484
ArcSec::SecHandlerConfig . . . . .	359
Arc::XMLNodeContainer . . . . .	482





## Chapter 3

# Hosting Environment (Daemon) Data Structure Index

### 3.1 Hosting Environment (Daemon) Data Structures

Here are the data structures with brief descriptions:

<a href="#">Arc::Adler32Sum</a> (Implementation of Adler32 checksum ) . . . . .	49
<a href="#">ArcSec::AlgFactory</a> (Interface for algorithm factory class ) . . . . .	52
<a href="#">Arc::ApplicationEnvironment</a> ( <a href="#">ApplicationEnvironment</a> ) . . . . .	53
<a href="#">Arc::ArcLocation</a> (Determines ARC installation location ) . . . . .	54
<a href="#">Arc::ArcVersion</a> (Determines ARC HED libraries version ) . . . . .	55
<a href="#">ArcSec::Attr</a> ( <a href="#">Attr</a> contains a tuple of attribute type and value ) . . . . .	56
<a href="#">ArcSec::AttributeFactory</a> . . . . .	57
<a href="#">Arc::AttributeIterator</a> (A const iterator class for accessing multiple values of an attribute ) . . . .	58
<a href="#">ArcSec::AttributeProxy</a> (Interface for creating the <a href="#">AttributeValue</a> object, it will be used by <a href="#">AttributeFactory</a> ) . . . . .	61
<a href="#">ArcSec::AttributeValue</a> (Interface for containing different type of <Attribute> node for both policy and request ) . . . . .	62
<a href="#">ArcSec::Attrs</a> ( <a href="#">Attrs</a> is a container for one or more <a href="#">Attr</a> ) . . . . .	64
<a href="#">ArcSec::AuthzRequestSection</a> . . . . .	65
<a href="#">Arc::AutoPointer&lt; T &gt;</a> (Wrapper for pointer with automatic destruction ) . . . . .	66
<a href="#">Arc::BaseConfig</a> . . . . .	68
<a href="#">Arc::ChainContext</a> (Interface to chain specific functionality ) . . . . .	70
<a href="#">Arc::Checksum</a> (Interface for checksum manipulations ) . . . . .	71
<a href="#">Arc::ChecksumAny</a> (Wrapper for <a href="#">Checksum</a> class ) . . . . .	74
<a href="#">Arc::CIStrngValue</a> (This class implements case insensitive strings as security attributes ) . . . .	77
<a href="#">Arc::ClientHTTP</a> (Class for setting up a <a href="#">MCC</a> chain for HTTP communication ) . . . . .	79
<a href="#">Arc::ClientInterface</a> (Utility base class for <a href="#">MCC</a> ) . . . . .	80
<a href="#">Arc::ClientSOAP</a> . . . . .	81
<a href="#">Arc::ClientTCP</a> (Class for setting up a <a href="#">MCC</a> chain for TCP communication ) . . . . .	83
<a href="#">ArcSec::CombiningAlg</a> (Interface for combining algorithm ) . . . . .	84
<a href="#">Arc::Config</a> (Configuration element - represents (sub)tree of ARC configuration ) . . . . .	86
<a href="#">Arc::ConfusaCertHandler</a> . . . . .	88
<a href="#">Arc::ConfusaParserUtils</a> . . . . .	89
<a href="#">Arc::CountedPointer&lt; T &gt;</a> (Wrapper for pointer with automatic destruction and mutiple references ) . . . . .	91
<a href="#">Arc::Counter</a> (A class defining a common interface for counters ) . . . . .	93

<a href="#">Arc::CounterTicket</a> (A class for "tickets" that correspond to counter reservations ) . . . . .	100
<a href="#">Arc::CRC32Sum</a> (Implementation of CRC32 checksum ) . . . . .	102
<a href="#">Arc::Credential</a> . . . . .	105
<a href="#">Arc::CredentialError</a> . . . . .	114
<a href="#">Arc::CredentialStore</a> . . . . .	115
<a href="#">Arc::Database</a> (Interface for calling database client library ) . . . . .	116
<a href="#">DataStaging::DataDelivery</a> ( <a href="#">DataDelivery</a> transfers data between specified physical locations ) . . . . .	119
<a href="#">DataStaging::DataDeliveryComm</a> (This class provides an abstract interface for the Delivery layer ) . . . . .	121
<a href="#">DataStaging::DataDeliveryComm::Status</a> (Plain C struct to pass information from executing process back to main thread ) . . . . .	126
<a href="#">DataStaging::DataDeliveryCommHandler</a> (Singleton class handling all active <a href="#">DataDeliveryComm</a> objects ) . . . . .	128
<a href="#">DataStaging::DataDeliveryLocalComm</a> (This class starts, monitors and controls a local Delivery process ) . . . . .	129
<a href="#">DataStaging::DataDeliveryRemoteComm</a> (This class contacts a remote service to make a Delivery request ) . . . . .	131
<a href="#">ArcSec::DateTimeAttribute</a> . . . . .	133
<a href="#">Arc::DelegationConsumer</a> . . . . .	135
<a href="#">Arc::DelegationConsumerSOAP</a> . . . . .	137
<a href="#">Arc::DelegationContainerSOAP</a> . . . . .	139
<a href="#">Arc::DelegationProvider</a> . . . . .	142
<a href="#">Arc::DelegationProviderSOAP</a> . . . . .	144
<a href="#">ArcSec::DenyOverridesCombiningAlg</a> (Implement the "Deny-Overrides" algorithm ) . . . . .	146
<a href="#">DataStaging::DTR</a> (Data Transfer Request ) . . . . .	148
<a href="#">DataStaging::DTRCacheParameters</a> (The configured cache directories ) . . . . .	160
<a href="#">DataStaging::DTRCallback</a> (The base class from which all callback-enabled classes should be derived ) . . . . .	162
<a href="#">DataStaging::DTRErrorStatus</a> (A class to represent error states reported by various components ) . . . . .	163
<a href="#">DataStaging::DTRLList</a> (Global list of all active DTRs in the system ) . . . . .	166
<a href="#">DataStaging::DTRStatus</a> (Class representing the status of a <a href="#">DTR</a> ) . . . . .	169
<a href="#">ArcSec::DurationAttribute</a> . . . . .	174
<a href="#">ArcSec::EqualFunction</a> (Evaluate whether the two values are equal ) . . . . .	176
<a href="#">ArcSec::EvalResult</a> (Struct to record the xml node and effect, which will be used by <a href="#">Evaluator</a> to get the information about which rule/policy(in xmlnode) is satisfied ) . . . . .	178
<a href="#">ArcSec::EvaluationCtx</a> ( <a href="#">EvaluationCtx</a> , in charge of storing some context information for ) . . . . .	179
<a href="#">ArcSec::Evaluator</a> (Interface for policy evaluation. Execute the policy evaluation, based on the request and policy ) . . . . .	180
<a href="#">ArcSec::EvaluatorContext</a> (Context for evaluator. It includes the factories which will be used to create related objects ) . . . . .	183
<a href="#">ArcSec::EvaluatorLoader</a> ( <a href="#">EvaluatorLoader</a> is implemented as a helper class for loading different <a href="#">Evaluator</a> objects, like <a href="#">ArcEvaluator</a> ) . . . . .	184
<a href="#">Arc::ExecutableType</a> (Executable ) . . . . .	186
<a href="#">Arc::ExecutionTarget</a> ( <a href="#">ExecutionTarget</a> ) . . . . .	187
<a href="#">Arc::ExpirationReminder</a> (A class intended for internal use within counters ) . . . . .	189
<a href="#">Arc::FileAccess</a> (Defines interface for accessing filesystems ) . . . . .	191
<a href="#">Arc::FileLock</a> (A general file locking class ) . . . . .	196
<a href="#">ArcSec::FnFactory</a> (Interface for function factory class ) . . . . .	199
<a href="#">ArcSec::Function</a> (Interface for function, which is in charge of evaluating two <a href="#">AttributeValue</a> ) . . . . .	200
<a href="#">DataStaging::Generator</a> (Simple <a href="#">Generator</a> implementation ) . . . . .	201
<a href="#">Arc::GLUE2</a> ( <a href="#">GLUE2</a> parser ) . . . . .	202
<a href="#">Arc::InfoCache</a> (Stores XML document in filesystem split into parts ) . . . . .	203
<a href="#">Arc::InfoFilter</a> (Filters information document according to identity of requestor ) . . . . .	204
<a href="#">Arc::InfoRegister</a> (Registration to ISIS interface ) . . . . .	205
<a href="#">Arc::InfoRegisterContainer</a> . . . . .	206

<a href="#">Arc::InfoRegisters</a> (Handling multiple registrations to ISISes ) . . . . .	207
<a href="#">Arc::InfoRegistrar</a> (Registration process associated with particular ISIS ) . . . . .	208
<a href="#">Arc::InformationContainer</a> (Information System document container and processor ) . . . . .	209
<a href="#">Arc::InformationInterface</a> (Information System message processor ) . . . . .	211
<a href="#">Arc::InformationRequest</a> (Request for information in InfoSystem ) . . . . .	213
<a href="#">Arc::InformationResponse</a> (Informational response from InfoSystem ) . . . . .	214
<a href="#">Arc::initializeCredentialsType</a> (Defines how user credentials are looked for ) . . . . .	215
<a href="#">Arc::IntraProcessCounter</a> (A class for counters used by threads within a single process ) . . . . .	216
<a href="#">Arc::Job</a> (Job ) . . . . .	220
<a href="#">Arc::JobControllerPluginLoader</a> . . . . .	227
<a href="#">Arc::JobDescription</a> . . . . .	229
<a href="#">Arc::JobDescriptionParser</a> (Abstract class for the different parsers ) . . . . .	232
<a href="#">Arc::JobDescriptionParserLoader</a> . . . . .	233
<a href="#">Arc::JobIdentificationType</a> (Job identification ) . . . . .	235
<a href="#">Arc::JobState</a> . . . . .	237
<a href="#">Arc::JobSupervisor</a> (% <a href="#">JobSupervisor</a> class ) . . . . .	238
<a href="#">Arc::Loader</a> (Plugins loader ) . . . . .	245
<a href="#">Arc::LogDestination</a> (A base class for log destinations ) . . . . .	246
<a href="#">Arc::LogFile</a> (A class for logging to files ) . . . . .	248
<a href="#">Arc::Logger</a> (A logger class ) . . . . .	251
<a href="#">Arc::LoggerContext</a> (Container for logger configuration ) . . . . .	255
<a href="#">Arc::LogMessage</a> (A class for log messages ) . . . . .	256
<a href="#">Arc::LogStream</a> (A class for logging to ostreams ) . . . . .	258
<a href="#">ArcSec::MatchFunction</a> (Evaluate whether arg1 (value in regular expression) matched arg0 (label in regular expression) ) . . . . .	260
<a href="#">Arc::MCC</a> (Message Chain Component - base class for every <a href="#">MCC</a> plugin ) . . . . .	262
<a href="#">Arc::MCC_Status</a> (A class for communication of <a href="#">MCC</a> processing results ) . . . . .	265
<a href="#">Arc::MCCInterface</a> (Interface for communication between <a href="#">MCC</a> , <a href="#">Service</a> and <a href="#">Plexer</a> objects ) . . . . .	268
<a href="#">Arc::MCCLoader</a> (Creator of <a href="#">Message</a> Component Chains ( <a href="#">MCC</a> ) ) . . . . .	270
<a href="#">Arc::MD5Sum</a> (Implementation of MD5 checksum ) . . . . .	272
<a href="#">Arc::Message</a> (Object being passed through chain of <a href="#">MCCs</a> ) . . . . .	275
<a href="#">Arc::MessageAttributes</a> (A class for storage of attribute values ) . . . . .	278
<a href="#">Arc::MessageAuth</a> (Contains authenticity information, authorization tokens and decisions ) . . . . .	281
<a href="#">Arc::MessageAuthContext</a> (Handler for content of message auth* context ) . . . . .	283
<a href="#">Arc::MessageContext</a> (Handler for content of message context ) . . . . .	284
<a href="#">Arc::MessageContextElement</a> (Top class for elements contained in message context ) . . . . .	285
<a href="#">Arc::MessagePayload</a> (Base class for content of message passed through chain ) . . . . .	286
<a href="#">Arc::ModuleDesc</a> (Description of loadable module ) . . . . .	287
<a href="#">Arc::ModuleManager</a> (Manager of shared libraries ) . . . . .	288
<a href="#">Arc::MultiSecAttr</a> (Container of multiple <a href="#">SecAttr</a> attributes ) . . . . .	291
<a href="#">Arc::MySQLDatabase</a> . . . . .	292
<a href="#">Arc::OAuthConsumer</a> . . . . .	294
<a href="#">Arc::PathIterator</a> (Class to iterate through elements of path ) . . . . .	296
<a href="#">Arc::PayloadRaw</a> (Raw byte multi-buffer ) . . . . .	298
<a href="#">Arc::PayloadRawInterface</a> (Random Access Payload for <a href="#">Message</a> objects ) . . . . .	300
<a href="#">Arc::PayloadSOAP</a> (Payload of <a href="#">Message</a> with SOAP content ) . . . . .	302
<a href="#">Arc::PayloadStream</a> (POSIX handle as Payload ) . . . . .	303
<a href="#">Arc::PayloadStreamInterface</a> (Stream-like Payload for <a href="#">Message</a> object ) . . . . .	306
<a href="#">Arc::PayloadWSRF</a> (This class combines <a href="#">MessagePayload</a> with <a href="#">WSRF</a> ) . . . . .	309
<a href="#">ArcSec::PDP</a> (Base class for <a href="#">Policy</a> Decision Point plugins ) . . . . .	310
<a href="#">ArcSec::PeriodAttribute</a> . . . . .	311
<a href="#">ArcSec::PermitOverridesCombiningAlg</a> (Implement the "Permit-Overrides" algorithm ) . . . . .	313
<a href="#">Arc::Plexer</a> (The <a href="#">Plexer</a> class, used for routing messages to services ) . . . . .	315
<a href="#">Arc::PlexerEntry</a> (A pair of label (regex) and pointer to <a href="#">MCC</a> ) . . . . .	317

<a href="#">Arc::Plugin</a> (Base class for loadable ARC components ) . . . . .	318
<a href="#">Arc::PluginArgument</a> (Base class for passing arguments to loadable ARC components ) . . . . .	320
<a href="#">Arc::PluginDesc</a> (Description of plugin ) . . . . .	321
<a href="#">Arc::PluginDescriptor</a> (Description of ARC lodable component ) . . . . .	322
<a href="#">Arc::PluginsFactory</a> (Generic ARC plugins loader ) . . . . .	323
<a href="#">ArcSec::Policy</a> (Interface for containing and processing different types of policy ) . . . . .	325
<a href="#">ArcSec::PolicyParser</a> (A interface which will isolate the policy object from actual policy storage (files, urls, database) ) . . . . .	328
<a href="#">ArcSec::PolicyStore</a> (Storage place for policy objects ) . . . . .	329
<a href="#">DataStaging::Processor</a> (The <a href="#">Processor</a> performs pre- and post-transfer operations ) . . . . .	330
<a href="#">Arc::RegisteredService</a> ( <a href="#">RegisteredService</a> - extension of <a href="#">Service</a> performing self-registration ) . . . . .	332
<a href="#">Arc::RegularExpression</a> (A regular expression class ) . . . . .	333
<a href="#">Arc::RemoteLoggingType</a> (Remote logging ) . . . . .	335
<a href="#">ArcSec::Request</a> (Base class/Interface for request, includes a container for RequestItems and some operations ) . . . . .	336
<a href="#">ArcSec::RequestAttribute</a> (Wrapper which includes <a href="#">AttributeValue</a> object which is generated ac- cording to date type of one spefic node in Request.xml ) . . . . .	338
<a href="#">ArcSec::RequestItem</a> (Interface for request item container, <subjects, actions, objects, ctxs> tuple ) . . . . .	339
<a href="#">ArcSec::Response</a> (Container for the evaluation results ) . . . . .	340
<a href="#">ArcSec::ResponseItem</a> (Evaluation result concerning one RequestTuple ) . . . . .	341
<a href="#">Arc::Run</a> . . . . .	342
<a href="#">Arc::SAMLToken</a> (Class for manipulating SAML Token Profile ) . . . . .	346
<a href="#">DataStaging::Scheduler</a> (The <a href="#">Scheduler</a> is the control centre of the data staging framework ) . . . . .	349
<a href="#">Arc::SecAttr</a> (This is an abstract interface to a security attribute ) . . . . .	352
<a href="#">Arc::SecAttrFormat</a> (Export/import format ) . . . . .	355
<a href="#">Arc::SecAttrValue</a> (This is an abstract interface to a security attribute ) . . . . .	356
<a href="#">ArcSec::SecHandler</a> (Base class for simple security handling plugins ) . . . . .	358
<a href="#">ArcSec::SecHandlerConfig</a> . . . . .	359
<a href="#">ArcSec::Security</a> (Common stuff used by security related slasses ) . . . . .	360
<a href="#">Arc::Service</a> ( <a href="#">Service</a> - last component in a <a href="#">Message</a> Chain ) . . . . .	361
<a href="#">Arc::SimpleCondition</a> (Simple triggered condition ) . . . . .	364
<a href="#">Arc::SimpleCounter</a> . . . . .	366
<a href="#">Arc::SOAPMessage</a> ( <a href="#">Message</a> restricted to SOAP payload ) . . . . .	368
<a href="#">Arc::Software</a> (Used to represent software (names and version) and comparison ) . . . . .	370
<a href="#">Arc::SoftwareRequirement</a> (Class used to express and resolve version requirements on software ) . . . . .	378
<a href="#">ArcSec::Source</a> (Acquires and parses XML document from specified source ) . . . . .	385
<a href="#">ArcSec::SourceFile</a> (Convenience class for obtaining XML document from file ) . . . . .	387
<a href="#">ArcSec::SourceURL</a> (Convenience class for obtaining XML document from remote URL ) . . . . .	388
<a href="#">Arc::SubmitterPlugin</a> (Base class for the SubmitterPlugins ) . . . . .	389
<a href="#">Arc::SubmitterPluginLoader</a> . . . . .	391
<a href="#">Arc::ThreadDataItem</a> (Base class for per-thread object ) . . . . .	393
<a href="#">Arc::ThreadedPointer&lt; T &gt;</a> (Wrapper for pointer with automatic destruction and mutiple refer- ences ) . . . . .	395
<a href="#">Arc::ThreadedPointerBase</a> (Helper class for <a href="#">ThreadedPointer</a> ) . . . . .	398
<a href="#">Arc::ThreadRegistry</a> . . . . .	399
<a href="#">Arc::Time</a> (A class for storing and manipulating times ) . . . . .	400
<a href="#">ArcSec::TimeAttribute</a> . . . . .	403
<a href="#">DataStaging::TransferParameters</a> . . . . .	405
<a href="#">DataStaging::TransferShares</a> ( <a href="#">TransferShares</a> is used to implement fair-sharing and priorities ) . . . . .	406
<a href="#">DataStaging::TransferSharesConf</a> ( <a href="#">TransferSharesConf</a> describes the configuration of <a href="#">Transfer-</a> <a href="#">Shares</a> ) . . . . .	408
<a href="#">Arc::URL</a> (Class to hold general URLs ) . . . . .	411
<a href="#">Arc::URLLocation</a> (Class to hold a resolved <a href="#">URL</a> location ) . . . . .	422

<a href="#">Arc::UserConfig</a> (User configuration class ) . . . . .	424
<a href="#">Arc::UsernameToken</a> (Interface for manipulation of WS-Security according to Username Token Profile ) . . . . .	450
<a href="#">Arc::UserSwitch</a> . . . . .	452
<a href="#">Arc::VOMSTrustList</a> . . . . .	453
<a href="#">Arc::WSAEndpointReference</a> (Interface for manipulation of WS-Adressing Endpoint Reference )	455
<a href="#">Arc::WSAHeader</a> (Interface for manipulation WS-Addressing information in SOAP header ) . .	457
<a href="#">Arc::WSRF</a> (Base class for every <a href="#">WSRF</a> message ) . . . . .	461
<a href="#">Arc::WSRFBaseFault</a> (Base class for <a href="#">WSRF</a> fault messages ) . . . . .	463
<a href="#">Arc::WSRP</a> (Base class for WS-ResourceProperties structures ) . . . . .	465
<a href="#">Arc::WSRPFault</a> (Base class for WS-ResourceProperties faults ) . . . . .	467
<a href="#">Arc::WSRPResourcePropertyChangeFailure</a> . . . . .	468
<a href="#">Arc::X509Token</a> (Class for manipulating X.509 Token Profile ) . . . . .	469
<a href="#">Arc::XMLNode</a> (Wrapper for LibXML library Tree interface ) . . . . .	471
<a href="#">Arc::XMLNodeContainer</a> . . . . .	482
<a href="#">Arc::XMLSecNode</a> (Extends <a href="#">XMLNode</a> class to support XML security operation ) . . . . .	484



## Chapter 4

# Hosting Environment (Daemon) Namespace Documentation

### 4.1 Arc Namespace Reference

[Arc](#) namespace contains all core ARC classes.

#### Data Structures

- class **BrokerPluginArgument**
- class **BrokerPlugin**
- class **BrokerPluginLoader**
- class **Broker**
- class **CountedBroker**
- class **ExecutionTargetSet**
- class [ClientInterface](#)

*Utility base class for [MCC](#).*

- class [ClientTCP](#)

*Class for setting up a [MCC](#) chain for TCP communication.*

- struct **HTTPClientInfo**
- class [ClientHTTP](#)

*Class for setting up a [MCC](#) chain for HTTP communication.*

- class [ClientSOAP](#)
- class **SecHandlerConfig**
- class **DNListHandlerConfig**
- class **ARCPolicyHandlerConfig**
- class **ClientHTTPwithSAML2SSO**
- class **ClientSOAPwithSAML2SSO**
- class **ClientX509Delegation**
- class **ComputingServiceRetriever**
- class [ConfusaCertHandler](#)
- class [ConfusaParserUtils](#)

- class **HakaClient**
- class **OpenIdpClient**
- class [OAuthConsumer](#)
- class **SAML2LoginClient**
- class **SAML2SSOHTTPClient**
- class **Endpoint**
- class **EndpointQueryingStatus**
- class **EndpointQueryOptions**
- class **EndpointQueryOptions**< **Endpoint** >
- class **EntityRetrieverPlugin**
- class **EntityRetrieverPluginLoader**
- class **EntityConsumer**
- class **EntityContainer**
- class **EntityRetriever**
- class [ApplicationEnvironment](#)

*[ApplicationEnvironment](#).*

- class **LocationAttributes**
- class **AdminDomainAttributes**
- class **ExecutionEnvironmentAttributes**
- class **ComputingManagerAttributes**
- class **ComputingShareAttributes**
- class **ComputingEndpointAttributes**
- class **ComputingServiceAttributes**
- class **LocationType**
- class **AdminDomainType**
- class **ExecutionEnvironmentType**
- class **ComputingManagerType**
- class **ComputingShareType**
- class **ComputingEndpointType**
- class **ComputingServiceType**
- class [ExecutionTarget](#)

*[ExecutionTarget](#).*

- class [GLUE2](#)
- [GLUE2](#) parser.*

- class **GLUE2Entity**
  - class [Job](#)
- [Job](#).*

- class **JobControllerPlugin**
- class [JobControllerPluginLoader](#)
- class **JobControllerPluginPluginArgument**
- class **OptIn**
- class **Range**
- class **ScalableTime**
- class **ScalableTime**< **int** >
- class [JobIdentificationType](#)

*[Job](#) identification.*



- class [ExecutableType](#)  
*Executable.*
- class [RemoteLoggingType](#)  
*Remote logging.*
- class **NotificationType**
- class **ApplicationType**
- class **SlotRequirementType**
- class **DiskSpaceRequirementType**
- class **ParallelEnvironmentType**
- class **ResourcesType**
- class **SourceType**
- class **TargetType**
- class **InputFileType**
- class **OutputFileType**
- class **DataStagingType**
- class **JobDescriptionResult**
- class [JobDescription](#)
- class **JobDescriptionParserResult**
- class [JobDescriptionParser](#)  
*Abstract class for the different parsers.*
- class [JobDescriptionParserLoader](#)
- class [JobState](#)
- class [JobSupervisor](#)  
*% JobSupervisor class*
- class [Software](#)  
*Used to represent software (names and version) and comparison.*
- class [SoftwareRequirement](#)  
*Class used to express and resolve version requirements on software.*
- class **Submitter**
- class [SubmitterPlugin](#)  
*Base class for the SubmitterPlugins.*
- class [SubmitterPluginLoader](#)
- class **SubmitterPluginArgument**
- class **BrokerPluginTestACCControl**
- class **JobDescriptionParserTestACCControl**
- class **JobControllerPluginTestACCControl**
- class **SubmitterPluginTestACCControl**
- class **JobStateTEST**
- class **JobListRetrieverPluginTESTControl**
- class **ServiceEndpointRetrieverPluginTESTControl**
- class **TargetInformationRetrieverPluginTESTControl**
- class [Config](#)

*Configuration element - represents (sub)tree of ARC configuration.*

- class [BaseConfig](#)
- class [ArcLocation](#)

*Determines ARC installation location.*

- class [RegularExpression](#)

*A regular expression class.*

- class [ArcVersion](#)

*Determines ARC HED libraries version.*

- class **Base64**
- class [Checksum](#)

*Interface for checksum manipulations.*

- class [CRC32Sum](#)

*Implementation of CRC32 checksum.*

- class [MD5Sum](#)

*Implementation of MD5 checksum.*

- class [Adler32Sum](#)

*Implementation of Adler32 checksum.*

- class [ChecksumAny](#)

*Wrapper for [Checksum](#) class.*

- class [Counter](#)

*A class defining a common interface for counters.*

- class [CounterTicket](#)

*A class for "tickets" that correspond to counter reservations.*

- class [ExpirationReminder](#)

*A class intended for internal use within counters.*

- class **Period**
- class [Time](#)

*A class for storing and manipulating times.*

- class [Database](#)

*Interface for calling database client library.*

- class **Query**
- class [FileAccess](#)

*Defines interface for accessing filesystems.*

- class [FileLock](#)

*A general file locking class.*

- class **IniConfig**
- class [IntraProcessCounter](#)  
*A class for counters used by threads within a single process.*
- class **PrintfBase**
- class **Printf**
- class **IString**
- struct **LoggerFormat**
- class [LogMessage](#)  
*A class for log messages.*
- class [LogDestination](#)  
*A base class for log destinations.*
- class [LogStream](#)  
*A class for logging to ostreams.*
- class [LogFile](#)  
*A class for logging to files.*
- class [LoggerContext](#)  
*Container for logger configuration.*
- class [Logger](#)  
*A logger class.*
- class [MySQLDatabase](#)
- class [MySQLQuery](#)
- class **OptionParser**
- class **Profile**
- class [Run](#)
- class [ThreadDataItem](#)  
*Base class for per-thread object.*
- class [SimpleCondition](#)  
*Simple triggered condition.*
- class [SimpleCounter](#)
- class **TimedMutex**
- class **SharedMutex**
- class [ThreadedPointerBase](#)  
*Helper class for [ThreadedPointer](#).*
- class [ThreadedPointer](#)  
*Wrapper for pointer with automatic destruction and mutiple references.*
- class [ThreadRegistry](#)
- class **ThreadInitializer**
- class [URL](#)

*Class to hold general URLs.*

- class [URLLocation](#)

*Class to hold a resolved [URL](#) location.*

- class [PathIterator](#)

*Class to iterate through elements of path.*

- class **User**
- class [UserSwitch](#)
- class **ConfigEndpoint**
- class [initializeCredentialsType](#)

*Defines how user credentials are looked for.*

- class [UserConfig](#)

*User configuration class*

- class **CertEnvLocker**
- class **EnvLockWrapper**
- class [AutoPointer](#)

*Wrapper for pointer with automatic destruction.*

- class [CountedPointer](#)

*Wrapper for pointer with automatic destruction and mutiple references.*

- class **NS**
- class [XMLNode](#)

*Wrapper for LibXML library Tree interface.*

- class [XMLNodeContainer](#)
- class [CredentialError](#)
- class [Credential](#)
- class **VOMSACInfo**
- class [VOMSTrustList](#)
- class [CredentialStore](#)
- class **XmlContainer**
- class **XmlDatabase**
- class [DelegationConsumer](#)
- class [DelegationProvider](#)
- class [DelegationConsumerSOAP](#)
- class [DelegationProviderSOAP](#)
- class [DelegationContainerSOAP](#)
- class **GlobusResult**
- class **GSSCredential**
- class [InfoCache](#)

*Stores XML document in filesystem split into parts.*

- class **InfoCacheInterface**
- class [InfoFilter](#)

*Filters information document according to identity of requestor.*

- class [InfoRegister](#)  
*Registration to ISIS interface.*
- class [InfoRegisters](#)  
*Handling multiple registrations to ISISes.*
- struct **Register\_Info\_Type**
- struct **ISIS\_description**
- class [InfoRegistrar](#)  
*Registration process associated with particular ISIS.*
- class [InfoRegisterContainer](#)
- class [InformationInterface](#)  
*Information System message processor.*
- class [InformationContainer](#)  
*Information System document container and processor.*
- class [InformationRequest](#)  
*Request for information in InfoSystem.*
- class [InformationResponse](#)  
*Informational response from InfoSystem.*
- class [RegisteredService](#)  
*[RegisteredService](#) - extension of [Service](#) performing self-registration.*
- class **FinderLoader**
- class [Loader](#)  
*Plugins loader.*
- class **LoadableModuleDescription**
- class [ModuleManager](#)  
*Manager of shared libraries.*
- class [PluginArgument](#)  
*Base class for passing arguments to loadable ARC components.*
- class [Plugin](#)  
*Base class for loadable ARC components.*
- struct [PluginDescriptor](#)  
*Description of ARC loadable component.*
- class [PluginDesc](#)  
*Description of plugin.*
- class [ModuleDesc](#)  
*Description of loadable module.*

- class [PluginsFactory](#)  
*Generic ARC plugins loader.*
- class [MCCInterface](#)  
*Interface for communication between [MCC](#), [Service](#) and [Plexer](#) objects.*
- class [MCC](#)  
*[Message](#) Chain Component - base class for every [MCC](#) plugin.*
- class **MCCConfig**
- class **MCCPluginArgument**
- class [MCC\\_Status](#)  
*A class for communication of [MCC](#) processing results.*
- class [MCCLoader](#)  
*Creator of [Message](#) Component Chains ([MCC](#)).*
- class [ChainContext](#)  
*Interface to chain specific functionality.*
- class [MessagePayload](#)  
*Base class for content of message passed through chain.*
- class [MessageContextElement](#)  
*Top class for elements contained in message context.*
- class [MessageContext](#)  
*Handler for content of message context.*
- class [MessageAuthContext](#)  
*Handler for content of message auth\* context.*
- class [Message](#)  
*Object being passed through chain of [MCCs](#).*
- class [AttributeIterator](#)  
*A const iterator class for accessing multiple values of an attribute.*
- class [MessageAttributes](#)  
*A class for storage of attribute values.*
- class [MessageAuth](#)  
*Contains authenticity information, authorization tokens and decisions.*
- class [PayloadRawInterface](#)  
*Random Access Payload for [Message](#) objects.*
- struct **PayloadRawBuf**
- class [PayloadRaw](#)

*Raw byte multi-buffer.*

- class [PayloadSOAP](#)  
*Payload of [Message](#) with SOAP content.*
- class [PayloadStreamInterface](#)  
*Stream-like Payload for [Message](#) object.*
- class [PayloadStream](#)  
*POSIX handle as Payload.*
- class [PlexerEntry](#)  
*A pair of label (regex) and pointer to [MCC](#).*
- class [Plexer](#)  
*The [Plexer](#) class, used for routing messages to services.*
- class [CIStringValue](#)  
*This class implements case insensitive strings as security attributes.*
- class [SecAttrValue](#)  
*This is an abstract interface to a security attribute.*
- class [SecAttrFormat](#)  
*Export/import format.*
- class [SecAttr](#)  
*This is an abstract interface to a security attribute.*
- class [MultiSecAttr](#)  
*Container of multiple [SecAttr](#) attributes.*
- class [Service](#)  
*[Service](#) - last component in a [Message](#) Chain.*
- class **ServicePluginArgument**
- class [SOAPMessage](#)  
*[Message](#) restricted to SOAP payload.*
- class **ClassLoader**
- class **ClassLoaderPluginArgument**
- class [WSAEndpointReference](#)  
*Interface for manipulation of WS-Addressing Endpoint Reference.*
- class [WSAHeader](#)  
*Interface for manipulation WS-Addressing information in SOAP header.*
- class [SAMLToken](#)  
*Class for manipulating SAML Token Profile.*

- class [UsernameToken](#)

*Interface for manipulation of WS-Security according to Username Token Profile.*

- class [X509Token](#)

*Class for manipulating X.509 Token Profile.*

- class [PayloadWSRF](#)

*This class combines [MessagePayload](#) with [WSRF](#).*

- class [WSRP](#)

*Base class for WS-ResourceProperties structures.*

- class [WSRPFault](#)

*Base class for WS-ResourceProperties faults.*

- class **WSRPInvalidResourcePropertyQNameFault**
- class [WSRPResourcePropertyChangeFailure](#)
- class **WSRPUnableToPutResourcePropertyDocumentFault**
- class **WSRPInvalidModificationFault**
- class **WSRPUnableToModifyResourcePropertyFault**
- class **WSRPSetResourcePropertyRequestFailedFault**
- class **WSRPInsertResourcePropertiesRequestFailedFault**
- class **WSRPUpdateResourcePropertiesRequestFailedFault**
- class **WSRPDeleteResourcePropertiesRequestFailedFault**
- class **WSRPGetResourcePropertyDocumentRequest**
- class **WSRPGetResourcePropertyDocumentResponse**
- class **WSRPGetResourcePropertyRequest**
- class **WSRPGetResourcePropertyResponse**
- class **WSRPGetMultipleResourcePropertiesRequest**
- class **WSRPGetMultipleResourcePropertiesResponse**
- class **WSRPPutResourcePropertyDocumentRequest**
- class **WSRPPutResourcePropertyDocumentResponse**
- class **WSRPModifyResourceProperties**
- class **WSRPInsertResourceProperties**
- class **WSRPUpdateResourceProperties**
- class **WSRPDeleteResourceProperties**
- class **WSRPSetResourcePropertiesRequest**
- class **WSRPSetResourcePropertiesResponse**
- class **WSRPInsertResourcePropertiesRequest**
- class **WSRPInsertResourcePropertiesResponse**
- class **WSRPUpdateResourcePropertiesRequest**
- class **WSRPUpdateResourcePropertiesResponse**
- class **WSRPDeleteResourcePropertiesRequest**
- class **WSRPDeleteResourcePropertiesResponse**
- class **WSRPQueryResourcePropertiesRequest**
- class **WSRPQueryResourcePropertiesResponse**
- class [WSRF](#)

*Base class for every [WSRF](#) message.*

- class [WSRFBBaseFault](#)



Base class for [WSRF](#) fault messages.

- class **WSRFResourceUnknownFault**
- class **WSRFResourceUnavailableFault**
- class [XMLSecNode](#)

Extends [XMLNode](#) class to support XML security operation.

## Typedefs

- typedef [Plugin](#) \*(\*) [get\\_plugin\\_instance](#) ([PluginArgument](#) \*arg)
- typedef std::multimap< std::string, std::string > [AttrMap](#)
- typedef [AttrMap](#)::const\_iterator [AttrConstIter](#)
- typedef [AttrMap](#)::iterator [AttrIter](#)

## Enumerations

- enum [TimeFormat](#)
- enum [LogLevel](#)
- enum [LogFormat](#)
- enum [escape\\_type](#) { , [escape\\_octal](#), [escape\\_hex](#) }
- enum [StatusKind](#) { ,  
[STATUS\\_OK](#) = 1, [GENERIC\\_ERROR](#) = 2, [PARSING\\_ERROR](#) = 4, [PROTOCOL\\_-RECOGNIZED\\_ERROR](#) = 8,  
[UNKNOWN\\_SERVICE\\_ERROR](#) = 16, [BUSY\\_ERROR](#) = 32, [SESSION\\_CLOSE](#) = 64 }
- enum [WSAFault](#) { , [WSAFaultUnknown](#), [WSAFaultInvalidAddressingHeader](#) }

## Functions

- std::ostream & [operator<<](#) (std::ostream &, const [Period](#) &)
- std::ostream & [operator<<](#) (std::ostream &, const [Time](#) &)
- std::string [TimeStamp](#) (const [TimeFormat](#) &=Time::GetFormat())
- std::string [TimeStamp](#) ([Time](#), const [TimeFormat](#) &=Time::GetFormat())
- bool [FileCopy](#) (const std::string &source\_path, const std::string &destination\_path, uid\_t uid, gid\_t gid)
- bool [FileCopy](#) (const std::string &source\_path, const std::string &destination\_path)
- bool [FileCopy](#) (const std::string &source\_path, int destination\_handle)
- bool [FileCopy](#) (int source\_handle, const std::string &destination\_path)
- bool [FileCopy](#) (int source\_handle, int destination\_handle)
- bool [FileRead](#) (const std::string &filename, std::list< std::string > &data, uid\_t uid=0, gid\_t gid=0)
- bool [FileRead](#) (const std::string &filename, std::string &data, uid\_t uid=0, gid\_t gid=0)
- bool [FileCreate](#) (const std::string &filename, const std::string &data, uid\_t uid=0, gid\_t gid=0, mode\_t mode=0)
- bool [FileStat](#) (const std::string &path, struct stat \*st, bool follow\_symlinks)
- bool [FileStat](#) (const std::string &path, struct stat \*st, uid\_t uid, gid\_t gid, bool follow\_symlinks)
- bool [FileLink](#) (const std::string &oldpath, const std::string &newpath, bool symbolic)
- bool [FileLink](#) (const std::string &oldpath, const std::string &newpath, uid\_t uid, gid\_t gid, bool symbolic)
- std::string [FileReadLink](#) (const std::string &path)

- `std::string FileReadLink` (const `std::string` &path, `uid_t` uid, `gid_t` gid)
- `bool FileDelete` (const `std::string` &path)
- `bool FileDelete` (const `std::string` &path, `uid_t` uid, `gid_t` gid)
- `bool DirCreate` (const `std::string` &path, `mode_t` mode, bool with\_parents=false)
- `bool DirCreate` (const `std::string` &path, `uid_t` uid, `gid_t` gid, `mode_t` mode, bool with\_parents=false)
- `bool DirDelete` (const `std::string` &path, bool recursive=true)
- `bool DirDelete` (const `std::string` &path, bool recursive, `uid_t` uid, `gid_t` gid)
- `bool TmpDirCreate` (`std::string` &path)
- `bool TmpFileCreate` (`std::string` &filename, const `std::string` &data, `uid_t` uid=0, `gid_t` gid=0, `mode_t` mode=0)
- `bool CanonicalDir` (`std::string` &name, bool leading\_slash=true)
- `void GUID` (`std::string` &guid)
- `std::string UUID` (void)
- `std::ostream & operator<<` (`std::ostream` &os, `LogLevel` level)
- `LogLevel string_to_level` (const `std::string` &str)
- `bool istring_to_level` (const `std::string` &llstr, `LogLevel` &ll)
- `bool string_to_level` (const `std::string` &str, `LogLevel` &ll)
- `std::string level_to_string` (const `LogLevel` &level)
- `LogLevel old_level_to_level` (unsigned int old\_level)
- `template<typename T> T stringto` (const `std::string` &s)
- `template<typename T> bool stringto` (const `std::string` &s, T &t)
- `bool strtoint` (const `std::string` &s, signed int &t, int base=10)
- `bool strtoint` (const `std::string` &s, unsigned int &t, int base=10)
- `bool strtoint` (const `std::string` &s, signed long &t, int base=10)
- `bool strtoint` (const `std::string` &s, unsigned long &t, int base=10)
- `bool strtoint` (const `std::string` &s, signed long long &t, int base=10)
- `bool strtoint` (const `std::string` &s, unsigned long long &t, int base=10)
- `template<typename T> std::string tostring` (T t, int width=0, int precision=0)
- `std::string inttostr` (signed long long t, int base=10, int width=0)
- `std::string inttostr` (unsigned long long t, int base=10, int width=0)
- `std::string inttostr` (signed int t, int base=10, int width=0)
- `std::string inttostr` (unsigned int t, int base=10, int width=0)
- `std::string inttostr` (signed long t, int base=10, int width=0)
- `std::string inttostr` (unsigned long t, int base=10, int width=0)
- `std::string booltostr` (bool b)
- `bool strtobool` (const `std::string` &s)
- `bool strtobool` (const `std::string` &s, bool &b)
- `std::string lower` (const `std::string` &s)
- `std::string upper` (const `std::string` &s)
- `void tokenize` (const `std::string` &str, `std::vector`< `std::string` > &tokens, const `std::string` &delimiters=" ", const `std::string` &start\_quotes="", const `std::string` &end\_quotes="")
- `void tokenize` (const `std::string` &str, `std::list`< `std::string` > &tokens, const `std::string` &delimiters=" ", const `std::string` &start\_quotes="", const `std::string` &end\_quotes="")
- `std::string::size_type get_token` (`std::string` &token, const `std::string` &str, `std::string::size_type` pos, const `std::string` &delimiters=" ", const `std::string` &start\_quotes="", const `std::string` &end\_quotes="")
- `std::string trim` (const `std::string` &str, const char \*sep=NULL)
- `std::string strip` (const `std::string` &str)
- `std::string uri_encode` (const `std::string` &str, bool encode\_slash)
- `std::string uri_unencode` (const `std::string` &str)

- std::string [convert\\_to\\_rdn](#) (const std::string &dn)
- std::string [escape\\_chars](#) (const std::string &str, const std::string &chars, char esc, bool excl, [escape\\_type](#) type=escape\_char)
- std::string [unescape\\_chars](#) (const std::string &str, char esc, [escape\\_type](#) type=escape\_char)
- bool [CreateThreadFunction](#) (void(\*func)(void \*), void \*arg, [SimpleCounter](#) \*count=NULL)
- std::list< [URL](#) > [ReadURLList](#) (const [URL](#) &urllist)
- std::string [GetEnv](#) (const std::string &var)
- std::string [GetEnv](#) (const std::string &var, bool &found)
- bool [SetEnv](#) (const std::string &var, const std::string &value, bool overwrite=true)
- void [UnsetEnv](#) (const std::string &var)
- void [EnvLockWrap](#) (bool all=false)
- void [EnvLockUnwrap](#) (bool all=false)
- void [EnvLockUnwrapComplete](#) (void)
- std::string [StrError](#) (int errnum=errno)
- bool [MatchXMLName](#) (const [XMLNode](#) &node1, const [XMLNode](#) &node2)
- bool [MatchXMLName](#) (const [XMLNode](#) &node, const char \*name)
- bool [MatchXMLName](#) (const [XMLNode](#) &node, const std::string &name)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node1, const [XMLNode](#) &node2)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node, const char \*uri)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node, const std::string &uri)
- bool [createVOMSAC](#) (std::string &codedac, [Credential](#) &issuer\_cred, [Credential](#) &holder\_cred, std::vector< std::string > &fqan, std::vector< std::string > &targets, std::vector< std::string > &attributes, std::string &voname, std::string &uri, int lifetime)
- bool [addVOMSAC](#) (ArcCredential::AC \*\*&aclist, std::string &acorder, std::string &decodedac)
- bool [parseVOMSAC](#) (X509 \*holder, const std::string &ca\_cert\_dir, const std::string &ca\_cert\_file, const std::string &vomkdir, [VOMSTrustList](#) &vomscert\_trust\_dn, std::vector< VOMSACInfo > &output, bool verify=true, bool reportall=false)
- bool [parseVOMSAC](#) (const [Credential](#) &holder\_cred, const std::string &ca\_cert\_dir, const std::string &ca\_cert\_file, const std::string &vomkdir, [VOMSTrustList](#) &vomscert\_trust\_dn, std::vector< VOMSACInfo > &output, bool verify=true, bool reportall=false)
- char \* [VOMSDecode](#) (const char \*data, int size, int \*j)
- std::string [getCredentialProperty](#) (const [Arc::Credential](#) &u, const std::string &property, const std::string &ca\_cert\_dir=std::string(""), const std::string &ca\_cert\_file=std::string(""), const std::string &vomkdir=std::string(""), const std::vector< std::string > &vom\_trust\_list=std::vector< std::string >())
- bool [OpenSSLInit](#) (void)
- void [HandleOpenSSLError](#) (void)
- void [HandleOpenSSLError](#) (int code)
- std::string [string](#) ([StatusKind](#) kind)
- const char \* [ContentFromPayload](#) (const [MessagePayload](#) &payload)
- void [WSAFaultAssign](#) (SOAPEnvelope &message, [WSAFault](#) fid)
- [WSAFault](#) [WSAFaultExtract](#) (SOAPEnvelope &message)
- int [passphrase\\_callback](#) (char \*buf, int size, int rwflag, void \*)
- bool [init\\_xmlsec](#) (void)
- bool [final\\_xmlsec](#) (void)
- std::string [get\\_cert\\_str](#) (const char \*certfile)
- xmlSecKey \* [get\\_key\\_from\\_keystr](#) (const std::string &value)
- xmlSecKey \* [get\\_key\\_from\\_keyfile](#) (const char \*keyfile)
- std::string [get\\_key\\_from\\_certfile](#) (const char \*certfile)
- xmlSecKey \* [get\\_key\\_from\\_certstr](#) (const std::string &value)

- xmlSecKeysMngrPtr [load\\_key\\_from\\_keyfile](#) (xmlSecKeysMngrPtr \*keys\_manager, const char \*keyfile)
- xmlSecKeysMngrPtr [load\\_key\\_from\\_certfile](#) (xmlSecKeysMngrPtr \*keys\_manager, const char \*certfile)
- xmlSecKeysMngrPtr [load\\_key\\_from\\_certstr](#) (xmlSecKeysMngrPtr \*keys\_manager, const std::string &certstr)
- xmlSecKeysMngrPtr [load\\_trusted\\_cert\\_file](#) (xmlSecKeysMngrPtr \*keys\_manager, const char \*cert\_file)
- xmlSecKeysMngrPtr [load\\_trusted\\_cert\\_str](#) (xmlSecKeysMngrPtr \*keys\_manager, const std::string &cert\_str)
- xmlSecKeysMngrPtr [load\\_trusted\\_certs](#) (xmlSecKeysMngrPtr \*keys\_manager, const char \*cafile, const char \*capath)
- [XMLNode get\\_node](#) (XMLNode &parent, const char \*name)

## Variables

- const Glib::TimeVal [ETERNAL](#)
- const Glib::TimeVal [HISTORIC](#)
- const size\_t [thread\\_stacksize](#) = (16 \* 1024 \* 1024)
- [Logger CredentialLogger](#)
- const char \* [plugins\\_table\\_name](#)

### 4.1.1 Detailed Description

[Arc](#) namespace contains all core ARC classes.

### 4.1.2 Typedef Documentation

#### 4.1.2.1 typedef [Plugin](#)\*(\*) [Arc::get\\_plugin\\_instance](#)([PluginArgument](#) \*arg)

Constructor function of ARC loadable component.

This function is called with plugin-specific argument and should produce and return valid instance of plugin. If plugin can't be produced by any reason (for example because passed argument is not applicable) then NULL is returned. No exceptions should be raised.

#### 4.1.2.2 typedef std::multimap<std::string, std::string> [Arc::AttrMap](#)

A typedef of a multimap for storage of message attributes.

This typedef is used as a shorthand for a multimap that uses strings for keys as well as values. It is used within the [MessageAttributes](#) class for internal storage of message attributes, but is not visible externally.

#### 4.1.2.3 typedef [AttrMap::const\\_iterator](#) [Arc::AttrConstIter](#)

A typedef of a const\_iterator for [AttrMap](#).

This typedef is used as a shorthand for a const\_iterator for [AttrMap](#). It is used extensively within the [MessageAttributes](#) class as well as the [AttributesIterator](#) class, but is not visible externally.

#### 4.1.2.4 typedef AttrMap::iterator [Arc::AttrIter](#)

A typedef of an (non-const) iterator for AttrMap.

This typedef is used as a shorthand for a (non-const) iterator for AttrMap. It is used in one method within the [MessageAttributes](#) class, but is not visible externally.

### 4.1.3 Enumeration Type Documentation

#### 4.1.3.1 enum [Arc::TimeFormat](#)

An enumeration that contains the possible textual timeformats.

#### 4.1.3.2 enum [Arc::LogLevel](#)

Logging levels.

Logging levels for tagging and filtering log messages. FATAL level designates very severe error events that will presumably lead the application to abort. ERROR level designates error events that might still allow the application to continue running. WARNING level designates potentially harmful situations. INFO level designates informational messages that highlight the progress of the application at coarse-grained level. VERBOSE level designates fine-grained informational events that will give additional information about the application. DEBUG level designates finer-grained informational events which should only be used for debugging purposes.

#### 4.1.3.3 enum [Arc::LogFormat](#)

Output formats.

Defines prefix for every message. LongFormat - all informatino about message is printed ShortFormat - only message level is printed DebugFormat - message time (microsecond precision) and time difference from previous message are printed. This format is mostly meant for profiling. EmptyFormat - only message is printed

#### 4.1.3.4 enum [Arc::escape\\_type](#)

Type of escaping or encoding to use.

**Enumerator:**

*escape\_octal* place the escape character before the character being escaped

*escape\_hex* hex encoding of the character

#### 4.1.3.5 enum [Arc::StatusKind](#)

Status kinds (types).

This enum defines a set of possible status kinds.

**Enumerator:**

*STATUS\_OK* Default status - undefined error.

**GENERIC\_ERROR** No error.

**PARSING\_ERROR** Error does not fit any class.

**PROTOCOL\_RECOGNIZED\_ERROR** Error detected while parsing request/response.

**UNKNOWN\_SERVICE\_ERROR** [Message](#) does not fit into expected protocol.

**BUSY\_ERROR** There is no destination configured for this message.

**SESSION\_CLOSE** [Message](#) can't be processed now.

#### 4.1.3.6 enum [Arc::WSAFault](#)

WS-Addressing possible faults.

Enumerator:

**WSAFaultUnknown** This is not a fault

**WSAFaultInvalidAddressingHeader** This is not a WS-Addressing fault

### 4.1.4 Function Documentation

#### 4.1.4.1 `std::ostream& Arc::operator<< (std::ostream &, const Period &)`

Prints a Period-object to the given ostream – typically cout.

#### 4.1.4.2 `std::ostream& Arc::operator<< (std::ostream &, const Time &)`

Prints a Time-object to the given ostream – typically cout.

#### 4.1.4.3 `std::string Arc::TimeStamp (const TimeFormat & = Time::GetFormat())`

Returns a time-stamp of the current time in some format.

#### 4.1.4.4 `std::string Arc::TimeStamp (Time, const TimeFormat & = Time::GetFormat())`

Returns a time-stamp of some specified time in some format.

#### 4.1.4.5 `bool Arc::FileCopy (const std::string & source_path, const std::string & destination_path, uid_t uid, gid_t gid)`

Copy file source\_path to file destination\_path. Specified uid and gid are used for accessing filesystem.

#### 4.1.4.6 `bool Arc::FileCopy (const std::string & source_path, const std::string & destination_path)`

Copy file source\_path to file destination\_path.

#### 4.1.4.7 `bool Arc::FileCopy (const std::string & source_path, int destination_handle)`

Copy file source\_path to file handle destination\_handle.

**4.1.4.8** `bool Arc::FileCopy (int source_handle, const std::string & destination_path)`

Copy from file handle *source\_handle* to file *destination\_path*.

**4.1.4.9** `bool Arc::FileCopy (int source_handle, int destination_handle)`

Copy from file handle *source\_handle* to file handle *destination\_handle*.

**4.1.4.10** `bool Arc::FileRead (const std::string & filename, std::list< std::string > & data, uid_t uid = 0, gid_t gid = 0)`

The content is split into lines with the new line character removed, and the lines are returned in the data list. If protected access is required, [FileLock](#) should be used in addition to FileRead.

**4.1.4.11** `bool Arc::FileRead (const std::string & filename, std::string & data, uid_t uid = 0, gid_t gid = 0)`

Simple method to read whole file content from *filename*. Specified *uid* and *gid* are used for accessing filesystem.

**4.1.4.12** `bool Arc::FileCreate (const std::string & filename, const std::string & data, uid_t uid = 0, gid_t gid = 0, mode_t mode = 0)`

An existing file is overwritten with the new data. Permissions of the created file are determined using the current umask. If protected access is required, [FileLock](#) should be used in addition to FileRead. If *uid/gid* are zero then no real switch of *uid/gid* is done.

**4.1.4.13** `bool Arc::FileStat (const std::string & path, struct stat * st, bool follow_symlinks)`

Stat a file and put info into the *st* struct.

**4.1.4.14** `bool Arc::FileStat (const std::string & path, struct stat * st, uid_t uid, gid_t gid, bool follow_symlinks)`

Stat a file using the specified *uid* and *gid* and put info into the *st* struct Specified *uid* and *gid* are used for accessing filesystem.

**4.1.4.15** `bool Arc::FileLink (const std::string & oldpath, const std::string & newpath, bool symbolic)`

Make symbolic or hard link of file.

**4.1.4.16** `bool Arc::FileLink (const std::string & oldpath, const std::string & newpath, uid_t uid, gid_t gid, bool symbolic)`

Make symbolic or hard link of file using the specified *uid* and *gid* Specified *uid* and *gid* are used for accessing filesystem.

**4.1.4.17** `std::string Arc::FileReadLink (const std::string & path)`

Returns path at which symbolic link is pointing.

**4.1.4.18** `std::string Arc::FileReadLink (const std::string & path, uid_t uid, gid_t gid)`

Returns path at which symbolic link is pointing using the specified uid and gid Specified uid and gid are used for accessing filesystem.

**4.1.4.19** `bool Arc::FileDelete (const std::string & path)`

Deletes file at path.

**4.1.4.20** `bool Arc::FileDelete (const std::string & path, uid_t uid, gid_t gid)`

Deletes file at path using the specified uid and gid Specified uid and gid are used for accessing filesystem.

**4.1.4.21** `bool Arc::DirCreate (const std::string & path, mode_t mode, bool with_parents = false)`

Create a new directory.

**4.1.4.22** `bool Arc::DirCreate (const std::string & path, uid_t uid, gid_t gid, mode_t mode, bool with_parents = false)`

Create a new directory using the specified uid and gid Specified uid and gid are used for accessing filesystem.

**4.1.4.23** `bool Arc::DirDelete (const std::string & path, bool recursive = true)`

Delete a directory, and its content if recursive is true. If the directory is not empty and recursive is false DirDelete will fail.

**4.1.4.24** `bool Arc::DirDelete (const std::string & path, bool recursive, uid_t uid, gid_t gid)`

Delete a directory, and its content if recursive is true. If the directory is not empty and recursive is false DirDelete will fail. Specified uid and gid are used for accessing filesystem.

**4.1.4.25** `bool Arc::TmpDirCreate (std::string & path)`

Create a temporary directory under the system defined temp location, and return its path.

Uses mkdtemp if available, and a combination of random parameters if not. This latter method is not as safe as mkdtemp.



#### 4.1.4.26 **bool Arc::TmpFileCreate (std::string & filename, const std::string & data, uid\_t uid = 0, gid\_t gid = 0, mode\_t mode = 0)**

Permissions of the created file are determined using the current umask. If uid/gid are zero then no real switch of uid/gid is done. Input value of filename argument is ignored. On output it contains path to created file. Content of data argument is written into created file.

#### 4.1.4.27 **bool Arc::CanonicalDir (std::string & name, bool leading\_slash = true)**

Removes `../` from 'name'. If `leading_slash=true` `'/'` will be added at the beginning of 'name' if missing. Otherwise it will be removed. The directory separator used here depends on the platform. Returns false if it is not possible to remove all the `../`

#### 4.1.4.28 **void Arc::GUID (std::string & guid)**

Generates a unique identifier using information such as IP address, current time etc.

#### 4.1.4.29 **std::string Arc::UUID (void)**

Generates a unique identifier using the system uuid libraries.

#### 4.1.4.30 **std::ostream& Arc::operator<< (std::ostream & os, LogLevel level)**

Printing of LogLevel values to ostreams.

Output operator so that LogLevel values can be printed in a nicer way.

#### 4.1.4.31 **LogLevel Arc::string\_to\_level (const std::string & str)**

Convert string to a LogLevel.

#### 4.1.4.32 **bool Arc::istring\_to\_level (const std::string & lStr, LogLevel & ll)**

Case-insensitive parsing of a string to a LogLevel with error response.

The method will try to parse (case-insensitive) the argument string to a corresponding LogLevel. If the method succeeds, true will be returned and the argument `ll` will be set to the parsed LogLevel. If the parsing fails `false` will be returned. The parsing succeeds if `lStr` match (case-insensitively) one of the names of the LogLevel members.

#### Parameters:

`lStr` a string which should be parsed to a [Arc::LogLevel](#).

`ll` a [Arc::LogLevel](#) reference which will be set to the matching [Arc::LogLevel](#) upon successful parsing.

#### Returns:

`true` in case of successful parsing, otherwise `false`.

#### See also:

[LogLevel](#)

**4.1.4.33** `bool Arc::string_to_level (const std::string & str, LogLevel & ll)`

Same as `istring_to_level` except it is case-sensitive.

**4.1.4.34** `std::string Arc::level_to_string (const LogLevel & level)`

Convert `LogLevel` to a string.

**4.1.4.35** `LogLevel Arc::old_level_to_level (unsigned int old_level)`

Convert an old-style log level (int from 0 to 5) to a `LogLevel`.

**4.1.4.36** `template<typename T> T Arc::stringto (const std::string & s)`

This method converts a string to any type.

**4.1.4.37** `template<typename T> bool Arc::stringto (const std::string & s, T & t)`

This method converts a string to any type but lets calling function process errors.

**4.1.4.38** `bool Arc::strtoint (const std::string & s, signed int & t, int base = 10)`

Convert string to integer with specified base. Returns false if any argument is wrong.

**4.1.4.39** `bool Arc::strtoint (const std::string & s, unsigned int & t, int base = 10)`

Convert string to unsigned integer with specified base. Returns false if any argument is wrong.

**4.1.4.40** `bool Arc::strtoint (const std::string & s, signed long & t, int base = 10)`

Convert string to long integer with specified base. Returns false if any argument is wrong.

**4.1.4.41** `bool Arc::strtoint (const std::string & s, unsigned long & t, int base = 10)`

Convert string to unsigned long integer with specified base. Returns false if any argument is wrong.

**4.1.4.42** `bool Arc::strtoint (const std::string & s, signed long long & t, int base = 10)`

Convert string to long long integer with specified base. Returns false if any argument is wrong.

**4.1.4.43** `bool Arc::strtoint (const std::string & s, unsigned long long & t, int base = 10)`

Convert string to unsigned long long integer with specified base. Returns false if any argument is wrong.

**4.1.4.44** `template<typename T> std::string Arc::tostring (T t, int width = 0, int precision = 0)`

This method converts any type to a string of the width given.

**4.1.4.45** `std::string Arc::inttostr (signed long long t, int base = 10, int width = 0)`

Convert long long integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.46** `std::string Arc::inttostr (unsigned long long t, int base = 10, int width = 0)`

Convert unsigned long long integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.47** `std::string Arc::inttostr (signed int t, int base = 10, int width = 0) [inline]`

Convert integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.48** `std::string Arc::inttostr (unsigned int t, int base = 10, int width = 0) [inline]`

Convert unsigned integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.49** `std::string Arc::inttostr (signed long t, int base = 10, int width = 0) [inline]`

Convert long integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.50** `std::string Arc::inttostr (unsigned long t, int base = 10, int width = 0) [inline]`

Convert unsigned long integer to textual representation for specied base. Result is padded with zeroes on left till width.

**4.1.4.51** `std::string Arc::booltostr (bool b) [inline]`

Convert bool to textual representation, i.e. "true" or "false".

**4.1.4.52** `bool Arc::strtobool (const std::string & s) [inline]`

Convert string to bool. Simply checks string if equal to "true" or "1".

**4.1.4.53** `bool Arc::strtobool (const std::string & s, bool & b) [inline]`

Convert string to bool Checks whether string is equal to one of "true", "false", "1" or "0", and if not returns false. If equal, true is returned and the bool reference is set to true, if string equals "true" or "1", otherwise it is set to false.

**4.1.4.54** `std::string Arc::lower (const std::string & s)`

This method converts to lower case of the string.

**4.1.4.55** `std::string Arc::upper (const std::string & s)`

This method converts to upper case of the string.

**4.1.4.56** `void Arc::tokenize (const std::string & str, std::vector< std::string > & tokens, const std::string & delimiters = " ", const std::string & start_quotes = "", const std::string & end_quotes = "")`

This method tokenizes string.

**4.1.4.57** `void Arc::tokenize (const std::string & str, std::list< std::string > & tokens, const std::string & delimiters = " ", const std::string & start_quotes = "", const std::string & end_quotes = "")`

This method tokenizes string.

**4.1.4.58** `std::string::size_type Arc::get_token (std::string & token, const std::string & str, std::string::size_type pos, const std::string & delimiters = " ", const std::string & start_quotes = "", const std::string & end_quotes = "")`

This method extracts first token in string str starting at pos.

**4.1.4.59** `std::string Arc::trim (const std::string & str, const char * sep = NULL)`

This method removes given separators from the beginning and the end of the string.

**4.1.4.60** `std::string Arc::strip (const std::string & str)`

This method removes blank lines from the passed text string. Lines with only space on them are considered blank.

**4.1.4.61** `std::string Arc::uri_encode (const std::string & str, bool encode_slash)`

be encoded

Characters which are not unreserved according to RFC 3986 are encoded. If encode\_slash is true forward slashes will also be encoded. It is useful to set encode\_slash to false when encoding full paths.

**4.1.4.62** `std::string Arc::uri_unencode (const std::string & str)`

This method unencodes the -encoded URI str.

**4.1.4.63** `std::string Arc::convert_to_rdn (const std::string & dn)`

Convert dn to rdn: /O=Grid/OU=Knowarc/CN=abc —> CN=abc,OU=Knowarc,O=Grid.

**4.1.4.64** `std::string Arc::escape_chars (const std::string & str, const std::string & chars, char esc, bool excl, escape_type type = escape_char)`

Escape or encode the given chars in str using the escape character esc. If excl is true then escape all characters not in chars

**4.1.4.65** `std::string Arc::unescape_chars (const std::string & str, char esc, escape_type type = escape_char)`

Unescape or unencode characters in str escaped with esc.

**4.1.4.66** `bool Arc::CreateThreadFunction (void(*) (void *) func, void * arg, SimpleCounter * count = NULL)`

Helper function to create simple thread.

It takes care of all peculiarities of Glib::Thread API. As result it runs function 'func' with argument 'arg' in a separate thread. If count parameter not NULL then corresponding object will be incremented before function returns and then decremented then thread finished. Returns true on success.

**4.1.4.67** `std::list<URL> Arc::ReadURLList (const URL & urllist)`

Reads a list of URLs from a file.

**4.1.4.68** `std::string Arc::GetEnv (const std::string & var)`

Portable function for getting environment variables.

**4.1.4.69** `std::string Arc::GetEnv (const std::string & var, bool & found)`

Portable function for getting environment variables.

**4.1.4.70** `bool Arc::SetEnv (const std::string & var, const std::string & value, bool overwrite = true)`

Portable function for setting environment variables.

**4.1.4.71** `void Arc::UnsetEnv (const std::string & var)`

Portable function for unsetting environment variables.

**4.1.4.72 void Arc::EnvLockWrap (bool *all* = false)**

Start code which is using setenv/getenv. Use all=true for setenv and all=false for getenv. Must always have corresponding EnvLockUnwrap.

**4.1.4.73 void Arc::EnvLockUnwrap (bool *all* = false)**

End code which is using setenv/getenv. Value of all must be same as in corresponding EnvLockWrap.

**4.1.4.74 void Arc::EnvLockUnwrapComplete (void)**

Use after fork() to reset all internal variables and release all locks.

**4.1.4.75 std::string Arc::StrError (int *errnum* = errno)**

Portable function for obtaining description of last system error.

**4.1.4.76 bool Arc::MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*)**

Returns true if underlying XML elements have same names

**4.1.4.77 bool Arc::MatchXMLName (const XMLNode & *node*, const char \* *name*)**

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**4.1.4.78 bool Arc::MatchXMLName (const XMLNode & *node*, const std::string & *name*)**

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**4.1.4.79 bool Arc::MatchXMLNamespace (const XMLNode & *node1*, const XMLNode & *node2*)**

Returns true if underlying XML elements belong to same namespaces

**4.1.4.80 bool Arc::MatchXMLNamespace (const XMLNode & *node*, const char \* *uri*)**

Returns true if 'namespace' matches 'node's namespace.

**4.1.4.81 bool Arc::MatchXMLNamespace (const XMLNode & *node*, const std::string & *uri*)**

Returns true if 'namespace' matches 'node's namespace.

**4.1.4.82 bool Arc::createVOMSAC (std::string & *codedac*, Credential & *issuer\_cred*, Credential & *holder\_cred*, std::vector< std::string > & *fqn*, std::vector< std::string > & *targets*, std::vector< std::string > & *attributes*, std::string & *voname*, std::string & *uri*, int *lifetime*)**

Create AC(Attribute Certificate) with voms specific format.

**Parameters:**

*codedac* The coded AC as output of this method

*issuer\_cred* The issuer credential which is used to sign the AC

*holder\_cred* The holder credential, the holder certificate is the one which carries AC The rest arguments are the same as the above method

#### 4.1.4.83 **bool Arc::addVOMSAC (ArcCredential::AC \*\*& aclist, std::string & acorder, std::string & decodedac)**

Add decoded AC string into a list of AC objects

**Parameters:**

*aclist* The list of AC objects (output)

*acorder* The order of AC objects (output)

*decodedac* The AC string that is decoded from the string returned from voms server (input)

#### 4.1.4.84 **bool Arc::parseVOMSAC (X509 \* holder, const std::string & ca\_cert\_dir, const std::string & ca\_cert\_file, const std::string & vomkdir, VOMSTrustList & vomscert\_trust\_dn, std::vector< VOMSACInfo > & output, bool verify = true, bool reportall = false)**

Parse the certificate, and output the attributes.

**Parameters:**

*holder* The proxy certificate which includes the voms specific formatted AC.

*ca\_cert\_dir* The trusted certificates which are used to verify the certificate which is used to sign the AC

*ca\_cert\_file* The same as ca\_cert\_dir except it is a file instead of a directory. Only one of them need to be set

*vomkdir* The directory which include \*.lsc file for each vo. For instance, a vo called "knowarc.eu" should have file vomkdir/knowarc/voms.knowarc.eu.lsc which contains on the first line the DN of the VOMS server, and on the second line the corresponding CA DN: /O=Grid/O=NorduGrid/OU=KnowARC/CN=voms.knowarc.eu /O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority See more in : <https://twiki.cern.ch/twiki/bin/view/LCG/VomsFAQforServiceManagers>

*output* The parsed attributes (Role and Generic Attribute) . Each attribute is stored in element of a vector as a string. It is up to the consumer to understand the meaning of the attribute. There are two types of attributes stored in VOMS AC: AC\_IETFATTR, AC\_FULL\_ATTRIBUTES. The AC\_IETFATTR will be like /Role=Employee/Group=Tester/Capability=NULL The AC\_FULL\_ATTRIBUTES will be like knowarc:Degree=PhD (qualifier::name=value) In order to make the output attribute values be identical, the voms server information is added as prefix of the original attributes in AC. for AC\_FULL\_ATTRIBUTES, the voname + hostname is added: /voname=knowarc.eu/hostname=arthur.hep.lu.se:15001//knowarc.eu/coredev:attribute1=1 for AC\_IETFATTR, the 'VO' (voname) is added: /VO=knowarc.eu/Group=coredev/Role=NULL/Capability=NULL /VO=knowarc.eu/Group=testers/Role=NULL/Capability=NULL

some other redundant attributes is provided: voname=knowarc.eu/hostname=arthur.hep.lu.se:15001

**Parameters:**

**verify** true: Verify the voms certificate is trusted based on the `ca_cert_dir/ca_cert_file` which specifies the CA certificates, and the `vomscert_trust_dn` which specifies the trusted DN chain from voms server certificate to CA certificate. false: Not verify, which means the issuer of AC (voms server certificate is supposed to be trusted by default). In this case the parameters '`ca_cert_dir`', '`ca_cert_file`' and '`vomscert_trust_dn`' will not effect, and may be left empty. This case is specifically used by '`arcproxy -info`' to list all of the attributes in AC, and not to need to verify if the AC's issuer is trusted.

**reportall** If set to true fills output with all attributes including those which failed passing test procedures. Validity of attributes can be checked through status members of output items. Combination of `verify=true` and `reportall=true` provides most information.

**4.1.4.85** `bool Arc::parseVOMSAC (const Credential & holder_cred, const std::string & ca_cert_dir, const std::string & ca_cert_file, const std::string & vomkdir, VOMSTrustList & vomscert_trust_dn, std::vector< VOMSACInfo > & output, bool verify = true, bool reportall = false)`

Parse the certificate. Similar to above one, but collects information From all certificates in a chain.

**4.1.4.86** `char* Arc::VOMSDecode (const char * data, int size, int * j)`

Decode the data which is encoded by voms server. Since voms code uses some specific coding method (not base64 encoding), we simply copy the method from voms code to here

**4.1.4.87** `std::string Arc::getCredentialProperty (const Arc::Credential & u, const std::string & property, const std::string & ca_cert_dir = std::string(""), const std::string & ca_cert_file = std::string(""), const std::string & vomkdir = std::string(""), const std::vector< std::string > & voms_trust_list = std::vector< std::string >())`

Extract the needed field from the certificate.

**Parameters:**

**u** The proxy certificate which includes the voms specific formatted AC.

**property** The property that caller would get, including: `dn`, `voms:vo`, `voms:role`, `voms:group`

**ca\_cert\_dir**

**ca\_cert\_file**

**vomkdir**

**voms\_trust\_list** the dn chain that is trusted when parsing voms AC

**4.1.4.88** `bool Arc::OpenSSLInit (void)`

This function initializes OpenSSL library.

It may be called multiple times and makes sure everything is done properly and OpenSSL may be used in multi-threaded environment. Because this function makes use of [ArcLocation](#) it is advisable to call it after [ArcLocation::Init\(\)](#).



**4.1.4.89 void Arc::HandleOpenSSLError (void)**

Prints chain of accumulaed OpenSSL errors if any available.

**4.1.4.90 void Arc::HandleOpenSSLError (int *code*)**

Prints chain of accumulaed OpenSSL errors if any available.

**4.1.4.91 std::string Arc::string (StatusKind *kind*)**

Conversion to string.

Conversion from StatusKind to string.

**Parameters:**

*kind* The StatusKind to convert.

**4.1.4.92 const char\* Arc::ContentFromPayload (const MessagePayload & *payload*)**

Returns pointer to main memory chunk of [Message](#) payload.

If no buffer is present or if payload is not of [PayloadRawInterface](#) type NULL is returned.

**4.1.4.93 void Arc::WSAFaultAssign (SOAPEnvelope & *mesage*, WSAFault *fid*)**

Makes WS-Addressing fault.

It fills SOAP Fault message with WS-Addressing fault related information.

**4.1.4.94 [WSAFault](#) Arc::WSAFaultExtract (SOAPEnvelope & *message*)**

Gets WS-addressing fault.

Analyzes SOAP Fault message and returns WS-Addressing fault it represents.

**4.1.4.95 int Arc::passphrase\_callback (char \* *buf*, int *size*, int *rwflag*, void \*)**

callback method for inputing passphrase of key file

**4.1.4.96 bool Arc::init\_xmlsec (void)**

Initialize the xml security library, it should be called before the xml security functionality is used.

**4.1.4.97 bool Arc::final\_xmlsec (void)**

Finalize the xml security library

**4.1.4.98** `std::string Arc::get_cert_str (const char * certfile)`

Get certificate in string format from certificate file

**4.1.4.99** `xmlSecKey* Arc::get_key_from_keyst (const std::string & value)`

Get key in xmlSecKey structure from key in string format

**4.1.4.100** `xmlSecKey* Arc::get_key_from_keyfile (const char * keyfile)`

Get key in xmlSecKey structure from key file

**4.1.4.101** `std::string Arc::get_key_from_certfile (const char * certfile)`

Get public key in string format from certificate file

**4.1.4.102** `xmlSecKey* Arc::get_key_from_certstr (const std::string & value)`

Get public key in xmlSecKey structure from certificate string (the string under "—BEGIN CERTIFICATE—" and "—END CERTIFICATE—")

**4.1.4.103** `xmlSecKeysMngrPtr Arc::load_key_from_keyfile (xmlSecKeysMngrPtr * keys_manager, const char * keyfile)`

Load private or public key from a key file into key manager

**4.1.4.104** `xmlSecKeysMngrPtr Arc::load_key_from_certfile (xmlSecKeysMngrPtr * keys_manager, const char * certfile)`

Load public key from a certificate file into key manager

**4.1.4.105** `xmlSecKeysMngrPtr Arc::load_key_from_certstr (xmlSecKeysMngrPtr * keys_manager, const std::string & certstr)`

Load public key from a certificate string into key manager

**4.1.4.106** `xmlSecKeysMngrPtr Arc::load_trusted_cert_file (xmlSecKeysMngrPtr * keys_manager, const char * cert_file)`

Load trusted certificate from certificate file into key manager

**4.1.4.107** `xmlSecKeysMngrPtr Arc::load_trusted_cert_str (xmlSecKeysMngrPtr * keys_manager, const std::string & cert_str)`

Load trusted certificate from certificate string into key manager

**4.1.4.108** `xmlSecKeysMngrPtr Arc::load_trusted_certs (xmlSecKeysMngrPtr * keys_manager,  
const char * cafile, const char * capath)`

Load trusted certificates from a file or directory into key manager

**4.1.4.109** `XMLNode Arc::get_node (XMLNode & parent, const char * name)`

Generate a new child `XMLNode` with specified name

**4.1.5 Variable Documentation****4.1.5.1** `const Glib::TimeVal Arc::ETERNAL`

A time very far in the future.

**4.1.5.2** `const Glib::TimeVal Arc::HISTORIC`

A time very far in the past.

**4.1.5.3** `const size_t Arc::thread_stacksize = (16 * 1024 * 1024)`

Defines size of stack assigned to every new thread.

So far it takes care of automatic initialization of threading environment and creation of simple detached threads. Always use it instead of `glibmm/thread.h` and keep among first includes. It safe to use it multiple times and to include it both from source files and other include files.

**4.1.5.4** `Logger Arc::CredentialLogger`

`Logger` to be used by all modules of credentials library

**4.1.5.5** `const char* Arc::plugins_table_name`

Name of symbol referring to table of plugins.

This C null terminated string specifies name of symbol which shared library should export to give an access to an array of `PluginDescriptor` elements. The array is terminated by element with all components set to `NULL`.

## 4.2 ArcCredential Namespace Reference

### Data Structures

- struct **cert\_verify\_context**
- struct **PROXYPOLICY\_st**
- struct **PROXYCERTINFO\_st**
- struct **ACDIGEST**
- struct **ACIS**
- struct **ACFORM**
- struct **ACACI**
- struct **ACHOLDER**
- struct **ACVAL**
- struct **ACIETFATTR**
- struct **ACTARGET**
- struct **ACTARGETS**
- struct **ACATTR**
- struct **ACINFO**
- struct **ACC**
- struct **ACSEQ**
- struct **ACCERTS**
- struct **ACATTRIBUTE**
- struct **ACATTHOLDER**
- struct **ACFULLATTRIBUTES**

### Enumerations

- enum **certType** {  
CERT\_TYPE\_EEC, CERT\_TYPE\_CA, CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY,  
CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY,  
CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY, CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY,  
CERT\_TYPE\_GSI\_2\_PROXY, CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY,  
CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY, CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY,  
CERT\_TYPE\_RFC\_LIMITED\_PROXY, CERT\_TYPE\_RFC\_RESTRICTED\_PROXY,  
CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY }

#### 4.2.1 Detailed Description

The code is derived from globus gsi, voms, and openssl-0.9.8e. The existing code for maintaining proxy certificates in OpenSSL only covers standard proxies and does not cover old Globus proxies, so here the Globus code is introduced.

#### 4.2.2 Enumeration Type Documentation

##### 4.2.2.1 enum **ArcCredential::certType**

Enumerator:

**CERT\_TYPE\_EEC** A end entity certificate

***CERT\_TYPE\_CA*** A CA certificate

***CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY*** A X.509 Proxy Certificate Profile (pre-RFC) compliant impersonation proxy

***CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY*** A X.509 Proxy Certificate Profile (pre-RFC) compliant independent proxy

***CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY*** A X.509 Proxy Certificate Profile (pre-RFC) compliant limited proxy

***CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY*** A X.509 Proxy Certificate Profile (pre-RFC) compliant restricted proxy

***CERT\_TYPE\_GSI\_2\_PROXY*** A legacy Globus impersonation proxy

***CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY*** A legacy Globus limited impersonation proxy

***CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY*** A X.509 Proxy Certificate Profile RFC compliant impersonation proxy; RFC inheritAll proxy

***CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY*** A X.509 Proxy Certificate Profile RFC compliant independent proxy; RFC independent proxy

***CERT\_TYPE\_RFC\_LIMITED\_PROXY*** A X.509 Proxy Certificate Profile RFC compliant limited proxy

***CERT\_TYPE\_RFC\_RESTRICTED\_PROXY*** A X.509 Proxy Certificate Profile RFC compliant restricted proxy

***CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY*** RFC anyLanguage proxy

## 4.3 DataStaging Namespace Reference

[DataStaging](#) contains all components for data transfer scheduling and execution.

### Data Structures

- class [DataDelivery](#)  
*[DataDelivery](#) transfers data between specified physical locations.*
- class [DataDeliveryComm](#)  
*This class provides an abstract interface for the Delivery layer.*
- class [DataDeliveryCommHandler](#)  
*Singleton class handling all active [DataDeliveryComm](#) objects.*
- class [DataDeliveryLocalComm](#)  
*This class starts, monitors and controls a local Delivery process.*
- class [DataDeliveryRemoteComm](#)  
*This class contacts a remote service to make a Delivery request.*
- class [TransferParameters](#)
- class [DTRCacheParameters](#)  
*The configured cache directories.*
- class [DTRCallback](#)  
*The base class from which all callback-enabled classes should be derived.*
- class [DTR](#)  
*Data Transfer Request.*
- class [DTRLList](#)  
*Global list of all active DTRs in the system.*
- class [DTRStatus](#)  
*Class representing the status of a [DTR](#).*
- class [DTRErrorStatus](#)  
*A class to represent error states reported by various components.*
- class [Generator](#)  
*Simple [Generator](#) implementation.*
- class [Processor](#)  
*The [Processor](#) performs pre- and post-transfer operations.*
- class [Scheduler](#)  
*The [Scheduler](#) is the control centre of the data staging framework.*

- class [TransferSharesConf](#)  
*TransferSharesConf describes the configuration of TransferShares.*
- class [TransferShares](#)  
*TransferShares is used to implement fair-sharing and priorities.*

## Typedefs

- typedef [Arc::ThreadedPointer](#)< [DTR](#) > [DTR\\_ptr](#)
- typedef [Arc::ThreadedPointer](#)< [Arc::Logger](#) > [DTRLogger](#)

## Enumerations

- enum [StagingProcesses](#)
- enum [ProcessState](#)
- enum [CacheState](#) {  
    [CACHEABLE](#),      [NON\\_CACHEABLE](#),      [CACHE\\_ALREADY\\_PRESENT](#),      [CACHE\\_-](#)  
    [DOWNLOADED](#),  
    [CACHE\\_LOCKED](#), [CACHE\\_SKIP](#), [CACHE\\_NOT\\_USED](#) }

### 4.3.1 Detailed Description

[DataStaging](#) contains all components for data transfer scheduling and execution.

### 4.3.2 Typedef Documentation

#### 4.3.2.1 typedef [Arc::ThreadedPointer](#)<[DTR](#)> [DataStaging::DTR\\_ptr](#)

Provides automatic memory management of DTRs and thread-safe destruction.

#### 4.3.2.2 typedef [Arc::ThreadedPointer](#)<[Arc::Logger](#)> [DataStaging::DTRLogger](#)

The DTR's Logger object can be used outside the [DTR](#) object with [DTRLogger](#).

### 4.3.3 Enumeration Type Documentation

#### 4.3.3.1 enum [DataStaging::StagingProcesses](#)

Components of the data staging framework.

#### 4.3.3.2 enum [DataStaging::ProcessState](#)

Internal state of staging processes.

#### 4.3.3.3 enum [DataStaging::CacheState](#)

Represents possible cache states of this [DTR](#).

**Enumerator:**

***CACHEABLE*** Source should be cached.

***NON\_CACHEABLE*** Source should not be cached.

***CACHE\_ALREADY\_PRESENT*** Source is available in cache from before.

***CACHE\_DOWNLOADED*** Source has just been downloaded and put in cache.

***CACHE\_LOCKED*** Cache file is locked.

***CACHE\_SKIP*** Source is cacheable but due to some problem should not be cached.

***CACHE\_NOT\_USED*** Cache was started but was not used.



## Chapter 5

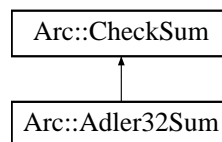
# Hosting Environment (Daemon) Data Structure Documentation

### 5.1 Arc::Adler32Sum Class Reference

Implementation of Adler32 checksum.

```
#include <CheckSum.h>
```

Inheritance diagram for Arc::Adler32Sum::



#### Public Member Functions

- virtual void [start](#) (void)
- virtual void [add](#) (void \*buf, unsigned long long int len)
- virtual void [end](#) (void)
- virtual void [result](#) (unsigned char \*&res, unsigned int &len) const
- virtual int [print](#) (char \*buf, int len) const
- virtual void [scan](#) (const char \*)
- virtual [operator bool](#) (void) const
- virtual bool [operator!](#) (void) const

#### 5.1.1 Detailed Description

Implementation of Adler32 checksum.

This class is a specialized class of the [Checksum](#) class. It provides an implementation of the Adler-32 checksum algorithm.

## 5.1.2 Member Function Documentation

### 5.1.2.1 `virtual void Arc::Adler32Sum::add (void * buf, unsigned long long int len)` [`inline`, `virtual`]

Add data to be checksummed.

This method calculates the checksum of the passed data chunk, taking into account the previous state of this object.

#### Parameters:

*buf* pointer to data chunk to be checksummed.

*len* size of the data chunk.

Implements [Arc::Checksum](#).

### 5.1.2.2 `virtual void Arc::Adler32Sum::end (void)` [`inline`, `virtual`]

Finalize the checksumming.

This method finalizes the checksum algorithm, that is calculating the final checksum result.

Implements [Arc::Checksum](#).

### 5.1.2.3 `virtual Arc::Adler32Sum::operator bool (void) const` [`inline`, `virtual`]

Indicates whether the checksum has been calculated.

Reimplemented from [Arc::Checksum](#).

### 5.1.2.4 `virtual bool Arc::Adler32Sum::operator! (void) const` [`inline`, `virtual`]

Indicates whether the checksum has not been calculated.

Reimplemented from [Arc::Checksum](#).

### 5.1.2.5 `virtual int Arc::Adler32Sum::print (char * buf, int len) const` [`inline`, `virtual`]

Retrieve result of checksum into a string.

The passed string *buf* is filled with result of checksum algorithm in base 16. At most *len* characters is filled into buffer *buf*. The hexadecimal value is prepended with "<algorithm>:", where <algorithm> is one of "cksum", "md5" or "adler32" respectively corresponding to the result from the [CRC32Sum](#), [MD5Sum](#) and [Adler32](#) classes.

#### Parameters:

*buf* pointer to buffer which should be filled with checksum result.

*len* max number of character filled into buffer.

Reimplemented from [Arc::Checksum](#).

**5.1.2.6 virtual void Arc::Adler32Sum::result (unsigned char \*& *res*, unsigned int & *len*) const**  
[inline, virtual]

Retrieve result of checksum as binary blob.

Implements [Arc::Checksum](#).

**5.1.2.7 virtual void Arc::Adler32Sum::scan (const char \*)** [inline, virtual]

Set internal checksum state.

This method sets the internal state to that of the passed textual representation. The format passed to this method must be the same as retrieved from the [Checksum::print](#) method.

**Parameters:**

*buf* string containing textual representation of checksum

**See also:**

[Checksum::print](#)

Implements [Arc::Checksum](#).

**5.1.2.8 virtual void Arc::Adler32Sum::start (void)** [inline, virtual]

Initiate the checksum algorithm.

This method must be called before starting a new checksum calculation.

Implements [Arc::Checksum](#).

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

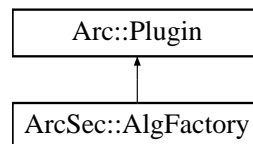
- CheckSum.h

## 5.2 ArcSec::AlgFactory Class Reference

Interface for algorithm factory class.

```
#include <AlgFactory.h>
```

Inheritance diagram for ArcSec::AlgFactory::



### Public Member Functions

- virtual [CombiningAlg](#) \* [createAlg](#) (const std::string &type)=0

#### 5.2.1 Detailed Description

Interface for algorithm factory class.

[AlgFactory](#) is in charge of creating [CombiningAlg](#) according to the algorithm type given as argument of method [createAlg](#). This class can be inherited for implementing a factory class which can create some specific combining algorithm objects.

#### 5.2.2 Member Function Documentation

**5.2.2.1** virtual [CombiningAlg](#)\* [ArcSec::AlgFactory::createAlg](#) (const std::string & *type*) [pure virtual]

creat algorithm object based on the type algorithm type

##### Parameters:

*type* The type of combining algorithm

##### Returns:

The object of [CombiningAlg](#)

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

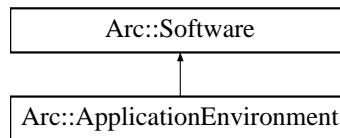
- AlgFactory.h

## 5.3 Arc::ApplicationEnvironment Class Reference

[ApplicationEnvironment](#).

```
#include <ExecutionTarget.h>
```

Inheritance diagram for Arc::ApplicationEnvironment::



### 5.3.1 Detailed Description

[ApplicationEnvironment](#).

The ApplicationEnvironment is closely related to the definition given in [GLUE2](#). By extending the [Software](#) class the two [GLUE2](#) attributes AppName and AppVersion are mapped to two private members. However these can be obtained through the inherited member methods getName and getVersion.

[GLUE2](#) description: A description of installed application software or software environment characteristics available within one or more Execution Environments.

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

- ExecutionTarget.h

## 5.4 Arc::ArcLocation Class Reference

Determines ARC installation location.

```
#include <ArcLocation.h>
```

### Static Public Member Functions

- static void [Init](#) (std::string path)
- static const std::string & [Get](#) ()
- static std::list< std::string > [GetPlugins](#) ()

#### 5.4.1 Detailed Description

Determines ARC installation location.

#### 5.4.2 Member Function Documentation

##### 5.4.2.1 static const std::string& Arc::ArcLocation::Get () [static]

Returns ARC installation location.

##### 5.4.2.2 static std::list<std::string> Arc::ArcLocation::GetPlugins () [static]

Returns ARC plugins directory location.

Main source is value of variable ARC\_PLUGIN\_PATH, otherwise path is derived from installation location.

##### 5.4.2.3 static void Arc::ArcLocation::Init (std::string *path*) [static]

Initializes location information.

Main source is value of variable ARC\_LOCATION, otherwise path to executable provided in is used. If nothing works then warning message is sent to logger and initial installation prefix is used.

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

- ArcLocation.h

## 5.5 Arc::ArcVersion Class Reference

Determines ARC HED libraries version.

```
#include <ArcVersion.h>
```

### 5.5.1 Detailed Description

Determines ARC HED libraries version.

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

- ArcVersion.h

## 5.6 ArcSec::Attr Struct Reference

[Attr](#) contains a tuple of attribute type and value.

```
#include <Request.h>
```

### 5.6.1 Detailed Description

[Attr](#) contains a tuple of attribute type and value.

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

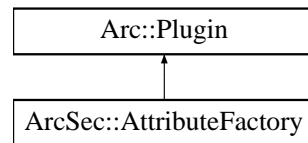
- Request.h



## 5.7 ArcSec::AttributeFactory Class Reference

```
#include <AttributeFactory.h>
```

Inheritance diagram for ArcSec::AttributeFactory::



### 5.7.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

## 5.8 Arc::AttributeIterator Class Reference

A const iterator class for accessing multiple values of an attribute.

```
#include <MessageAttributes.h>
```

### Public Member Functions

- [AttributeIterator](#) ()
- const std::string & [operator \\*](#) () const
- const std::string \* [operator →](#) () const
- const std::string & [key](#) (void) const
- const [AttributeIterator](#) & [operator++](#) ()
- [AttributeIterator](#) [operator++](#) (int)
- bool [hasMore](#) () const

### Protected Member Functions

- [AttributeIterator](#) ([AttrConstIter](#) begin, [AttrConstIter](#) end)

### Protected Attributes

- [AttrConstIter](#) [current\\_](#)
- [AttrConstIter](#) [end\\_](#)

### Friends

- class [MessageAttributes](#)

### 5.8.1 Detailed Description

A const iterator class for accessing multiple values of an attribute.

This is an iterator class that is used when accessing multiple values of an attribute. The `getAll()` method of the [MessageAttributes](#) class returns an [AttributeIterator](#) object that can be used to access the values of the attribute.

Typical usage is:

```
MessageAttributes attributes;
...
for (AttributeIterator iterator=attributes.getAll("Foo:Bar");
     iterator.hasMore(); ++iterator)
    std::cout << *iterator << std::endl;
```

### 5.8.2 Constructor & Destructor Documentation

#### 5.8.2.1 Arc::AttributeIterator::AttributeIterator ()

Default constructor.

The default constructor. Does nothing since all attributes are instances of well-behaving STL classes.

### 5.8.2.2 Arc::AttributeIterator::AttributeIterator ([AttrConstIter](#) *begin*, [AttrConstIter](#) *end*) [protected]

Protected constructor used by the [MessageAttributes](#) class.

This constructor is used to create an iterator for iteration over all values of an attribute. It is not supposed to be visible externally, but is only used from within the `getAll()` method of [MessageAttributes](#) class.

#### Parameters:

*begin* A `const_iterator` pointing to the first matching key-value pair in the internal multimap of the [MessageAttributes](#) class.

*end* A `const_iterator` pointing to the first key-value pair in the internal multimap of the [MessageAttributes](#) class where the key is larger than the key searched for.

## 5.8.3 Member Function Documentation

### 5.8.3.1 bool Arc::AttributeIterator::hasMore () const

Predicate method for iteration termination.

This method determines whether there are more values for the iterator to refer to.

#### Returns:

Returns true if there are more values, otherwise false.

### 5.8.3.2 const std::string& Arc::AttributeIterator::key (void) const

The key of attribute.

This method returns reference to key of attribute to which iterator refers.

### 5.8.3.3 const std::string& Arc::AttributeIterator::operator \* () const

The dereference operator.

This operator is used to access the current value referred to by the iterator.

#### Returns:

A (constant reference to a) string representation of the current value.

### 5.8.3.4 [AttributeIterator](#) Arc::AttributeIterator::operator++ (int)

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

#### Returns:

An iterator referring to the value referred to by this iterator before the advance.

### 5.8.3.5 `const AttributeIterator& Arc::AttributeIterator::operator++ ()`

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

#### Returns:

A const reference to this iterator.

### 5.8.3.6 `const std::string* Arc::AttributeIterator::operator → () const`

The arrow operator.

Used to call methods for value objects (strings) conveniently.

## 5.8.4 Friends And Related Function Documentation

### 5.8.4.1 `friend class MessageAttributes [friend]`

The [MessageAttributes](#) class is a friend.

The constructor that creates an [AttributeIterator](#) that is connected to the internal multimap of the [MessageAttributes](#) class should not be exposed to the outside, but it still needs to be accessible from the `getAll()` method of the [MessageAttributes](#) class. Therefore, that class is a friend.

## 5.8.5 Field Documentation

### 5.8.5.1 `AttrConstIter Arc::AttributeIterator::current_ [protected]`

A `const_iterator` pointing to the current key-value pair.

This iterator is the internal representation of the current value. It points to the corresponding key-value pair in the internal multimap of the [MessageAttributes](#) class.

### 5.8.5.2 `AttrConstIter Arc::AttributeIterator::end_ [protected]`

A `const_iterator` pointing beyond the last key-value pair.

A `const_iterator` pointing to the first key-value pair in the internal multimap of the [MessageAttributes](#) class where the key is larger than the key searched for.

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

- `MessageAttributes.h`

## 5.9 ArcSec::AttributeProxy Class Reference

Interface for creating the [AttributeValue](#) object, it will be used by [AttributeFactory](#).

```
#include <AttributeProxy.h>
```

### Public Member Functions

- virtual [AttributeValue](#) \* [getAttribute](#) (const [Arc::XMLNode](#) &node)=0

#### 5.9.1 Detailed Description

Interface for creating the [AttributeValue](#) object, it will be used by [AttributeFactory](#).

The [AttributeProxy](#) object will be insert into AttributeFactoty; and the [getAttribute\(node\)](#) method will be called inside AttributeFacroty.createvalue(node), in order to create a specific [AttributeValue](#)

#### 5.9.2 Member Function Documentation

**5.9.2.1** virtual [AttributeValue](#)\* [ArcSec::AttributeProxy::getAttribute](#) (const [Arc::XMLNode](#) &  
*node*) [pure virtual]

Create a [AttributeValue](#) object according to the information inside the XMLNode as parameter.

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

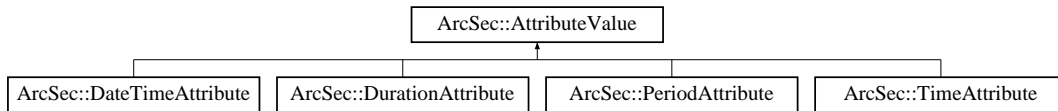
- AttributeProxy.h

## 5.10 ArcSec::AttributeValue Class Reference

Interface for containing different type of <Attribute> node for both policy and request.

```
#include <AttributeValue.h>
```

Inheritance diagram for ArcSec::AttributeValue::



### Public Member Functions

- virtual bool [equal](#) ([AttributeValue](#) \*value, bool check\_id=true)=0
- virtual std::string [encode](#) ()=0
- virtual std::string [getType](#) ()=0
- virtual std::string [getId](#) ()=0

#### 5.10.1 Detailed Description

Interface for containing different type of <Attribute> node for both policy and request.

<Attribute> contains different "Type" definition; Each type of <Attribute> needs different approach to compare the value. Any specific class which is for processing specific "Type" should inherit this class. The "Type" supported so far is: StringAttribute, DateAttribute, [TimeAttribute](#), [DurationAttribute](#), [PeriodAttribute](#), AnyURIAttribute, X500NameAttribute

#### 5.10.2 Member Function Documentation

##### 5.10.2.1 virtual std::string ArcSec::AttributeValue::encode () [pure virtual]

encode the value in a string format

Implemented in [ArcSec::DateTimeAttribute](#), [ArcSec::TimeAttribute](#), [ArcSec::DurationAttribute](#), and [ArcSec::PeriodAttribute](#).

##### 5.10.2.2 virtual bool ArcSec::AttributeValue::equal ([AttributeValue](#) \* value, bool check\_id = true) [pure virtual]

Evaluate whether "this" equals to the parameter value

Implemented in [ArcSec::DateTimeAttribute](#), [ArcSec::TimeAttribute](#), [ArcSec::DurationAttribute](#), and [ArcSec::PeriodAttribute](#).

##### 5.10.2.3 virtual std::string ArcSec::AttributeValue::getId () [pure virtual]

Get the AttributeId of the <Attribute>

Implemented in [ArcSec::DateTimeAttribute](#), [ArcSec::TimeAttribute](#), [ArcSec::DurationAttribute](#), and [ArcSec::PeriodAttribute](#).

#### 5.10.2.4 virtual std::string ArcSec::AttributeValue::getType () [pure virtual]

Get the DataType of the <Attribute>

Implemented in [ArcSec::DateTimeAttribute](#), [ArcSec::TimeAttribute](#), [ArcSec::DurationAttribute](#), and [ArcSec::PeriodAttribute](#).

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

- AttributeValue.h

## 5.11 ArcSec::Attrs Class Reference

[Attrs](#) is a container for one or more [Attr](#).

```
#include <Request.h>
```

### 5.11.1 Detailed Description

[Attrs](#) is a container for one or more [Attr](#).

[Attrs](#) includes includes methods for inserting, getting items, and counting size as well

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

- Request.h



## 5.12 ArcSec::AuthzRequestSection Struct Reference

```
#include <PDP.h>
```

### 5.12.1 Detailed Description

These structure are based on the request schema for [PDP](#), so far it can apply to the ArcPDP's request schema, see `src/hed/pdc/Request.xsd` and `src/hed/pdc/Request.xml`. It could also apply to the XACMLPDP's request schema, since the difference is minor.

Another approach is, the service composes/marshalls the xml structure directly, then the service should use difference code to compose for ArcPDP's request schema and XACMLPDP's schema, which is not so good.

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

- PDP.h

## 5.13 Arc::AutoPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction.

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

### Public Member Functions

- [AutoPointer](#) (void)
- [AutoPointer](#) (T \*o)
- [~AutoPointer](#) (void)
- T & [operator \\*](#) (void) const
- T \* [operator →](#) (void) const
- [operator bool](#) (void) const
- bool [operator!](#) (void) const
- T \* [Ptr](#) (void) const
- T \* [Release](#) (void)

### 5.13.1 Detailed Description

```
template<typename T> class Arc::AutoPointer< T >
```

Wrapper for pointer with automatic destruction.

If ordinary pointer is wrapped in instance of this class it will be automatically destroyed when instance is destroyed. This is useful for maintaing pointers in scope of one function. Only pointers returned by new() are supported.

### 5.13.2 Constructor & Destructor Documentation

**5.13.2.1** `template<typename T> Arc::AutoPointer< T >::AutoPointer (void) [inline]`

NULL pointer constructor.

**5.13.2.2** `template<typename T> Arc::AutoPointer< T >::AutoPointer (T * o) [inline]`

Constructor which wraps pointer.

**5.13.2.3** `template<typename T> Arc::AutoPointer< T >::~AutoPointer (void) [inline]`

Destructor destroys wrapped object using delete().

### 5.13.3 Member Function Documentation

**5.13.3.1** `template<typename T> T& Arc::AutoPointer< T >::operator \* (void) const [inline]`

For refering wrapped object.

**5.13.3.2** `template<typename T> Arc::AutoPointer< T >::operator bool (void) const` `[inline]`

Returns false if pointer is NULL and true otherwise.

**5.13.3.3** `template<typename T> bool Arc::AutoPointer< T >::operator! (void) const`  
`[inline]`

Returns true if pointer is NULL and false otherwise.

**5.13.3.4** `template<typename T> T* Arc::AutoPointer< T >::operator → (void) const`  
`[inline]`

For refering wrapped object.

**5.13.3.5** `template<typename T> T* Arc::AutoPointer< T >::Ptr (void) const` `[inline]`

Cast to original pointer.

**5.13.3.6** `template<typename T> T* Arc::AutoPointer< T >::Release (void)` `[inline]`

Release refred object so that it can be passed to other container.

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

- Utils.h

## 5.14 Arc::BaseConfig Class Reference

```
#include <ArcConfig.h>
```

### Public Member Functions

- void [AddPluginsPath](#) (const std::string &path)
- void [AddPrivateKey](#) (const std::string &path)
- void [AddCertificate](#) (const std::string &path)
- void [AddProxy](#) (const std::string &path)
- void [AddCAFile](#) (const std::string &path)
- void [AddCADir](#) (const std::string &path)
- void [AddOverlay](#) ([XMLNode](#) cfg)
- void [GetOverlay](#) (std::string fname)
- virtual [XMLNode](#) [MakeConfig](#) ([XMLNode](#) cfg) const

### 5.14.1 Detailed Description

Configuration for client interface. It contains information which can't be expressed in class constructor arguments. Most probably common things like software installation location, identity of user, etc.

### 5.14.2 Member Function Documentation

#### 5.14.2.1 void Arc::BaseConfig::AddCADir (const std::string & *path*)

Add CA directory

#### 5.14.2.2 void Arc::BaseConfig::AddCAFile (const std::string & *path*)

Add CA file

#### 5.14.2.3 void Arc::BaseConfig::AddCertificate (const std::string & *path*)

Add certificate

#### 5.14.2.4 void Arc::BaseConfig::AddOverlay ([XMLNode](#) *cfg*)

Add configuration overlay

#### 5.14.2.5 void Arc::BaseConfig::AddPluginsPath (const std::string & *path*)

Adds non-standard location of plugins

#### 5.14.2.6 void Arc::BaseConfig::AddPrivateKey (const std::string & *path*)

Add private key

**5.14.2.7 void Arc::BaseConfig::AddProxy (const std::string & *path*)**

Add credentials proxy

**5.14.2.8 void Arc::BaseConfig::GetOverlay (std::string *fname*)**

Read overlay from file

**5.14.2.9 virtual XMLNode Arc::BaseConfig::MakeConfig (XMLNode *cfg*) const** [virtual]

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument. Returns reference to XML node representing configuration of [ModuleManager](#)

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

- ArcConfig.h

## 5.15 Arc::ChainContext Class Reference

Interface to chain specific functionality.

```
#include <MCCLoader.h>
```

### Public Member Functions

- [operator PluginsFactory \\*](#) ()

#### 5.15.1 Detailed Description

Interface to chain specific functionality.

Object of this class is associated with every [MCCLoader](#) object. It is accessible for [MCC](#) and [Service](#) components and provides an interface to manipulate chains stored in [Loader](#). This makes it possible to modify chains dynamically - like deploying new services on demand.

#### 5.15.2 Member Function Documentation

##### 5.15.2.1 Arc::ChainContext::operator [PluginsFactory](#) \* () [inline]

Returns associated [PluginsFactory](#) object

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

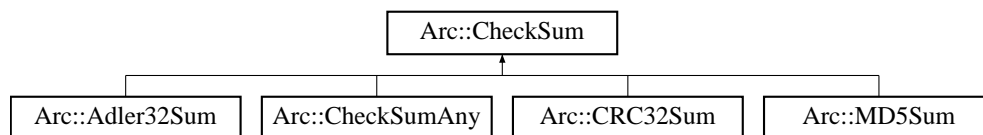
- MCCLoader.h

## 5.16 Arc::Checksum Class Reference

Interface for checksum manipulations.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::Checksum::



### Public Member Functions

- [Checksum](#) (void)
- virtual void [start](#) (void)=0
- virtual void [add](#) (void \*buf, unsigned long long int len)=0
- virtual void [end](#) (void)=0
- virtual void [result](#) (unsigned char \*&res, unsigned int &len) const =0
- virtual int [print](#) (char \*buf, int len) const
- virtual void [scan](#) (const char \*buf)=0
- virtual [operator bool](#) (void) const
- virtual bool [operator!](#) (void) const

### 5.16.1 Detailed Description

Interface for checksum manipulations.

This class is an interface and is extended in the specialized classes [CRC32Sum](#), [MD5Sum](#) and [Adler32Sum](#). The interface is among others used during data transfers through [DataBuffer](#) class

See also:

[CRC32Sum](#)  
[MD5Sum](#)  
[Adler32Sum](#)

### 5.16.2 Constructor & Destructor Documentation

#### 5.16.2.1 Arc::Checksum::Checksum (void) [inline]

Default constructor.

### 5.16.3 Member Function Documentation

#### 5.16.3.1 virtual void Arc::Checksum::add (void \* *buf*, unsigned long long int *len*) [pure virtual]

Add data to be checksummed.

This method calculates the checksum of the passed data chunk, taking into account the previous state of this object.

**Parameters:**

*buf* pointer to data chunk to be checksummed.

*len* size of the data chunk.

Implemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

### 5.16.3.2 **virtual void Arc::Checksum::end (void)** [pure virtual]

Finalize the checksumming.

This method finalizes the checksum algorithm, that is calculating the final checksum result.

Implemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

### 5.16.3.3 **virtual Arc::Checksum::operator bool (void) const** [inline, virtual]

Indicates whether the checksum has been calculated.

Reimplemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

### 5.16.3.4 **virtual bool Arc::Checksum::operator! (void) const** [inline, virtual]

Indicates whether the checksum has not been calculated.

Reimplemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

### 5.16.3.5 **virtual int Arc::Checksum::print (char \* buf, int len) const** [inline, virtual]

Retrieve result of checksum into a string.

The passed string buf is filled with result of checksum algorithm in base 16. At most len chacters is filled into buffer buf. The hexadecimal value is prepended with "<algorithm>:", where <algorithm> is one of "cksum", "md5" or "adler32" respectively corresponding to the result from the [CRC32Sum](#), [MD5Sum](#) and [Adler32](#) classes.

**Parameters:**

*buf* pointer to buffer which should be filled with checksum result.

*len* max number of character filled into buffer.

Reimplemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

### 5.16.3.6 **virtual void Arc::Checksum::result (unsigned char \*& res, unsigned int & len) const** [pure virtual]

Retrieve result of checksum as binary blob.

Implemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).



**5.16.3.7 virtual void Arc::Checksum::scan (const char \* *buf*)** [pure virtual]

Set internal checksum state.

This method sets the internal state to that of the passed textural representation. The format passed to this method must be the same as retrieved from the [Checksum::print](#) method.

**Parameters:**

*buf* string containing textural representation of checksum

**See also:**

[Checksum::print](#)

Implemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

**5.16.3.8 virtual void Arc::Checksum::start (void)** [pure virtual]

Initiate the checksum algorithm.

This method must be called before starting a new checksum calculation.

Implemented in [Arc::CRC32Sum](#), [Arc::MD5Sum](#), [Arc::Adler32Sum](#), and [Arc::ChecksumAny](#).

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

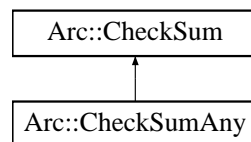
- CheckSum.h

## 5.17 Arc::ChecksumAny Class Reference

Wrapper for [Checksum](#) class.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::ChecksumAny::



### Public Member Functions

- virtual void [start](#) (void)
- virtual void [add](#) (void \*buf, unsigned long long int len)
- virtual void [end](#) (void)
- virtual void [result](#) (unsigned char \*&res, unsigned int &len) const
- virtual int [print](#) (char \*buf, int len) const
- virtual void [scan](#) (const char \*buf)
- virtual [operator bool](#) (void) const
- virtual bool [operator!](#) (void) const

### Static Public Member Functions

- static std::string [FileChecksum](#) (const std::string &filepath, type tp=md5, bool decimalbase=false)

#### 5.17.1 Detailed Description

Wrapper for [Checksum](#) class.

To be used for manipulation of any supported checksum type in a transparent way.

#### 5.17.2 Member Function Documentation

**5.17.2.1** virtual void Arc::ChecksumAny::add (void \* *buf*, unsigned long long int *len*) [inline, virtual]

Add data to be checksummed.

This method calculates the checksum of the passed data chunk, taking into account the previous state of this object.

##### Parameters:

*buf* pointer to data chunk to be checksummed.

*len* size of the data chunk.

Implements [Arc::Checksum](#).

**5.17.2.2 virtual void Arc::ChecksumAny::end (void) [inline, virtual]**

Finalize the checksumming.

This method finalizes the checksum algorithm, that is calculating the final checksum result.

Implements [Arc::Checksum](#).

**5.17.2.3 static std::string Arc::ChecksumAny::FileChecksum (const std::string & filepath, type tp = md5, bool decimalbase = false) [static]**

Get checksum of a file.

This method provide an easy way to get the checksum of a file, by only specifying the path to the file. Optionally the checksum type can be specified, if not the MD5 algorithm will be used.

**Parameters:**

*filepath* path to file of which checksum should be calculated

*tp* type of checksum algorithm to use, default is md5.

*decimalbase* specifies whether output should be in base 10 or 16

**Returns:**

a string containing the calculated checksum is returned.

**5.17.2.4 virtual Arc::ChecksumAny::operator bool (void) const [inline, virtual]**

Indicates whether the checksum has been calculated.

Reimplemented from [Arc::Checksum](#).

**5.17.2.5 virtual bool Arc::ChecksumAny::operator! (void) const [inline, virtual]**

Indicates whether the checksum has not been calculated.

Reimplemented from [Arc::Checksum](#).

**5.17.2.6 virtual int Arc::ChecksumAny::print (char \* buf, int len) const [inline, virtual]**

Retrieve result of checksum into a string.

The passed string buf is filled with result of checksum algorithm in base 16. At most len chacters is filled into buffer buf. The hexadecimal value is prepended with "<algorithm>:", where <algorithm> is one of "cksum", "md5" or "adler32" respectively corresponding to the result from the [CRC32Sum](#), [MD5Sum](#) and Adler32 classes.

**Parameters:**

*buf* pointer to buffer which should be filled with checksum result.

*len* max number of character filled into buffer.

Reimplemented from [Arc::Checksum](#).

**5.17.2.7 virtual void Arc::ChecksumAny::result (unsigned char \*& *res*, unsigned int & *len*) const**  
[inline, virtual]

Retrieve result of checksum as binary blob.

Implements [Arc::Checksum](#).

**5.17.2.8 virtual void Arc::ChecksumAny::scan (const char \* *buf*)** [inline, virtual]

Set internal checksum state.

This method sets the internal state to that of the passed textural representation. The format passed to this method must be the same as retrieved from the [Checksum::print](#) method.

**Parameters:**

*buf* string containing textural representation of checksum

**See also:**

[Checksum::print](#)

Implements [Arc::Checksum](#).

**5.17.2.9 virtual void Arc::ChecksumAny::start (void)** [inline, virtual]

Initiate the checksum algorithm.

This method must be called before starting a new checksum calculation.

Implements [Arc::Checksum](#).

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

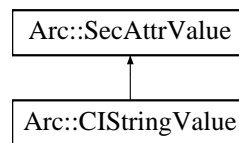
- CheckSum.h

## 5.18 Arc::CIStrStringValue Class Reference

This class implements case insensitive strings as security attributes.

```
#include <CIStrStringValue.h>
```

Inheritance diagram for Arc::CIStrStringValue::



### Public Member Functions

- [CIStrStringValue](#) ()
- [CIStrStringValue](#) (const char \*ss)
- [CIStrStringValue](#) (const std::string &ss)
- virtual [operator bool](#) ()

### Protected Member Functions

- virtual bool [equal](#) ([SecAttrValue](#) &b)

#### 5.18.1 Detailed Description

This class implements case insensitive strings as security attributes.

This is an example of how to inherit [SecAttrValue](#). The class is meant to implement security attributes that are case insensitive strings.

#### 5.18.2 Constructor & Destructor Documentation

##### 5.18.2.1 Arc::CIStrStringValue::CIStrStringValue ()

Default constructor

##### 5.18.2.2 Arc::CIStrStringValue::CIStrStringValue (const char \* ss)

This is a constructor that takes a string literal.

##### 5.18.2.3 Arc::CIStrStringValue::CIStrStringValue (const std::string & ss)

This is a constructor that takes a string object.

### 5.18.3 Member Function Documentation

#### 5.18.3.1 `virtual bool Arc::CStringValue::equal (SecAttrValue & b)` [protected, virtual]

This function returns true if two strings are the same apart from letter case

Reimplemented from [Arc::SecAttrValue](#).

#### 5.18.3.2 `virtual Arc::CStringValue::operator bool ()` [virtual]

This function returns false if the string is empty or uninitialized

Reimplemented from [Arc::SecAttrValue](#).

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

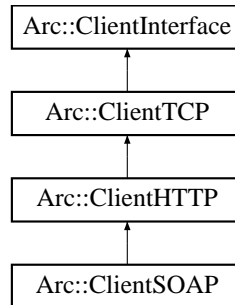
- `CStringValue.h`

## 5.19 Arc::ClientHTTP Class Reference

Class for setting up a [MCC](#) chain for HTTP communication.

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientHTTP::



### 5.19.1 Detailed Description

Class for setting up a [MCC](#) chain for HTTP communication.

The [ClientHTTP](#) class inherits from the [ClientTCP](#) class and adds an HTTP [MCC](#) to the chain.

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

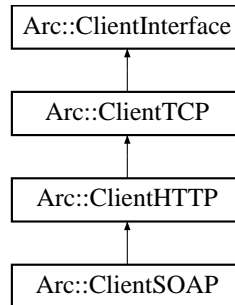
- ClientInterface.h

## 5.20 Arc::ClientInterface Class Reference

Utility base class for [MCC](#).

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientInterface::



### 5.20.1 Detailed Description

Utility base class for [MCC](#).

The [ClientInterface](#) class is a utility base class used for configuring a client side [Message](#) Chain Component ([MCC](#)) chain and loading it into memory. It has several specializations of increasing complexity of the [MCC](#) chains.

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

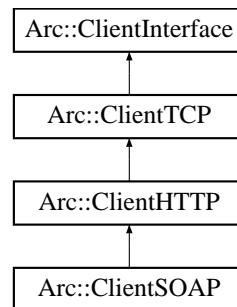
- ClientInterface.h



## 5.21 Arc::ClientSOAP Class Reference

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientSOAP::



### Public Member Functions

- [ClientSOAP](#) ()
- [MCC\\_Status process](#) ([PayloadSOAP](#) \*request, [PayloadSOAP](#) \*\*response)
- [MCC\\_Status process](#) (const std::string &action, [PayloadSOAP](#) \*request, [PayloadSOAP](#) \*\*response)
- [MCC](#) \* [GetEntry](#) ()
- void [AddSecHandler](#) ([XMLNode](#) handlercfg, const std::string &libanme="", const std::string &libpath="")
- virtual bool [Load](#) ()

### 5.21.1 Detailed Description

Class with easy interface for sending/receiving SOAP messages over HTTP(S/G). It takes care of configuring [MCC](#) chain and making an entry point.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 Arc::ClientSOAP::ClientSOAP () [inline]

Constructor creates [MCC](#) chain and connects to server.

### 5.21.3 Member Function Documentation

#### 5.21.3.1 void Arc::ClientSOAP::AddSecHandler ([XMLNode](#) handlercfg, const std::string &libanme = "", const std::string &libpath = "")

Adds security handler to configuration of SOAP [MCC](#)

Reimplemented from [Arc::ClientHTTP](#).

**5.21.3.2** **MCC\*** **Arc::ClientSOAP::GetEntry ()** [inline]

Returns entry point to SOAP **MCC** in configured chain. To initialize entry point **Load()** method must be called.

Reimplemented from [Arc::ClientHTTP](#).

**5.21.3.3** **virtual bool Arc::ClientSOAP::Load ()** [virtual]

Instantiates pluggable elements according to generated configuration

Reimplemented from [Arc::ClientHTTP](#).

**5.21.3.4** **MCC\_Status Arc::ClientSOAP::process (const std::string & action, PayloadSOAP \* request, PayloadSOAP \*\* response)**

Send SOAP request with specified SOAP action and receive response.

**5.21.3.5** **MCC\_Status Arc::ClientSOAP::process (PayloadSOAP \* request, PayloadSOAP \*\* response)**

Send SOAP request and receive response.

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

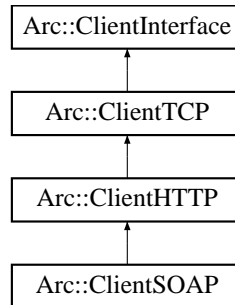
- ClientInterface.h

## 5.22 Arc::ClientTCP Class Reference

Class for setting up a [MCC](#) chain for TCP communication.

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientTCP::



### 5.22.1 Detailed Description

Class for setting up a [MCC](#) chain for TCP communication.

The [ClientTCP](#) class is a specialization of the [ClientInterface](#) which sets up a client [MCC](#) chain for TCP communication, and optionally with a security layer on top which can be either TLS, GSI or SSL3.

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

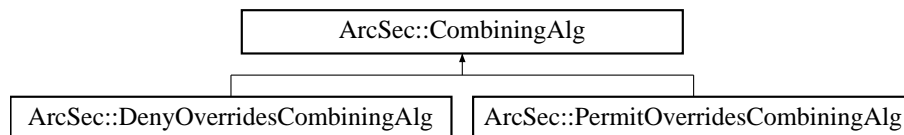
- ClientInterface.h

## 5.23 ArcSec::CombiningAlg Class Reference

Interface for combining algrithm.

```
#include <CombiningAlg.h>
```

Inheritance diagram for ArcSec::CombiningAlg::



### Public Member Functions

- virtual Result [combine](#) (EvaluationCtx \*ctx, std::list< [Policy](#) \* > policies)=0
- virtual const std::string & [getalgId](#) (void) const =0

#### 5.23.1 Detailed Description

Interface for combining algrithm.

This class is used to implement a specific combining algorithm for combining policies.

#### 5.23.2 Member Function Documentation

##### 5.23.2.1 virtual Result ArcSec::CombiningAlg::combine ([EvaluationCtx](#) \* ctx, std::list< [Policy](#) \* > *policies*) [pure virtual]

Evaluate request against policy, and if there are more than one policies, combine the evaluation results according to the combing algorithm implemented inside in the method combine(ctx, policies) itself.

##### Parameters:

*ctx* The information about request is included

*policies* The "match" and "eval" method inside each policy will be called, and then those results from each policy will be combined according to the combining algorithm inside CombiningAlg class.

Implemented in [ArcSec::DenyOverridesCombiningAlg](#), and [ArcSec::PermitOverridesCombiningAlg](#).

##### 5.23.2.2 virtual const std::string& ArcSec::CombiningAlg::getalgId (void) const [pure virtual]

Get the identifier of the combining algorithm class

##### Returns:

The identity of the algorithm

Implemented in [ArcSec::DenyOverridesCombiningAlg](#), and [ArcSec::PermitOverridesCombiningAlg](#).

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

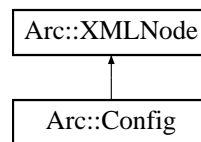
- CombiningAlg.h

## 5.24 Arc::Config Class Reference

Configuration element - represents (sub)tree of ARC configuration.

```
#include <ArcConfig.h>
```

Inheritance diagram for Arc::Config::



### Public Member Functions

- [Config](#) ()
- [Config](#) (const char \*filename)
- [Config](#) (const std::string &xml\_str)
- [Config](#) (XMLNode xml)
- [Config](#) (long cfg\_ptr\_addr)
- [Config](#) (const [Config](#) &cfg)
- void [print](#) (void)
- bool [parse](#) (const char \*filename)
- const std::string & [getFileName](#) (void) const
- void [setFileName](#) (const std::string &filename)
- void [save](#) (const char \*filename)

### 5.24.1 Detailed Description

Configuration element - represents (sub)tree of ARC configuration.

This class is intended to be used to pass configuration details to various parts of HED and external modules. Currently it's just a wrapper over XML tree. But than may change in a future, although interface should be preserved. Currently it is capable of loading XML configuration document from file. In future it will be capable of loading more user-readable format and process it into tree-like structure convenient for machine processing (XML-like). So far there are no schema and/or namespaces assigned.

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 Arc::Config::Config () [inline]

Creates empty XML tree

#### 5.24.2.2 Arc::Config::Config (const char \*filename)

Loads configuration document from file 'filename'

**5.24.2.3 Arc::Config::Config (const std::string & *xml\_str*) [inline]**

Parse configuration document from memory

**5.24.2.4 Arc::Config::Config (XMLNode *xml*) [inline]**

Acquire existing XML (sub)tree. Content is not copied. Make sure XML tree is not destroyed while in use by this object.

**5.24.2.5 Arc::Config::Config (long *cfg\_ptr\_addr*)**

Copy constructor used by language bindings

**5.24.2.6 Arc::Config::Config (const Config & *cfg*)**

Copy constructor used by language bindings

**5.24.3 Member Function Documentation****5.24.3.1 const std::string& Arc::Config::getFileName (void) const [inline]**

Gives back file name of config file or empty string if it was generated from the XMLNode subtree

**5.24.3.2 bool Arc::Config::parse (const char \* *filename*)**

Parse configuration document from file 'filename'

**5.24.3.3 void Arc::Config::print (void)**

Print structure of document. For debugging purposes. Printed content is not an XML document.

**5.24.3.4 void Arc::Config::save (const char \* *filename*)**

Save to file

**5.24.3.5 void Arc::Config::setFileName (const std::string & *filename*) [inline]**

Set the file name of config file

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

- ArcConfig.h

## 5.25 Arc::ConfusaCertHandler Class Reference

```
#include <ConfusaCertHandler.h>
```

### Public Member Functions

- [ConfusaCertHandler](#) (int keysize, const std::string dn)
- std::string [getCertRequestB64](#) ()
- bool [createCertRequest](#) (std::string password="", std::string storedir=".")

### 5.25.1 Detailed Description

Wrapper around [Credential](#) handling the Confusa specifics.

### 5.25.2 Constructor & Destructor Documentation

#### 5.25.2.1 Arc::ConfusaCertHandler::ConfusaCertHandler (int keysize, const std::string dn)

Create a new [ConfusaCertHandler](#) for DN dn and given keysize Basically Confusa cert handler wraps around [Credential](#)

### 5.25.3 Member Function Documentation

#### 5.25.3.1 bool Arc::ConfusaCertHandler::createCertRequest (std::string password = "", std::string storedir = ". / ")

Create a new end entity certificate, with a private key encrypted with password password. Private key and certificate will be stored in directory storedir.

#### 5.25.3.2 std::string Arc::ConfusaCertHandler::getCertRequestB64 ()

Get the certificate request managed by this confusa cert handler in base 64 encoding

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

- ConfusaCertHandler.h



## 5.26 Arc::ConfusaParserUtils Class Reference

```
#include <ConfusaParserUtils.h>
```

### Static Public Member Functions

- static std::string [urlencode](#) (const std::string url)
- static std::string [urlencode\\_params](#) (const std::string url)
- static xmlDocPtr [get\\_doc](#) (const std::string xml\_file)
- static void [destroy\\_doc](#) (xmlDocPtr doc)
- static std::string [extract\\_body\\_information](#) (const std::string html\_string)
- static std::string [handle\\_redirect\\_step](#) (Arc::MCCConfig cfg, const std::string remote\_url, std::string \*cookies=NULL, std::multimap< std::string, std::string > \*httpAttributes=NULL)
- static std::string [evaluate\\_path](#) (xmlDocPtr doc, const std::string xpathExpr, std::list< std::string > \*contentList=NULL)

### 5.26.1 Detailed Description

Methods often needed in evaluation web pages from the Confusa WebSSO workflow

### 5.26.2 Member Function Documentation

#### 5.26.2.1 static void Arc::ConfusaParserUtils::destroy\_doc (xmlDocPtr *doc*) [static]

Destroy a libxml2 doc representation

#### 5.26.2.2 static std::string Arc::ConfusaParserUtils::evaluate\_path (xmlDocPtr *doc*, const std::string *xpathExpr*, std::list< std::string > \* *contentList* = NULL) [static]

Evaluate the given xPathExpr on the document ptr. Return a string with the FIRST result if contentList is NULL. Return a string with the first result and all results, including the first one, in contentList if contentList is not null.

#### 5.26.2.3 static std::string Arc::ConfusaParserUtils::extract\_body\_information (const std::string *html\_string*) [static]

Get the part only within <body> and </body> in a HTML string For parsing, usually only this part is interesting.

#### 5.26.2.4 static xmlDocPtr Arc::ConfusaParserUtils::get\_doc (const std::string *xml\_file*) [static]

Construct a libxml2 doc representation from the xml file

**5.26.2.5** `static std::string Arc::ConfusaParserUtils::handle_redirect_step (Arc::MCCConfig cfg,  
const std::string remote_url, std::string * cookies = NULL, std::multimap< std::string,  
std::string > * httpAttributes = NULL) [static]`

Handle a single redirect step from the SAML2 WebSSO profile. Store the received cookie in *\*cookie* and pass the given *httpAttributes* to the site during redirect.

**5.26.2.6** `static std::string Arc::ConfusaParserUtils::urlencode (const std::string url) [static]`

urlencode the passed string

**5.26.2.7** `static std::string Arc::ConfusaParserUtils::urlencode_params (const std::string url)  
[static]`

Urlencode the passed string with respect to the parameters. The difference to `urlencode` is that the parameters will keep their separators, i.e. the `?` and `&` separating parameters will be preserved.

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

- `ConfusaParserUtils.h`

## 5.27 Arc::CountedPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction and mutiple references.

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

### Public Member Functions

- T & [operator \\*](#) (void) const
- T \* [operator →](#) (void) const
- [operator bool](#) (void) const
- bool [operator!](#) (void) const
- bool [operator==](#) (const [CountedPointer](#) &p) const
- bool [operator!=](#) (const [CountedPointer](#) &p) const
- bool [operator<](#) (const [CountedPointer](#) &p) const
- T \* [Ptr](#) (void) const
- T \* [Release](#) (void)

### Data Structures

- class [Base](#)

#### 5.27.1 Detailed Description

```
template<typename T> class Arc::CountedPointer< T >
```

Wrapper for pointer with automatic destruction and mutiple references.

If ordinary pointer is wrapped in instance of this class it will be automatically destroyed when all instances refering to it are destroyed. This is useful for maintaing pointers refered from multiple structures wiith automatic destruction of original object when last reference is destroyed. It is similar to Java approach with a difference that desctruction time is strictly defined. Only pointers returned by new() are supported. This class is not thread-safe

#### 5.27.2 Member Function Documentation

**5.27.2.1** `template<typename T> T& Arc::CountedPointer< T >::operator * (void) const`  
[inline]

For refering wrapped object.

**5.27.2.2** `template<typename T> Arc::CountedPointer< T >::operator bool (void) const`  
[inline]

Returns false if pointer is NULL and true otherwise.

**5.27.2.3** `template<typename T> bool Arc::CountedPointer< T >::operator! (void) const`  
[inline]

Returns true if pointer is NULL and false otherwise.

**5.27.2.4** `template<typename T> bool Arc::CountedPointer< T >::operator!= (const CountedPointer< T > &p) const` `[inline]`

Returns true if pointers are not equal.

**5.27.2.5** `template<typename T> T* Arc::CountedPointer< T >::operator → (void) const` `[inline]`

For referring wrapped object.

**5.27.2.6** `template<typename T> bool Arc::CountedPointer< T >::operator< (const CountedPointer< T > &p) const` `[inline]`

Comparison operator.

**5.27.2.7** `template<typename T> bool Arc::CountedPointer< T >::operator== (const CountedPointer< T > &p) const` `[inline]`

Returns true if pointers are equal.

**5.27.2.8** `template<typename T> T* Arc::CountedPointer< T >::Ptr (void) const` `[inline]`

Cast to original pointer.

**5.27.2.9** `template<typename T> T* Arc::CountedPointer< T >::Release (void)` `[inline]`

Release refred object so that it can be passed to other container.

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

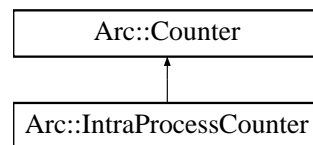
- Utils.h

## 5.28 Arc::Counter Class Reference

A class defining a common interface for counters.

```
#include <Counter.h>
```

Inheritance diagram for Arc::Counter::



### Public Member Functions

- virtual `~Counter ()`
- virtual int `getLimit ()=0`
- virtual int `setLimit (int newLimit)=0`
- virtual int `changeLimit (int amount)=0`
- virtual int `getExcess ()=0`
- virtual int `setExcess (int newExcess)=0`
- virtual int `changeExcess (int amount)=0`
- virtual int `getValue ()=0`
- virtual `CounterTicket reserve (int amount=1, Glib::TimeVal duration=ETERNAL, bool prioritized=false, Glib::TimeVal timeOut=ETERNAL)=0`

### Protected Types

- typedef unsigned long long int `IDType`

### Protected Member Functions

- `Counter ()`
- virtual void `cancel (IDType reservationID)=0`
- virtual void `extend (IDType &reservationID, Glib::TimeVal &expiryTime, Glib::TimeVal duration=ETERNAL)=0`
- Glib::TimeVal `getCurrentTime ()`
- Glib::TimeVal `getExpiryTime (Glib::TimeVal duration)`
- `CounterTicket getCounterTicket (Counter::IDType reservationID, Glib::TimeVal expiryTime, Counter *counter)`
- `ExpirationReminder getExpirationReminder (Glib::TimeVal expTime, Counter::IDType resID)`

### Friends

- class `CounterTicket`
- class `ExpirationReminder`

### 5.28.1 Detailed Description

A class defining a common interface for counters.

This class defines a common interface for counters as well as some common functionality.

The purpose of a counter is to provide housekeeping some resource such as e.g. disk space, memory or network bandwidth. The counter itself will not be aware of what kind of resource it limits the use of. Neither will it be aware of what unit is being used to measure that resource. Counters are thus very similar to semaphores. Furthermore, counters are designed to handle concurrent operations from multiple threads/processes in a consistent manner.

Every counter has a limit, an excess limit and a value. The limit is a number that specify how many units are available for reservation. The value is the number of units that are currently available for reservation, i.e. has not allready been reserved. The excess limit specify how many extra units can be reserved for high priority needs even if there are no normal units available for reservation. The excess limit is similar to the credit limit of e.g. a VISA card.

The users of the resource must thus first call the counter in order to make a reservation of an appropriate amount of the resource, then allocate and use the resource and finally call the counter again to cancel the reservation.

Typical usage is:

```
// Declare a counter. Replace XYZ by some appropriate kind of
// counter and provide required parameters. Unit is MB.
XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
CounterTicket tick = memory.reserve(2*sizeof(double));
// Use the memory.
double* A=new double[2000000];
doSomething(A);
delete[] A;
// Cancel the reservation.
tick.cancel();
```

There are also alternative ways to make reservations, including self-expiring reservations, prioritized reservations and reservations that fail if they cannot be made fast enough.

For self expiring reservations, a duration is provided in the reserve call:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0));
```

A self-expiring reservation can be cancelled explicitly before it expires, but if it is not cancelled it will expire automatically when the duration has passed. The default value for the duration is ETERNAL, which means that the reservation will not be cancelled automatically.

Prioritized reservations may use the excess limit and succeed immediately even if there are no normal units available for reservation. The value of the counter will in this case become negative. A prioritized reservation looks like this:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0), true);
```

Finally, a time out option can be provided for a reservation. If some task should be performed within two seconds or not at all, the reservation can look like this:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0),
                    true, Glib::TimeVal(2,0));
if (tick.isValid())
    doSomething(...);
```

## 5.28.2 Member Typedef Documentation

### 5.28.2.1 typedef unsigned long long int Arc::Counter::IDType [protected]

A typedef of identification numbers for reservation.

This is a type that is used as identification numbers (keys) for referencing of reservations. It is used internally in counters for book keeping of reservations as well as in the [CounterTicket](#) class in order to be able to cancel and extend reservations.

## 5.28.3 Constructor & Destructor Documentation

### 5.28.3.1 Arc::Counter::Counter () [protected]

Default constructor.

This is the default constructor. Since [Counter](#) is an abstract class, it should only be used by subclasses. Therefore it is protected. Furthermore, since the [Counter](#) class has no attributes, nothing needs to be initialized and thus this constructor is empty.

### 5.28.3.2 virtual Arc::Counter::~~Counter () [virtual]

The destructor.

This is the destructor of the [Counter](#) class. Since the [Counter](#) class has no attributes, nothing needs to be cleaned up and thus the destructor is empty.

## 5.28.4 Member Function Documentation

### 5.28.4.1 virtual void Arc::Counter::cancel (IDType reservationID) [protected, pure virtual]

Cancellation of a reservation.

This method cancels a reservation. It is called by the [CounterTicket](#) that corresponds to the reservation.

#### Parameters:

*reservationID* The identity number (key) of the reservation to cancel.

### 5.28.4.2 virtual int Arc::Counter::changeExcess (int amount) [pure virtual]

Changes the excess limit of the counter.

Changes the excess limit of the counter by adding a certain amount to the current excess limit.

#### Parameters:

*amount* The amount by which to change the excess limit.

#### Returns:

The new excess limit.

Implemented in [Arc::IntraProcessCounter](#).

#### 5.28.4.3 `virtual int Arc::Counter::changeLimit (int amount)` [pure virtual]

Changes the limit of the counter.

Changes the limit of the counter by adding a certain amount to the current limit.

##### Parameters:

*amount* The amount by which to change the limit.

##### Returns:

The new limit.

Implemented in [Arc::IntraProcessCounter](#).

#### 5.28.4.4 `virtual void Arc::Counter::extend (IDType & reservationID, Glib::TimeVal & expiryTime, Glib::TimeVal duration = ETERNAL)` [protected, pure virtual]

Extension of a reservation.

This method extends a reservation. It is called by the [CounterTicket](#) that corresponds to the reservation.

##### Parameters:

*reservationID* Used for input as well as output. Contains the identification number of the original reservation on entry and the new identification number of the extended reservation on exit.

*expiryTime* Used for input as well as output. Contains the expiry time of the original reservation on entry and the new expiry time of the extended reservation on exit.

*duration* The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

#### 5.28.4.5 `CounterTicket Arc::Counter::getCounterTicket (Counter::IDType reservationID, Glib::TimeVal expiryTime, Counter * counter)` [protected]

A "relay method" for a constructor of the [CounterTicket](#) class.

This method acts as a relay for one of the constructors of the [CounterTicket](#) class. That constructor is private, but needs to be accessible from the subclasses of [Counter](#) (but not from anywhere else). In order not to have to declare every possible subclass of [Counter](#) as a friend of [CounterTicket](#), only the base class [Counter](#) is a friend and its subclasses access the constructor through this method. (If C++ had supported "package access", as Java does, this trick would not have been necessary.)

##### Parameters:

*reservationID* The identity number of the reservation corresponding to the [CounterTicket](#).

*expiryTime* the expiry time of the reservation corresponding to the [CounterTicket](#).

*counter* The [Counter](#) from which the reservation has been made.

##### Returns:

The counter ticket that has been created.



**5.28.4.6 Glib::TimeVal Arc::Counter::getCurrentTime ()** [protected]

Get the current time.

Returns the current time. An "adapter method" for the assign\_current\_time() method in the Glib::TimeVal class. return The current time.

**5.28.4.7 virtual int Arc::Counter::getExcess ()** [pure virtual]

Returns the excess limit of the counter.

Returns the excess limit of the counter, i.e. by how much the usual limit may be exceeded by prioritized reservations.

**Returns:**

The excess limit.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.4.8 ExpirationReminder Arc::Counter::getExpirationReminder (Glib::TimeVal *expTime*, Counter::IDType *resID*)** [protected]

A "relay method" for the constructor of [ExpirationReminder](#).

This method acts as a relay for one of the constructors of the [ExpirationReminder](#) class. That constructor is private, but needs to be accessible from the subclasses of [Counter](#) (but not from anywhere else). In order not to have to declare every possible subclass of [Counter](#) as a friend of [ExpirationReminder](#), only the base class [Counter](#) is a friend and its subclasses access the constructor through this method. (If C++ had supported "package access", as Java does, this trick would not have been necessary.)

**Parameters:**

*expTime* the expiry time of the reservation corresponding to the [ExpirationReminder](#).

*resID* The identity number of the reservation corresponding to the [ExpirationReminder](#).

**Returns:**

The [ExpirationReminder](#) that has been created.

**5.28.4.9 Glib::TimeVal Arc::Counter::getExpiryTime (Glib::TimeVal *duration*)** [protected]

Computes an expiry time.

This method computes an expiry time by adding a duration to the current time.

**Parameters:**

*duration* The duration.

**Returns:**

The expiry time.

**5.28.4.10 virtual int Arc::Counter::getLimit ()** [pure virtual]

Returns the current limit of the counter.

This method returns the current limit of the counter, i.e. how many units can be reserved simultaneously by different threads without claiming high priority.

**Returns:**

The current limit of the counter.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.4.11 virtual int Arc::Counter::getValue ()** [pure virtual]

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the value is decreased. Normally, the value should never be negative, but this may happen if there are prioritized reservations. It can also happen if the limit is decreased after some reservations have been made, since reservations are never revoked.

**Returns:**

The current value of the counter.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.4.12 virtual CounterTicket Arc::Counter::reserve (int amount = 1, Glib::TimeVal duration = ETERNAL, bool prioritized = false, Glib::TimeVal timeOut = ETERNAL)** [pure virtual]

Makes a reservation from the counter.

This method makes a reservation from the counter. If the current value of the counter is too low to allow for the reservation, the method blocks until the reservation is possible or times out.

**Parameters:**

*amount* The amount to reserve, default value is 1.

*duration* The duration of a self expiring reservation, default is that it lasts forever.

*prioritized* Whether this reservation is prioritized and thus allowed to use the excess limit.

*timeOut* The maximum time to block if the value of the counter is too low, default is to allow "eternal" blocking.

**Returns:**

A [CounterTicket](#) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.4.13** `virtual int Arc::Counter::setExcess (int newExcess)` [pure virtual]

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

**Parameters:**

*newExcess* The new excess limit, an absolute number.

**Returns:**

The new excess limit.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.4.14** `virtual int Arc::Counter::setLimit (int newLimit)` [pure virtual]

Sets the limit of the counter.

This method sets a new limit for the counter.

**Parameters:**

*newLimit* The new limit, an absolute number.

**Returns:**

The new limit.

Implemented in [Arc::IntraProcessCounter](#).

**5.28.5 Friends And Related Function Documentation****5.28.5.1** `friend class CounterTicket` [friend]

The [CounterTicket](#) class needs to be a friend.

**5.28.5.2** `friend class ExpirationReminder` [friend]

The [ExpirationReminder](#) class needs to be a friend.

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

- Counter.h

## 5.29 Arc::CounterTicket Class Reference

A class for "tickets" that correspond to counter reservations.

```
#include <Counter.h>
```

### Public Member Functions

- [CounterTicket](#) ()
- bool [isValid](#) ()
- void [extend](#) (Glib::TimeVal duration)
- void [cancel](#) ()

### Friends

- class [Counter](#)

### 5.29.1 Detailed Description

A class for "tickets" that correspond to counter reservations.

This is a class for reservation tickets. When a reservation is made from a [Counter](#), a [ReservationTicket](#) is returned. This ticket can then be queried about the validity of a reservation. It can also be used for cancelation and extension of reservations.

Typical usage is:

```
// Declare a counter. Replace XYZ by some appropriate kind of
// counter and provide required parameters. Unit is MB.
XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
CounterTicket tick = memory.reserve(2*sizeof(double));
// Use the memory.
double* A=new double[2000000];
doSomething(A);
delete[] A;
// Cancel the reservation.
tick.cancel();
```

### 5.29.2 Constructor & Destructor Documentation

#### 5.29.2.1 Arc::CounterTicket::CounterTicket ()

The default constructor.

This is the default constructor. It creates a [CounterTicket](#) that is not valid. The ticket object that is created can later be assigned a ticket that is returned by the [reserve\(\)](#) method of a [Counter](#).

### 5.29.3 Member Function Documentation

#### 5.29.3.1 void Arc::CounterTicket::cancel ()

Cancels a reservation.

This method is called to cancel a reservation. It may be called also for self-expiring reservations, which will then be cancelled before they were originally planned to expire.

#### 5.29.3.2 void Arc::CounterTicket::extend (Glib::TimeVal *duration*)

Extends a reservation.

Extends a self-expiring reservation. In order to succeed the extension should be made before the previous reservation expires.

##### Parameters:

*duration* The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

#### 5.29.3.3 bool Arc::CounterTicket::isValid ()

Returns the validity of a [CounterTicket](#).

This method checks whether a [CounterTicket](#) is valid. The ticket was probably returned earlier by the `reserve()` method of a [Counter](#) but the corresponding reservation may have expired.

##### Returns:

The validity of the ticket.

### 5.29.4 Friends And Related Function Documentation

#### 5.29.4.1 friend class [Counter](#) [friend]

The [Counter](#) class needs to be a friend.

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

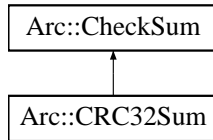
- Counter.h

## 5.30 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

```
#include <CheckSum.h>
```

Inheritance diagram for Arc::CRC32Sum::



### Public Member Functions

- virtual void [start](#) (void)
- virtual void [add](#) (void \*buf, unsigned long long int len)
- virtual void [end](#) (void)
- virtual void [result](#) (unsigned char \*&res, unsigned int &len) const
- virtual int [print](#) (char \*buf, int len) const
- virtual void [scan](#) (const char \*buf)
- virtual [operator bool](#) (void) const
- virtual bool [operator!](#) (void) const

### 5.30.1 Detailed Description

Implementation of CRC32 checksum.

This class is a specialized class of the [CheckSum](#) class. It provides an implementation for the CRC-32 IEEE 802.3 standard.

### 5.30.2 Member Function Documentation

#### 5.30.2.1 virtual void Arc::CRC32Sum::add (void \* *buf*, unsigned long long int *len*) [virtual]

Add data to be checksummed.

This method calculates the checksum of the passed data chunk, taking into account the previous state of this object.

#### Parameters:

*buf* pointer to data chunk to be checksummed.

*len* size of the data chunk.

Implements [Arc::CheckSum](#).

**5.30.2.2 virtual void Arc::CRC32Sum::end (void) [virtual]**

Finalize the checksumming.

This method finalizes the checksum algorithm, that is calculating the final checksum result.

Implements [Arc::Checksum](#).

**5.30.2.3 virtual Arc::CRC32Sum::operator bool (void) const [inline, virtual]**

Indicates whether the checksum has been calculated.

Reimplemented from [Arc::Checksum](#).

**5.30.2.4 virtual bool Arc::CRC32Sum::operator! (void) const [inline, virtual]**

Indicates whether the checksum has not been calculated.

Reimplemented from [Arc::Checksum](#).

**5.30.2.5 virtual int Arc::CRC32Sum::print (char \* *buf*, int *len*) const [virtual]**

Retrieve result of checksum into a string.

The passed string *buf* is filled with result of checksum algorithm in base 16. At most *len* characters is filled into buffer *buf*. The hexadecimal value is prepended with "<algorithm>:", where <algorithm> is one of "cksum", "md5" or "adler32" respectively corresponding to the result from the [CRC32Sum](#), [MD5Sum](#) and [Adler32](#) classes.

**Parameters:**

*buf* pointer to buffer which should be filled with checksum result.

*len* max number of character filled into buffer.

Reimplemented from [Arc::Checksum](#).

**5.30.2.6 virtual void Arc::CRC32Sum::result (unsigned char \*& *res*, unsigned int & *len*) const [inline, virtual]**

Retrieve result of checksum as binary blob.

Implements [Arc::Checksum](#).

**5.30.2.7 virtual void Arc::CRC32Sum::scan (const char \* *buf*) [virtual]**

Set internal checksum state.

This method sets the internal state to that of the passed textual representation. The format passed to this method must be the same as retrieved from the [Checksum::print](#) method.

**Parameters:**

*buf* string containing textual representation of checksum

See also:

[Checksum::print](#)

Implements [Arc::Checksum](#).

#### **5.30.2.8** `virtual void Arc::CRC32Sum::start (void)` [virtual]

Initiate the checksum algorithm.

This method must be called before starting a new checksum calculation.

Implements [Arc::Checksum](#).

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

- CheckSum.h



## 5.31 Arc::Credential Class Reference

```
#include <Credential.h>
```

### Public Member Functions

- [Credential](#) ()
- [Credential](#) (int keybits)
- [Credential](#) (const std::string &CAfile, const std::string &CAkey, const std::string &CAserial, const std::string &extfile, const std::string &extsect, const std::string &passphrase4key)
- [Credential](#) (Time start, Period lifetime=Period("PT12H"), int keybits=1024, std::string proxyversion="rfc", std::string policylang="inheritAll", std::string policy="", int pathlength=-1)
- [Credential](#) (const std::string &cert, const std::string &key, const std::string &cadir, const std::string &cafile, const std::string &passphrase4key="", const bool is\_file=true)
- [Credential](#) (const [UserConfig](#) &usercfg, const std::string &passphrase4key="")
- void [AddCertExtObj](#) (std::string &sn, std::string &oid)
- void [LogError](#) (void) const
- bool [GetVerification](#) (void) const
- EVP\_PKEY \* [GetPrivKey](#) (void) const
- EVP\_PKEY \* [GetPubKey](#) (void) const
- X509 \* [GetCert](#) (void) const
- X509\_REQ \* [GetCertReq](#) (void) const
- STACK\_OF (X509) \* [GetCertChain](#) (void) const
- int [GetCertNumofChain](#) (void) const
- Credformat [getFormat\\_BIO](#) (BIO \*in, const bool is\_file=true) const
- std::string [GetDN](#) (void) const
- std::string [GetIdentityName](#) (void) const
- [ArcCredential::certType](#) [GetType](#) (void) const
- std::string [GetIssuerName](#) (void) const
- std::string [GetCAName](#) (void) const
- std::string [GetProxyPolicy](#) (void) const
- void [SetProxyPolicy](#) (const std::string &proxyversion, const std::string &policylang, const std::string &policy, int pathlength)
- bool [OutputPrivatekey](#) (std::string &content, bool encryption=false, const std::string &passphrase="")
- bool [OutputPublickey](#) (std::string &content)
- bool [OutputCertificate](#) (std::string &content, bool is\_der=false)
- bool [OutputCertificateChain](#) (std::string &content, bool is\_der=false)
- Period [GetLifeTime](#) (void) const
- Time [GetStartTime](#) () const
- Time [GetEndTime](#) () const
- void [SetLifeTime](#) (const Period &period)
- void [SetStartTime](#) (const Time &start\_time)
- bool [IsValid](#) (void)
- bool [AddExtension](#) (const std::string &name, const std::string &data, bool crit=false)
- bool [AddExtension](#) (const std::string &name, char \*\*binary)
- std::string [GetExtension](#) (const std::string &name)
- bool [GenerateEECRequest](#) (BIO \*reqbio, BIO \*keybio, const std::string &dn="")
- bool [GenerateEECRequest](#) (std::string &reqcontent, std::string &keycontent, const std::string &dn="")

- bool [GenerateEECRequest](#) (const char \*request\_filename, const char \*key\_filename, const std::string &dn="")
- bool [GenerateRequest](#) (BIO \*bio, bool if\_der=false)
- bool [GenerateRequest](#) (std::string &content, bool if\_der=false)
- bool [GenerateRequest](#) (const char \*filename, bool if\_der=false)
- bool [InquireRequest](#) (BIO \*reqbio, bool if\_eec=false, bool if\_der=false)
- bool [InquireRequest](#) (std::string &content, bool if\_eec=false, bool if\_der=false)
- bool [InquireRequest](#) (const char \*filename, bool if\_eec=false, bool if\_der=false)
- bool [SignRequest](#) ([Credential](#) \*proxy, BIO \*outputbio, bool if\_der=false)
- bool [SignRequest](#) ([Credential](#) \*proxy, std::string &content, bool if\_der=false)
- bool [SignRequest](#) ([Credential](#) \*proxy, const char \*filename, bool foamat=false)
- bool [SelfSignEECRequest](#) (const std::string &dn, const char \*extfile, const std::string &extsect, const char \*certfile)
- bool [SignEECRequest](#) ([Credential](#) \*eec, const std::string &dn, BIO \*outputbio)
- bool [SignEECRequest](#) ([Credential](#) \*eec, const std::string &dn, std::string &content)
- bool [SignEECRequest](#) ([Credential](#) \*eec, const std::string &dn, const char \*filename)

## Static Public Member Functions

- static void [InitProxyCertInfo](#) (void)
- static bool [IsCredentialsValid](#) (const [UserConfig](#) &usercfg)

### 5.31.1 Detailed Description

[Credential](#) class covers the functionality about general processing about certificate/key files, including:

1. certificate/key parsing, information extracting (such as subject name, issuer name, lifetime, etc.), chain verifying, extension processing about proxy certinfo, extension processing about other general certificate extension (such as voms attributes, it should be the extension-specific code itself to create, parse and verify the extension, not the [Credential](#) class. For voms, it is some code about writing and parsing voms-implementing Attribute Certificate/ RFC3281, the voms-attribute is then be looked as a binary part and embeded into extension of X509 certificate/proxy certificate);
2. certificate request, extension emeding and certificate signing, for both proxy certificate and EEC (end entity certificate) certificate The [Credential](#) class support PEM, DER PKCS12 credential.

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 [Arc::Credential::Credential \(\)](#)

Default constructor, only acts as a container for inquiring certificate request, is meaningless for any other use.

#### 5.31.2.2 [Arc::Credential::Credential \(int keybits\)](#)

Constructor with user-defined keylength. Needed for creation of EE certs, since some applications will only support keys with a certain minimum length > 1024

### 5.31.2.3 Arc::Credential::Credential (const std::string & *CAfile*, const std::string & *CAkey*, const std::string & *CAserial*, const std::string & *extfile*, const std::string & *extsect*, const std::string & *passphrase4key*)

Constructor, specific constructor for CA certificate is meaningless for any other use.

### 5.31.2.4 Arc::Credential::Credential (**Time** *start*, Period *lifetime* = Period("PT12H"), int *keybits* = 1024, std::string *proxyversion* = "rfc", std::string *policylang* = "inheritAll", std::string *policy* = "", int *pathlength* = -1)

Constructor, specific constructor for proxy certificate, only acts as a container for constraining certificate signing and/or generating certificate request(only keybits is useful for creating certificate request), is meaningless for any other use. The proxyversion and policylang is for specifying the proxy certificate type and the policy language inside proxy. The definition of proxyversion and policy language is based on [http://dev.globus.org/wiki/Security/ProxyCertTypes#RFC\\_3820\\_Proxy\\_Certificates](http://dev.globus.org/wiki/Security/ProxyCertTypes#RFC_3820_Proxy_Certificates) The code is supposed to support proxy version: GSI2(legacy proxy), GSI3(Proxy draft) and RFC(RFC3820 proxy), and corresponding policy language. GSI2(GSI2, GSI2\_LIMITED) GSI3 and RFC (IMPERSONATION\_PROXY-1.3.6.1.5.5.7.21.1, INDEPENDENT\_PROXY-1.3.6.1.5.5.7.21.2, LIMITED\_PROXY-1.3.6.1.4.1.3536.1.1.1.9, RESTRICTED\_PROXY-policy language undefined) In openssl>=0.9.8, there are three types of policy languages: id-ppl-inheritAll-1.3.6.1.5.5.7.21.1, id-ppl-independent-1.3.6.1.5.5.7.21.2, and id-ppl-anyLanguage-1.3.6.1.5.5.7.21.0

#### Parameters:

*start, start* time of proxy certificate

*lifetime, lifetime* of proxy certificate

*keybits, modulus* size for RSA key generation, it should be greater than 1024 if 'this' class is used for generating X509 request; it should be '0' if 'this' class is used for constraining certificate signing.

### 5.31.2.5 Arc::Credential::Credential (const std::string & *cert*, const std::string & *key*, const std::string & *cadir*, const std::string & *cafile*, const std::string & *passphrase4key* = "", const bool *is\_file* = true)

Constructor, specific constructor for usual certificate, constructing from credential files. only acts as a container for parsing the certificate and key files, is meaningless for any other use. this constructor will parse the credential information, and put them into "this" object

#### Parameters:

*passphrase4key, specifies* the password for decrypting private key (if needed). If value is empty then password will be asked interactively. To avoid asking for password use value provided by No-Password() method.

*is\_file, specifies* if the cert/key are from file, otherwise they are supposed to be from string. default is from file

### 5.31.2.6 Arc::Credential::Credential (const **UserConfig** & *usercfg*, const std::string & *passphrase4key* = "")

Constructor, specific constructor for usual certificate, constructing from information in **UserConfig** object. Only acts as a container for parsing the certificate and key files, is meaningless for any other use. this constructor will parse the credential \* information, and put them into "this" object

**Parameters:**

*is\_file,specify* if the cert/key are from file, otherwise they are supposed to be from string. default is from file

**5.31.3 Member Function Documentation****5.31.3.1 void Arc::Credential::AddCertExtObj (std::string & *sn*, std::string & *oid*)**

General method for adding a new nid into openssl's global const

**5.31.3.2 bool Arc::Credential::AddExtension (const std::string & *name*, char \*\* *binary*)**

Add an extension to the extension part of the certificate

**Parameters:**

*binary,the* data which will be inserted into certificate extension part as a specific extension there should be specific methods defined inside specific X509V3\_EXT\_METHOD structure to parse the specific extension format. For example, VOMS attribute certificate is a specific extension to proxy certificate. There is specific X509V3\_EXT\_METHOD defined in [VOMSAttribute.h](#) and VOMSAttribute.c for parsing attribute certificate. In openssl, the specific X509V3\_EXT\_METHOD can be got according to the extension name/id, see X509V3\_EXT\_get\_nid(ext\_nid)

**5.31.3.3 bool Arc::Credential::AddExtension (const std::string & *name*, const std::string & *data*, bool *crit* = false)**

Add an extension to the extension part of the certificate

**Parameters:**

*name,the* name of the extension, there OID related with the name should be registered into openssl firstly

*data,the* data which will be inserted into certificate extension

**5.31.3.4 bool Arc::Credential::GenerateEECRequest (const char \* *request\_filename*, const char \* *key\_filename*, const std::string & *dn* = "")**

Generate an EEC request, output the certificate request and the key to a file

**5.31.3.5 bool Arc::Credential::GenerateEECRequest (std::string & *reqcontent*, std::string & *keycontent*, const std::string & *dn* = "")**

Generate an EEC request, output the certificate request to a string

**5.31.3.6 bool Arc::Credential::GenerateEECRequest (BIO \* *reqbio*, BIO \* *keybio*, const std::string & *dn* = "")**

Generate an EEC request, based on the keybits and signing algorithm information inside this object output the certificate request to output BIO

The user will be asked for a private key password

#### 5.31.3.7 **bool Arc::Credential::GenerateRequest (const char \* *filename*, bool *if\_der* = false)**

Generate a proxy request, output the certificate request to a file

#### 5.31.3.8 **bool Arc::Credential::GenerateRequest (std::string & *content*, bool *if\_der* = false)**

Generate a proxy request, output the certificate request to a string

#### 5.31.3.9 **bool Arc::Credential::GenerateRequest (BIO \* *bio*, bool *if\_der* = false)**

Generate a proxy request, base on the keybits and signing algorithm information inside this object output the certificate request to output BIO

#### 5.31.3.10 **std::string Arc::Credential::GetCAName (void) const**

Get CA of the certificate attached to this object, if the certificate is an EEC, GetCAName get the same value as GetIssuerName

#### 5.31.3.11 **X509\* Arc::Credential::GetCert (void) const**

Get the certificate attached to this object

#### 5.31.3.12 **int Arc::Credential::GetCertNumofChain (void) const**

Get the number of certificates in the certificate chain attached to this object

#### 5.31.3.13 **X509\_REQ\* Arc::Credential::GetCertReq (void) const**

Get the certificate request, if there is any

#### 5.31.3.14 **std::string Arc::Credential::GetDN (void) const**

Get the DN of the certificate attached to this object

#### 5.31.3.15 **Time Arc::Credential::GetEndTime () const**

Returns validity end time of certificate or proxy

#### 5.31.3.16 **std::string Arc::Credential::GetExtension (const std::string & *name*)**

Get the specific extension (named by the parameter) in a certificate this function is only supposed to be called after certificate and key are loaded by the constructor for usual certificate

**Parameters:**

*name,the* name of the extension to get

**5.31.3.17 Credformat Arc::Credential::getFormat\_BIO (BIO \* *in*, const bool *is\_file* = true) const**

Get the certificate format, PEM PKCS12 or DER BIO could be memory or file, they should be processed differently.

**5.31.3.18 std::string Arc::Credential::GetIdentityName (void) const**

Get the Identity name of the certificate attached to this object, the result will not include proxy CN

**5.31.3.19 std::string Arc::Credential::GetIssuerName (void) const**

Get issuer of the certificate attached to this object

**5.31.3.20 Period Arc::Credential::GetLifeTime (void) const**

Returns lifetime of certificate or proxy

**5.31.3.21 EVP\_PKEY\* Arc::Credential::GetPrivKey (void) const**

Get the private key attached to this object

**5.31.3.22 std::string Arc::Credential::GetProxyPolicy (void) const**

Get the proxy policy attached to the "proxy certificate information" extension of the proxy certificate

**5.31.3.23 EVP\_PKEY\* Arc::Credential::GetPubKey (void) const**

Get the public key attached to this object

**5.31.3.24 Time Arc::Credential::GetStartTime () const**

Returns validity start time of certificate or proxy

**5.31.3.25 ArcCredential::certType Arc::Credential::GetType (void) const**

Get type of the certificate attached to this object

**5.31.3.26 bool Arc::Credential::GetVerification (void) const [inline]**

Get the verification result about certificate chain checking

**5.31.3.27 static void Arc::Credential::InitProxyCertInfo (void) [static]**

Initiate nid for proxy certificate extension

**5.31.3.28 bool Arc::Credential::InquireRequest (const char \*filename, bool if\_eec = false, bool if\_der = false)**

Inquire the certificate request from a file

**5.31.3.29 bool Arc::Credential::InquireRequest (std::string &content, bool if\_eec = false, bool if\_der = false)**

Inquire the certificate request from a string

**5.31.3.30 bool Arc::Credential::InquireRequest (BIO \*reqbio, bool if\_eec = false, bool if\_der = false)**

Inquire the certificate request from BIO, and put the request information to X509\_REQ inside this object, and parse the certificate type from the PROXYCERTINFO of request' extension

**Parameters:**

*if\_der* false for PEM; true for DER

**5.31.3.31 static bool Arc::Credential::IsCredentialsValid (const UserConfig &usercfg) [static]**

Returns true if credentials are valid. Credentials are read from locations specified in UserConfig object. This method is deprecated. User per-instance method IsValid() instead.

**5.31.3.32 bool Arc::Credential::IsValid (void)**

Returns true if credentials are valid

**5.31.3.33 void Arc::Credential::LogError (void) const**

Log error information related with openssl

**5.31.3.34 bool Arc::Credential::OutputCertificate (std::string &content, bool is\_der = false)**

Output the certificate into string

**Parameters:**

*is\_der* false for PEM, true for DER

**5.31.3.35** `bool Arc::Credential::OutputCertificateChain (std::string & content, bool is_der = false)`

Output the certificate chain into string

**Parameters:**

*is\_der* false for PEM, true for DER

**5.31.3.36** `bool Arc::Credential::OutputPrivatekey (std::string & content, bool encryption = false, const std::string & passphrase = "")`

Output the private key into string

**Parameters:**

*encryption,whether* encrypt the output private key or not

*passphrase,the* passphrase to encrypt the output private key

**5.31.3.37** `bool Arc::Credential::OutputPublickey (std::string & content)`

Output the public key into string

**5.31.3.38** `bool Arc::Credential::SelfSignEECRequest (const std::string & dn, const char * extfile, const std::string & extsect, const char * certfile)`

Self sign a certificate. This functionality is specific for creating a CA credential by using this [Credential](#) class.

**Parameters:**

*dn* the DN for the subject

*extfile* the configuration file which includes the extension information, typically the openssl.cnf file

*extsect* the section/group name for the extension, e.g. in openssl.cnf, usr\_cert and v3\_ca

*certfile* the certificate file, which contains the signed certificate

**5.31.3.39** `void Arc::Credential::SetLifeTime (const Period & period)`

Set lifetime of certificate or proxy

**5.31.3.40** `void Arc::Credential::SetProxyPolicy (const std::string & proxyversion, const std::string & policylang, const std::string & policy, int pathlength)`

Set the proxy policy attached to the "proxy certificate information" extension of the proxy certificate

**5.31.3.41** `void Arc::Credential::SetStartTime (const Time & start_time)`

Set start time of certificate or proxy



**5.31.3.42** `bool Arc::Credential::SignEECRequest (Credential * eec, const std::string & dn, const char * filename)`

Sign request and output the signed certificate to a file

**5.31.3.43** `bool Arc::Credential::SignEECRequest (Credential * eec, const std::string & dn, std::string & content)`

Sign request and output the signed certificate to a string

**5.31.3.44** `bool Arc::Credential::SignEECRequest (Credential * eec, const std::string & dn, BIO * outputbio)`

Sign eec request, and output the signed certificate to output BIO

**5.31.3.45** `bool Arc::Credential::SignRequest (Credential * proxy, const char * filename, bool foamat = false)`

Sign request and output the signed certificate to a file

**Parameters:**

*if\_der* false for PEM, true for DER

**5.31.3.46** `bool Arc::Credential::SignRequest (Credential * proxy, std::string & content, bool if_der = false)`

Sign request and output the signed certificate to a string

**Parameters:**

*if\_der* false for PEM, true for DER

**5.31.3.47** `bool Arc::Credential::SignRequest (Credential * proxy, BIO * outputbio, bool if_der = false)`

Sign request based on the information inside proxy, and output the signed certificate to output BIO

**Parameters:**

*if\_der* false for PEM, true for DER

**5.31.3.48** `Arc::Credential::STACK_OF (X509) const`

Get the certificate chain attached to this object

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

- Credential.h

## 5.32 Arc::CredentialError Class Reference

```
#include <Credential.h>
```

### Public Member Functions

- [CredentialError](#) (const std::string &what="")

#### 5.32.1 Detailed Description

This is an exception class that is used to handle runtime errors discovered in the [Credential](#) class.

#### 5.32.2 Constructor & Destructor Documentation

##### 5.32.2.1 Arc::CredentialError::CredentialError (const std::string & *what* = "")

This is the constructor of the [CredentialError](#) class.

#### Parameters:

- what* An explanation of the error.

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

- Credential.h

## 5.33 Arc::CredentialStore Class Reference

```
#include <CredentialStore.h>
```

### 5.33.1 Detailed Description

This class provides functionality for storing delegated credentials and retrieving them from some store services. This is very preliminary implementation and currently support only one type of credentials - X.509 proxies, and only one type of store service - MyProxy. Later it will be extended to support at least following services: ARC delegation service, VOMS service, local file system.

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

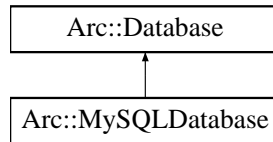
- CredentialStore.h

## 5.34 Arc::Database Class Reference

Interface for calling database client library.

```
#include <DBInterface.h>
```

Inheritance diagram for Arc::Database::



### Public Member Functions

- [Database](#) ()
- [Database](#) (std::string &server, int port)
- [Database](#) (const [Database](#) &other)
- virtual [~Database](#) ()
- virtual bool [connect](#) (std::string &dbname, std::string &user, std::string &password)=0
- virtual bool [isconnected](#) () const =0
- virtual void [close](#) ()=0
- virtual bool [enable\\_ssl](#) (const std::string &keyfile="", const std::string &certfile="", const std::string &cafile="", const std::string &capath="")=0
- virtual bool [shutdown](#) ()=0

### 5.34.1 Detailed Description

Interface for calling database client library.

For different types of database client library, different classes should be implemented by implementing this interface.

### 5.34.2 Constructor & Destructor Documentation

#### 5.34.2.1 Arc::Database::Database () [inline]

Default constructor

#### 5.34.2.2 Arc::Database::Database (std::string & *server*, int *port*) [inline]

Constructor which uses the server's name(or IP address) and port as parametes

#### 5.34.2.3 Arc::Database::Database (const [Database](#) & *other*) [inline]

Copy constructor

#### 5.34.2.4 virtual Arc::Database::~Database () [inline, virtual]

Deconstructor

### 5.34.3 Member Function Documentation

#### 5.34.3.1 virtual void Arc::Database::close () [pure virtual]

Close the connection with database server

Implemented in [Arc::MySQLDatabase](#).

#### 5.34.3.2 virtual bool Arc::Database::connect (std::string & *dbname*, std::string & *user*, std::string & *password*) [pure virtual]

Do connection with database server

##### Parameters:

*dbname* The database name which will be used.

*user* The username which will be used to access database.

*password* The password which will be used to access database.

Implemented in [Arc::MySQLDatabase](#).

#### 5.34.3.3 virtual bool Arc::Database::enable\_ssl (const std::string & *keyfile* = "", const std::string & *certfile* = "", const std::string & *cafile* = "", const std::string & *capath* = "") [pure virtual]

Enable ssl communication for the connection

##### Parameters:

*keyfile* The location of key file.

*certfile* The location of certificate file.

*cafile* The location of ca file.

*capath* The location of ca directory

Implemented in [Arc::MySQLDatabase](#).

#### 5.34.3.4 virtual bool Arc::Database::isconnected () const [pure virtual]

Get the connection status

Implemented in [Arc::MySQLDatabase](#).

#### 5.34.3.5 virtual bool Arc::Database::shutdown () [pure virtual]

Ask database server to shutdown

Implemented in [Arc::MySQLDatabase](#).

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

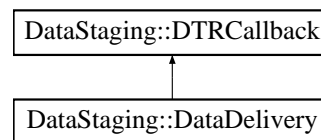
- DBInterface.h

## 5.35 DataStaging::DataDelivery Class Reference

[DataDelivery](#) transfers data between specified physical locations.

```
#include <DataDelivery.h>
```

Inheritance diagram for DataStaging::DataDelivery::



### Public Member Functions

- [DataDelivery](#) ()
- [~DataDelivery](#) ()
- virtual void [receiveDTR](#) ([DTR\\_ptr](#) request)
- bool [cancelDTR](#) ([DTR\\_ptr](#) request)
- bool [start](#) ()
- bool [stop](#) ()
- void [SetTransferParameters](#) (const [TransferParameters](#) &params)

### 5.35.1 Detailed Description

[DataDelivery](#) transfers data between specified physical locations.

All meta-operations for a [DTR](#) such as resolving replicas must be done before sending to [DataDelivery](#). Calling [receiveDTR\(\)](#) starts a new process which performs data transfer as specified in [DTR](#).

### 5.35.2 Constructor & Destructor Documentation

#### 5.35.2.1 DataStaging::DataDelivery::DataDelivery ()

Constructor.

#### 5.35.2.2 DataStaging::DataDelivery::~~DataDelivery () [inline]

Destructor calls [stop\(\)](#) and waits for cancelled processes to exit.

### 5.35.3 Member Function Documentation

#### 5.35.3.1 bool DataStaging::DataDelivery::cancelDTR ([DTR\\_ptr](#) request)

Kill the process corresponding to the given [DTR](#).

**5.35.3.2 virtual void DataStaging::DataDelivery::receiveDTR ([DTR\\_ptr request](#))** [virtual]

Pass a [DTR](#) to Delivery.

This method is called by the scheduler to pass a [DTR](#) to the delivery. The [DataDelivery](#) starts a process to do the processing, and then returns. [DataDelivery](#)'s own thread then monitors the started process.

Implements [DataStaging::DTRCallback](#).

**5.35.3.3 void DataStaging::DataDelivery::SetTransferParameters (const [TransferParameters](#) & *params*)**

Set transfer limits.

**5.35.3.4 bool DataStaging::DataDelivery::start ()**

Start the Delivery thread, which runs until [stop\(\)](#) is called.

**5.35.3.5 bool DataStaging::DataDelivery::stop ()**

Tell the delivery to shut down all processes and threads and exit.

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

- [DataDelivery.h](#)

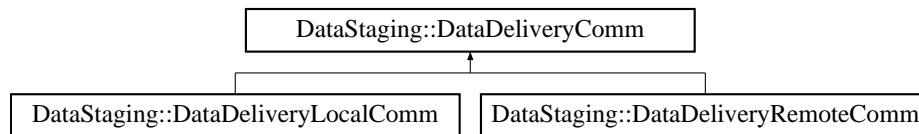


## 5.36 DataStaging::DataDeliveryComm Class Reference

This class provides an abstract interface for the Delivery layer.

```
#include <DataDeliveryComm.h>
```

Inheritance diagram for DataStaging::DataDeliveryComm::



### Public Types

- [CommInit](#)
- [CommNoError](#)
- [CommTimeout](#)
- [CommClosed](#)
- [CommExited](#)
- [CommFailed](#)
- enum [CommStatusType](#) {  
[CommInit](#), [CommNoError](#), [CommTimeout](#), [CommClosed](#),  
[CommExited](#), [CommFailed](#) }

### Public Member Functions

- virtual [~DataDeliveryComm](#) ()
- [Status](#) [GetStatus](#) () const
- std::string [GetError](#) () const
- virtual [operator bool](#) () const =0
- virtual bool [operator!](#) () const =0

### Static Public Member Functions

- static [DataDeliveryComm](#) \* [CreateInstance](#) ([DTR\\_ptr](#) dtr, const [TransferParameters](#) &params)
- static bool [CheckComm](#) ([DTR\\_ptr](#) dtr, std::vector< std::string > &allowed\_dirs)

### Protected Member Functions

- virtual void [PullStatus](#) ()=0
- [DataDeliveryComm](#) ([DTR\\_ptr](#) dtr, const [TransferParameters](#) &params)

## Protected Attributes

- [Status](#) `status_`
- [Status](#) `status_buf_`
- unsigned int `status_pos_`
- [Glib::Mutex](#) `lock_`
- [DataDeliveryCommHandler](#) \* `handler_`
- `std::string` `dtr_id`
- [TransferParameters](#) `transfer_params`
- [Arc::Time](#) `start_`
- [DTRLogger](#) `logger_`

## Data Structures

- struct [Status](#)

*Plain C struct to pass information from executing process back to main thread.*

### 5.36.1 Detailed Description

This class provides an abstract interface for the Delivery layer.

Different implementations provide different ways of providing Delivery functionality. [DataDeliveryLocalComm](#) launches a local process to perform the transfer and [DataDeliveryRemoteComm](#) contacts a remote service which performs the transfer. The implementation is chosen depending on what is set in the [DTR](#), which the [Scheduler](#) should set based on various factors.

[CreateInstance\(\)](#) should be used to get a pointer to the instantiated object. This also starts the transfer. Deleting this object stops the transfer and cleans up any used resources. A singleton instance of [DataDeliveryCommHandler](#) regularly polls all active transfers using [PullStatus\(\)](#) and fills the [Status](#) object with current information, which can be obtained through [GetStatus\(\)](#).

### 5.36.2 Member Enumeration Documentation

#### 5.36.2.1 enum [DataStaging::DataDeliveryComm::CommStatusType](#)

Communication status with transfer.

#### Enumerator:

***CommInit*** Initializing/starting transfer, rest of information not valid.

***CommNoError*** Communication going on smoothly.

***CommTimeout*** Communication experienced timeout.

***CommClosed*** Communication channel was closed.

***CommExited*** Transfer exited. Mostly same as *CommClosed* but exit detected before pipe closed.

***CommFailed*** Transfer failed. If we have *CommFailed* and no error code reported that normally means segfault or external kill.

### 5.36.3 Constructor & Destructor Documentation

#### 5.36.3.1 DataStaging::DataDeliveryComm::DataDeliveryComm ([DTR\\_ptr](#) *dtr*, const [TransferParameters](#) & *params*) [protected]

Start transfer with parameters taken from [DTR](#) and supplied transfer limits.

Constructor should not be used directly, [CreateInstance\(\)](#) should be used instead.

#### 5.36.3.2 virtual DataStaging::DataDeliveryComm::~~DataDeliveryComm () [inline, virtual]

Destroy object. This stops any ongoing transfer and cleans up resources.

### 5.36.4 Member Function Documentation

#### 5.36.4.1 static bool DataStaging::DataDeliveryComm::CheckComm ([DTR\\_ptr](#) *dtr*, std::vector< std::string > & *allowed\_dirs*) [static]

Check the delivery is available. Calls CheckComm of the appropriate subclass.

##### Parameters:

*dtr* [DTR](#) from which credentials are used

*allowed\_dirs* List of dirs that this comm is allowed to read/write

Reimplemented in [DataStaging::DataDeliveryLocalComm](#), and [DataStaging::DataDeliveryRemoteComm](#).

#### 5.36.4.2 static [DataDeliveryComm\\*](#) DataStaging::DataDeliveryComm::CreateInstance ([DTR\\_ptr](#) *dtr*, const [TransferParameters](#) & *params*) [static]

Factory method to get concrete instance.

#### 5.36.4.3 std::string DataStaging::DataDeliveryComm::GetError (void) const [inline]

Get explanation of error.

#### 5.36.4.4 [Status](#) DataStaging::DataDeliveryComm::GetStatus () const

Obtain status of transfer.

#### 5.36.4.5 virtual DataStaging::DataDeliveryComm::operator bool () const [pure virtual]

Returns true if transfer is currently active.

Implemented in [DataStaging::DataDeliveryLocalComm](#), and [DataStaging::DataDeliveryRemoteComm](#).

**5.36.4.6** `virtual bool DataStaging::DataDeliveryComm::operator! () const` [pure virtual]

Returns true if transfer is currently not active.

Implemented in [DataStaging::DataDeliveryLocalComm](#), and [DataStaging::DataDeliveryRemoteComm](#).

**5.36.4.7** `virtual void DataStaging::DataDeliveryComm::PullStatus ()` [protected, pure virtual]

Check for new state and fill state accordingly.

This method is periodically called by the comm handler to obtain status info. It detects communication and delivery failures and delivery termination.

Implemented in [DataStaging::DataDeliveryLocalComm](#), and [DataStaging::DataDeliveryRemoteComm](#).

**5.36.5 Field Documentation****5.36.5.1** `std::string DataStaging::DataDeliveryComm::dtr_id` [protected]

ID of the [DTR](#) this object is handling. Used in log messages.

**5.36.5.2** `DataDeliveryCommHandler* DataStaging::DataDeliveryComm::handler_` [protected]

Pointer to singleton handler of all [DataDeliveryComm](#) objects.

**5.36.5.3** `Glib::Mutex DataStaging::DataDeliveryComm::lock_` [protected]

Lock to protect access to status.

**5.36.5.4** `DTRLogger DataStaging::DataDeliveryComm::logger_` [protected]

Logger object. Pointer to DTR's Logger.

**5.36.5.5** `Arc::Time DataStaging::DataDeliveryComm::start_` [protected]

Time transfer was started.

**5.36.5.6** `Status DataStaging::DataDeliveryComm::status_` [protected]

Current status of transfer.

**5.36.5.7** `Status DataStaging::DataDeliveryComm::status_buf_` [protected]

Latest status of transfer is read into this buffer.

**5.36.5.8** unsigned int [DataStaging::DataDeliveryComm::status\\_pos\\_](#) [protected]

Reading position of [Status](#) buffer.

**5.36.5.9** [TransferParameters](#) [DataStaging::DataDeliveryComm::transfer\\_params](#)  
[protected]

Transfer limits.

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

- [DataDeliveryComm.h](#)

## 5.37 DataStaging::DataDeliveryComm::Status Struct Reference

Plain C struct to pass information from executing process back to main thread.

```
#include <DataDeliveryComm.h>
```

### Data Fields

- [CommStatusType](#) `commstatus`
- `time_t` `timestamp_s`
- `time_t` `timestamp_ns`
- [DTRStatus::DTRStatusType](#) `status`
- [DTRErrorStatus::DTRErrorStatusType](#) `error`
- [DTRErrorStatus::DTRErrorLocation](#) `error_location`
- `char` `error_desc` [256]
- `unsigned int` `streams`
- `unsigned long long int` `transferred`
- `unsigned long long int` `offset`
- `unsigned long long int` `size`
- `unsigned int` `speed`
- `char` `checksum` [128]

### 5.37.1 Detailed Description

Plain C struct to pass information from executing process back to main thread.

### 5.37.2 Field Documentation

#### 5.37.2.1 `char DataStaging::DataDeliveryComm::Status::checksum`[128]

Calculated checksum.

#### 5.37.2.2 `CommStatusType DataStaging::DataDeliveryComm::Status::commstatus`

Communication state (filled by main thread).

#### 5.37.2.3 `DTRErrorStatus::DTRErrorStatusType DataStaging::DataDeliveryComm::Status::error`

Error type.

#### 5.37.2.4 `char DataStaging::DataDeliveryComm::Status::error_desc`[256]

Error description.

#### 5.37.2.5 `DTRErrorStatus::DTRErrorLocation DataStaging::DataDeliveryComm::Status::error_location`

Where error happened.

**5.37.2.6 unsigned long long int DataStaging::DataDeliveryComm::Status::offset**

Last position to which file has no missing pieces.

**5.37.2.7 unsigned long long int DataStaging::DataDeliveryComm::Status::size**

File size as obtained by protocol.

**5.37.2.8 unsigned int DataStaging::DataDeliveryComm::Status::speed**

Current transfer speed in bytes/sec during last ~minute.

**5.37.2.9 DTRStatus::DTRStatusType DataStaging::DataDeliveryComm::Status::status**

Generic status.

**5.37.2.10 unsigned int DataStaging::DataDeliveryComm::Status::streams**

Number of transfer streams active.

**5.37.2.11 time\_t DataStaging::DataDeliveryComm::Status::timestamp\_ns**

Nanosecond-precision in timestamp\_s.

**5.37.2.12 time\_t DataStaging::DataDeliveryComm::Status::timestamp\_s**

Time when information was generated (filled externally).

**5.37.2.13 unsigned long long int DataStaging::DataDeliveryComm::Status::transferred**

Number of bytes transferred.

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

- DataDeliveryComm.h

## 5.38 DataStaging::DataDeliveryCommHandler Class Reference

Singleton class handling all active [DataDeliveryComm](#) objects.

```
#include <DataDeliveryComm.h>
```

### Public Member Functions

- void [Add](#) ([DataDeliveryComm](#) \*item)
- void [Remove](#) ([DataDeliveryComm](#) \*item)

### Static Public Member Functions

- static [DataDeliveryCommHandler](#) \* [getInstance](#) ()

#### 5.38.1 Detailed Description

Singleton class handling all active [DataDeliveryComm](#) objects.

#### 5.38.2 Member Function Documentation

##### 5.38.2.1 void DataStaging::DataDeliveryCommHandler::Add ([DataDeliveryComm](#) \* *item*)

Add a new [DataDeliveryComm](#) instance to the handler.

##### 5.38.2.2 static [DataDeliveryCommHandler](#)\* DataStaging::DataDeliveryCommHandler::get-Instance () [static]

Get the singleton instance of the handler.

##### 5.38.2.3 void DataStaging::DataDeliveryCommHandler::Remove ([DataDeliveryComm](#) \* *item*)

Remove a [DataDeliveryComm](#) instance from the handler.

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

- [DataDeliveryComm.h](#)

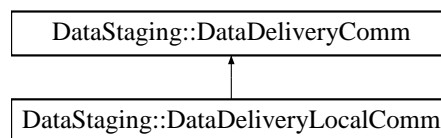


## 5.39 DataStaging::DataDeliveryLocalComm Class Reference

This class starts, monitors and controls a local Delivery process.

```
#include <DataDeliveryLocalComm.h>
```

Inheritance diagram for DataStaging::DataDeliveryLocalComm::



### Public Member Functions

- [DataDeliveryLocalComm](#) ([DTR\\_ptr](#) dtr, const [TransferParameters](#) &params)
- virtual [~DataDeliveryLocalComm](#) ()
- virtual void [PullStatus](#) ()
- virtual [operator bool](#) () const
- virtual bool [operator!](#) () const

### Static Public Member Functions

- static bool [CheckComm](#) ([DTR\\_ptr](#) dtr, std::vector< std::string > &allowed\_dirs)

### 5.39.1 Detailed Description

This class starts, monitors and controls a local Delivery process.

### 5.39.2 Constructor & Destructor Documentation

#### 5.39.2.1 DataStaging::DataDeliveryLocalComm::DataDeliveryLocalComm ([DTR\\_ptr](#) dtr, const [TransferParameters](#) &params)

Starts child process.

#### 5.39.2.2 virtual DataStaging::DataDeliveryLocalComm::~~DataDeliveryLocalComm () [virtual]

This stops the child process.

### 5.39.3 Member Function Documentation

#### 5.39.3.1 static bool DataStaging::DataDeliveryLocalComm::CheckComm ([DTR\\_ptr](#) dtr, std::vector< std::string > &allowed\_dirs) [static]

Returns "/" since local Delivery can access everywhere.

Reimplemented from [DataStaging::DataDeliveryComm](#).

**5.39.3.2** `virtual DataStaging::DataDeliveryLocalComm::operator bool (void) const` `[inline, virtual]`

Returns true if child process exists.

Implements [DataStaging::DataDeliveryComm](#).

**5.39.3.3** `virtual bool DataStaging::DataDeliveryLocalComm::operator! (void) const` `[inline, virtual]`

Returns true if child process does not exist.

Implements [DataStaging::DataDeliveryComm](#).

**5.39.3.4** `virtual void DataStaging::DataDeliveryLocalComm::PullStatus ()` `[virtual]`

Read from stdout of child to get status.

Implements [DataStaging::DataDeliveryComm](#).

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

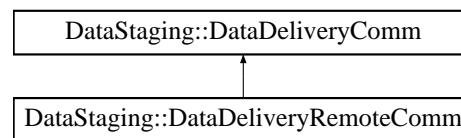
- `DataDeliveryLocalComm.h`

## 5.40 DataStaging::DataDeliveryRemoteComm Class Reference

This class contacts a remote service to make a Delivery request.

```
#include <DataDeliveryRemoteComm.h>
```

Inheritance diagram for DataStaging::DataDeliveryRemoteComm::



### Public Member Functions

- virtual void [PullStatus](#) ()
- virtual [operator bool](#) () const
- virtual bool [operator!](#) () const

### Static Public Member Functions

- static bool [CheckComm](#) ([DTR\\_ptr](#) dtr, std::vector< std::string > &allowed\_dirs)

#### 5.40.1 Detailed Description

This class contacts a remote service to make a Delivery request.

#### 5.40.2 Member Function Documentation

**5.40.2.1** static bool DataStaging::DataDeliveryRemoteComm::CheckComm ([DTR\\_ptr](#) dtr, std::vector< std::string > & *allowed\_dirs*) [static]

Pings service to find allowed dirs.

Reimplemented from [DataStaging::DataDeliveryComm](#).

**5.40.2.2** virtual DataStaging::DataDeliveryRemoteComm::operator bool (void) const [inline, virtual]

Returns true if service is still processing request.

Implements [DataStaging::DataDeliveryComm](#).

**5.40.2.3** virtual bool DataStaging::DataDeliveryRemoteComm::operator! (void) const [inline, virtual]

Returns true if service is not processing request or down.

Implements [DataStaging::DataDeliveryComm](#).

**5.40.2.4 virtual void DataStaging::DataDeliveryRemoteComm::PullStatus ()** [virtual]

Read status from service.

Implements [DataStaging::DataDeliveryComm](#).

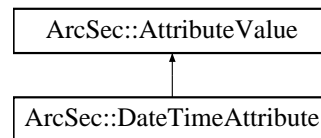
The documentation for this class was generated from the following file:

- DataDeliveryRemoteComm.h

## 5.41 ArcSec::DateTimeAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::DateTimeAttribute::



### Public Member Functions

- virtual bool [equal](#) ([AttributeValue](#) \*other, bool check\_id=true)
- virtual std::string [encode](#) ()
- virtual std::string [getType](#) ()
- virtual std::string [getId](#) ()

#### 5.41.1 Detailed Description

Format: YYYYMMDDHHMMSSZ Day Month DD HH:MM:SS YYYY YYYY-MM-DD HH:MM:SS  
 YYYY-MM-DDTHH:MM:SS+HH:MM YYYY-MM-DDTHH:MM:SSZ

#### 5.41.2 Member Function Documentation

##### 5.41.2.1 virtual std::string ArcSec::DateTimeAttribute::encode () [virtual]

encode the value in a string format

Implements [ArcSec::AttributeValue](#).

##### 5.41.2.2 virtual bool ArcSec::DateTimeAttribute::equal ([AttributeValue](#) \* other, bool check\_id = true) [virtual]

Evaluate whether "this" equals to the parameter value

Implements [ArcSec::AttributeValue](#).

##### 5.41.2.3 virtual std::string ArcSec::DateTimeAttribute::getId () [inline, virtual]

Get the AttributeId of the <Attribute>

Implements [ArcSec::AttributeValue](#).

##### 5.41.2.4 virtual std::string ArcSec::DateTimeAttribute::getType () [inline, virtual]

Get the DataType of the <Attribute>

Implements [ArcSec::AttributeValue](#).

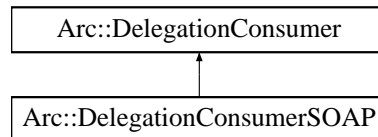
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

## 5.42 Arc::DelegationConsumer Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationConsumer::



### Public Member Functions

- [DelegationConsumer](#) (void)
- [DelegationConsumer](#) (const std::string &content)
- const std::string & [ID](#) (void)
- bool [Backup](#) (std::string &content)
- bool [Restore](#) (const std::string &content)
- bool [Request](#) (std::string &content)
- bool [Acquire](#) (std::string &content)
- bool [Acquire](#) (std::string &content, std::string &identity)

### Protected Member Functions

- bool [Generate](#) (void)
- void [LogError](#) (void)

#### 5.42.1 Detailed Description

A consumer of delegated X509 credentials. During delegation procedure this class acquires delegated credentials aka proxy - certificate, private key and chain of previous certificates. Delegation procedure consists of calling [Request\(\)](#) method for generating certificate request followed by call to [Acquire\(\)](#) method for making complete credentials from certificate chain.

#### 5.42.2 Constructor & Destructor Documentation

##### 5.42.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

##### 5.42.2.2 Arc::DelegationConsumer::DelegationConsumer (const std::string & *content*)

Creates object with provided private key

### 5.42.3 Member Function Documentation

#### 5.42.3.1 `bool Arc::DelegationConsumer::Acquire (std::string & content, std::string & identity)`

Includes the functionality of `Acquire(content)` plus extracting the credential identity.

#### 5.42.3.2 `bool Arc::DelegationConsumer::Acquire (std::string & content)`

Ads private key into certificates chain in '*content*' On exit *content* contains complete delegated credentials.

#### 5.42.3.3 `bool Arc::DelegationConsumer::Backup (std::string & content)`

Stores content of this object into a string

#### 5.42.3.4 `bool Arc::DelegationConsumer::Generate (void)` `[protected]`

Private key

#### 5.42.3.5 `const std::string& Arc::DelegationConsumer::ID (void)`

Return identifier of this object - not implemented

#### 5.42.3.6 `void Arc::DelegationConsumer::LogError (void)` `[protected]`

Creates private key

#### 5.42.3.7 `bool Arc::DelegationConsumer::Request (std::string & content)`

Make X509 certificate request from internal private key

#### 5.42.3.8 `bool Arc::DelegationConsumer::Restore (const std::string & content)`

Restores content of object from string

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

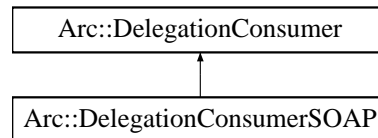
- `DelegationInterface.h`



## 5.43 Arc::DelegationConsumerSOAP Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationConsumerSOAP::



### Public Member Functions

- [DelegationConsumerSOAP](#) (void)
- [DelegationConsumerSOAP](#) (const std::string &content)
- bool [DelegateCredentialsInit](#) (const std::string &id, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool [UpdateCredentials](#) (std::string &credentials, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool [UpdateCredentials](#) (std::string &credentials, std::string &identity, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool [DelegatedToken](#) (std::string &credentials, [XMLNode](#) token)

#### 5.43.1 Detailed Description

This class extends [DelegationConsumer](#) to support SOAP message exchange. Implements WS interface <http://www.nordugrid.org/schemas/delegation> described in delegation.wsdl.

#### 5.43.2 Constructor & Destructor Documentation

##### 5.43.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

##### 5.43.2.2 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (const std::string &content)

Creates object with specified private key

#### 5.43.3 Member Function Documentation

##### 5.43.3.1 bool Arc::DelegationConsumerSOAP::DelegateCredentialsInit (const std::string &id, const SOAPEnvelope &in, SOAPEnvelope &out)

Process SOAP message which starts delegation. Generated message in 'out' is meant to be sent back to DelagationProviderSOAP. Argument 'id' contains identifier of procedure and is used only to produce SOAP message.

**5.43.3.2** `bool Arc::DelegationConsumerSOAP::DelegatedToken (std::string & credentials,  
XMLNode token)`

Similar to UpdateCredentials but takes only DelegatedToken XML element

**5.43.3.3** `bool Arc::DelegationConsumerSOAP::UpdateCredentials (std::string & credentials,  
std::string & identity, const SOAPEnvelope & in, SOAPEnvelope & out)`

Includes the functionality in above UpdateCredentials method; plus extracting the credential identity

**5.43.3.4** `bool Arc::DelegationConsumerSOAP::UpdateCredentials (std::string & credentials, const  
SOAPEnvelope & in, SOAPEnvelope & out)`

Accepts delegated credentials. Process 'in' SOAP message and stores full proxy credentials in 'credentials'. 'out' message is generated for sending to DelagationProviderSOAP.

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

- DelegationInterface.h

## 5.44 Arc::DelegationContainerSOAP Class Reference

```
#include <DelegationInterface.h>
```

### Public Member Functions

- bool [DelegatedToken](#) (std::string &credentials, [XMLNode](#) token, const std::string &client="")

### Protected Member Functions

- virtual [DelegationConsumerSOAP](#) \* [AddConsumer](#) (std::string &id, const std::string &client)
- virtual [DelegationConsumerSOAP](#) \* [FindConsumer](#) (const std::string &id, const std::string &client)
- virtual bool [TouchConsumer](#) ([DelegationConsumerSOAP](#) \*c, const std::string &credentials)
- virtual bool [QueryConsumer](#) ([DelegationConsumerSOAP](#) \*c, std::string &credentials)
- virtual void [ReleaseConsumer](#) ([DelegationConsumerSOAP](#) \*c)
- virtual void [RemoveConsumer](#) ([DelegationConsumerSOAP](#) \*c)
- virtual void [CheckConsumers](#) (void)
- bool [DelegateCredentialsInit](#) (const [SOAPEnvelope](#) &in, [SOAPEnvelope](#) &out, const std::string &client="")
- bool [UpdateCredentials](#) (std::string &credentials, const [SOAPEnvelope](#) &in, [SOAPEnvelope](#) &out, const std::string &client="")

### Protected Attributes

- int [max\\_size\\_](#)
- int [max\\_duration\\_](#)
- int [max\\_usage\\_](#)
- bool [context\\_lock\\_](#)

#### 5.44.1 Detailed Description

Manages multiple delegated credentials. Delegation consumers are created automatically with [DelegateCredentialsInit](#) method up to [max\\_size\\_](#) and assigned unique identifier. It's methods are similar to those of [DelegationConsumerSOAP](#) with identifier included in SOAP message used to route execution to one of managed [DelegationConsumerSOAP](#) instances.

#### 5.44.2 Member Function Documentation

**5.44.2.1** virtual [DelegationConsumerSOAP](#)\* [Arc::DelegationContainerSOAP::AddConsumer](#) (std::string & *id*, const std::string & *client*) [protected, virtual]

Creates new consumer object, if empty assigns id and stores in intenal store

**5.44.2.2** virtual void [Arc::DelegationContainerSOAP::CheckConsumers](#) (void) [protected, virtual]

Periodic management of stored consumers

**5.44.2.3** `bool Arc::DelegationContainerSOAP::DelegateCredentialsInit (const SOAPEnvelope & in, SOAPEnvelope & out, const std::string & client = "")` [protected]

See [DelegationConsumerSOAP::DelegateCredentialsInit](#) If 'client' is not empty then all subsequent calls involving access to generated credentials must contain same value in their 'client' arguments.

**5.44.2.4** `bool Arc::DelegationContainerSOAP::DelegatedToken (std::string & credentials, XMLNode token, const std::string & client = "")`

See [DelegationConsumerSOAP::DelegatedToken](#)

**5.44.2.5** `virtual DelegationConsumerSOAP* Arc::DelegationContainerSOAP::FindConsumer (const std::string & id, const std::string & client)` [protected, virtual]

Finds previously created consumer in internal store

**5.44.2.6** `virtual bool Arc::DelegationContainerSOAP::QueryConsumer (DelegationConsumerSOAP * c, std::string & credentials)` [protected, virtual]

Obtain stored credentials - not all containers may provide this functionality

**5.44.2.7** `virtual void Arc::DelegationContainerSOAP::ReleaseConsumer (DelegationConsumerSOAP * c)` [protected, virtual]

Releases consumer obtained by call to [AddConsumer\(\)](#) or [FindConsumer\(\)](#)

**5.44.2.8** `virtual void Arc::DelegationContainerSOAP::RemoveConsumer (DelegationConsumerSOAP * c)` [protected, virtual]

Releases consumer obtained by call to [AddConsumer\(\)](#) or [FindConsumer\(\)](#) and deletes it

**5.44.2.9** `virtual bool Arc::DelegationContainerSOAP::TouchConsumer (DelegationConsumerSOAP * c, const std::string & credentials)` [protected, virtual]

Marks consumer as recently used and acquire new credentials

**5.44.2.10** `bool Arc::DelegationContainerSOAP::UpdateCredentials (std::string & credentials, const SOAPEnvelope & in, SOAPEnvelope & out, const std::string & client = "")` [protected]

See [DelegationConsumerSOAP::UpdateCredentials](#)

### 5.44.3 Field Documentation

**5.44.3.1** `bool Arc::DelegationContainerSOAP::context_lock` [protected]

If true delegation consumer is deleted when connection context is destroyed

**5.44.3.2** int [Arc::DelegationContainerSOAP::max\\_duration\\_](#) [protected]

Lifetime of unused delegation consumer

**5.44.3.3** int [Arc::DelegationContainerSOAP::max\\_size\\_](#) [protected]

Max. number of delegation consumers

**5.44.3.4** int [Arc::DelegationContainerSOAP::max\\_usage\\_](#) [protected]

Max. times same delegation consumer may accept credentials

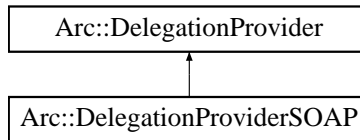
The documentation for this class was generated from the following file:

- DelegationInterface.h

## 5.45 Arc::DelegationProvider Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationProvider::



### Public Member Functions

- [DelegationProvider](#) (const std::string &credentials)
- [DelegationProvider](#) (const std::string &cert\_file, const std::string &key\_file, std::istream \*inpwd=NULL)
- std::string [Delegate](#) (const std::string &request, const DelegationRestrictions &restrictions=DelegationRestrictions())

### 5.45.1 Detailed Description

A provider of delegated credentials. During delegation procedure this class generates new credential to be used in proxy/delegated credential.

### 5.45.2 Constructor & Destructor Documentation

#### 5.45.2.1 Arc::DelegationProvider::DelegationProvider (const std::string & *credentials*)

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain PEM-encoded certificate, private key and optionally certificates chain.

#### 5.45.2.2 Arc::DelegationProvider::DelegationProvider (const std::string & *cert\_file*, const std::string & *key\_file*, std::istream \* *inpwd* = NULL)

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain filesystem path to PEM-encoded certificate and private key. Optionally cert\_file may contain certificates chain.

### 5.45.3 Member Function Documentation

#### 5.45.3.1 std::string Arc::DelegationProvider::Delegate (const std::string & *request*, const DelegationRestrictions & *restrictions* = DelegationRestrictions())

Perform delegation. Takes X509 certificate request and creates proxy credentials excluding private key. Result is then to be fed into [DelegationConsumer::Acquire](#)

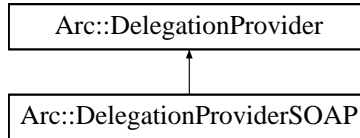
The documentation for this class was generated from the following file:

- [DelegationInterface.h](#)

## 5.46 Arc::DelegationProviderSOAP Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationProviderSOAP::



### Public Member Functions

- [DelegationProviderSOAP](#) (const std::string &credentials)
- [DelegationProviderSOAP](#) (const std::string &cert\_file, const std::string &key\_file, std::istream \*inpwd=NULL)
- bool [DelegateCredentialsInit](#) ([MCCInterface](#) &mcc\_interface, [MessageContext](#) \*context, ServiceType type=ARCDelegation)
- bool [DelegateCredentialsInit](#) ([MCCInterface](#) &mcc\_interface, [MessageAttributes](#) \*attributes\_in, [MessageAttributes](#) \*attributes\_out, [MessageContext](#) \*context, ServiceType type=ARCDelegation)
- bool [UpdateCredentials](#) ([MCCInterface](#) &mcc\_interface, [MessageContext](#) \*context, const [DelegationRestrictions](#) &restrictions=DelegationRestrictions(), ServiceType type=ARCDelegation)
- bool [UpdateCredentials](#) ([MCCInterface](#) &mcc\_interface, [MessageAttributes](#) \*attributes\_in, [MessageAttributes](#) \*attributes\_out, [MessageContext](#) \*context, const [DelegationRestrictions](#) &restrictions=DelegationRestrictions(), ServiceType type=ARCDelegation)
- bool [DelegatedToken](#) ([XMLNode](#) parent)
- const std::string & [ID](#) (void)

### 5.46.1 Detailed Description

Extension of [DelegationProvider](#) with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

### 5.46.2 Constructor & Destructor Documentation

#### 5.46.2.1 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & credentials)

Creates instance from provided credentials. Credentials are used to sign delegated credentials.

#### 5.46.2.2 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & cert\_file, const std::string & key\_file, std::istream \* inpwd = NULL)

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain filesystem path to PEM-encoded certificate and private key. Optionally cert\_file may contain certificates chain.



### 5.46.3 Member Function Documentation

**5.46.3.1** `bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & mcc_interface, MessageAttributes * attributes_in, MessageAttributes * attributes_out, MessageContext * context, ServiceType stype = ARCDelagation)`

Extended version of DelegateCredentialsInit(MCCInterface&,MessageContext\*). Additionally takes attributes for request and response message to make fine control on message processing possible.

**5.46.3.2** `bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & mcc_interface, MessageContext * context, ServiceType stype = ARCDelagation)`

Performs DelegateCredentialsInit SOAP operation. As result request for delegated credentials is received by this instance and stored internally. Call to UpdateCredentials should follow.

**5.46.3.3** `bool Arc::DelegationProviderSOAP::DelegatedToken (XMLNode parent)`

Generates DelegatedToken element. Element is created as child of provided XML element and contains structure described in delegation.wsdl.

**5.46.3.4** `const std::string& Arc::DelegationProviderSOAP::ID (void) [inline]`

Returns the identifier provided by service accepting delegated credentials. This identifier may then be used to refer to credentials stored at service.

**5.46.3.5** `bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & mcc_interface, MessageAttributes * attributes_in, MessageAttributes * attributes_out, MessageContext * context, const DelegationRestrictions & restrictions = DelegationRestrictions(), ServiceType stype = ARCDelagation)`

Extended version of UpdateCredentials(MCCInterface&,MessageContext\*). Additionally takes attributes for request and response message to make fine control on message processing possible.

**5.46.3.6** `bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & mcc_interface, MessageContext * context, const DelegationRestrictions & restrictions = DelegationRestrictions(), ServiceType stype = ARCDelagation)`

Performs UpdateCredentials SOAP operation. This concludes delegation procedure and passes delegated credentials to [DelegationConsumerSOAP](#) instance.

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

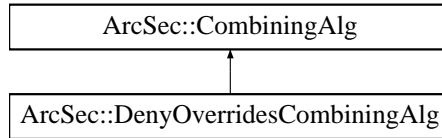
- DelegationInterface.h

## 5.47 ArcSec::DenyOverridesCombiningAlg Class Reference

Implement the "Deny-Overrides" algorithm.

```
#include <DenyOverridesAlg.h>
```

Inheritance diagram for ArcSec::DenyOverridesCombiningAlg::



### Public Member Functions

- virtual Result [combine](#) (EvaluationCtx \*ctx, std::list< Policy \* > policies)
- virtual const std::string & [getalgId](#) (void) const

#### 5.47.1 Detailed Description

Implement the "Deny-Overrides" algorithm.

Deny-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "deny" result from any policy, then stops scanning and gives "deny" as result, otherwise gives "permit".

#### 5.47.2 Member Function Documentation

**5.47.2.1** virtual Result ArcSec::DenyOverridesCombiningAlg::combine (EvaluationCtx \* ctx, std::list< Policy \* > policies) [virtual]

If there is one policy which return negative evaluation result, then omit the other policies and return DECISION\_DENY

##### Parameters:

- ctx* This object contains request information which will be used to evaluated against policy.
- policies* This is a container which contains policy objects.

##### Returns:

The combined result according to the algorithm.

Implements [ArcSec::CombiningAlg](#).

**5.47.2.2** virtual const std::string& ArcSec::DenyOverridesCombiningAlg::getalgId (void) const [inline, virtual]

Get the identifier

Implements [ArcSec::CombiningAlg](#).

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

- DenyOverridesAlg.h

## 5.48 DataStaging::DTR Class Reference

Data Transfer Request.

```
#include <DTR.h>
```

### Public Member Functions

- [DTR](#) ()
- [DTR](#) (const std::string &source, const std::string &destination, const [Arc::UserConfig](#) &usercfg, const std::string &jobid, const uid\_t &uid, [DTRLogger](#) log)
- [~DTR](#) ()
- [operator bool](#) () const
- [bool operator!](#) () const
- void [registerCallback](#) ([DTRCallback](#) \*cb, [StagingProcesses](#) owner)
- std::list< [DTRCallback](#) \* > [get\\_callbacks](#) (const std::map< [StagingProcesses](#), std::list< [DTRCallback](#) \* > > &proc\_callback, [StagingProcesses](#) owner)
- void [reset](#) ()
- void [set\\_id](#) (const std::string &id)
- std::string [get\\_id](#) () const
- std::string [get\\_short\\_id](#) () const
- [Arc::DataHandle](#) & [get\\_source](#) ()
- [Arc::DataHandle](#) & [get\\_destination](#) ()
- std::string [get\\_source\\_str](#) () const
- std::string [get\\_destination\\_str](#) () const
- const [Arc::UserConfig](#) & [get\\_usercfg](#) () const
- void [set\\_timeout](#) (time\_t value)
- [Arc::Time](#) [get\\_timeout](#) () const
- void [set\\_process\\_time](#) (const [Arc::Period](#) &process\_time)
- [Arc::Time](#) [get\\_process\\_time](#) () const
- [Arc::Time](#) [get\\_creation\\_time](#) () const
- [Arc::Time](#) [get\\_modification\\_time](#) () const
- std::string [get\\_parent\\_job\\_id](#) () const
- void [set\\_priority](#) (int pri)
- int [get\\_priority](#) () const
- void [set\\_rfc\\_proxy](#) (bool rfc)
- bool [is\\_rfc\\_proxy](#) () const
- void [set\\_transfer\\_share](#) (const std::string &share\_name)
- std::string [get\\_transfer\\_share](#) () const
- void [set\\_sub\\_share](#) (const std::string &share)
- std::string [get\\_sub\\_share](#) () const
- void [set\\_tries\\_left](#) (unsigned int tries)
- unsigned int [get\\_tries\\_left](#) () const
- unsigned int [get\\_initial\\_tries](#) () const
- void [decrease\\_tries\\_left](#) ()
- void [set\\_status](#) ([DTRStatus](#) stat)
- [DTRStatus](#) [get\\_status](#) ()
- void [set\\_error\\_status](#) ([DTRErrorStatus::DTRErrorStatusType](#) error\_stat, [DTRErrorStatus::DTRErrorLocation](#) error\_loc, const std::string &desc="")
- void [reset\\_error\\_status](#) ()

- [DTRErrorStatus get\\_error\\_status](#) ()
- void [set\\_bytes\\_transferred](#) (unsigned long long int bytes)
- unsigned long long int [get\\_bytes\\_transferred](#) () const
- void [set\\_cancel\\_request](#) ()
- bool [cancel\\_requested](#) () const
- void [set\\_delivery\\_endpoint](#) (const [Arc::URL](#) &endpoint)
- const [Arc::URL](#) & [get\\_delivery\\_endpoint](#) () const
- void [add\\_problematic\\_delivery\\_service](#) (const [Arc::URL](#) &endpoint)
- const std::vector< [Arc::URL](#) > & [get\\_problematic\\_delivery\\_services](#) () const
- void [host\\_cert\\_for\\_remote\\_delivery](#) (bool host)
- bool [host\\_cert\\_for\\_remote\\_delivery](#) () const
- void [set\\_cache\\_file](#) (const std::string &filename)
- std::string [get\\_cache\\_file](#) () const
- void [set\\_cache\\_parameters](#) (const [DTRCacheParameters](#) &param)
- const [DTRCacheParameters](#) & [get\\_cache\\_parameters](#) () const
- void [set\\_cache\\_state](#) ([CacheState](#) state)
- [CacheState](#) [get\\_cache\\_state](#) () const
- void [set\\_mapped\\_source](#) (const std::string &file="")
- std::string [get\\_mapped\\_source](#) () const
- [StagingProcesses](#) [get\\_owner](#) () const
- [Arc::User](#) [get\\_local\\_user](#) () const
- void [set\\_replication](#) (bool rep)
- bool [is\\_replication](#) () const
- void [set\\_force\\_registration](#) (bool force)
- bool [is\\_force\\_registration](#) () const
- void [set\\_bulk\\_start](#) (bool value)
- bool [get\\_bulk\\_start](#) () const
- void [set\\_bulk\\_end](#) (bool value)
- bool [get\\_bulk\\_end](#) () const
- bool [bulk\\_possible](#) ()
- const [DTRLogger](#) & [get\\_logger](#) () const
- void [connect\\_logger](#) ()
- void [disconnect\\_logger](#) ()
- bool [suspend](#) ()
- bool [error](#) () const
- bool [is\\_destined\\_for\\_pre\\_processor](#) () const
- bool [is\\_destined\\_for\\_post\\_processor](#) () const
- bool [is\\_destined\\_for\\_delivery](#) () const
- bool [came\\_from\\_pre\\_processor](#) () const
- bool [came\\_from\\_post\\_processor](#) () const
- bool [came\\_from\\_delivery](#) () const
- bool [came\\_from\\_generator](#) () const
- bool [is\\_in\\_final\\_state](#) () const

## Static Public Member Functions

- static void [push](#) ([DTR\\_ptr](#) dtr, [StagingProcesses](#) new\_owner)

## Static Public Attributes

- static const [Arc::URL LOCAL\\_DELIVERY](#)

### 5.48.1 Detailed Description

Data Transfer Request.

[DTR](#) stands for Data Transfer Request and a [DTR](#) describes a data transfer between two endpoints, a source and a destination. There are several parameters and options relating to the transfer contained in a [DTR](#). The normal workflow is for a [Generator](#) to create a [DTR](#) and send it to the [Scheduler](#) for processing using `dtr.push(SCHEDULER)`. If the [Generator](#) is a subclass of [DTRCallback](#), when the [Scheduler](#) has finished with the [DTR](#) the `receivedDTR()` callback method is called.

DTRs should always be used through the ThreadedPointer `DTR_ptr`. This ensures proper memory management when passing DTRs among various threads. To enforce this policy the copy constructor and assignment operator are private.

`registerCallback(this,DataStaging::GENERATOR)` can be used to activate the callback. The following simple [Generator](#) code sample illustrates how to use DTRs:

```
class MyGenerator : public DTRCallback {
public:
    void receiveDTR(DTR_ptr dtr);
    void run();
private:
    Arc::SimpleCondition cond;
};

void MyGenerator::receiveDTR(DTR_ptr dtr) {
    // DTR received back, so notify waiting condition
    std::cout << "Received DTR " << dtr->get_id() << std::endl;
    cond.signal();
}

void MyGenerator::run() {
    // start Scheduler thread
    Scheduler scheduler;
    scheduler.start();

    // create a DTR
    DTR_ptr dtr(new DTR(source, destination,...));

    // register this callback
    dtr->registerCallback(this,DataStaging::GENERATOR);
    // this line must be here in order to pass the DTR to the Scheduler
    dtr->registerCallback(&scheduler,DataStaging::SCHEDULER);

    // push the DTR to the Scheduler
    DataStaging::DTR::push(dtr, DataStaging::SCHEDULER);

    // wait until callback is called
    cond.wait();
    // DTR is finished, so stop Scheduler
    scheduler.stop();
}
```

A lock protects member variables that are likely to be accessed and modified by multiple threads.

## 5.48.2 Constructor & Destructor Documentation

### 5.48.2.1 DataStaging::DTR::DTR ()

Public empty constructor.

### 5.48.2.2 DataStaging::DTR::DTR (const std::string & *source*, const std::string & *destination*, const [Arc::UserConfig](#) & *usercfg*, const std::string & *jobid*, const uid\_t & *uid*, [DTRLogger](#) *log*)

Normal constructor.

Construct a new [DTR](#).

#### Parameters:

*source* Endpoint from which to read data

*destination* Endpoint to which to write data

*usercfg* Provides some user configuration information

*jobid* ID of the job associated with this data transfer

*uid* UID to use when accessing local file system if source or destination is a local file. If this is different to the current uid then the current uid must have sufficient privileges to change uid.

*log* ThreadedPointer containing log object. If NULL the root logger is used.

### 5.48.2.3 DataStaging::DTR::~~DTR () `[inline]`

Empty destructor.

## 5.48.3 Member Function Documentation

### 5.48.3.1 void DataStaging::DTR::add\_problematic\_delivery\_service (const [Arc::URL](#) & *endpoint*) `[inline]`

Add problematic endpoint. Should only be those endpoints where there is a problem with the service itself and not the transfer.

### 5.48.3.2 bool DataStaging::DTR::bulk\_possible ()

Whether bulk operation is possible according to current state and src/dest.

### 5.48.3.3 bool DataStaging::DTR::came\_from\_delivery () const

Returns true if this [DTR](#) just came from delivery.

### 5.48.3.4 bool DataStaging::DTR::came\_from\_generator () const

Returns true if this [DTR](#) just came from the generator.

**5.48.3.5 bool DataStaging::DTR::came\_from\_post\_processor () const**

Returns true if this [DTR](#) just came from the post-processor.

**5.48.3.6 bool DataStaging::DTR::came\_from\_pre\_processor () const**

Returns true if this [DTR](#) just came from the pre-processor.

**5.48.3.7 bool DataStaging::DTR::cancel\_requested () const [inline]**

Returns true if cancellation has been requested.

**5.48.3.8 void DataStaging::DTR::connect\_logger () [inline]**

Connect log destinations to logger. Only needs to be done after disconnect().

**5.48.3.9 void DataStaging::DTR::decrease\_tries\_left ()**

Decrease attempt number.

**5.48.3.10 void DataStaging::DTR::disconnect\_logger () [inline]**

Disconnect log destinations from logger.

**5.48.3.11 bool DataStaging::DTR::error () const [inline]**

Did an error happen?

**5.48.3.12 bool DataStaging::DTR::get\_bulk\_end () const [inline]**

Get bulk start flag.

**5.48.3.13 bool DataStaging::DTR::get\_bulk\_start () const [inline]**

Get bulk start flag.

**5.48.3.14 unsigned long long int DataStaging::DTR::get\_bytes\_transferred () const [inline]**

Get current number of bytes transferred.

**5.48.3.15 std::string DataStaging::DTR::get\_cache\_file () const [inline]**

Get cache filename.



**5.48.3.16** `const DTRCacheParameters& DataStaging::DTR::get_cache_parameters () const` `[inline]`

Get cache parameters.

**5.48.3.17** `CacheState DataStaging::DTR::get_cache_state () const` `[inline]`

Get the cache state.

**5.48.3.18** `std::list<DTRCallback*> DataStaging::DTR::get_callbacks (const std::map<StagingProcesses, std::list< DTRCallback * > > & proc_callback, StagingProcesses owner)`

Get the list of callbacks for this owner. Protected by lock.

**5.48.3.19** `Arc::Time DataStaging::DTR::get_creation_time () const` `[inline]`

Get the creation time.

**5.48.3.20** `const Arc::URL& DataStaging::DTR::get_delivery_endpoint () const` `[inline]`

Returns delivery endpoint.

**5.48.3.21** `Arc::DataHandle& DataStaging::DTR::get_destination ()` `[inline]`

Get destination handle. Return by reference since DataHandle cannot be copied.

**5.48.3.22** `std::string DataStaging::DTR::get_destination_str () const` `[inline]`

Get destination as a string.

**5.48.3.23** `DTRErrorStatus DataStaging::DTR::get_error_status ()`

Get the error status.

**5.48.3.24** `std::string DataStaging::DTR::get_id () const` `[inline]`

Get the ID of this [DTR](#).

**5.48.3.25** `unsigned int DataStaging::DTR::get_initial_tries () const` `[inline]`

Get the initial number of attempts (set by [set\\_tries\\_left\(\)](#)).

**5.48.3.26** `Arc::User DataStaging::DTR::get_local_user () const` `[inline]`

Get the local user information.

**5.48.3.27** `const DTRLogger& DataStaging::DTR::get_logger () const` `[inline]`

Get Logger object, so that processes can log to this DTR's log.

**5.48.3.28** `std::string DataStaging::DTR::get_mapped_source () const` `[inline]`

Get the mapped file.

**5.48.3.29** `Arc::Time DataStaging::DTR::get_modification_time () const` `[inline]`

Get the modification time.

**5.48.3.30** `StagingProcesses DataStaging::DTR::get_owner () const` `[inline]`

Find the [DTR](#) owner.

**5.48.3.31** `std::string DataStaging::DTR::get_parent_job_id () const` `[inline]`

Get the parent job ID.

**5.48.3.32** `int DataStaging::DTR::get_priority () const` `[inline]`

Get the priority.

**5.48.3.33** `const std::vector<Arc::URL>& DataStaging::DTR::get_problematic_delivery_services () const` `[inline]`

Get all problematic endpoints.

**5.48.3.34** `Arc::Time DataStaging::DTR::get_process_time () const` `[inline]`

Get the next processing time for the [DTR](#).

**5.48.3.35** `std::string DataStaging::DTR::get_short_id () const`

Get an abbreviated version of the [DTR](#) ID - useful to reduce logging verbosity.

**5.48.3.36** `Arc::DataHandle& DataStaging::DTR::get_source ()` `[inline]`

Get source handle. Return by reference since DataHandle cannot be copied.

**5.48.3.37** `std::string DataStaging::DTR::get_source_str () const` `[inline]`

Get source as a string.

**5.48.3.38** [DTRStatus](#) DataStaging::DTR::get\_status ()

Get the status. Protected by lock.

**5.48.3.39** `std::string` DataStaging::DTR::get\_sub\_share () const [inline]

Get sub-share.

**5.48.3.40** [Arc::Time](#) DataStaging::DTR::get\_timeout () const [inline]

Get the timeout for processing this [DTR](#).

**5.48.3.41** `std::string` DataStaging::DTR::get\_transfer\_share () const [inline]

Get the transfer share. sub\_share is automatically added to transfershare.

**5.48.3.42** `unsigned int` DataStaging::DTR::get\_tries\_left () const [inline]

Get the number of attempts remaining.

**5.48.3.43** `const` [Arc::UserConfig&](#) DataStaging::DTR::get\_usercfg () const [inline]

Get the UserConfig object associated with this [DTR](#).

**5.48.3.44** `bool` DataStaging::DTR::host\_cert\_for\_remote\_delivery () const [inline]

Set the flag for using host certificate for contacting remote delivery services.

**5.48.3.45** `void` DataStaging::DTR::host\_cert\_for\_remote\_delivery (`bool host`) [inline]

Set the flag for using host certificate for contacting remote delivery services.

**5.48.3.46** `bool` DataStaging::DTR::is\_destined\_for\_delivery () const

Returns true if this [DTR](#) is about to go into delivery.

**5.48.3.47** `bool` DataStaging::DTR::is\_destined\_for\_post\_processor () const

Returns true if this [DTR](#) is about to go into the post-processor.

**5.48.3.48** `bool` DataStaging::DTR::is\_destined\_for\_pre\_processor () const

Returns true if this [DTR](#) is about to go into the pre-processor.

**5.48.3.49** `bool DataStaging::DTR::is_force_registration () const` `[inline]`

Get force replication flag.

**5.48.3.50** `bool DataStaging::DTR::is_in_final_state () const`

Returns true if this [DTR](#) is in a final state (finished, failed or cancelled).

**5.48.3.51** `bool DataStaging::DTR::is_replication () const` `[inline]`

Get replication flag.

**5.48.3.52** `bool DataStaging::DTR::is_rfc_proxy () const` `[inline]`

Get whether credentials are type RFC proxy.

**5.48.3.53** `DataStaging::DTR::operator bool (void) const` `[inline]`

Is [DTR](#) valid?

**5.48.3.54** `bool DataStaging::DTR::operator! (void) const` `[inline]`

Is [DTR](#) not valid?

**5.48.3.55** `static void DataStaging::DTR::push (DTR\_ptr dtr, StagingProcesses new_owner)`  
`[static]`

Pass the [DTR](#) from one process to another. Protected by lock.

**5.48.3.56** `void DataStaging::DTR::registerCallback (DTRCallback * cb, StagingProcesses owner)`

Register callback objects to be used during [DTR](#) processing.

Objects deriving from [DTRCallback](#) can be registered with this method. The callback method of these objects will then be called when the [DTR](#) is passed to the specified owner. Protected by lock.

**5.48.3.57** `void DataStaging::DTR::reset ()`

Reset information held on this [DTR](#), such as resolved replicas, error state etc.

Useful when a failed [DTR](#) is to be retried.

**5.48.3.58** `void DataStaging::DTR::reset_error_status ()`

Set the error status back to NONE\_ERROR and clear other fields.

**5.48.3.59** void DataStaging::DTR::set\_bulk\_end (bool *value*) [inline]

Set bulk end flag.

**5.48.3.60** void DataStaging::DTR::set\_bulk\_start (bool *value*) [inline]

Set bulk start flag.

**5.48.3.61** void DataStaging::DTR::set\_bytes\_transferred (unsigned long long int *bytes*)

Set bytes transferred (should be set by whatever is controlling the transfer).

**5.48.3.62** void DataStaging::DTR::set\_cache\_file (const std::string &*filename*)

Set cache filename.

**5.48.3.63** void DataStaging::DTR::set\_cache\_parameters (const [DTRCacheParameters](#) &*param*)  
[inline]

Set cache parameters.

**5.48.3.64** void DataStaging::DTR::set\_cache\_state ([CacheState](#) *state*)

Set the cache state.

**5.48.3.65** void DataStaging::DTR::set\_cancel\_request ()

Set the [DTR](#) to be cancelled.

**5.48.3.66** void DataStaging::DTR::set\_delivery\_endpoint (const [Arc::URL](#) &*endpoint*)  
[inline]

Set delivery endpoint.

**5.48.3.67** void DataStaging::DTR::set\_error\_status ([DTRErrorStatus::DTRErrorStatusType](#)  
*error\_stat*, [DTRErrorStatus::DTRErrorLocation](#) *error\_loc*, const std::string &*desc* =  
" ")

Set the error status.

The [DTRErrorStatus](#) last error state field is set to the current status of the [DTR](#). Protected by lock.

**5.48.3.68** void DataStaging::DTR::set\_force\_registration (bool *force*) [inline]

Set force replication flag.

**5.48.3.69 void DataStaging::DTR::set\_id (const std::string & *id*)**

Set the ID of this [DTR](#). Useful when passing [DTR](#) between processes.

**5.48.3.70 void DataStaging::DTR::set\_mapped\_source (const std::string & *file* = "") [inline]**

Set the mapped file.

**5.48.3.71 void DataStaging::DTR::set\_priority (int *pri*)**

Set the priority.

**5.48.3.72 void DataStaging::DTR::set\_process\_time (const Arc::Period & *process\_time*)**

Set the next processing time to current time + given time.

**5.48.3.73 void DataStaging::DTR::set\_replication (bool *rep*) [inline]**

Set replication flag.

**5.48.3.74 void DataStaging::DTR::set\_rfc\_proxy (bool *rfc*) [inline]**

Set whether credentials are type RFC proxy.

**5.48.3.75 void DataStaging::DTR::set\_status ([DTRStatus](#) *stat*)**

Set the status. Protected by lock.

**5.48.3.76 void DataStaging::DTR::set\_sub\_share (const std::string & *share*) [inline]**

Set sub-share.

**5.48.3.77 void DataStaging::DTR::set\_timeout (time\_t *value*) [inline]**

Set the timeout for processing this [DTR](#).

**5.48.3.78 void DataStaging::DTR::set\_transfer\_share (const std::string & *share\_name*)**

Set the transfer share. `sub_share` is automatically added to `transfershare`.

**5.48.3.79 void DataStaging::DTR::set\_tries\_left (unsigned int *tries*)**

Set the number of attempts remaining.

#### 5.48.3.80 bool DataStaging::DTR::suspend ()

Suspend the [DTR](#) which is in doing transfer in the delivery process.

### 5.48.4 Field Documentation

#### 5.48.4.1 const [Arc::URL](#) DataStaging::DTR::LOCAL\_DELIVERY [static]

URL that is used to denote local Delivery should be used.

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

- DTR.h

## 5.49 DataStaging::DTRCacheParameters Class Reference

The configured cache directories.

```
#include <DTR.h>
```

### Public Member Functions

- [DTRCacheParameters](#) (void)
- [DTRCacheParameters](#) (std::vector< std::string > caches, std::vector< std::string > remote\_caches, std::vector< std::string > drain\_caches)

### Data Fields

- std::vector< std::string > [cache\\_dirs](#)
- std::vector< std::string > [remote\\_cache\\_dirs](#)
- std::vector< std::string > [drain\\_cache\\_dirs](#)

#### 5.49.1 Detailed Description

The configured cache directories.

#### 5.49.2 Constructor & Destructor Documentation

##### 5.49.2.1 DataStaging::DTRCacheParameters::DTRCacheParameters (void) [inline]

Constructor with empty lists initialised.

##### 5.49.2.2 DataStaging::DTRCacheParameters::DTRCacheParameters (std::vector< std::string > caches, std::vector< std::string > remote\_caches, std::vector< std::string > drain\_caches)

Constructor with supplied cache lists.

#### 5.49.3 Field Documentation

##### 5.49.3.1 std::vector<std::string> DataStaging::DTRCacheParameters::cache\_dirs

List of (cache dir [link dir]).

##### 5.49.3.2 std::vector<std::string> DataStaging::DTRCacheParameters::drain\_cache\_dirs

List of draining caches. Not necessary for data staging but here for completeness.

##### 5.49.3.3 std::vector<std::string> DataStaging::DTRCacheParameters::remote\_cache\_dirs

List of (cache dir [link dir]) for remote caches.

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



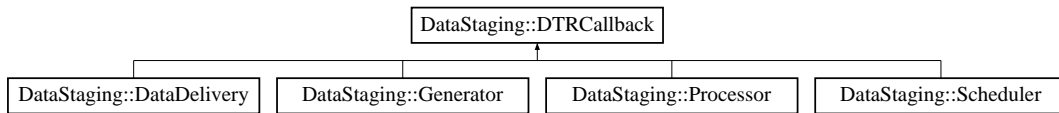
- [DTR.h](#)

## 5.50 DataStaging::DTRCallback Class Reference

The base class from which all callback-enabled classes should be derived.

```
#include <DTR.h>
```

Inheritance diagram for DataStaging::DTRCallback::



### Public Member Functions

- virtual [~DTRCallback](#) ()
- virtual void [receiveDTR](#) ([DTR\\_ptr](#) dtr)=0

#### 5.50.1 Detailed Description

The base class from which all callback-enabled classes should be derived.

This class is a container for a callback method which is called when a [DTR](#) is to be passed to a component. Several components in data staging (eg [Scheduler](#), [Generator](#)) are subclasses of [DTRCallback](#), which allows them to receive DTRs through the callback system.

#### 5.50.2 Constructor & Destructor Documentation

##### 5.50.2.1 virtual DataStaging::DTRCallback::~~DTRCallback () [inline, virtual]

Empty virtual destructor

#### 5.50.3 Member Function Documentation

##### 5.50.3.1 virtual void DataStaging::DTRCallback::receiveDTR ([DTR\\_ptr](#) dtr) [pure virtual]

Defines the callback method called when a [DTR](#) is pushed to this object. The automatic memory management of [DTR\\_ptr](#) ensures that the [DTR](#) object is only deleted when the last copy is deleted.

Implemented in [DataStaging::DataDelivery](#), [DataStaging::Generator](#), [DataStaging::Processor](#), and [DataStaging::Scheduler](#).

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

- [DTR.h](#)

## 5.51 DataStaging::DTRErrorStatus Class Reference

A class to represent error states reported by various components.

```
#include <DTRStatus.h>
```

### Public Types

- [NONE\\_ERROR](#)
- [INTERNAL\\_LOGIC\\_ERROR](#)
- [INTERNAL\\_PROCESS\\_ERROR](#)
- [SELF\\_REPLICATION\\_ERROR](#)
- [CACHE\\_ERROR](#)
- [TEMPORARY\\_REMOTE\\_ERROR](#)
- [PERMANENT\\_REMOTE\\_ERROR](#)
- [LOCAL\\_FILE\\_ERROR](#)
- [TRANSFER\\_SPEED\\_ERROR](#)
- [STAGING\\_TIMEOUT\\_ERROR](#)
- [NO\\_ERROR\\_LOCATION](#)
- [ERROR\\_SOURCE](#)
- [ERROR\\_DESTINATION](#)
- [ERROR\\_TRANSFER](#)
- [ERROR\\_UNKNOWN](#)
- [enum DTRErrorStatusType {](#)  
    [NONE\\_ERROR, INTERNAL\\_LOGIC\\_ERROR, INTERNAL\\_PROCESS\\_ERROR, SELF\\_](#)  
    [REPLICATION\\_ERROR,](#)  
    [CACHE\\_ERROR, TEMPORARY\\_REMOTE\\_ERROR, PERMANENT\\_REMOTE\\_ERROR,](#)  
    [LOCAL\\_FILE\\_ERROR,](#)  
    [TRANSFER\\_SPEED\\_ERROR, STAGING\\_TIMEOUT\\_ERROR }](#)
- [enum DTRErrorLocation {](#)  
    [NO\\_ERROR\\_LOCATION, ERROR\\_SOURCE, ERROR\\_DESTINATION, ERROR\\_TRANSFER,](#)  
    [ERROR\\_UNKNOWN }](#)

### Public Member Functions

- [DTRErrorStatus](#) ([DTRErrorStatusType](#) status, [DTRStatus::DTRStatusType](#) error\_state, [DTRErrorLocation](#) location, const std::string &desc="")
- [DTRErrorStatus](#) ()
- [DTRErrorStatusType](#) [GetErrorStatus](#) () const
- [DTRStatus::DTRStatusType](#) [GetLastErrorState](#) () const
- [DTRErrorLocation](#) [GetErrorLocation](#) () const
- std::string [GetDesc](#) () const
- bool [operator==](#) (const [DTRErrorStatusType](#) &s) const
- bool [operator==](#) (const [DTRErrorStatus](#) &s) const
- bool [operator!=](#) (const [DTRErrorStatusType](#) &s) const
- bool [operator!=](#) (const [DTRErrorStatus](#) &s) const
- [DTRErrorStatus](#) & [operator=](#) (const [DTRErrorStatusType](#) &s)

### 5.51.1 Detailed Description

A class to represent error states reported by various components.

### 5.51.2 Member Enumeration Documentation

#### 5.51.2.1 enum [DataStaging::DTRErrorStatus::DTRErrorLocation](#)

Describes where the error occurred.

**Enumerator:**

***NO\_ERROR\_LOCATION*** No error.  
***ERROR\_SOURCE*** Error with source.  
***ERROR\_DESTINATION*** Error with destination.  
***ERROR\_TRANSFER*** Error during transfer not directly related to source or destination.  
***ERROR\_UNKNOWN*** Error occurred in an unknown location.

#### 5.51.2.2 enum [DataStaging::DTRErrorStatus::DTRErrorStatusType](#)

A list of error types.

**Enumerator:**

***NONE\_ERROR*** No error.  
***INTERNAL\_LOGIC\_ERROR*** Internal error in Data Staging logic.  
***INTERNAL\_PROCESS\_ERROR*** Internal processing error, like losing contact with external process.  
***SELF\_REPLICATION\_ERROR*** Attempt to replicate a file to itself.  
***CACHE\_ERROR*** Permanent error with cache.  
***TEMPORARY\_REMOTE\_ERROR*** Temporary error with remote service.  
***PERMANENT\_REMOTE\_ERROR*** Permanent error with remote service.  
***LOCAL\_FILE\_ERROR*** Error with local file.  
***TRANSFER\_SPEED\_ERROR*** Transfer rate was too slow.  
***STAGING\_TIMEOUT\_ERROR*** Waited for too long to become staging.

### 5.51.3 Constructor & Destructor Documentation

#### 5.51.3.1 [DataStaging::DTRErrorStatus::DTRErrorStatus](#) ([DTRErrorStatusType](#) *status*, [DTRStatus::DTRStatusType](#) *error\_state*, [DTRErrorLocation](#) *location*, const std::string & *desc* = "") `[inline]`

Create a new [DTRErrorStatus](#) with given error states.

#### 5.51.3.2 [DataStaging::DTRErrorStatus::DTRErrorStatus](#) () `[inline]`

Create a new [DTRErrorStatus](#) with default none/null error states.

## 5.51.4 Member Function Documentation

### 5.51.4.1 `std::string DataStaging::DTRErrorStatus::GetDesc () const` [inline]

Returns the error description.

### 5.51.4.2 `DTRErrorLocation DataStaging::DTRErrorStatus::GetErrorLocation () const` [inline]

Returns the location at which the error occurred.

### 5.51.4.3 `DTRErrorStatusType DataStaging::DTRErrorStatus::GetErrorStatus () const` [inline]

Returns the error type.

### 5.51.4.4 `DTRStatus::DTRStatusType DataStaging::DTRErrorStatus::GetLastErrorState () const` [inline]

Returns the state in which the error occurred.

### 5.51.4.5 `bool DataStaging::DTRErrorStatus::operator!= (const DTRErrorStatus & s) const` [inline]

Returns true if this error status is not the same as the given [DTRErrorStatus](#).

### 5.51.4.6 `bool DataStaging::DTRErrorStatus::operator!= (const DTRErrorStatusType & s) const` [inline]

Returns true if this error status is not the same as the given [DTRErrorStatusType](#).

### 5.51.4.7 `DTRErrorStatus& DataStaging::DTRErrorStatus::operator= (const DTRErrorStatusType & s)` [inline]

Make a new [DTRErrorStatus](#) with the same error status as the given [DTRErrorStatusType](#).

### 5.51.4.8 `bool DataStaging::DTRErrorStatus::operator== (const DTRErrorStatus & s) const` [inline]

Returns true if this error status is the same as the given [DTRErrorStatus](#).

### 5.51.4.9 `bool DataStaging::DTRErrorStatus::operator== (const DTRErrorStatusType & s) const` [inline]

Returns true if this error status is the same as the given [DTRErrorStatusType](#).

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

- [DTRStatus.h](#)

## 5.52 DataStaging::DTRLList Class Reference

Global list of all active DTRs in the system.

```
#include <DTRLList.h>
```

### Public Member Functions

- bool [add\\_dtr](#) ([DTR\\_ptr](#) DTRToAdd)
- bool [delete\\_dtr](#) ([DTR\\_ptr](#) DTRToDelete)
- bool [filter\\_dtrs\\_by\\_owner](#) ([StagingProcesses](#) OwnerToFilter, std::list< [DTR\\_ptr](#) > &FilteredList)
- int [number\\_of\\_dtrs\\_by\\_owner](#) ([StagingProcesses](#) OwnerToFilter)
- bool [filter\\_dtrs\\_by\\_status](#) ([DTRStatus::DTRStatusType](#) StatusToFilter, std::list< [DTR\\_ptr](#) > &FilteredList)
- bool [filter\\_dtrs\\_by\\_statuses](#) (const std::vector< [DTRStatus::DTRStatusType](#) > &StatusesToFilter, std::list< [DTR\\_ptr](#) > &FilteredList)
- bool [filter\\_dtrs\\_by\\_statuses](#) (const std::vector< [DTRStatus::DTRStatusType](#) > &StatusesToFilter, std::map< [DTRStatus::DTRStatusType](#), std::list< [DTR\\_ptr](#) > > &FilteredList)
- bool [filter\\_dtrs\\_by\\_next\\_receiver](#) ([StagingProcesses](#) NextReceiver, std::list< [DTR\\_ptr](#) > &FilteredList)
- bool [filter\\_pending\\_dtrs](#) (std::list< [DTR\\_ptr](#) > &FilteredList)
- bool [filter\\_dtrs\\_by\\_job](#) (const std::string &jobid, std::list< [DTR\\_ptr](#) > &FilteredList)
- void [caching\\_started](#) ([DTR\\_ptr](#) request)
- void [caching\\_finished](#) ([DTR\\_ptr](#) request)
- bool [is\\_being\\_cached](#) ([DTR\\_ptr](#) DTRToCheck)
- bool [empty](#) ()
- std::list< std::string > [all\\_jobs](#) ()
- void [dumpState](#) (const std::string &path)

### 5.52.1 Detailed Description

Global list of all active DTRs in the system.

This class contains several methods for filtering the list by owner, state etc

### 5.52.2 Member Function Documentation

#### 5.52.2.1 bool DataStaging::DTRLList::add\_dtr ([DTR\\_ptr](#) DTRToAdd)

Put a new [DTR](#) into the list.

#### 5.52.2.2 std::list<std::string> DataStaging::DTRLList::all\_jobs ()

Get the list of all job IDs.

#### 5.52.2.3 void DataStaging::DTRLList::caching\_finished ([DTR\\_ptr](#) request)

Update the caching set, removing a [DTR](#).

**5.52.2.4 void DataStaging::DTRLList::caching\_started (*DTR\_ptr request*)**

Update the caching set, add a [DTR](#) (only if it is CACHEABLE).

**5.52.2.5 bool DataStaging::DTRLList::delete\_dtr (*DTR\_ptr DTRToDelete*)**

Remove a [DTR](#) from the list.

**5.52.2.6 void DataStaging::DTRLList::dumpState (const std::string & *path*)**

Dump state of all current DTRs to a destination, eg file, database, url...

Currently only file is supported.

**Parameters:**

*path* Path to the file in which to dump state.

**5.52.2.7 bool DataStaging::DTRLList::empty ()**

Returns true if there are no DTRs in the list.

**5.52.2.8 bool DataStaging::DTRLList::filter\_dtrs\_by\_job (const std::string & *jobid*, std::list<[DTR\\_ptr](#)> & *FilteredList*)**

Get the list of DTRs corresponding to the given job ID.

**Parameters:**

*FilteredList* This list is filled with filtered DTRs

**5.52.2.9 bool DataStaging::DTRLList::filter\_dtrs\_by\_next\_receiver ([StagingProcesses](#) *NextReceiver*, std::list<[DTR\\_ptr](#)> & *FilteredList*)**

Select DTRs that are about to go to the specified process.

This selection is actually a virtual queue for pre-, post-processor and delivery.

**Parameters:**

*FilteredList* This list is filled with filtered DTRs

**5.52.2.10 bool DataStaging::DTRLList::filter\_dtrs\_by\_owner ([StagingProcesses](#) *OwnerToFilter*, std::list<[DTR\\_ptr](#)> & *FilteredList*)**

Filter the queue to select DTRs owned by a specified process.

**Parameters:**

*FilteredList* This list is filled with filtered DTRs

#### 5.52.2.11 `bool DataStaging::DTRLList::filter_dtrs_by_status (DTRStatus::DTRStatusType StatusToFilter, std::list< DTR_ptr > & FilteredList)`

Filter the queue to select DTRs with particular status.

If we have only one common queue for all DTRs, this method is necessary to make virtual queues for the DTRs about to go into the pre-, post-processor or delivery stages.

##### Parameters:

*FilteredList* This list is filled with filtered DTRs

#### 5.52.2.12 `bool DataStaging::DTRLList::filter_dtrs_by_statuses (const std::vector< DTRStatus::DTRStatusType > & StatusesToFilter, std::map< DTRStatus::DTRStatusType, std::list< DTR_ptr > > & FilteredList)`

Filter the queue to select DTRs with particular statuses.

##### Parameters:

*FilteredList* This map is filled with filtered DTRs, one list per state.

#### 5.52.2.13 `bool DataStaging::DTRLList::filter_dtrs_by_statuses (const std::vector< DTRStatus::DTRStatusType > & StatusesToFilter, std::list< DTR_ptr > & FilteredList)`

Filter the queue to select DTRs with particular statuses.

##### Parameters:

*FilteredList* This list is filled with filtered DTRs

#### 5.52.2.14 `bool DataStaging::DTRLList::filter_pending_dtrs (std::list< DTR_ptr > & FilteredList)`

Select DTRs that have just arrived from pre-, post-processor, delivery or generator.

These DTRs need some reaction from the scheduler. This selection is actually a virtual queue of DTRs that need to be processed.

##### Parameters:

*FilteredList* This list is filled with filtered DTRs

#### 5.52.2.15 `bool DataStaging::DTRLList::is_being_cached (DTR_ptr DTRToCheck)`

Returns true if the DTR's source is currently in the caching set.

#### 5.52.2.16 `int DataStaging::DTRLList::number_of_dtrs_by_owner (StagingProcesses OwnerToFilter)`

Returns the number of DTRs owned by a particular process.

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

- DTRLList.h



## 5.53 DataStaging::DTRStatus Class Reference

Class representing the status of a [DTR](#).

```
#include <DTRStatus.h>
```

### Public Types

- [NEW](#)
- [CHECK\\_CACHE](#)
- [CHECKING\\_CACHE](#)
- [CACHE\\_WAIT](#)
- [CACHE\\_CHECKED](#)
- [RESOLVE](#)
- [RESOLVING](#)
- [RESOLVED](#)
- [QUERY\\_REPLICA](#)
- [QUERYING\\_REPLICA](#)
- [REPLICA\\_QUERIED](#)
- [PRE\\_CLEAN](#)
- [PRE\\_CLEANING](#)
- [PRE\\_CLEARED](#)
- [STAGE\\_PREPARE](#)
- [STAGING\\_PREPARING](#)
- [STAGING\\_PREPARING\\_WAIT](#)
- [STAGED\\_PREPARED](#)
- [TRANSFER](#)
- [TRANSFERRING](#)
- [TRANSFERRING\\_CANCEL](#)
- [TRANSFERRED](#)
- [RELEASE\\_REQUEST](#)
- [RELEASING\\_REQUEST](#)
- [REQUEST\\_RELEASED](#)
- [REGISTER\\_REPLICA](#)
- [REGISTERING\\_REPLICA](#)
- [REPLICA\\_REGISTERED](#)
- [PROCESS\\_CACHE](#)
- [PROCESSING\\_CACHE](#)
- [CACHE\\_PROCESSED](#)
- [DONE](#)
- [CANCELLED](#)
- [CANCELLED\\_FINISHED](#)
- [ERROR](#)
- [NULL\\_STATE](#)
- `enum DTRStatusType {`
  - [NEW](#), [CHECK\\_CACHE](#), [CHECKING\\_CACHE](#), [CACHE\\_WAIT](#),
  - [CACHE\\_CHECKED](#), [RESOLVE](#), [RESOLVING](#), [RESOLVED](#),
  - [QUERY\\_REPLICA](#), [QUERYING\\_REPLICA](#), [REPLICA\\_QUERIED](#), [PRE\\_CLEAN](#),
  - [PRE\\_CLEANING](#), [PRE\\_CLEARED](#), [STAGE\\_PREPARE](#), [STAGING\\_PREPARING](#),`}`

STAGING\_PREPARING\_WAIT, STAGED\_PREPARED, TRANSFER, TRANSFERRING,  
 TRANSFERRING\_CANCEL, TRANSFERRED, RELEASE\_REQUEST, RELEASING\_  
 REQUEST,  
 REQUEST\_RELEASED, REGISTER\_REPLICA, REGISTERING\_REPLICA, REPLICA\_  
 REGISTERED,  
 PROCESS\_CACHE, PROCESSING\_CACHE, CACHE\_PROCESSED, DONE,  
 CANCELLED, CANCELLED\_FINISHED, ERROR, NULL\_STATE }

## Public Member Functions

- [DTRStatus](#) (const [DTRStatusType](#) &status, std::string desc="")
- [DTRStatus](#) ()
- bool [operator==](#) (const [DTRStatusType](#) &s) const
- bool [operator==](#) (const [DTRStatus](#) &s) const
- bool [operator!=](#) (const [DTRStatusType](#) &s) const
- bool [operator!=](#) (const [DTRStatus](#) &s) const
- [DTRStatus](#) & [operator=](#) (const [DTRStatusType](#) &s)
- std::string [str](#) () const
- void [SetDesc](#) (const std::string &d)
- std::string [GetDesc](#) () const
- [DTRStatusType](#) [GetStatus](#) () const

## Static Public Attributes

- static const std::vector< [DTRStatus::DTRStatusType](#) > [ToProcessStates](#)
- static const std::vector< [DTRStatus::DTRStatusType](#) > [ProcessingStates](#)
- static const std::vector< [DTRStatus::DTRStatusType](#) > [StagedStates](#)

### 5.53.1 Detailed Description

Class representing the status of a [DTR](#).

### 5.53.2 Member Enumeration Documentation

#### 5.53.2.1 enum [DataStaging::DTRStatus::DTRStatusType](#)

Possible state values.

#### Enumerator:

**NEW** Just created.

**CHECK\_CACHE** Check the cache for the file may be already there.

**CHECKING\_CACHE** Checking the cache.

**CACHE\_WAIT** Cache file is locked, waiting for its release.

**CACHE\_CHECKED** Cache check completed.

**RESOLVE** Resolve a meta-protocol.

**RESOLVING** Resolving replicas.

**RESOLVED** Replica resolution completed.

**QUERY\_REPLICA** Query a replica.

**QUERYING\_REPLICA** Replica is being queried.

**REPLICA\_QUERIED** Replica was queried.

**PRE\_CLEAN** The destination should be deleted.

**PRE\_CLEANNING** Deleting the destination.

**PRE\_CLEANNED** The destination file has been deleted.

**STAGE\_PREPARE** Prepare or stage the source and/or destination.

**STAGING\_PREPARING** Making a staging or preparing request.

**STAGING\_PREPARING\_WAIT** Wait for the status of the staging/preparing request.

**STAGED\_PREPARED** Staging/preparing request completed.

**TRANSFER** Transfer ready and can be started.

**TRANSFERRING** Transfer is going.

**TRANSFERRING\_CANCEL** Transfer is on-going but scheduled for cancellation.

**TRANSFERRED** Transfer completed.

**RELEASE\_REQUEST** Transfer finished, release requests on the storage.

**RELEASING\_REQUEST** Releasing staging/preparing request.

**REQUEST\_RELEASED** Release of staging/preparing request completed.

**REGISTER\_REPLICA** Register a new replica of the destination.

**REGISTERING\_REPLICA** Registering a replica in an index service.

**REPLICA\_REGISTERED** Replica registration completed.

**PROCESS\_CACHE** Destination is cacheable, process cache.

**PROCESSING\_CACHE** Releasing locks and copying/linking cache files to the session dir.

**CACHE\_PROCESSED** Cache processing completed.

**DONE** Everything completed successfully.

**CANCELLED** Cancellation request fulfilled successfully.

**CANCELLED\_FINISHED** Cancellation request fulfilled but [DTR](#) also completed transfer successfully.

**ERROR** Error occurred.

**NULL\_STATE** "Stateless" [DTR](#)

### 5.53.3 Constructor & Destructor Documentation

**5.53.3.1** `DataStaging::DTRStatus::DTRStatus (const DTRStatusType & status, std::string desc = "")` `[inline]`

Make new [DTRStatus](#) with given status.

**5.53.3.2** `DataStaging::DTRStatus::DTRStatus ()` `[inline]`

Make new [DTRStatus](#) with default NEW status.

### 5.53.4 Member Function Documentation

#### 5.53.4.1 `std::string DataStaging::DTRStatus::GetDesc () const` [inline]

Get the detailed description of the current state.

#### 5.53.4.2 `DTRStatusType DataStaging::DTRStatus::GetStatus () const` [inline]

Get the DTRStatusType of the current state.

#### 5.53.4.3 `bool DataStaging::DTRStatus::operator!= (const DTRStatus & s) const` [inline]

Returns true if this status is not the same as the given [DTRStatus](#).

#### 5.53.4.4 `bool DataStaging::DTRStatus::operator!= (const DTRStatusType & s) const` [inline]

Returns true if this status is not the same as the given DTRStatusType.

#### 5.53.4.5 `DTRStatus& DataStaging::DTRStatus::operator= (const DTRStatusType & s)` [inline]

Make a new [DTRStatus](#) with the same status as the given DTRStatusType.

#### 5.53.4.6 `bool DataStaging::DTRStatus::operator== (const DTRStatus & s) const` [inline]

Returns true if this status is the same as the given [DTRStatus](#).

#### 5.53.4.7 `bool DataStaging::DTRStatus::operator== (const DTRStatusType & s) const` [inline]

Returns true if this status is the same as the given DTRStatusType.

#### 5.53.4.8 `void DataStaging::DTRStatus::SetDesc (const std::string & d)` [inline]

Set the detailed description of the current state.

#### 5.53.4.9 `std::string DataStaging::DTRStatus::str () const`

Returns a string representation of the current state.

### 5.53.5 Field Documentation

#### 5.53.5.1 `const std::vector<DTRStatus::DTRStatusType> DataStaging::DTRStatus::Processing-States` [static]

Vector of states with a processing action, eg CHECKING\_CACHE.

**5.53.5.2** `const std::vector<DTRStatus::DTRStatusType> DataStaging::DTRStatus::StagedStates`  
[static]

Vector of states where a [DTR](#) is staged - used to limit the number of staged files.

**5.53.5.3** `const std::vector<DTRStatus::DTRStatusType> DataStaging::DTRStatus::ToProcessStates`  
[static]

Vector of states with a to be processed action, eg CHECK\_CACHE.

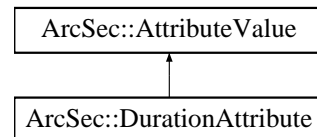
The documentation for this class was generated from the following file:

- DTRStatus.h

## 5.54 ArcSec::DurationAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::DurationAttribute::



### Public Member Functions

- virtual bool [equal](#) ([AttributeValue](#) \*other, bool check\_id=true)
- virtual std::string [encode](#) ()
- virtual std::string [getType](#) ()
- virtual std::string [getId](#) ()

#### 5.54.1 Detailed Description

Formate: P??Y??M??DT??H??M??S

#### 5.54.2 Member Function Documentation

##### 5.54.2.1 virtual std::string ArcSec::DurationAttribute::encode () [virtual]

encode the value in a string format

Implements [ArcSec::AttributeValue](#).

##### 5.54.2.2 virtual bool ArcSec::DurationAttribute::equal ([AttributeValue](#) \* other, bool check\_id = true) [virtual]

Evaluate whether "this" equale to the parameter value

Implements [ArcSec::AttributeValue](#).

##### 5.54.2.3 virtual std::string ArcSec::DurationAttribute::getId () [inline, virtual]

Get the AttributeId of the <Attribute>

Implements [ArcSec::AttributeValue](#).

##### 5.54.2.4 virtual std::string ArcSec::DurationAttribute::getType () [inline, virtual]

Get the DataType of the <Attribute>

Implements [ArcSec::AttributeValue](#).

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

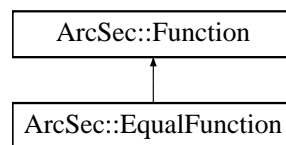
- [DateTimeAttribute.h](#)

## 5.55 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

```
#include <EqualFunction.h>
```

Inheritance diagram for ArcSec::EqualFunction::



### Public Member Functions

- virtual [AttributeValue](#) \* [evaluate](#) ([AttributeValue](#) \*arg0, [AttributeValue](#) \*arg1, bool check\_id=true)
- virtual std::list< [AttributeValue](#) \* > [evaluate](#) (std::list< [AttributeValue](#) \* > args, bool check\_id=true)

### Static Public Member Functions

- static std::string [getFunctionName](#) (std::string datatype)

#### 5.55.1 Detailed Description

Evaluate whether the two values are equal.

#### 5.55.2 Member Function Documentation

**5.55.2.1** virtual std::list<[AttributeValue](#)\*> ArcSec::EqualFunction::evaluate (std::list<[AttributeValue](#) \* > args, bool check\_id = true) [virtual]

Evaluate a list of [AttributeValue](#) objects, and return a list of Attribute objects

Implements [ArcSec::Function](#).

**5.55.2.2** virtual [AttributeValue](#)\* ArcSec::EqualFunction::evaluate ([AttributeValue](#) \* arg0, [AttributeValue](#) \* arg1, bool check\_id = true) [virtual]

Evaluate two [AttributeValue](#) objects, and return one [AttributeValue](#) object

Implements [ArcSec::Function](#).

**5.55.2.3** static std::string ArcSec::EqualFunction::getFunctionName (std::string datatype) [static]

help function to get the FunctionName

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



- [EqualFunction.h](#)

## 5.56 ArcSec::EvalResult Struct Reference

Struct to record the xml node and effect, which will be used by [Evaluator](#) to get the information about which rule/policy(in xmlnode) is satisfied.

```
#include <Result.h>
```

### 5.56.1 Detailed Description

Struct to record the xml node and effect, which will be used by [Evaluator](#) to get the information about which rule/policy(in xmlnode) is satisfied.

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

- Result.h

## 5.57 ArcSec::EvaluationCtx Class Reference

[EvaluationCtx](#), in charge of storing some context information for.

```
#include <EvaluationCtx.h>
```

### Public Member Functions

- [EvaluationCtx](#) ([Request](#) \*request)

#### 5.57.1 Detailed Description

[EvaluationCtx](#), in charge of storing some context information for.

#### 5.57.2 Constructor & Destructor Documentation

##### 5.57.2.1 ArcSec::EvaluationCtx::EvaluationCtx ([Request](#) \*request) [inline]

Construct a new [EvaluationCtx](#) based on the given request

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

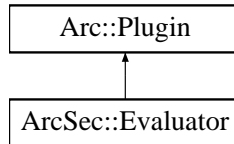
- [EvaluationCtx.h](#)

## 5.58 ArcSec::Evaluator Class Reference

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

```
#include <Evaluator.h>
```

Inheritance diagram for ArcSec::Evaluator::



### Public Member Functions

- virtual [Response](#) \* [evaluate](#) ([Request](#) \*request)=0
- virtual [Response](#) \* [evaluate](#) (const [Source](#) &request)=0
- virtual [Response](#) \* [evaluate](#) ([Request](#) \*request, const [Source](#) &policy)=0
- virtual [Response](#) \* [evaluate](#) (const [Source](#) &request, const [Source](#) &policy)=0
- virtual [Response](#) \* [evaluate](#) ([Request](#) \*request, [Policy](#) \*policyobj)=0
- virtual [Response](#) \* [evaluate](#) (const [Source](#) &request, [Policy](#) \*policyobj)=0
- virtual [AttributeFactory](#) \* [getAttrFactory](#) ()=0
- virtual [FnFactory](#) \* [getFnFactory](#) ()=0
- virtual [AlgFactory](#) \* [getAlgFactory](#) ()=0
- virtual void [addPolicy](#) (const [Source](#) &policy, const std::string &id="")=0
- virtual void [addPolicy](#) ([Policy](#) \*policy, const std::string &id="")=0
- virtual void [setCombiningAlg](#) ([EvaluatorCombiningAlg](#) alg)=0
- virtual void [setCombiningAlg](#) ([CombiningAlg](#) \*alg=NULL)=0
- virtual const char \* [getName](#) (void) const =0

### Protected Member Functions

- virtual [Response](#) \* [evaluate](#) ([EvaluationCtx](#) \*ctx)=0

#### 5.58.1 Detailed Description

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

#### 5.58.2 Member Function Documentation

##### 5.58.2.1 virtual void ArcSec::Evaluator::addPolicy ([Policy](#) \*policy, const std::string &id = "") [pure virtual]

Add policy to the evaluator. [Policy](#) will be marked with id. The policy object is taken over by this instance and will be destroyed in destructor.

**5.58.2.2** `virtual void ArcSec::Evaluator::addPolicy (const Source & policy, const std::string & id = "")` [pure virtual]

Add policy from specified source to the evaluator. Policy will be marked with id.

**5.58.2.3** `virtual Response* ArcSec::Evaluator::evaluate (EvaluationCtx * ctx)` [protected, pure virtual]

Evaluate the request by using the EvaluationCtx object (which includes the information about request). The ctx is destroyed inside this method (why?!?!?).

**5.58.2.4** `virtual Response* ArcSec::Evaluator::evaluate (const Source & request, Policy * policyobj)` [pure virtual]

Evaluate the request from specified source against the specified policy. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

**5.58.2.5** `virtual Response* ArcSec::Evaluator::evaluate (Request * request, Policy * policyobj)` [pure virtual]

Evaluate the specified request against the specified policy. In some implementations all of the existing policy inside the evaluator may be destroyed by this method.

**5.58.2.6** `virtual Response* ArcSec::Evaluator::evaluate (const Source & request, const Source & policy)` [pure virtual]

Evaluate the request from specified source against the policy from specified source. In some implementations all of the existing policie inside the evaluator may be destroyed by this method.

**5.58.2.7** `virtual Response* ArcSec::Evaluator::evaluate (Request * request, const Source & policy)` [pure virtual]

Evaluate the specified request against the policy from specified source. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

**5.58.2.8** `virtual Response* ArcSec::Evaluator::evaluate (const Source & request)` [pure virtual]

Evaluates the request by using a specified source

**5.58.2.9** `virtual Response* ArcSec::Evaluator::evaluate (Request * request)` [pure virtual]

Evaluates the request by using a Request object. Evaluation is done till at least one of policies is satisfied.

**5.58.2.10** `virtual AlgFactory* ArcSec::Evaluator::getAlgFactory ()` [pure virtual]

Get the AlgFactory object

**5.58.2.11** `virtual AttributeFactory* ArcSec::Evaluator::getAttrFactory ()` [pure virtual]

Get the [AttributeFactory](#) object

**5.58.2.12** `virtual FnFactory* ArcSec::Evaluator::getFnFactory ()` [pure virtual]

Get the [FnFactory](#) object

**5.58.2.13** `virtual const char* ArcSec::Evaluator::getName (void) const` [pure virtual]

Get the name of this evaluator

**5.58.2.14** `virtual void ArcSec::Evaluator::setCombiningAlg (CombiningAlg * alg = NULL)`  
[pure virtual]

Specifies loadable combining algorithms. In case of multiple policies their results will be combined using this algorithm. To switch to simple algorithm specify NULL argument.

**5.58.2.15** `virtual void ArcSec::Evaluator::setCombiningAlg (EvaluatorCombiningAlg alg)`  
[pure virtual]

Specifies one of simple combining algorithms. In case of multiple policies their results will be combined using this algorithm.

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

- Evaluator.h

## 5.59 ArcSec::EvaluatorContext Class Reference

Context for evaluator. It includes the factories which will be used to create related objects.

```
#include <Evaluator.h>
```

### Public Member Functions

- [operator AttributeFactory \\* \(\)](#)
- [operator FnFactory \\* \(\)](#)
- [operator AlgFactory \\* \(\)](#)

### 5.59.1 Detailed Description

Context for evaluator. It includes the factories which will be used to create related objects.

### 5.59.2 Member Function Documentation

#### 5.59.2.1 ArcSec::EvaluatorContext::operator [AlgFactory](#) \* () [inline]

Returns associated [AlgFactory](#) object

#### 5.59.2.2 ArcSec::EvaluatorContext::operator [AttributeFactory](#) \* () [inline]

Returns associated [AttributeFactory](#) object

#### 5.59.2.3 ArcSec::EvaluatorContext::operator [FnFactory](#) \* () [inline]

Returns associated [FnFactory](#) object

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

- [Evaluator.h](#)

## 5.60 ArcSec::EvaluatorLoader Class Reference

[EvaluatorLoader](#) is implemented as a helper class for loading different [Evaluator](#) objects, like ArcEvaluator.

```
#include <EvaluatorLoader.h>
```

### Public Member Functions

- [Evaluator](#) \* [getEvaluator](#) (const std::string &classname)
- [Evaluator](#) \* [getEvaluator](#) (const [Policy](#) \*policy)
- [Evaluator](#) \* [getEvaluator](#) (const [Request](#) \*request)
- [Request](#) \* [getRequest](#) (const std::string &classname, const [Source](#) &requestsource)
- [Request](#) \* [getRequest](#) (const [Source](#) &requestsource)
- [Policy](#) \* [getPolicy](#) (const std::string &classname, const [Source](#) &polycysource)
- [Policy](#) \* [getPolicy](#) (const [Source](#) &polycysource)

### 5.60.1 Detailed Description

[EvaluatorLoader](#) is implemented as a helper class for loading different [Evaluator](#) objects, like ArcEvaluator.

The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

### 5.60.2 Member Function Documentation

#### 5.60.2.1 [Evaluator](#)\* ArcSec::EvaluatorLoader::getEvaluator (const [Request](#) \* *request*)

Get evaluator object suitable for presented request

#### 5.60.2.2 [Evaluator](#)\* ArcSec::EvaluatorLoader::getEvaluator (const [Policy](#) \* *policy*)

Get evaluator object suitable for presented policy

#### 5.60.2.3 [Evaluator](#)\* ArcSec::EvaluatorLoader::getEvaluator (const std::string & *classname*)

Get evaluator object according to the class name

#### 5.60.2.4 [Policy](#)\* ArcSec::EvaluatorLoader::getPolicy (const [Source](#) & *polycysource*)

Get proper policy object according to the policy source

#### 5.60.2.5 [Policy](#)\* ArcSec::EvaluatorLoader::getPolicy (const std::string & *classname*, const [Source](#) & *polycysource*)

Get policy object according to the class name, based on the policy source



**5.60.2.6 Request\*** ArcSec::EvaluatorLoader::getRequest (const **Source** & *requestsource*)

Get request object according to the request source

**5.60.2.7 Request\*** ArcSec::EvaluatorLoader::getRequest (const std::string & *classname*, const **Source** & *requestsource*)

Get request object according to the class name, based on the request source

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

- EvaluatorLoader.h

## 5.61 Arc::ExecutableType Class Reference

Executable.

```
#include <JobDescription.h>
```

### Data Fields

- `std::string` [Path](#)
- `std::list< std::string >` [Argument](#)
- `std::pair< bool, int >` [SuccessExitCode](#)

### 5.61.1 Detailed Description

Executable.

The [ExecutableType](#) class is used to specify path to an executable, arguments to pass to it when invoked and the exit code for successful execution.

NOTE: The Name string member has been renamed to Path.

### 5.61.2 Field Documentation

#### 5.61.2.1 `std::list<std::string>` [Arc::ExecutableType::Argument](#)

List of arguments to executable.

The Argument list is used to specify arguments which should be passed to the executable upon invocation.

#### 5.61.2.2 `std::string` [Arc::ExecutableType::Path](#)

Path to executable.

The Path string should specify the path to an executable. Note that some implementations might only accept a relative path, while others might also accept a absolute one.

#### 5.61.2.3 `std::pair<bool, int>` [Arc::ExecutableType::SuccessExitCode](#)

Exit code at successful execution.

The SuccessExitCode pair is used to specify the exit code returned by the executable in case of successful execution. For some scenarios the exit code returned by the executable should be ignored, which is specified by setting the first member of this object to false. If the exit code should be used for validation at the execution service, the first member of pair must be set to true, while the second member should be the exit code returned at successful execution.

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

- JobDescription.h

## 5.62 Arc::ExecutionTarget Class Reference

[ExecutionTarget](#).

```
#include <ExecutionTarget.h>
```

### Public Member Functions

- [ExecutionTarget](#) ()
- [ExecutionTarget](#) (const [ExecutionTarget](#) &t)
- [ExecutionTarget](#) (long int addrptr)
- void [RegisterJobSubmission](#) (const [JobDescription](#) &jobdesc) const
- void [SaveToStream](#) (std::ostream &out, bool longlist) const

### 5.62.1 Detailed Description

[ExecutionTarget](#).

This class describe a target which accept computing jobs. All of the members contained in this class, with a few exceptions, are directly linked to attributes defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

### 5.62.2 Constructor & Destructor Documentation

#### 5.62.2.1 Arc::ExecutionTarget::ExecutionTarget () [inline]

Create an [ExecutionTarget](#).

Default constructor to create an [ExecutionTarget](#). Takes no arguments.

#### 5.62.2.2 Arc::ExecutionTarget::ExecutionTarget (const [ExecutionTarget](#) &t) [inline]

Create an [ExecutionTarget](#).

Copy constructor.

##### Parameters:

*target* [ExecutionTarget](#) to copy.

#### 5.62.2.3 Arc::ExecutionTarget::ExecutionTarget (long int *addrptr*) [inline]

Create an [ExecutionTarget](#).

Copy constructor? Needed from Python?

##### Parameters:

*addrptr*

### 5.62.3 Member Function Documentation

#### 5.62.3.1 `void Arc::ExecutionTarget::RegisterJobSubmission (const JobDescription & jobdesc) const`

Update [ExecutionTarget](#) after succesful job submission.

Method to update the [ExecutionTarget](#) after a job succesfully has been submitted to the computing resource it represents. E.g. if a job is sent to the computing resource and is expected to enter the queue, then the `WaitingJobs` attribute is incremented with 1.

**Parameters:**

*jobdesc* contains all information about the job submitted.

#### 5.62.3.2 `void Arc::ExecutionTarget::SaveToStream (std::ostream & out, bool longlist) const`

Print the [ExecutionTarget](#) information to a `std::ostream` object.

Method to print the [ExecutionTarget](#) attributes to a `std::ostream` object.

**Parameters:**

*out* is the `std::ostream` to print the attributes to.

*longlist* should be set to true for printing a long list.

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

- `ExecutionTarget.h`

## 5.63 Arc::ExpirationReminder Class Reference

A class intended for internal use within counters.

```
#include <Counter.h>
```

### Public Member Functions

- bool [operator<](#) (const [ExpirationReminder](#) &other) const
- Glib::TimeVal [getExpiryTime](#) () const
- Counter::IDType [getReservationID](#) () const

### Friends

- class [Counter](#)

### 5.63.1 Detailed Description

A class intended for internal use within counters.

This class is used for "reminder objects" that are used for automatic deallocation of self-expiring reservations.

### 5.63.2 Member Function Documentation

#### 5.63.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime () const

Returns the expiry time.

This method returns the expiry time of the reservation that this [ExpirationReminder](#) is associated with.

#### Returns:

The expiry time.

#### 5.63.2.2 Counter::IDType Arc::ExpirationReminder::getReservationID () const

Returns the identification number of the reservation.

This method returns the identification number of the self-expiring reservation that this [ExpirationReminder](#) is associated with.

#### Returns:

The identification number.

#### 5.63.2.3 bool Arc::ExpirationReminder::operator< (const [ExpirationReminder](#) &other) const

Less than operator, compares "soonness".

This is the less than operator for the [ExpirationReminder](#) class. It compares the priority of such objects with respect to which reservation expires first. It is used when reminder objects are inserted in a priority queue in order to always place the next reservation to expire at the top.

### 5.63.3 Friends And Related Function Documentation

#### 5.63.3.1 friend class [Counter](#) [`friend`]

The [Counter](#) class needs to be a friend.

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

- Counter.h

## 5.64 Arc::FileAccess Class Reference

Defines interface for accessing filesystems.

```
#include <FileAccess.h>
```

### Public Member Functions

- bool [ping](#) (void)
- bool [setuid](#) (int uid, int gid)
- bool [mkdir](#) (const std::string &path, mode\_t mode)
- bool [mkdirp](#) (const std::string &path, mode\_t mode)
- bool [link](#) (const std::string &oldpath, const std::string &newpath)
- bool [softlink](#) (const std::string &oldpath, const std::string &newpath)
- bool [copy](#) (const std::string &oldpath, const std::string &newpath, mode\_t mode)
- bool [chmod](#) (const std::string &path, mode\_t mode)
- bool [stat](#) (const std::string &path, struct stat &st)
- bool [lstat](#) (const std::string &path, struct stat &st)
- bool [fstat](#) (struct stat &st)
- bool [ftruncate](#) (off\_t length)
- off\_t [fallocate](#) (off\_t length)
- bool [readlink](#) (const std::string &path, std::string &linkpath)
- bool [remove](#) (const std::string &path)
- bool [unlink](#) (const std::string &path)
- bool [rmdir](#) (const std::string &path)
- bool [rmdirr](#) (const std::string &path)
- bool [opendir](#) (const std::string &path)
- bool [closedir](#) (void)
- bool [readdir](#) (std::string &name)
- bool [open](#) (const std::string &path, int flags, mode\_t mode)
- bool [close](#) (void)
- bool [mkstemp](#) (std::string &path, mode\_t mode)
- off\_t [lseek](#) (off\_t offset, int whence)
- ssize\_t [read](#) (void \*buf, size\_t size)
- ssize\_t [write](#) (const void \*buf, size\_t size)
- ssize\_t [pread](#) (void \*buf, size\_t size, off\_t offset)
- ssize\_t [pwrite](#) (const void \*buf, size\_t size, off\_t offset)
- int [geterrno](#) ()
- [operator bool](#) (void)
- bool [operator!](#) (void)

### Static Public Member Functions

- static void [testtune](#) (void)

### Data Structures

- struct [header\\_t](#)

### 5.64.1 Detailed Description

Defines interface for accessing filesystems.

This class accesses local filesystem through proxy executable which allows to switch user id in multi-threaded systems without introducing conflict with other threads. Its methods are mostly replicas of corresponding POSIX functions with some convenience tweaking.

### 5.64.2 Member Function Documentation

#### 5.64.2.1 `bool Arc::FileAccess::chmod (const std::string & path, mode_t mode)`

Change mode of filesystem object.

#### 5.64.2.2 `bool Arc::FileAccess::close (void)`

Close open file.

#### 5.64.2.3 `bool Arc::FileAccess::closedir (void)`

Close open directory.

#### 5.64.2.4 `bool Arc::FileAccess::copy (const std::string & oldpath, const std::string & newpath, mode_t mode)`

Copy file to new location. If new file is created it is assigned specified mode.

#### 5.64.2.5 `off_t Arc::FileAccess::fallocate (off_t length)`

Allocate disk space for open file.

#### 5.64.2.6 `bool Arc::FileAccess::fstat (struct stat & st)`

stat open file.

#### 5.64.2.7 `bool Arc::FileAccess::ftruncate (off_t length)`

Truncate open file.

#### 5.64.2.8 `int Arc::FileAccess::geterrno () [inline]`

Get errno of last operation. Every operation resets errno.

#### 5.64.2.9 `bool Arc::FileAccess::link (const std::string & oldpath, const std::string & newpath)`

Create hard link.



**5.64.2.10** `off_t Arc::FileAccess::lseek (off_t offset, int whence)`

Change current position in open file.

**5.64.2.11** `bool Arc::FileAccess::lstat (const std::string & path, struct stat & st)`

stat symbolic link or file.

**5.64.2.12** `bool Arc::FileAccess::mkdir (const std::string & path, mode_t mode)`

Make a directory and assign it specified mode.

**5.64.2.13** `bool Arc::FileAccess::mkdirp (const std::string & path, mode_t mode)`

Make a directory and assign it specified mode. If missing all intermediate directories are created too.

**5.64.2.14** `bool Arc::FileAccess::mkstemp (std::string & path, mode_t mode)`

Open new temporary file for writing. On input path contains template of file name ending with XXXXXX. On output path is path to created file.

**5.64.2.15** `bool Arc::FileAccess::open (const std::string & path, int flags, mode_t mode)`

Open file. Only one file may be open at a time.

**5.64.2.16** `bool Arc::FileAccess::opendir (const std::string & path)`

Open directory. Only one directory may be open at a time.

**5.64.2.17** `Arc::FileAccess::operator bool (void) [inline]`

Returns true if this instance is in useful condition.

**5.64.2.18** `bool Arc::FileAccess::operator! (void) [inline]`

Returns true if this instance is not in useful condition.

**5.64.2.19** `bool Arc::FileAccess::ping (void)`

Check if communication with proxy works.

**5.64.2.20** `ssize_t Arc::FileAccess::pread (void * buf, size_t size, off_t offset)`

Read from open file at specified offset.

**5.64.2.21** `ssize_t Arc::FileAccess::pwrite (const void * buf, size_t size, off_t offset)`

Write to open file at specified offset.

**5.64.2.22** `ssize_t Arc::FileAccess::read (void * buf, size_t size)`

Read from open file.

**5.64.2.23** `bool Arc::FileAccess::readdir (std::string & name)`

Read relative name of object in open directory.

**5.64.2.24** `bool Arc::FileAccess::readlink (const std::string & path, std::string & linkpath)`

Read content of symbolic link.

**5.64.2.25** `bool Arc::FileAccess::remove (const std::string & path)`

Remove file system object.

**5.64.2.26** `bool Arc::FileAccess::rmdir (const std::string & path)`

Remove directory (if empty).

**5.64.2.27** `bool Arc::FileAccess::rmdirr (const std::string & path)`

Remove directory recursively.

**5.64.2.28** `bool Arc::FileAccess::setuid (int uid, int gid)`

Modify user uid and gid. If any is set to 0 then executable is switched to original uid/gid.

**5.64.2.29** `bool Arc::FileAccess::softlink (const std::string & oldpath, const std::string & newpath)`

Create symbolic (aka soft) link.

**5.64.2.30** `bool Arc::FileAccess::stat (const std::string & path, struct stat & st)`

stat file.

**5.64.2.31** `static void Arc::FileAccess::testtune (void)` `[static]`

Special method for using in unit tests.

**5.64.2.32** `bool Arc::FileAccess::unlink (const std::string & path)`

Remove file.

**5.64.2.33** `ssize_t Arc::FileAccess::write (const void * buf, size_t size)`

Write to open file.

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

- FileAccess.h

## 5.65 Arc::FileLock Class Reference

A general file locking class.

```
#include <FileLock.h>
```

### Public Member Functions

- [FileLock](#) (const std::string &filename, unsigned int timeout=[DEFAULT\\_LOCK\\_TIMEOUT](#), bool use\_pid=true)
- bool [acquire](#) (bool &lock\_removed)
- bool [acquire](#) ()
- bool [release](#) (bool force=false)
- int [check](#) (bool log\_error=true)

### Static Public Member Functions

- static std::string [getLockSuffix](#) ()

### Static Public Attributes

- static const int [DEFAULT\\_LOCK\\_TIMEOUT](#)
- static const std::string [LOCK\\_SUFFIX](#)

#### 5.65.1 Detailed Description

A general file locking class.

This class can be used when protected access is required to files which are used by multiple processes or threads. Call [acquire\(\)](#) to obtain a lock and [release\(\)](#) to release it when finished. [check\(\)](#) can be used to verify if a lock is valid for the current process. Locks are independent of [FileLock](#) objects - locks are only created and destroyed through [acquire\(\)](#) and [release\(\)](#), not on creation or destruction of [FileLock](#) objects.

Unless use\_pid is set false, the process ID and hostname of the calling process are stored in a file filename.lock in the form pid. This information is used to determine whether a lock is still valid. It is also possible to specify a timeout on the lock.

To ensure an atomic locking operation, [acquire\(\)](#) first creates a temporary lock file filename.lock.XXXXXXX, then attempts to rename this file to filename.lock. After a successful rename the lock file is checked to make sure the correct process ID and hostname are inside. This eliminates race conditions where multiple processes compete to obtain the lock.

#### 5.65.2 Constructor & Destructor Documentation

##### 5.65.2.1 Arc::FileLock::FileLock (const std::string &filename, unsigned int timeout = [DEFAULT\\_LOCK\\_TIMEOUT](#), bool use\_pid = true)

Create a new [FileLock](#) object.

#### Parameters:

*filename* The name of the file to be locked

*timeout* The timeout of the lock

*use\_pid* If true, use process id in the lock and to determine lock validity

### 5.65.3 Member Function Documentation

#### 5.65.3.1 bool Arc::FileLock::acquire ()

Acquire the lock.

Callers can use this version of [acquire\(\)](#) if they do not care whether an invalid lock was removed in the process of obtaining the lock.

#### 5.65.3.2 bool Arc::FileLock::acquire (bool & *lock\_removed*)

Acquire the lock.

Returns true if the lock was acquired successfully. Locks are acquired if no lock file currently exists, or if the current lock file is invalid. A lock is invalid if the process ID inside the lock no longer exists on the host inside the lock, or the age of the lock file is greater than the lock timeout.

##### Parameters:

*lock\_removed* Set to true if an existing lock was removed due to being invalid. In this case the caller may decide to check or delete the file as it is potentially corrupted.

##### Returns:

True if lock is successfully acquired

#### 5.65.3.3 int Arc::FileLock::check (bool *log\_error* = true)

Check the lock is valid.

Returns 0 if the lock is valid for the current process, the pid inside the lock if the lock is owned by another process on the same host, and -1 if the lock is owned by another host or any other error occurred. *log\_error* may be set to false to log error messages at INFO level, in cases where the lock not existing or being owned by another host are not errors.

#### 5.65.3.4 static std::string Arc::FileLock::getLockSuffix () [static]

Get the lock suffix used.

#### 5.65.3.5 bool Arc::FileLock::release (bool *force* = false)

Release the lock.

##### Parameters:

*force* Remove the lock without checking ownership or timeout

## 5.65.4 Field Documentation

**5.65.4.1** `const int Arc::FileLock::DEFAULT_LOCK_TIMEOUT` `[static]`

Default timeout for a lock.

**5.65.4.2** `const std::string Arc::FileLock::LOCK_SUFFIX` `[static]`

Suffix added to file name to make lock file.

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

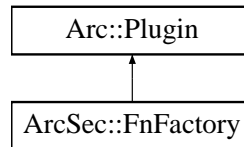
- FileLock.h

## 5.66 ArcSec::FnFactory Class Reference

Interface for function factory class.

```
#include <FnFactory.h>
```

Inheritance diagram for ArcSec::FnFactory::



### Public Member Functions

- virtual [Function](#) \* [createFn](#) (const std::string &type)=0

#### 5.66.1 Detailed Description

Interface for function factory class.

[FnFactory](#) is in charge of creating [Function](#) object according to the algorithm type given as argument of method [createFn](#). This class can be inherited for implementing a factory class which can create some specific [Function](#) objects.

#### 5.66.2 Member Function Documentation

**5.66.2.1** virtual [Function](#)\* ArcSec::FnFactory::createFn (const std::string & *type*) [pure virtual]

creat algorithm object based on the type algorithm type

##### Parameters:

*type* The type of [Function](#)

##### Returns:

The object of [Function](#)

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

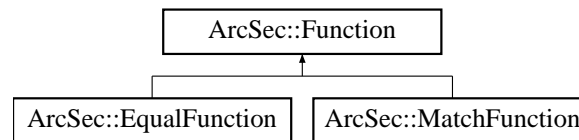
- FnFactory.h

## 5.67 ArcSec::Function Class Reference

Interface for function, which is in charge of evaluating two [AttributeValue](#).

```
#include <Function.h>
```

Inheritance diagram for ArcSec::Function::



### Public Member Functions

- virtual [AttributeValue](#) \* **evaluate** ([AttributeValue](#) \*arg0, [AttributeValue](#) \*arg1, bool check\_id=true)=0
- virtual std::list< [AttributeValue](#) \* > **evaluate** (std::list< [AttributeValue](#) \* > args, bool check\_id=true)=0

### 5.67.1 Detailed Description

Interface for function, which is in charge of evaluating two [AttributeValue](#).

### 5.67.2 Member Function Documentation

**5.67.2.1** virtual std::list<[AttributeValue](#)\*> ArcSec::Function::evaluate (std::list< [AttributeValue](#) \* > args, bool check\_id = true) [pure virtual]

Evaluate a list of [AttributeValue](#) objects, and return a list of Attribute objects

Implemented in [ArcSec::EqualFunction](#), and [ArcSec::MatchFunction](#).

**5.67.2.2** virtual [AttributeValue](#)\* ArcSec::Function::evaluate ([AttributeValue](#) \* arg0, [AttributeValue](#) \* arg1, bool check\_id = true) [pure virtual]

Evaluate two [AttributeValue](#) objects, and return one [AttributeValue](#) object

Implemented in [ArcSec::EqualFunction](#), and [ArcSec::MatchFunction](#).

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

- Function.h

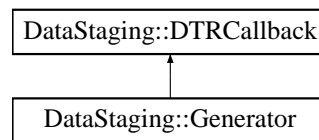


## 5.68 DataStaging::Generator Class Reference

Simple [Generator](#) implementation.

```
#include <Generator.h>
```

Inheritance diagram for DataStaging::Generator::



### Public Member Functions

- virtual void [receiveDTR](#) ([DTR\\_ptr](#) dtr)
- void [run](#) (const std::string &source, const std::string &destination)

### 5.68.1 Detailed Description

Simple [Generator](#) implementation.

This [Generator](#) implementation is included in the data staging library for for basic direct testing of the library and to show how a [Generator](#) can be written. It has one method, [run\(\)](#), which creates a single [DTR](#) and submits it to the [Scheduler](#).

### 5.68.2 Member Function Documentation

#### 5.68.2.1 virtual void DataStaging::Generator::receiveDTR ([DTR\\_ptr](#) dtr) [virtual]

Implementation of callback from [DTRCallback](#).

Callback method used when [DTR](#) processing is complete to pass back to the generator. The [DTR](#) is passed by value so that the scheduler can delete its copy of the object after calling this method.

Implements [DataStaging::DTRCallback](#).

#### 5.68.2.2 void DataStaging::Generator::run (const std::string & *source*, const std::string & *destination*)

Submit a [DTR](#) with given source and destination.

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

- Generator.h

## 5.69 Arc::GLUE2 Class Reference

GLUE2 parser.

```
#include <GLUE2.h>
```

### Static Public Member Functions

- static void [ParseExecutionTargets](#) ([XMLNode](#) glue2tree, std::list< ComputingServiceType > &targets)

#### 5.69.1 Detailed Description

GLUE2 parser.

This class parses [GLUE2](#) information rendered in XML and transfers information into various classes representing different types of objects which [GLUE2](#) information model can describe. This parser uses GLUE Specification v. 2.0 (GFD-R-P.147).

#### 5.69.2 Member Function Documentation

##### 5.69.2.1 static void Arc::GLUE2::ParseExecutionTargets ([XMLNode](#) glue2tree, std::list< ComputingServiceType > & targets) [static]

Parses ComputingService elements of [GLUE2](#) into ComputingServiceType objects. The glue2tree is either XML tree representing ComputingService object directly or ComputingService objects are immediate children of it. On exit targets contains ComputingServiceType objects found inside glue2tree. If targets contained any objects on entry those are not destroyed.

##### Parameters:

*glue2tree*

*targets*

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

- GLUE2.h

## 5.70 Arc::InfoCache Class Reference

Stores XML document in filesystem split into parts.

```
#include <InfoCache.h>
```

### Public Member Functions

- [InfoCache](#) (const [Config](#) &cfg, const std::string &service\_id)

#### 5.70.1 Detailed Description

Stores XML document in filesystem split into parts.

#### 5.70.2 Constructor & Destructor Documentation

##### 5.70.2.1 Arc::InfoCache::InfoCache (const [Config](#) & *cfg*, const std::string & *service\_id*)

Creates object according to configuration (see InfoCacheConfig.xsd).

XML configuration is passed in *cfg*. Argument *service\_id* is used to distinguish between various documents stored under same path - corresponding files will be stored in subdirectory with *service\_id* name.

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

- InfoCache.h

## 5.71 Arc::InfoFilter Class Reference

Filters information document according to identity of requestor.

```
#include <InfoFilter.h>
```

### Public Member Functions

- [InfoFilter](#) ([MessageAuth](#) &id)
- bool [Filter](#) ([XMLNode](#) doc) const
- bool [Filter](#) ([XMLNode](#) doc, const InfoFilterPolicies &policies, const NS &ns) const

### 5.71.1 Detailed Description

Filters information document according to identity of requestor.

Identity is compared to policies stored inside information document and external ones. Parts of document which do not pass policy evaluation are removed.

### 5.71.2 Constructor & Destructor Documentation

#### 5.71.2.1 Arc::InfoFilter::InfoFilter ([MessageAuth](#) &id)

Creates object and associates identity.

Associated identity is not copied, hence passed argument must not be destroyed while this method is used.

### 5.71.3 Member Function Documentation

#### 5.71.3.1 bool Arc::InfoFilter::Filter ([XMLNode](#) doc, const InfoFilterPolicies &policies, const NS &ns) const

Filter information document according to internal and external policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed. External policies are provided in policies argument. First element of every pair is XPath defining to which XML node policy must be applied. Second element is policy itself. Argument ns defines XML namespaces for XPath evaluation.

#### 5.71.3.2 bool Arc::InfoFilter::Filter ([XMLNode](#) doc) const

Filter information document according to internal policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed.

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

- InfoFilter.h

## 5.72 Arc::InfoRegister Class Reference

Registration to ISIS interface.

```
#include <InfoRegister.h>
```

### 5.72.1 Detailed Description

Registration to ISIS interface.

This class represents service registering to Information Indexing [Service](#). It does not perform registration itself. It only collects configuration information. Configuration is as described in InfoRegisterConfig.xsd for element InfoRegistration.

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

- InfoRegister.h

## 5.73 Arc::InfoRegisterContainer Class Reference

```
#include <InfoRegister.h>
```

### Public Member Functions

- [InfoRegistrar](#) \* [addRegistrar](#) ([XMLNode](#) doc)
- void [addService](#) ([InfoRegister](#) \*reg, const std::list< std::string > &ids, [XMLNode](#) cfg=[XMLNode](#)())
- void [removeService](#) ([InfoRegister](#) \*reg)

### 5.73.1 Detailed Description

Singleton class for scanning configuration and storing references to registration elements.

### 5.73.2 Member Function Documentation

#### 5.73.2.1 [InfoRegistrar](#)\* Arc::InfoRegisterContainer::addRegistrar ([XMLNode](#) doc)

Adds ISISes to list of handled services.

Supplied configuration document is scanned for [InfoRegistrar](#) elements and those are turned into [InfoRegistrar](#) classes for handling connection to ISIS service each.

#### 5.73.2.2 void Arc::InfoRegisterContainer::addService ([InfoRegister](#) \* reg, const std::list< std::string > &ids, [XMLNode](#) cfg = [XMLNode](#) ())

Adds service to list of handled.

This method must be called first time after last addRegistrar was called - services will be only associated with ISISes which are already added. Argument ids contains list of ISIS identifiers to which service is associated. If ids is empty then service is associated to all ISISes currently added. If argument cfg is available and no ISISes are configured then addRegistrars is called with cfg used as configuration document.

#### 5.73.2.3 void Arc::InfoRegisterContainer::removeService ([InfoRegister](#) \* reg)

This method must be called if service being destroyed.

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

- InfoRegister.h

## 5.74 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

```
#include <InfoRegister.h>
```

### Public Member Functions

- [InfoRegisters](#) ([XMLNode](#) cfg, [Service](#) \*service)
- bool [addRegister](#) ([XMLNode](#) cfg, [Service](#) \*service)

#### 5.74.1 Detailed Description

Handling multiple registrations to ISISes.

#### 5.74.2 Constructor & Destructor Documentation

##### 5.74.2.1 Arc::InfoRegisters::InfoRegisters ([XMLNode](#) cfg, [Service](#) \* service)

Constructor creates [InfoRegister](#) objects according to configuration.

Inside cfg elements [InfoRegister](#) are found and for each corresponding [InfoRegister](#) object is created. Those objects are destroyed in destructor of this class.

#### 5.74.3 Member Function Documentation

##### 5.74.3.1 bool Arc::InfoRegisters::addRegister ([XMLNode](#) cfg, [Service](#) \* service)

Dinamically add one more [InfoRegister](#) object.

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

- InfoRegister.h

## 5.75 Arc::InfoRegistrar Class Reference

Registration process associated with particular ISIS.

```
#include <InfoRegister.h>
```

### Public Member Functions

- void [registration](#) (void)
- bool [addService](#) ([InfoRegister \\*](#), [XMLNode](#))
- bool [removeService](#) ([InfoRegister \\*](#))

### 5.75.1 Detailed Description

Registration process associated with particular ISIS.

Instance of this class starts thread which takes care passing information about associated services to ISIS service defined in configuration. Configuration is as described in InfoRegister.xsd for element [InfoRegistrar](#).

### 5.75.2 Member Function Documentation

#### 5.75.2.1 bool Arc::InfoRegistrar::addService ([InfoRegister \\*](#), [XMLNode](#))

Adds new service to list of handled services.

[Service](#) is described by it's [InfoRegister](#) object which must be valid as long as this object is functional.

#### 5.75.2.2 void Arc::InfoRegistrar::registration (void)

Performs registartion in a loop.

Never exits unless there is a critical error or requested by destructor.

#### 5.75.2.3 bool Arc::InfoRegistrar::removeService ([InfoRegister \\*](#))

Removes service from list of handled services.

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

- InfoRegister.h

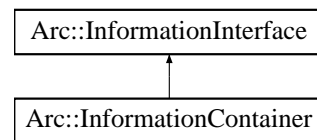


## 5.76 Arc::InformationContainer Class Reference

Information System document container and processor.

```
#include <InformationInterface.h>
```

Inheritance diagram for Arc::InformationContainer::



### Public Member Functions

- [InformationContainer](#) ([XMLNode](#) doc, bool copy=false)
- [XMLNode Acquire](#) (void)
- void [Assign](#) ([XMLNode](#) doc, bool copy=false)

### Protected Member Functions

- virtual void [Get](#) (const std::list< std::string > &path, [XMLNodeContainer](#) &result)

### Protected Attributes

- [XMLNode doc\\_](#)

#### 5.76.1 Detailed Description

Information System document container and processor.

This class inherits from [InformationInterface](#) and offers container for storing informational XML document.

#### 5.76.2 Constructor & Destructor Documentation

##### 5.76.2.1 Arc::InformationContainer::InformationContainer ([XMLNode](#) doc, bool copy = false)

Creates an instance with XML document . If is true this method makes a copy of for internal use.

#### 5.76.3 Member Function Documentation

##### 5.76.3.1 [XMLNode](#) Arc::InformationContainer::Acquire (void)

Get a lock on contained XML document. To be used in multi-threaded environment. Do not forget to release it with Release()

**5.76.3.2 void Arc::InformationContainer::Assign ([XMLNode](#) *doc*, bool *copy* = false)**

Replaces internal XML document with *doc*. If *copy* is true this method makes a copy of *doc* for internal use.

**5.76.3.3 virtual void Arc::InformationContainer::Get (const std::list< std::string > & *path*, [XMLNodeContainer](#) & *result*)** [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented from [Arc::InformationInterface](#).

**5.76.4 Field Documentation****5.76.4.1 [XMLNode](#) Arc::InformationContainer::doc\_** [protected]

Either link or container of XML document

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

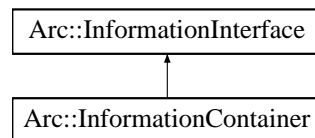
- InformationInterface.h

## 5.77 Arc::InformationInterface Class Reference

Information System message processor.

```
#include <InformationInterface.h>
```

Inheritance diagram for Arc::InformationInterface::



### Public Member Functions

- [InformationInterface](#) (bool safe=true)

### Protected Member Functions

- virtual void [Get](#) (const std::list< std::string > &path, [XMLNodeContainer](#) &result)

### Protected Attributes

- Glib::Mutex [lock\\_](#)

#### 5.77.1 Detailed Description

Information System message processor.

This class provides callback for 2 operations of WS-ResourceProperties and convenient parsing/generation of corresponding SOAP messages. In a future it may extend range of supported specifications.

#### 5.77.2 Constructor & Destructor Documentation

##### 5.77.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = true)

Constructor. If 'safe' is true all calls to Get will be locked.

#### 5.77.3 Member Function Documentation

##### 5.77.3.1 virtual void Arc::InformationInterface::Get (const std::list< std::string > & *path*, [XMLNodeContainer](#) & *result*) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented in [Arc::InformationContainer](#).

## 5.77.4 Field Documentation

### 5.77.4.1 Glib::Mutex [Arc::InformationInterface::lock\\_](#) [protected]

Mutex used to protect access to Get methods in multi-threaded env.

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

- InformationInterface.h

## 5.78 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

```
#include <InformationInterface.h>
```

### Public Member Functions

- [InformationRequest](#) (void)
- [InformationRequest](#) (const std::list< std::string > &path)
- [InformationRequest](#) (const std::list< std::list< std::string > > &paths)
- [InformationRequest](#) ([XMLNode](#) query)
- SOAPEnvelope \* [SOAP](#) (void)

### 5.78.1 Detailed Description

Request for information in InfoSystem.

This is a convenience wrapper creating proper WS-ResourceProperties request targeted InfoSystem interface of service.

### 5.78.2 Constructor & Destructor Documentation

#### 5.78.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

#### 5.78.2.2 Arc::InformationRequest::InformationRequest (const std::list< std::string > &path)

Request for attribute specified by elements of path. Currently only first element is used.

#### 5.78.2.3 Arc::InformationRequest::InformationRequest (const std::list< std::list< std::string > > &paths)

Request for attribute specified by elements of paths. Currently only first element of every path is used.

#### 5.78.2.4 Arc::InformationRequest::InformationRequest ([XMLNode](#) query)

Request for attributes specified by XPath query.

### 5.78.3 Member Function Documentation

#### 5.78.3.1 SOAPEnvelope\* Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

## 5.79 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

```
#include <InformationInterface.h>
```

### Public Member Functions

- [InformationResponse](#) (SOAPEnvelope &soap)
- std::list< [XMLNode](#) > [Result](#) (void)

#### 5.79.1 Detailed Description

Informational response from InfoSystem.

This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

#### 5.79.2 Constructor & Destructor Documentation

##### 5.79.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & soap)

Constructor parses WS-ResourceProperties response. Provided SOAPEnvelope object must be valid as long as this object is in use.

#### 5.79.3 Member Function Documentation

##### 5.79.3.1 std::list<[XMLNode](#)> Arc::InformationResponse::Result (void)

Returns set of attributes which were in SOAP message passed to constructor.

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

- InformationInterface.h

## 5.80 Arc::initializeCredentialsType Class Reference

Defines how user credentials are looked for.

```
#include <UserConfig.h>
```

### 5.80.1 Detailed Description

Defines how user credentials are looked for.

For complete information see description of UserConfig::InitializeCredentials(initializeCredentials) method.

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

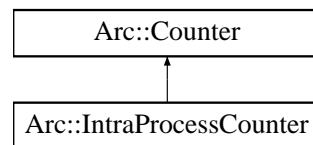
- UserConfig.h

## 5.81 Arc::IntraProcessCounter Class Reference

A class for counters used by threads within a single process.

```
#include <IntraProcessCounter.h>
```

Inheritance diagram for Arc::IntraProcessCounter::



### Public Member Functions

- [IntraProcessCounter](#) (int limit, int excess)
- virtual [~IntraProcessCounter](#) ()
- virtual int [getLimit](#) ()
- virtual int [setLimit](#) (int newLimit)
- virtual int [changeLimit](#) (int amount)
- virtual int [getExcess](#) ()
- virtual int [setExcess](#) (int newExcess)
- virtual int [changeExcess](#) (int amount)
- virtual int [getValue](#) ()
- virtual [CounterTicket reserve](#) (int amount=1, Glib::TimeVal duration=[ETERNAL](#), bool prioritized=false, Glib::TimeVal timeOut=[ETERNAL](#))

### Protected Member Functions

- virtual void [cancel](#) (IDType reservationID)
- virtual void [extend](#) (IDType &reservationID, Glib::TimeVal &expiryTime, Glib::TimeVal duration=[ETERNAL](#))

#### 5.81.1 Detailed Description

A class for counters used by threads within a single process.

This is a class for shared among different threads within a single process. See the [Counter](#) class for further information about counters and examples of usage.

#### 5.81.2 Constructor & Destructor Documentation

##### 5.81.2.1 Arc::IntraProcessCounter::IntraProcessCounter (int *limit*, int *excess*)

Creates an [IntraProcessCounter](#) with specified limit and excess.

This constructor creates a counter with the specified limit (amount of resources available for reservation) and excess limit (an extra amount of resources that may be used for prioritized reservations).



**Parameters:**

*limit* The limit of the counter.

*excess* The excess limit of the counter.

**5.81.2.2 virtual Arc::IntraProcessCounter::~~IntraProcessCounter () [virtual]**

Destructor.

This is the destructor of the [IntraProcessCounter](#) class. Does not need to do anything.

**5.81.3 Member Function Documentation****5.81.3.1 virtual void Arc::IntraProcessCounter::cancel (IDType reservationID) [protected, virtual]**

Cancellation of a reservation.

This method cancels a reservation. It is called by the [CounterTicket](#) that corresponds to the reservation.

**Parameters:**

*reservationID* The identity number (key) of the reservation to cancel.

**5.81.3.2 virtual int Arc::IntraProcessCounter::changeExcess (int amount) [virtual]**

Changes the excess limit of the counter.

Changes the excess limit of the counter by adding a certain amount to the current excess limit.

**Parameters:**

*amount* The amount by which to change the excess limit.

**Returns:**

The new excess limit.

Implements [Arc::Counter](#).

**5.81.3.3 virtual int Arc::IntraProcessCounter::changeLimit (int amount) [virtual]**

Changes the limit of the counter.

Changes the limit of the counter by adding a certain amount to the current limit.

**Parameters:**

*amount* The amount by which to change the limit.

**Returns:**

The new limit.

Implements [Arc::Counter](#).

**5.81.3.4 virtual void Arc::IntraProcessCounter::extend (IDType & reservationID, Glib::TimeVal & expiryTime, Glib::TimeVal duration = ETERNAL) [protected, virtual]**

Extension of a reservation.

This method extends a reservation. It is called by the [CounterTicket](#) that corresponds to the reservation.

**Parameters:**

**reservationID** Used for input as well as output. Contains the identification number of the original reservation on entry and the new identification number of the extended reservation on exit.

**expiryTime** Used for input as well as output. Contains the expiry time of the original reservation on entry and the new expiry time of the extended reservation on exit.

**duration** The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

**5.81.3.5 virtual int Arc::IntraProcessCounter::getExcess () [virtual]**

Returns the excess limit of the counter.

Returns the excess limit of the counter, i.e. by how much the usual limit may be exceeded by prioritized reservations.

**Returns:**

The excess limit.

Implements [Arc::Counter](#).

**5.81.3.6 virtual int Arc::IntraProcessCounter::getLimit () [virtual]**

Returns the current limit of the counter.

This method returns the current limit of the counter, i.e. how many units can be reserved simultaneously by different threads without claiming high priority.

**Returns:**

The current limit of the counter.

Implements [Arc::Counter](#).

**5.81.3.7 virtual int Arc::IntraProcessCounter::getValue () [virtual]**

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the value is decreased. Normally, the value should never be negative, but this may happen if there are prioritized reservations. It can also happen if the limit is decreased after some reservations have been made, since reservations are never revoked.

**Returns:**

The current value of the counter.

Implements [Arc::Counter](#).

**5.81.3.8** `virtual CounterTicket Arc::IntraProcessCounter::reserve (int amount = 1, Glib::TimeVal duration = ETERNAL, bool prioritized = false, Glib::TimeVal timeOut = ETERNAL) [virtual]`

Makes a reservation from the counter.

This method makes a reservation from the counter. If the current value of the counter is too low to allow for the reservation, the method blocks until the reservation is possible or times out.

**Parameters:**

*amount* The amount to reserve, default value is 1.

*duration* The duration of a self expiring reservation, default is that it lasts forever.

*prioritized* Whether this reservation is prioritized and thus allowed to use the excess limit.

*timeOut* The maximum time to block if the value of the counter is too low, default is to allow "eternal" blocking.

**Returns:**

A [CounterTicket](#) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implements [Arc::Counter](#).

**5.81.3.9** `virtual int Arc::IntraProcessCounter::setExcess (int newExcess) [virtual]`

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

**Parameters:**

*newExcess* The new excess limit, an absolute number.

**Returns:**

The new excess limit.

Implements [Arc::Counter](#).

**5.81.3.10** `virtual int Arc::IntraProcessCounter::setLimit (int newLimit) [virtual]`

Sets the limit of the counter.

This method sets a new limit for the counter.

**Parameters:**

*newLimit* The new limit, an absolute number.

**Returns:**

The new limit.

Implements [Arc::Counter](#).

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

- IntraProcessCounter.h

## 5.82 Arc::Job Class Reference

[Job](#).

```
#include <Job.h>
```

### Public Member Functions

- [Job](#) ()
- void [SaveToStream](#) (std::ostream &out, bool longlist) const
- [Job](#) & [operator=](#) ([XMLNode](#) job)
- void [Update](#) ([XMLNode](#) job)
- void [ToXML](#) ([XMLNode](#) job) const

### Static Public Member Functions

- static bool [ReadAllJobsFromFile](#) (const std::string &filename, std::list< [Job](#) > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [ReadJobsFromFile](#) (const std::string &filename, std::list< [Job](#) > &jobs, std::list< std::string > &jobIdentifiers, bool all=false, const std::list< std::string > &endpoints=std::list< std::string >(), const std::list< std::string > &rejectEndpoints=std::list< std::string >(), unsigned nTries=10, unsigned tryInterval=500000)
- static bool [WriteJobsToTruncatedFile](#) (const std::string &filename, const std::list< [Job](#) > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [WriteJobsToFile](#) (const std::string &filename, const std::list< [Job](#) > &jobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [WriteJobsToFile](#) (const std::string &filename, const std::list< [Job](#) > &jobs, std::list< const [Job](#) \* > &newJobs, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [RemoveJobsFromFile](#) (const std::string &filename, const std::list< [URL](#) > &jobids, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [ReadJobIDsFromFile](#) (const std::string &filename, std::list< std::string > &jobids, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [WriteJobIDToFile](#) (const [URL](#) &jobid, const std::string &filename, unsigned nTries=10, unsigned tryInterval=500000)
- static bool [WriteJobIDsToFile](#) (const std::list< [URL](#) > &jobids, const std::string &filename, unsigned nTries=10, unsigned tryInterval=500000)

### 5.82.1 Detailed Description

[Job](#).

This class describe a Grid job. Most of the members contained in this class are directly linked to the ComputingActivity defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

### 5.82.2 Constructor & Destructor Documentation

#### 5.82.2.1 Arc::Job::Job ()

Create a [Job](#) object.

Default constructor. Takes no arguments.

## 5.82.3 Member Function Documentation

### 5.82.3.1 Job& Arc::Job::operator= (XMLNode job)

Set [Job](#) attributes from a [XMLNode](#).

The attributes of the [Job](#) object is set to the values specified in the [XMLNode](#). The [XMLNode](#) should be a [ComputingActivity](#) type using the [GLUE2](#) XML hierarchical rendering, see <http://forge.gridforum.org/sf/wiki/do/viewPage/projects.glue-wg/wiki/GLUE2XMLSchema> for more information. Note that associations are not parsed.

#### Parameters:

*job* is a [XMLNode](#) of [GLUE2](#) [ComputingActivity](#) type.

#### See also:

[ToXML](#)

### 5.82.3.2 static bool Arc::Job::ReadAllJobsFromFile (const std::string & filename, std::list< Job > & jobs, unsigned nTries = 10, unsigned tryInterval = 500000) [static]

Read all jobs from file.

This static method will read jobs (in XML format) from the specified file, and they will be stored in the referenced list of jobs. The XML element in the file representing a job should be named "Job", and have the same format as accepted by the [operator=\(XMLNode\)](#) method.

File locking: To avoid simultaneous use (writing and reading) of the file, reading will not be initiated before a lock on the file has been acquired. For this purpose the [FileLock](#) class is used. *nTries* specifies the maximal number of times the method will try to acquire a lock on the file, with an interval of *tryInterval* micro seconds between each attempt. If a lock is not acquired\* this method returns false.

The method will also return false if the content of file is not in XML format. Otherwise it returns true.

#### Parameters:

*filename* is the filename of the job list to read jobs from.

*jobs* is a reference to a list of [Job](#) objects, which will be filled with the jobs read from file (cleared before use).

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

#### Returns:

true in case of success, otherwise false.

#### See also:

[operator=\(XMLNode\)](#)  
[ReadJobsFromFile](#)  
[WriteJobsToTruncatedFile](#)  
[WriteJobsToFile](#)  
[RemoveJobsFromFile](#)  
[FileLock](#)  
[XMLNode::ReadFromFile](#)

**5.82.3.3** `static bool Arc::Job::ReadJobIDsFromFile (const std::string &filename, std::list<std::string> &jobids, unsigned nTries = 10, unsigned tryInterval = 500000) [static]`

Read a list of [Job](#) IDs from a file, and append them to a list.

This static method will read job IDs from the given file, and append the strings to the string list given as parameter. File locking will be done as described for the `ReadAllJobsFromFile` method. It returns false if the file was not readable, true otherwise, even if there were no IDs in the file. The lines of the file will be trimmed, and lines starting with # will be ignored.

**Parameters:**

*filename* is the filename of the jobidfile

*jobids* is a list of strings, to which the IDs read from the file will be appended

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**5.82.3.4** `static bool Arc::Job::ReadJobsFromFile (const std::string &filename, std::list<Job> &jobs, std::list<std::string> &jobIdentifiers, bool all = false, const std::list<std::string> &endpoints = std::list<std::string>(), const std::list<std::string> &rejectEndpoints = std::list<std::string>(), unsigned nTries = 10, unsigned tryInterval = 500000) [static]`

Read specified jobs from file.

Extract job information for jobs specified by job identifiers and/or endpoints from job list file. The method read all jobs from specified job list file, using the `ReadAllJobsFromFile` method. If the all argument is false, jobs will only be put into the list of [Job](#) objects (jobs) if the `IDFromEndpoint` or `Name` attributes of the [Job](#) object matches one of the entries in the `jobIdentifiers` list argument or if the `Cluster` attribute of the [Job](#) object matches one of the entries in the `endpoints` list argument (if specified), using the [URL::StringMatches](#) method. If the all argument is true, none of those matchings is carried out, instead all jobs are put into the list of [Job](#) objects. For both values of the all argument, the entries in the `jobIdentifiers` list will be removed if corresponding to a job in the jobs list. In the end, if the `rejectEndpoints` list is non-empty, the jobs list will be filtered by removing [Job](#) objects for which the `Cluster` attribute matches those in the `rejectEndpoints` list, using the [URL::StringMatches](#) method. This method returns true, except when the `ReadAllJobsFromFile` method returns false.

**Parameters:**

*filename* is the filename of the job list to read jobs from.

*jobs* is a reference to a list of [Job](#) objects, which will be filled with the jobs read from file (cleared before use).

*jobIdentifiers* specifies the job IDs and names of jobs to be put into the jobs list. Entries in this list is removed if found among the jobs in the job list file.

*all* specifies whether all jobs from the jobs list should be put into the jobs list.

*endpoints* is a list of strings resembling endpoints for which [Job](#) objects having matching `Cluster` attribute should be added to the jobs list.

*rejectEndpoints* is a list of strings resembling endpoint for which [Job](#) objects having matching `Cluster` attribute should be removed from the jobs list. Overrides `jobIdentifiers`, `all` and `endpoints`.

*nTries* will be passed to the ReadAllJobsFromFile method

*tryInterval* will be passed to the ReadAllJobsFromFile method

**Returns:**

true in case of success, otherwise false.

**See also:**

[ReadAllJobsFromFile](#)  
[URL::StringMatches](#)  
[WriteJobsToTruncatedFile](#)  
[WriteJobsToFile](#)  
[RemoveJobsFromFile](#)

### 5.82.3.5 static bool Arc::Job::RemoveJobsFromFile (const std::string & *filename*, const std::list< [URL](#) > & *jobids*, unsigned *nTries* = 10, unsigned *tryInterval* = 500000) [static]

Remove job from file.

This static method will remove the jobs having IDFromEndpoint identical to any of those in the passed list jobids. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if reading from or writing jobs to the file fails. Otherwise it returns true.

**Parameters:**

*filename* is the filename of the job list to write jobs to.

*jobids* is a list of [URL](#) objects which specifies which jobs from the file to remove.

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**See also:**

[ReadAllJobsFromFile](#)  
[WriteJobsToTruncatedFile](#)  
[WriteJobsToFile](#)  
[FileLock](#)  
[XMLNode::ReadFromFile](#)  
[XMLNode::SaveToFile](#)

### 5.82.3.6 void Arc::Job::SaveToStream (std::ostream & *out*, bool *longlist*) const

Write job information to a std::ostream object.

This method will write job information to the passed std::ostream object. The longlist boolean specifies whether more (true) or less (false) information should be printed.

**Parameters:**

*out* is the std::ostream object to print the attributes to.

*longlist* is a boolean for switching on long listing (more details).

**5.82.3.7 void Arc::Job::ToXML (XMLNode job) const**

Add job information to a XMLNode.

Child nodes of GLUE ComputingActivity type containing job information of this object will be added to the passed XMLNode.

**Parameters:**

*job* is the XMLNode to add job information to in form of GLUE2 ComputingActivity type child nodes.

**See also:**

[operator=](#)

**5.82.3.8 void Arc::Job::Update (XMLNode job)**

Set Job attributes from a XMLNode representing GLUE2 ComputingActivity.

Because job XML representation follows GLUE2 model this method is similar to [operator=\(XMLNode\)](#). But it only covers job attributes which are part of GLUE2 computing activity. Also it treats Job object as being iextended with information provided by XMLNode. Contrary [operator=\(XMLNode\)](#) fully reinitializes Job, hence removing any associations to other objects.

**5.82.3.9 static bool Arc::Job::WriteJobIDsToFile (const std::list< URL > & jobids, const std::string & filename, unsigned nTries = 10, unsigned tryInterval = 500000) [static]**

Append list of URLs to a file.

This static method will put the ID given as a string, and append it to the given file. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file was not writable, true otherwise.

**Parameters:**

*jobid* is a list of URL objects to be written to file

*filename* is the filename of file, where the URL objects will be appended to.

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**5.82.3.10 static bool Arc::Job::WriteJobIDToFile (const URL & jobid, const std::string & filename, unsigned nTries = 10, unsigned tryInterval = 500000) [static]**

Append a jobID to a file.

This static method will put the ID represented by a URL object, and append it to the given file. File locking will be done as described for the ReadAllJobsFromFile method. It returns false if the file is not writable, true otherwise.



**Parameters:**

*jobid* is a jobID as a [URL](#) object

*filename* is the filename of the jobidfile, where the jobID will be appended

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**5.82.3.11** `static bool Arc::Job::WriteJobsToFile (const std::string &filename, const std::list< Job > &jobs, std::list< const Job * > &newJobs, unsigned nTries = 10, unsigned tryInterval = 500000) [static]`

Write jobs to file.

This static method will write (append) the passed list of jobs to the specified file. Jobs will be written in XML format as returned by the ToXML method, and each job will be contained in a element named "Job". If the passed list of jobs contains two identical jobs (i.e. IDFromEndpoint identical), only the latter [Job](#) object is stored. If a job in the list is identical to one in file, the one in file will be replaced with the one from the list. A pointer (no memory allocation) to those jobs from the list which are not in the file will be added to the newJobs list, thus these pointers goes out of scope when 'jobs' list goes out of scope. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if writing jobs to the file fails. Otherwise it returns true.

**Parameters:**

*filename* is the filename of the job list to write jobs to.

*jobs* is the list of [Job](#) objects which should be written to file.

*newJobs* is a reference to a list of pointers to [Job](#) objects which are not duplicates.

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**See also:**

[ToXML](#)  
[ReadAllJobsFromFile](#)  
[WriteJobsToTruncatedFile](#)  
[RemoveJobsFromFile](#)  
[FileLock](#)  
[XMLNode::SaveToFile](#)

**5.82.3.12** `static bool Arc::Job::WriteJobsToFile (const std::string &filename, const std::list< Job > &jobs, unsigned nTries = 10, unsigned tryInterval = 500000) [static]`

Write jobs to file.

This method is in all respects identical to the [WriteJobsToFile\(const std::string&, const std::list<Job>&, std::list<const Job\\*>&, unsigned, unsigned\)](#) method, except for the information about new jobs which is disregarded.

**See also:**

[WriteJobsToFile\(const std::string&, const std::list<Job>&, std::list<const Job\\*>&, unsigned, unsigned\)](#)

**5.82.3.13** `static bool Arc::Job::WriteJobsToTruncatedFile (const std::string & filename, const std::list< Job > & jobs, unsigned nTries = 10, unsigned tryInterval = 500000)`  
[static]

Truncate file and write jobs to it.

This static method will write the passed list of jobs to the specified file, but before writing the file will be truncated. Jobs will be written in XML format as returned by the ToXML method, and each job will be contained in a element named "Job". If the passed list of jobs contains two identical jobs (i.e. IDFrom-Endpoint identical), only the latter [Job](#) object is stored. File locking will be done as described for the ReadAllJobsFromFile method. The method will return false if writing jobs to the file fails. Otherwise it returns true.

**Parameters:**

*filename* is the filename of the job list to write jobs to.

*jobs* is the list of [Job](#) objects which should be written to file.

*nTries* specifies the maximal number of times the method will try to acquire a lock on file to read.

*tryInterval* specifies the interval (in micro seconds) between each attempt to acquire a lock.

**Returns:**

true in case of success, otherwise false.

**See also:**

[ToXML](#)  
[ReadAllJobsFromFile](#)  
[WriteJobsToFile](#)  
[RemoveJobsFromFile](#)  
[FileLock](#)  
[XMLNode::SaveToFile](#)

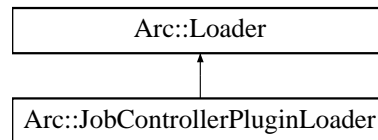
The documentation for this class was generated from the following file:

- Job.h

## 5.83 Arc::JobControllerPluginLoader Class Reference

```
#include <JobControllerPlugin.h>
```

Inheritance diagram for Arc::JobControllerPluginLoader::



### Public Member Functions

- [JobControllerPluginLoader](#) ()
- [~JobControllerPluginLoader](#) ()
- [JobControllerPlugin \\* load](#) (const std::string &name, const [UserConfig](#) &uc)

### 5.83.1 Detailed Description

Class responsible for loading JobControllerPlugin plugins The JobControllerPlugin objects returned by a [JobControllerPluginLoader](#) must not be used after the [JobControllerPluginLoader](#) goes out of scope.

### 5.83.2 Constructor & Destructor Documentation

#### 5.83.2.1 Arc::JobControllerPluginLoader::JobControllerPluginLoader ()

Constructor Creates a new [JobControllerPluginLoader](#).

#### 5.83.2.2 Arc::JobControllerPluginLoader::~~JobControllerPluginLoader ()

Destructor Calling the destructor destroys all JobControllerPlugins loaded by the [JobControllerPluginLoader](#) instance.

### 5.83.3 Member Function Documentation

#### 5.83.3.1 JobControllerPlugin\* Arc::JobControllerPluginLoader::load (const std::string & name, const [UserConfig](#) & uc)

Load a new JobControllerPlugin

##### Parameters:

*name* The name of the JobControllerPlugin to load.

*usercfg* The [UserConfig](#) object for the new JobControllerPlugin.

##### Returns:

A pointer to the new JobControllerPlugin (NULL on error).

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

- JobControllerPlugin.h

## 5.84 Arc::JobDescription Class Reference

```
#include <JobDescription.h>
```

### Public Member Functions

- JobDescriptionResult [UnParse](#) (std::string &product, std::string language, const std::string &dialect="") const
- const std::string & [GetSourceLanguage](#) () const
- JobDescriptionResult [SaveToStream](#) (std::ostream &out, const std::string &format) const
- bool [Prepare](#) (const [ExecutionTarget](#) &et)

### Static Public Member Functions

- static JobDescriptionResult [Parse](#) (const std::string &source, std::list< [JobDescription](#) > &jobdescs, const std::string &language="", const std::string &dialect="")

### Data Fields

- std::map< std::string, std::string > [OtherAttributes](#)

#### 5.84.1 Detailed Description

The [JobDescription](#) class is the internal representation of a job description in the ARC-lib. It is structured into a number of other classes/objects which should strictly follow the description given in the job description document <[http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/tech\\_doc/client/job\\_description.odt](http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/tech_doc/client/job_description.odt)>.

The class consist of a parsing method [JobDescription::Parse](#) which tries to parse the passed source using a number of different parsers. The parser method is complemented by the [JobDescription::UnParse](#) method, a method to generate a job description document in one of the supported formats. Additionally the internal representation is contained in public members which makes it directly accessible and modifiable from outside the scope of the class.

#### 5.84.2 Member Function Documentation

##### 5.84.2.1 const std::string& Arc::JobDescription::GetSourceLanguage () const [inline]

Get input source language.

If this object was created by a [JobDescriptionParser](#), then this method returns a string which indicates the job description language of the parsed source. If not created by a [JobDescriptionParser](#) the string returned is empty.

##### Returns:

const std::string& source language of parsed input source.

**5.84.2.2** `static JobDescriptionResult Arc::JobDescription::Parse (const std::string & source,  
std::list< JobDescription > & jobdescs, const std::string & language = "", const  
std::string & dialect = "") [static]`

Parse string into [JobDescription](#) objects.

The passed string will be tried parsed into the list of [JobDescription](#) objects. The available specialized [JobDescriptionParser](#) classes will be tried one by one, parsing the string, and if one succeeds the list of [JobDescription](#) objects is filled with the parsed contents and true is returned, otherwise false is returned. If no language specified, each [JobDescriptionParser](#) will try all its supported languages. On the other hand if a language is specified, only the [JobDescriptionParser](#) supporting that language will be tried. A dialect can also be specified, which only has an effect on the parsing if the [JobDescriptionParser](#) supports that dialect.

**Parameters:**

*source*  
*jobdescs*  
*language*  
*dialect*

**Returns:**

true if the passed string can be parsed successfully by any of the available parsers.

**5.84.2.3** `bool Arc::JobDescription::Prepare (const ExecutionTarget & et)`

Prepare for submission to target.

The Prepare method, is used to check and adapt the [JobDescription](#) object to the passed [ExecutionTarget](#) object before submitting the job description to the target. This method is normally called by [SubmitterPlugin](#) plugin classes, before submitting the job description. First the method checks the [DataStaging.InputFiles](#) list, for identical file names, and non-existent local input files. If any of such files are found, the method returns false. Then if the [Application.Executable](#) and [Application.Input](#) objects are specified as local input files, and they are not among the files in the [DataStaging.InputFiles](#) list a existence check will be done and if not found, false will be returned, otherwise they will be added to the list. Likewise if the [Application.Output](#), [Application.Error](#) and [Application.LogDir](#) attributes have been specified, and is not among the files in the [DataStaging.OutputFiles](#) list, they will be added to this list. After the file check, the [Resources.RunTimeEnvironment](#), [Resources.CEType](#) and [Resources.OperatingSystem](#) [SoftwareRequirement](#) objects are respectively resolved against the [ExecutionTarget::ApplicationEnvironments](#), [ExecutionTarget::Implementation](#) and [ExecutionTarget::OperatingSystem](#) [Software](#) objects using the [SoftwareRequirement::selectSoftware](#) method. If that method returns false i.e. unable to resolve the requirements false will be returned. After resolving software requirements, the value of the [Resources.QueueName](#) attribute will be set to that of the [ExecutionTarget::ComputingShareName](#) attribute, and then true is returned.

**Parameters:**

*et* [ExecutionTarget](#) object which to resolve software requirements against, and to pick up queue name from.

**Returns:**

false is returned is file checks fails, or if unable to resolve software requirements.

#### 5.84.2.4 JobDescriptionResult Arc::JobDescription::SaveToStream (std::ostream & *out*, const std::string & *format*) const

Print job description to a std::ostream object.

The job description will be written to the passed std::ostream object out in the format indicated by the format parameter. The format parameter should specify the format of one of the job description languages supported by the library. Or by specifying the special "user" or "userlong" format the job description will be written as a attribute/value pair list with respectively less or more attributes.

The mote

##### Returns:

true if writing the job description to the out object succeeds, otherwise false.

##### Parameters:

*out* a std::ostream reference specifying the ostream to write the job description to.

*format* specifies the format the job description should written in.

#### 5.84.2.5 JobDescriptionResult Arc::JobDescription::UnParse (std::string & *product*, std::string *language*, const std::string & *dialect* = "") const

Output contents in the specified language.

##### Parameters:

*product*

*language*

*dialect*

##### Returns:

### 5.84.3 Field Documentation

#### 5.84.3.1 std::map<std::string, std::string> [Arc::JobDescription::OtherAttributes](#)

Holds attributes not fitting into this class.

This member is used by [JobDescriptionParser](#) classes to store attribute/value pairs not fitting into attributes stored in this class. The form of the attribute (the key in the map) should be as follows: <language>;<attribute-name> E.g.: "nordugrid:xrsl;hostname".

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

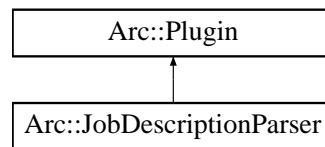
- JobDescription.h

## 5.85 Arc::JobDescriptionParser Class Reference

Abstract class for the different parsers.

```
#include <JobDescriptionParser.h>
```

Inheritance diagram for Arc::JobDescriptionParser::



### 5.85.1 Detailed Description

Abstract class for the different parsers.

The [JobDescriptionParser](#) class is abstract which provide a interface for job description parsers. A job description parser should inherit this class and overwrite the JobDescriptionParser::Parse and JobDescriptionParser::UnParse methods.

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

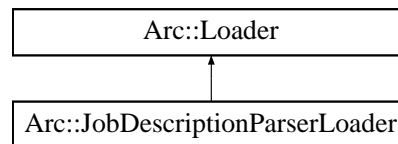
- JobDescriptionParser.h



## 5.86 Arc::JobDescriptionParserLoader Class Reference

```
#include <JobDescriptionParser.h>
```

Inheritance diagram for Arc::JobDescriptionParserLoader::



### Public Member Functions

- [JobDescriptionParserLoader](#) ()
- [~JobDescriptionParserLoader](#) ()
- [JobDescriptionParser \\* load](#) (const std::string &name)
- const std::list< [JobDescriptionParser](#) \* > & [GetJobDescriptionParsers](#) () const

### Data Structures

- class [iterator](#)

#### 5.86.1 Detailed Description

Class responsible for loading [JobDescriptionParser](#) plugins The [JobDescriptionParser](#) objects returned by a [JobDescriptionParserLoader](#) must not be used after the [JobDescriptionParserLoader](#) goes out of scope.

#### 5.86.2 Constructor & Destructor Documentation

##### 5.86.2.1 Arc::JobDescriptionParserLoader::JobDescriptionParserLoader ()

Constructor Creates a new [JobDescriptionParserLoader](#).

##### 5.86.2.2 Arc::JobDescriptionParserLoader::~~JobDescriptionParserLoader ()

Destructor Calling the destructor destroys all [JobDescriptionParser](#) object loaded by the [JobDescriptionParserLoader](#) instance.

#### 5.86.3 Member Function Documentation

##### 5.86.3.1 const std::list<[JobDescriptionParser](#)\*>& Arc::JobDescriptionParserLoader::GetJobDescriptionParsers () const [inline]

Retrieve the list of loaded [JobDescriptionParser](#) objects.

##### Returns:

A reference to the list of [JobDescriptionParser](#) objects.

### 5.86.3.2 [JobDescriptionParser](#)\* Arc::JobDescriptionParserLoader::load (const std::string & *name*)

Load a new [JobDescriptionParser](#)

#### Parameters:

*name* The name of the [JobDescriptionParser](#) to load.

#### Returns:

A pointer to the new [JobDescriptionParser](#) (NULL on error).

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

- JobDescriptionParser.h

## 5.87 Arc::JobIdentificationType Class Reference

Job identification.

```
#include <JobDescription.h>
```

### Data Fields

- std::string [JobName](#)
- std::string [Description](#)
- std::string [Type](#)
- std::list< std::string > [Annotation](#)
- std::list< std::string > [ActivityOldID](#)

### 5.87.1 Detailed Description

Job identification.

This class serves to provide human readable information about a job description. Some of this information might also be passed to the execution service for providing information about the job created from this job description. An object of this class is part of the [JobDescription](#) class as the Identification public member.

### 5.87.2 Field Documentation

#### 5.87.2.1 std::list<std::string> [Arc::JobIdentificationType::ActivityOldID](#)

ID of old activity.

The ActivityOldID object is used to store a list of IDs corresponding to activities which were performed from this job description. This information is not intended to be used by the execution service, but rather used for keeping track of activities, e.g. when doing a job resubmission the old activity ID is appended to this list.

#### 5.87.2.2 std::list<std::string> [Arc::JobIdentificationType::Annotation](#)

Annotation.

The Annotation list is used for human readable comments, tags for free grouping or identifying different activities.

#### 5.87.2.3 std::string [Arc::JobIdentificationType::Description](#)

Human readable description.

The Description string can be used to provide a human readable description of e.g. the task which should be performed when processing the job description.

#### 5.87.2.4 std::string [Arc::JobIdentificationType::JobName](#)

Name of job.

The JobName string is used to specify a name of the job description, and it will most likely also be the name given to the job when created at the execution service.

#### 5.87.2.5 `std::string Arc::JobIdentificationType::Type`

[Job](#) type.

The Type string specifies a classification of the activity in compliance with [GLUE2](#). The possible values should follow those defined in the ComputingActivityType\_t enumeration of [GLUE2](#).

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

- JobDescription.h

## 5.88 Arc::JobState Class Reference

```
#include <JobState.h>
```

### Public Member Functions

- bool [IsFinished](#) () const

#### 5.88.1 Detailed Description

ARC general state model. The class comprise the general state model of the ARC-lib, and are herein used to compare job states from the different middlewares supported by the plugin structure of the ARC-lib. Which is why every ACC plugin should contain a class derived from this class. The derived class should consist of a constructor and a mapping function (a JobStateMap) which maps a std::string to a [JobState](#):StateType. An example of a constructor in a plugin could be: `JobStatePlugin::JobStatePlugging(const std::string& state) : JobState(state, &pluginStateMap) {}` where `&pluginStateMap` is a reference to the JobStateMap defined by the derived class.

#### 5.88.2 Member Function Documentation

##### 5.88.2.1 bool Arc::JobState::IsFinished () const `[inline]`

Check if state is finished.

##### Returns:

true is returned if the StateType is equal to FINISHED, KILLED, FAILED or DELETED, otherwise false is returned.

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

- JobState.h

## 5.89 Arc::JobSupervisor Class Reference

% [JobSupervisor](#) class

```
#include <JobSupervisor.h>
```

### Public Member Functions

- [JobSupervisor](#) (const [UserConfig](#) &usercfg, const std::list< [Job](#) > &jobs=std::list< [Job](#) >())
- bool [AddJob](#) (const [Job](#) &job)
- std::list< [Job](#) > [GetJobs](#) (bool includeJobsWithoutStatus=true) const
- void [Update](#) ()
- bool [Retrieve](#) (const std::string &downloaddirprefix, bool usejobname, bool force, std::list< std::string > &downloaddirectories)
- bool [Renew](#) ()
- bool [Resume](#) ()
- bool [Resubmit](#) (int destination, const std::list< Endpoint > &, std::list< [Job](#) > &resubmittedJobs, const std::list< std::string > &=std::list< std::string >())
- bool [Migrate](#) (bool forcemigration, const std::list< Endpoint > &, std::list< [Job](#) > &migratedJobs, const std::list< std::string > &=std::list< std::string >())
- bool [Cancel](#) ()
- bool [Clean](#) ()

### 5.89.1 Detailed Description

% [JobSupervisor](#) class

The [JobSupervisor](#) class is tool for loading JobControllerPlugin plugins for managing Grid jobs.

### 5.89.2 Constructor & Destructor Documentation

#### 5.89.2.1 Arc::JobSupervisor::JobSupervisor (const [UserConfig](#) & usercfg, const std::list< [Job](#) > &jobs = std::list< [Job](#) >())

Create a [JobSupervisor](#).

The list of [Job](#) objects passed to the constructor will be managed by this [JobSupervisor](#), through the JobControllerPlugin class. It is important that the InterfaceName member of each [Job](#) object is set and names a interface supported by one of the available JobControllerPlugin plugins. The JobControllerPlugin plugin will be loaded using the [JobControllerPluginLoader](#) class, loading a plugin of type "HED:JobController-Plugin" which supports the particular interface, and the a reference to the [UserConfig](#) object usercfg will be passed to the plugin. Additionally a reference to the [UserConfig](#) object usercfg will be stored, thus usercfg must exist throughout the scope of the created object. If the InterfaceName member of a [Job](#) object is unset, a VERBOSE log message will be reported and that [Job](#) object will be ignored. If the JobControllerPlugin plugin for a given interface cannot be loaded, a WARNING log message will be reported and any [Job](#) object requesting that interface will be ignored. If loading of a specific plugin failed, that plugin will not be tried loaded for subsequent [Job](#) objects requiring that plugin. [Job](#) objects will be added to the corresponding JobControllerPlugin plugin, if loaded successfully.

#### Parameters:

- usercfg* [UserConfig](#) object to pass to JobControllerPlugin plugins and to use in member methods.
- jobs* List of [Job](#) objects which will be managed by the created object.

### 5.89.3 Member Function Documentation

#### 5.89.3.1 bool Arc::JobSupervisor::AddJob (const Job & job)

Add job.

Add [Job](#) object to this [JobSupervisor](#) for job management. The [Job](#) object will be passed to the corresponding specialized [JobControllerPlugin](#).

##### Parameters:

*job* [Job](#) object to add for job management

##### Returns:

true is returned if the passed [Job](#) object was added to the underlying [JobControllerPlugin](#), otherwise false is returned and a log message emitted with the reason.

#### 5.89.3.2 bool Arc::JobSupervisor::Cancel ()

Cancel jobs.

This method cancels jobs managed by this [JobSupervisor](#).

Before identifying jobs to cancel, the [JobControllerPlugin::UpdateJobs](#) method is called for each loaded [JobControllerPlugin](#) in order to retrieve the most up to date job information.

Since jobs in the [JobState::DELETED](#), [JobState::FINISHED](#), [JobState::KILLED](#) or [JobState::FAILED](#) states is already in a terminal state, a cancel request will not be send for those. Also no request will be send for jobs in the [JobState::UNDEFINED](#) state, since job information is not available. If the status-filter is non-empty, a cancel request will only be send to jobs with a general or specific state (see [JobState](#)) identical to any of the entries in the status-filter, excluding the states mentioned above.

For each job to be cancelled, the specialized [JobControllerPlugin::CancelJob](#) method is called and is responsible for cancelling the given job. If the method fails to cancel a job, this method will return false (otherwise true), and the job ID ([IDFromEndpoint](#)) of such jobs is appended to the notcancelled list. The job ID of successfully cancelled jobs will be appended to the passed cancelled list.

##### Returns:

false if any call to [JobControllerPlugin::CancelJob](#) failed, true otherwise.

##### See also:

[JobControllerPlugin::CancelJob](#).

#### 5.89.3.3 bool Arc::JobSupervisor::Clean ()

Clean jobs.

This method removes from services jobs managed by this [JobSupervisor](#). Before cleaning jobs, the [JobController::GetInformation](#) method is called in order to update job information, and that jobs are selected by job status instead of by job IDs. The status list argument should contain states for which cleaning of job in any of those states should be carried out. The states are compared using both the [JobState::operator\(\)](#) and [JobState::GetGeneralState\(\)](#) methods. If the status list is empty, all jobs will be selected for cleaning.

**Returns:**

false if calls to JobControllerPlugin::CleanJob fails, true otherwise.

#### 5.89.3.4 `std::list<Job> Arc::JobSupervisor::GetJobs (bool includeJobsWithoutStatus = true) const`

Get list of managed jobs.

The list of jobs managed by this [JobSupervisor](#) is returned when calling this method. If the includeJobsWithoutStatus argument is set to false, only [Job](#) objects with a valid State attribute is returned.

**Parameters:**

*includeJobWithoutStatus* specifies whether jobs with invalid status should be included in the returned list

**Returns:**

list of [Job](#) objects managed by this [JobSupervisor](#)

**See also:**

JobState::operator bool

#### 5.89.3.5 `bool Arc::JobSupervisor::Migrate (bool forcemigration, const std::list< Endpoint > &, std::list< Job > & migratedJobs, const std::list< std::string > & = std::list< std::string >())`

Migrate jobs.

Jobs managed by this [JobSupervisor](#) will be migrated when invoking this method, that is the job description of a job will be tried obtained, and if successful a job migration request will be sent, based on that job description.

Before identifying jobs to be migrated, the JobControllerPlugin::UpdateJobs method is called for each loaded JobControllerPlugin in order to retrieve the most up to date job information. Only jobs for which the State member of the [Job](#) object has the value JobState::QUEUEING, will be considered for migration. Furthermore the job description must be obtained (either locally or remote) and successfully parsed in order for a job to be migrated. If the job description cannot be obtained or parsed an ERROR log message is reported, and the IDFromEndpoint [URL](#) of the [Job](#) object is appended to the notmigrated list. If no jobs have been identified for migration, false will be returned in case ERRORS were reported, otherwise true is returned.

The execution services which can be targeted for migration are those specified in the [UserConfig](#) object of this class, as selected services. Before initiating any job migration request, resource discovery and broker\* loading is carried out using the TargetGenerator and Broker classes, initialised by the [UserConfig](#) object of this class. If Broker loading fails, or no ExecutionTargets are found, an ERROR log message is reported and all IDFromEndpoint [URLs](#) for job considered for migration will be appended to the notmigrated list and then false will be returned.

When the above checks have been carried out successfully, the following is done for each job considered for migration. The ActivityOldID member of the Identification member in the job description will be set to that of the [Job](#) object, and the IDFromEndpoint [URL](#) will be appended to ActivityOldID member of the job description. After that the Broker object will be used to find a suitable [ExecutionTarget](#) object, and



if found a migrate request will tried sent using the `ExecutionTarget::Migrate` method, passing the `User-Config` object of this class. The passed forcemigration boolean indicates whether the migration request at the service side should ignore failures in cancelling the existing queuing job. If the request succeeds, the corresponding new `Job` object is appended to the `migratedJobs` list. If no suitable `ExecutionTarget` objects are found an ERROR log message is reported and the `IDFromEndpoint URL` of the `Job` object is appended to the `notmigrated` list. When all jobs have been processed, false is returned if any ERRORS were reported, otherwise true.

#### Parameters:

***forcemigration*** indicates whether migration should succeed if service fails to cancel the existing queuing job.

***migratedJobs*** list of `Job` objects which migrated jobs will be appended to.

***TODO***

#### Returns:

false if any error is encountered, otherwise true.

### 5.89.3.6 bool Arc::JobSupervisor::Renew ()

Renew job credentials.

This method will renew credentials of jobs managed by this `JobSupervisor`.

Before identifying jobs for which to renew credentials, the `JobControllerPlugin::UpdateJobs` method is called for each loaded `JobControllerPlugin` in order to retrieve the most up to date job information.

Since jobs in the `JobState::DELETED`, `JobState::FINISHED` or `JobState::KILLED` states is in a terminal state credentials for those jobs will not be renewed. Also jobs in the `JobState::UNDEFINED` state will not get their credentials renewed, since job information is not available. The `JobState::FAILED` state is also a terminal state, but since jobs in this state can be restarted, credentials for such jobs can be renewed. If the status-filter is non-empty, a renewal of credentials will be done for jobs with a general or specific state (see `JobState`) identical to any of the entries in the status-filter, excluding the already filtered states as mentioned above.

For each job for which to renew credentials, the specialized `JobControllerPlugin::RenewJob` method is called and is responsible for renewing the credentials for the given job. If the method fails to renew any job credentials, this method will return false (otherwise true), and the job ID (`IDFromEndpoint`) of such jobs is appended to the `notrenewed` list. The job ID of successfully renewed jobs will be appended to the passed renewed list.

#### Returns:

false if any call to `JobControllerPlugin::RenewJob` fails, true otherwise.

#### See also:

`JobControllerPlugin::RenewJob`.

### 5.89.3.7 bool Arc::JobSupervisor::Resubmit (int destination, const std::list< Endpoint > &, std::list< Job > & resubmittedJobs, const std::list< std::string > & = std::list< std::string >())

Resubmit jobs.

Jobs managed by this [JobSupervisor](#) will be resubmitted when invoking this method, that is the job description of a job will be tried obtained, and if successful a new job will be submitted.

Before identifying jobs to be resubmitted, the `JobControllerPlugin::UpdateJobs` method is called for each loaded `JobControllerPlugin` in order to retrieve the most up to date job information. If an empty status-filter is specified, all jobs managed by this [JobSupervisor](#) will be considered for resubmission, except jobs in the undefined state (see [JobState](#)). If the status-filter is not empty, then only jobs with a general or specific state (see [JobState](#)) identical to any of the entries in the status-filter will be considered, except jobs in the undefined state. Jobs for which a job description cannot be obtained and successfully parsed will not be considered and an ERROR log message is reported, and the `IDFromEndpoint URL` is appended to the `notresubmitted` list. [Job](#) descriptions will be tried obtained either from [Job](#) object itself, or fetching them remotely. Furthermore if a [Job](#) object has the `LocalInputFiles` object set, then the checksum of each of the local input files specified in that object (key) will be calculated and verified to match the checksum `LocalInputFiles` object (value). If checksums are not matching the job will be filtered, and an ERROR log message is reported and the `IDFromEndpoint URL` is appended to the `notresubmitted` list. If no job have been identified for resubmission, false will be returned if ERRORS were reported, otherwise true is returned.

The destination for jobs is partly determined by the destination parameter. If a value of 1 is specified a job will only be targeted to the execution service (ES) on which it reside. A value of 2 indicates that a job should not be targeted to the ES it currently reside. Specifying any other value will target any ES. The ESs which can be targeted are those specified in the [UserConfig](#) object of this class, as selected services. Before initiating any job submission, resource discovery and broker loading is carried out using the `Target-Generator` and `Broker` classes, initialised by the [UserConfig](#) object of this class. If Broker loading fails, or no `ExecutionTargets` are found, an ERROR log message is reported and all `IDFromEndpoint URLs` for job considered for resubmission will be appended to the `notresubmitted` list and then false will be returned.

When the above checks have been carried out successfully, then the `Broker::Submit` method will be invoked for each considered for resubmission. If it fails the `IDFromEndpoint URL` for the job is appended to the `notresubmitted` list, and an ERROR is reported. If submission succeeds the new job represented by a [Job](#) object will be appended to the `resubmittedJobs` list - it will not be added to this [JobSupervisor](#). The method returns false if ERRORS were reported otherwise true is returned.

#### Parameters:

*destination* specifies how target destination should be determined (1 = same target, 2 = not same, any other = any target).

*resubmittedJobs* list of [Job](#) objects which resubmitted jobs will be appended to.

**TODO**

#### Returns:

false if any error is encountered, otherwise true.

### 5.89.3.8 bool Arc::JobSupervisor::Resume ()

Resume jobs by status.

This method resumes jobs managed by this [JobSupervisor](#).

Before identifying jobs to resume, the `JobControllerPlugin::UpdateJobs` method is called for each loaded `JobControllerPlugin` in order to retrieve the most up to date job information.

Since jobs in the `JobState::DELETED`, `JobState::FINISHED` or `JobState::KILLED` states is in a terminal state credentials for those jobs will not be renewed. Also jobs in the `JobState::UNDEFINED` state will not be resumed, since job information is not available. The `JobState::FAILED` state is also a terminal state, but

jobs in this state are allowed to be restarted. If the status-filter is non-empty, only jobs with a general or specific state (see [JobState](#)) identical to any of the entries in the status-filter will be resumed, excluding the already filtered states as mentioned above.

For each job to resume, the specialized `JobControllerPlugin::ResumeJob` method is called and is responsible for resuming the particular job. If the method fails to resume a job, this method will return false, otherwise true is returned. The job ID of successfully resumed jobs will be appended to the passed `resumedJobs` list.

#### Returns:

false if any call to `JobControllerPlugin::ResumeJob` fails, true otherwise.

#### See also:

`JobControllerPlugin::ResumeJob`.

### 5.89.3.9 bool Arc::JobSupervisor::Retrieve (const std::string & *downloaddirprefix*, bool *usejobname*, bool *force*, std::list< std::string > & *downloaddirectories*)

Retrieve job output files.

This method retrieves output files of jobs managed by this [JobSupervisor](#).

Before identifying jobs for which to retrieve output files, the `JobControllerPlugin::UpdateJobs` method is called for each loaded `JobControllerPlugin` in order to retrieve the most up to date job information. If an empty status-filter is specified, all jobs managed by this [JobSupervisor](#) will be considered for retrieval, except jobs in the undefined state (see [JobState](#)). If the status-filter is not empty, then only jobs with a general or specific state (see [JobState](#)) identical to any of the entries in the status-filter will be considered, except jobs in the undefined state. Jobs in the state `JobState::DELETED` and unfinished jobs (see [JobState::IsFinished](#)) will also not be considered.

For each of the jobs considered for retrieval, the files will be downloaded to a directory named either as the last part of the job ID or the job name, which is determined by the '`usejobname`' argument. The download directories will be located in the directory specified by the '`downloaddir`' argument, as either a relative or absolute path. If the '`force`' argument is set to 'true', and a download directory for a given job already exist it will be overwritten, otherwise files for that job will not be downloaded. This method calls the `JobControllerPlugin::GetJob` method in order to download jobs, and if a job is successfully retrieved the job ID will be appended to the '`retrievedJobs`' list. If all jobs are successfully retrieved this method returns true, otherwise false.

#### Parameters:

*downloaddir* specifies the path to in which job download directories will be located.

*usejobname* specifies whether to use the job name or job ID as directory name to store job output files in.

*force* indicates whether existing job directories should be overwritten or not.

#### See also:

`JobControllerPlugin::RetrieveJob`.

#### Returns:

true if all jobs are successfully retrieved, otherwise false.

**5.89.3.10 void Arc::JobSupervisor::Update ()**

Update job information.

When invoking this method the job information for the jobs managed by this [JobSupervisor](#) will be updated. Internally, for each loaded JobControllerPlugin the JobControllerPlugin::UpdateJobs method will be called, which will be responsible for updating job information.

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

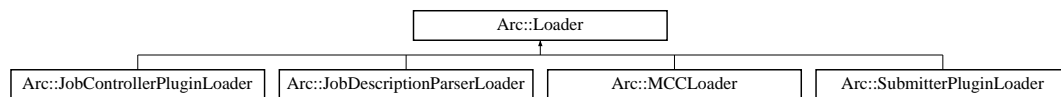
- JobSupervisor.h

## 5.90 Arc::Loader Class Reference

Plugins loader.

```
#include <Loader.h>
```

Inheritance diagram for Arc::Loader::



### Public Member Functions

- [Loader](#) ([XMLNode](#) cfg)
- [~Loader](#) ()

### Protected Attributes

- [PluginsFactory](#) \* [factory\\_](#)

### 5.90.1 Detailed Description

Plugins loader.

This class processes XML configuration and loads specified plugins. Accepted configuration is defined by XML schema mcc.xsd. "Plugins" elements are parsed by this class and corresponding libraries are loaded.

### 5.90.2 Constructor & Destructor Documentation

#### 5.90.2.1 Arc::Loader::Loader ([XMLNode](#) cfg)

Constructor that takes whole XML configuration and performs common configuration part

#### 5.90.2.2 Arc::Loader::~~Loader ()

Destructor destroys all components created by constructor

### 5.90.3 Field Documentation

#### 5.90.3.1 [PluginsFactory](#)\* [Arc::Loader::factory\\_](#) [protected]

Link to Factory responsible for loading and creation of [Plugin](#) and derived objects

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

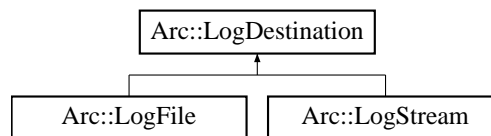
- [Loader.h](#)

## 5.91 Arc::LogDestination Class Reference

A base class for log destinations.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogDestination::



### Public Member Functions

- virtual void [log](#) (const [LogMessage](#) &message)=0

### Protected Member Functions

- [LogDestination](#) ()
- [LogDestination](#) (const std::string &locale)

#### 5.91.1 Detailed Description

A base class for log destinations.

This class defines an interface for LogDestinations. [LogDestination](#) objects will typically contain synchronization mechanisms and should therefore never be copied.

#### 5.91.2 Constructor & Destructor Documentation

##### 5.91.2.1 Arc::LogDestination::LogDestination () [protected]

Default constructor.

This destination will use the default locale.

##### 5.91.2.2 Arc::LogDestination::LogDestination (const std::string & locale) [protected]

Constructor with specific locale.

This destination will use the specified locale.

#### 5.91.3 Member Function Documentation

##### 5.91.3.1 virtual void Arc::LogDestination::log (const [LogMessage](#) & message) [pure virtual]

Logs a [LogMessage](#) to this [LogDestination](#).

Implemented in [Arc::LogStream](#), and [Arc::LogFile](#).

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

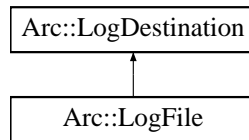
- [Logger.h](#)

## 5.92 Arc::LogFile Class Reference

A class for logging to files.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogFile::



### Public Member Functions

- [LogFile](#) (const std::string &path)
- [LogFile](#) (const std::string &path, const std::string &locale)
- void [setMaxSize](#) (int newsize)
- void [setBackups](#) (int newbackup)
- void [setReopen](#) (bool newreopen)
- [operator bool](#) (void)
- bool [operator!](#) (void)
- virtual void [log](#) (const [LogMessage](#) &message)

### 5.92.1 Detailed Description

A class for logging to files.

This class is used for logging to files. It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. It is possible to limit size of created file. Whenever specified size is exceeded file is deleted and new one is created. Old files may be moved into backup files instead of being deleted. Those files have names same as initial file with additional number suffix - similar to those found in /var/log of many Unix-like systems.

### 5.92.2 Constructor & Destructor Documentation

#### 5.92.2.1 Arc::LogFile::LogFile (const std::string & path)

Creates a [LogFile](#) connected to a file.

Creates a [LogFile](#) connected to the file located at specified path. In order not to break synchronization, it is important not to connect more than one [LogFile](#) object to a certain file. If file does not exist it will be created.

#### Parameters:

**path** The path to file to which to write LogMessages.



### 5.92.2.2 Arc::LogFile::LogFile (const std::string & path, const std::string & locale)

Creates a [LogFile](#) connected to a file.

Creates a [LogFile](#) connected to the file located at specified path. The output will be localised to the specified locale.

## 5.92.3 Member Function Documentation

### 5.92.3.1 virtual void Arc::LogFile::log (const [LogMessage](#) & message) [virtual]

Writes a [LogMessage](#) to the file.

This method writes a [LogMessage](#) to the file that is connected to this [LogFile](#) object. If after writitng size of file exceeds one set by [setMaxSize\(\)](#) file is moved to backup and new one is created.

#### Parameters:

*message* The [LogMessage](#) to write.

Implements [Arc::LogDestination](#).

### 5.92.3.2 Arc::LogFile::operator bool (void)

Returns true if this instance is valid.

### 5.92.3.3 bool Arc::LogFile::operator! (void)

Returns true if this instance is invalid.

### 5.92.3.4 void Arc::LogFile::setBackups (int newbackup)

Set number of backups to store.

Set number of backups to store. When file size exceeds one specified with [setMaxSize\(\)](#) file is closed and moved to one named path.1. If path.1 exists it is moved to path.2 and so on. Number of path.# files is one set in newbackup.

#### Parameters:

*newbackup* Number of backup files.

### 5.92.3.5 void Arc::LogFile::setMaxSize (int newsize)

Set maximal allowed size of file.

Set maximal allowed size of file. This value is not obeyed exactly. Spesified size may be exceeded by amount of one [LogMessage](#). To disable limit specify -1.

#### Parameters:

*newsize* Max size of log file.

**5.92.3.6 void Arc::LogFile::setReopen (bool *newreopen*)**

Set file reopen on every write.

Set file reopen on every write. If set to true file is opened before writing every log record and closed afterward.

**Parameters:**

*newreopen* If file to be reopened for every log record.

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

- Logger.h

## 5.93 Arc::Logger Class Reference

A logger class.

```
#include <Logger.h>
```

### Public Member Functions

- [Logger](#) ([Logger](#) &parent, const std::string &subdomain)
- [Logger](#) ([Logger](#) &parent, const std::string &subdomain, [LogLevel](#) threshold)
- [~Logger](#) ()
- void [addDestination](#) ([LogDestination](#) &destination)
- void [addDestinations](#) (const std::list< [LogDestination](#) \* > &destinations)
- const std::list< [LogDestination](#) \* > & [getDestinations](#) (void) const
- void [removeDestinations](#) (void)
- void [deleteDestinations](#) (void)
- void [setThreshold](#) ([LogLevel](#) threshold)
- [LogLevel](#) [getThreshold](#) () const
- void [setThreadContext](#) (void)
- void [msg](#) ([LogMessage](#) message)
- void [msg](#) ([LogLevel](#) level, const std::string &str)

### Static Public Member Functions

- static [Logger](#) & [getRootLogger](#) ()
- static void [setThresholdForDomain](#) ([LogLevel](#) threshold, const std::list< std::string > &subdomains)
- static void [setThresholdForDomain](#) ([LogLevel](#) threshold, const std::string &domain)

### 5.93.1 Detailed Description

A logger class.

This class defines a [Logger](#) to which LogMessages can be sent.

Every [Logger](#) (except for the rootLogger) has a parent [Logger](#). The domain of a [Logger](#) (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent [Logger](#).

A [Logger](#) also has a threshold. Every [LogMessage](#) that have a level that is greater than or equal to the threshold is forwarded to any [LogDestination](#) connected to this [Logger](#) as well as to the parent [Logger](#).

Typical usage of the [Logger](#) class is to declare a global [Logger](#) object for each library/module/component to be used by all classes and methods there.

### 5.93.2 Constructor & Destructor Documentation

#### 5.93.2.1 Arc::Logger::Logger ([Logger](#) &parent, const std::string &subdomain)

Creates a logger.

Creates a logger. The threshold is inherited from its parent [Logger](#).

**Parameters:**

*parent* The parent [Logger](#) of the new [Logger](#).

*subdomain* The subdomain of the new logger.

### 5.93.2.2 `Arc::Logger::Logger` ([Logger](#) & *parent*, const std::string & *subdomain*, [LogLevel](#) *threshold*)

Creates a logger.

Creates a logger.

**Parameters:**

*parent* The parent [Logger](#) of the new [Logger](#).

*subdomain* The subdomain of the new logger.

*threshold* The threshold of the new logger.

### 5.93.2.3 `Arc::Logger::~~Logger` ()

Destroys a logger.

Destructor

## 5.93.3 Member Function Documentation

### 5.93.3.1 `void Arc::Logger::addDestination` ([LogDestination](#) & *destination*)

Adds a [LogDestination](#).

Adds a [LogDestination](#) to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoin should not be copied, the new [LogDestination](#) is passed by reference and a pointer to it is kept for later use. It is therefore important that the [LogDestination](#) passed to this [Logger](#) exists at least as long as the [Logger](#) itself.

### 5.93.3.2 `void Arc::Logger::addDestinations` (const std::list< [LogDestination](#) \* > & *destinations*)

Adds LogDestinations.

See [addDestination\(LogDestination& destination\)](#).

### 5.93.3.3 `void Arc::Logger::deleteDestinations` (void)

Remove all LogDestinations and delete [LogDestination](#) objects.

### 5.93.3.4 `const std::list<LogDestination*>& Arc::Logger::getDestinations` (void) const

Obtains current LogDestinations.

Returns list of pointers to [LogDestination](#) objects. Returned result refers directly to internal member of [Logger](#) instance. Hence it should not be used after this [Logger](#) is destroyed.

**5.93.3.5 static [Logger](#)& Arc::Logger::getRootLogger () [static]**

The root [Logger](#).

This is the root [Logger](#). It is an ancestor of any other [Logger](#) and allways exists.

**5.93.3.6 [LogLevel](#) Arc::Logger::getThreshold () const**

Returns the threshold.

Returns the threshold.

**Returns:**

The threshold of this [Logger](#).

**5.93.3.7 void Arc::Logger::msg ([LogLevel](#) *level*, const std::string & *str*) [inline]**

Logs a message text.

Logs a message text string at the specified LogLevel. This is a convenience method to save some typing. It simply creates a [LogMessage](#) and sends it to the other [msg\(\)](#) method.

**Parameters:**

*level* The level of the message.

*str* The message text.

**5.93.3.8 void Arc::Logger::msg ([LogMessage](#) *message*)**

Sends a [LogMessage](#).

Sends a [LogMessage](#).

**Parameters:**

*The* [LogMessage](#) to send.

**5.93.3.9 void Arc::Logger::removeDestinations (void)**

Removes all LogDestinations.

**5.93.3.10 void Arc::Logger::setThreadContext (void)**

Creates per-thread context.

Creates new context for this logger which becomes effective for operations initiated by this thread. All new threads started by this one will inherit new context. Context stores current threshold and pointers to destinations. Hence new context is identical to current one. One can modify new context using [setThreshold\(\)](#), [removeDestinations\(\)](#) and [addDestination\(\)](#). All such operations will not affect old context.

**5.93.3.11 void Arc::Logger::setThreshold (LogLevel *threshold*)**

Sets the threshold.

This method sets the threshold of the [Logger](#). Any message sent to this [Logger](#) that has a level below this threshold will be discarded.

**Parameters:**

*The* threshold

**5.93.3.12 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold*, const std::string & *domain*) [static]**

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The domain is composed of all subdomains of all loggers in chain by merging them with '.' as separator.

**Parameters:**

*threshold* The threshold

*domain* The domain of logger

**5.93.3.13 static void Arc::Logger::setThresholdForDomain (LogLevel *threshold*, const std::list< std::string > & *subdomains*) [static]**

Sets the threshold for domain.

This method sets the default threshold of the domain. All new loggers created with specified domain will have specified threshold set by default. The subdomains of all loggers in chain are matched against list of provided subdomains.

**Parameters:**

*threshold* The threshold

*subdomains* The subdomains of all loggers in chain

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

- [Logger.h](#)

## 5.94 Arc::LoggerContext Class Reference

Container for logger configuration.

```
#include <Logger.h>
```

### 5.94.1 Detailed Description

Container for logger configuration.

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

- Logger.h

## 5.95 Arc::LogMessage Class Reference

A class for log messages.

```
#include <Logger.h>
```

### Public Member Functions

- [LogMessage](#) ([LogLevel](#) level, const [IString](#) &message)
- [LogMessage](#) ([LogLevel](#) level, const [IString](#) &message, const [std::string](#) &identifier)
- [LogLevel](#) [getLevel](#) () const

### Protected Member Functions

- void [setIdentifier](#) ([std::string](#) identifier)

### Friends

- class [Logger](#)
- [std::ostream](#) & [operator<<](#) ([std::ostream](#) &os, const [LogMessage](#) &message)

### 5.95.1 Detailed Description

A class for log messages.

This class is used to represent log messages internally. It contains the time the message was created, its level, from which domain it was sent, an identifier and the message text itself.

### 5.95.2 Constructor & Destructor Documentation

#### 5.95.2.1 Arc::LogMessage::LogMessage ([LogLevel](#) level, const [IString](#) & message)

Creates a [LogMessage](#) with the specified level and message text.

This constructor creates a [LogMessage](#) with the specified level and message text. The time is set automatically, the domain is set by the [Logger](#) to which the [LogMessage](#) is sent and the identifier is composed from the process ID and the address of the Thread object corresponding to the calling thread.

#### Parameters:

*level* The level of the [LogMessage](#).

*message* The message text.

#### 5.95.2.2 Arc::LogMessage::LogMessage ([LogLevel](#) level, const [IString](#) & message, const [std::string](#) & identifier)

Creates a [LogMessage](#) with the specified attributes.

This constructor creates a [LogMessage](#) with the specified level, message text and identifier. The time is set automatically and the domain is set by the [Logger](#) to which the [LogMessage](#) is sent.



**Parameters:**

*level* The level of the [LogMessage](#).  
*message* The message text.  
*ident* The identifier of the [LogMessage](#).

**5.95.3 Member Function Documentation****5.95.3.1 [LogLevel](#) Arc::LogMessage::getLevel () const**

Returns the level of the [LogMessage](#).

Returns the level of the [LogMessage](#).

**Returns:**

The level of the [LogMessage](#).

**5.95.3.2 void Arc::LogMessage::setIdentifier (std::string *identifier*) [protected]**

Sets the identifier of the [LogMessage](#).

The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a [LogMessage](#).

**Parameters:**

*The* identifier.

**5.95.4 Friends And Related Function Documentation****5.95.4.1 friend class [Logger](#) [friend]**

The [Logger](#) class is a friend.

The [Logger](#) class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

**5.95.4.2 std::ostream& operator<< (std::ostream & *os*, const [LogMessage](#) & *message*) [friend]**

Printing of LogMessages to ostreams.

Output operator so that LogMessages can be printed conveniently by LogDestinations.

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

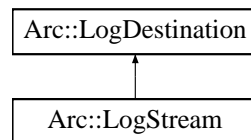
- [Logger.h](#)

## 5.96 Arc::LogStream Class Reference

A class for logging to ostreams.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogStream::



### Public Member Functions

- [LogStream](#) (std::ostream &destination)
- [LogStream](#) (std::ostream &destination, const std::string &locale)
- virtual void [log](#) (const [LogMessage](#) &message)

### 5.96.1 Detailed Description

A class for logging to ostreams.

This class is used for logging to ostreams (cout, cerr, files). It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. In order not to break the synchronization, LogStreams should never be copied. Therefore the copy constructor and assignment operator are private. Furthermore, it is important to keep a [LogStream](#) object as long as the [Logger](#) to which it has been registered.

### 5.96.2 Constructor & Destructor Documentation

#### 5.96.2.1 Arc::LogStream::LogStream (std::ostream & *destination*)

Creates a [LogStream](#) connected to an ostream.

Creates a [LogStream](#) connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one [LogStream](#) object to a certain stream.

#### Parameters:

*destination* The ostream to which to write LogMessages.

#### 5.96.2.2 Arc::LogStream::LogStream (std::ostream & *destination*, const std::string & *locale*)

Creates a [LogStream](#) connected to an ostream.

Creates a [LogStream](#) connected to the specified ostream. The output will be localised to the specified locale.

### 5.96.3 Member Function Documentation

#### 5.96.3.1 virtual void Arc::LogStream::log (const [LogMessage](#) & *message*) [virtual]

Writes a [LogMessage](#) to the stream.

This method writes a [LogMessage](#) to the ostream that is connected to this [LogStream](#) object. It is synchronized so that not more than one [LogMessage](#) can be written at a time.

##### Parameters:

*message* The [LogMessage](#) to write.

Implements [Arc::LogDestination](#).

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

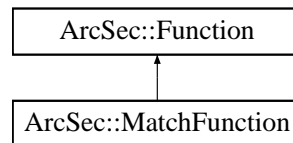
- [Logger.h](#)

## 5.97 ArcSec::MatchFunction Class Reference

Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).

```
#include <MatchFunction.h>
```

Inheritance diagram for ArcSec::MatchFunction::



### Public Member Functions

- virtual [AttributeValue](#) \* [evaluate](#) ([AttributeValue](#) \*arg0, [AttributeValue](#) \*arg1, bool check\_id=true)
- virtual std::list< [AttributeValue](#) \* > [evaluate](#) (std::list< [AttributeValue](#) \* > args, bool check\_id=true)

### Static Public Member Functions

- static std::string [getFunctionName](#) (std::string datatype)

#### 5.97.1 Detailed Description

Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).

#### 5.97.2 Member Function Documentation

**5.97.2.1** virtual std::list<[AttributeValue](#)\*> ArcSec::MatchFunction::evaluate (std::list<[AttributeValue](#) \* > args, bool check\_id = true) [virtual]

Evaluate a list of [AttributeValue](#) objects, and return a list of Attribute objects

Implements [ArcSec::Function](#).

**5.97.2.2** virtual [AttributeValue](#)\* ArcSec::MatchFunction::evaluate ([AttributeValue](#) \* arg0, [AttributeValue](#) \* arg1, bool check\_id = true) [virtual]

Evaluate two [AttributeValue](#) objects, and return one [AttributeValue](#) object

Implements [ArcSec::Function](#).

**5.97.2.3** static std::string ArcSec::MatchFunction::getFunctionName (std::string datatype) [static]

help function to get the FunctionName

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

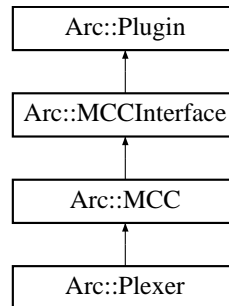
- [MatchFunction.h](#)

## 5.98 Arc::MCC Class Reference

[Message](#) Chain Component - base class for every [MCC](#) plugin.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCC::



### Public Member Functions

- [MCC](#) ([Config](#) \*, [PluginArgument](#) \*arg)
- virtual void [Next](#) ([MCCInterface](#) \*next, const std::string &label="")
- virtual void [AddSecHandler](#) ([Config](#) \*cfg, [ArcSec::SecHandler](#) \*sechandler, const std::string &label="")
- virtual void [Unlink](#) ()
- virtual [MCC\\_Status](#) process ([Message](#) &, [Message](#) &)

### Protected Member Functions

- bool [ProcessSecHandlers](#) ([Message](#) &message, const std::string &label="") const

### Protected Attributes

- std::map< std::string, [MCCInterface](#) \* > [next\\_](#)
- std::map< std::string, std::list< [ArcSec::SecHandler](#) \* > > [sechandlers\\_](#)

### Static Protected Attributes

- static [Logger](#) [logger](#)

#### 5.98.1 Detailed Description

[Message](#) Chain Component - base class for every [MCC](#) plugin.

This is partially virtual class which defines interface and common functionality for every [MCC](#) plugin needed for managing of component in a chain.

## 5.98.2 Constructor & Destructor Documentation

### 5.98.2.1 Arc::MCC::MCC ([Config](#) \*, [PluginArgument](#) \* *arg*) [inline]

Example constructor - [MCC](#) takes at least it's configuration subtree

## 5.98.3 Member Function Documentation

### 5.98.3.1 virtual void Arc::MCC::AddSecHandler ([Config](#) \* *cfg*, [ArcSec::SecHandler](#) \* *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this [MCC](#). Security handlers are stacked into a few queues with each queue identified by its label. The queue labelled 'incoming' is executed for every 'request' message after the message is processed by the [MCC](#) on the service side and before processing on the client side. The queue labelled 'outgoing' is run for response message before it is processed by [MCC](#) algorithms on the service side and after processing on the client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

### 5.98.3.2 virtual void Arc::MCC::Next ([MCCInterface](#) \* *next*, const std::string & *label* = "") [virtual]

Add reference to next [MCC](#) in chain. This method is called by [Loader](#) for every potentially labeled link to next component which implements [MCCInterface](#). If next is NULL corresponding link is removed.

Reimplemented in [Arc::Plexer](#).

### 5.98.3.3 virtual [MCC\\_Status](#) Arc::MCC::process ([Message](#) &, [Message](#) &) [inline, virtual]

Dummy [Message](#) processing method. Just a placeholder.

Implements [Arc::MCCInterface](#).

Reimplemented in [Arc::Plexer](#).

### 5.98.3.4 bool Arc::MCC::ProcessSecHandlers ([Message](#) & *message*, const std::string & *label* = "") const [protected]

Executes security handlers of specified queue. Returns true if the message is authorized for further processing or if there are no security handlers which implement authorization functionality. This is a convenience method and has to be called by the implementation of the [MCC](#).

### 5.98.3.5 virtual void Arc::MCC::Unlink () [virtual]

Removing all links. Useful for destroying chains.

## 5.98.4 Field Documentation

### 5.98.4.1 [Logger](#) Arc::MCC::logger [static, protected]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented in [Arc::Plexer](#).

#### **5.98.4.2** `std::map<std::string, MCCInterface *> Arc::MCC::next_` [protected]

Set of labeled "next" components. Each implemented [MCC](#) must call `process()` method of corresponding [MCCInterface](#) from this set in own `process()` method.

#### **5.98.4.3** `std::map<std::string, std::list<ArcSec::SecHandler *> > Arc::MCC::sechandlers_` [protected]

Set of labeled authentication and authorization handlers. [MCC](#) calls sequence of handlers at specific point depending on associated identifier. In most cases those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- [MCC.h](#)



## 5.99 Arc::MCC\_Status Class Reference

A class for communication of [MCC](#) processing results.

```
#include <MCC_Status.h>
```

### Public Member Functions

- [MCC\\_Status](#) ([StatusKind](#) kind=STATUS\_UNDEFINED, const std::string &origin="???", const std::string &explanation="No explanation.")
- bool [isOk](#) () const
- [StatusKind](#) [getKind](#) () const
- const std::string & [getOrigin](#) () const
- const std::string & [getExplanation](#) () const
- [operator std::string](#) () const
- [operator bool](#) (void) const
- bool [operator!](#) (void) const

### 5.99.1 Detailed Description

A class for communication of [MCC](#) processing results.

This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin ([MCC](#)) of the status object and an explanation.

### 5.99.2 Constructor & Destructor Documentation

**5.99.2.1** [Arc::MCC\\_Status::MCC\\_Status](#) ([StatusKind](#) *kind* = STATUS\_UNDEFINED, const std::string & *origin* = "???", const std::string & *explanation* = "No explanation.")

The constructor.

Creates a [MCC\\_Status](#) object.

#### Parameters:

- kind* The StatusKind (default: STATUS\_UNDEFINED)  
*origin* The origin [MCC](#) (default: "??")  
*explanation* An explanation (default: "No explanation.")

### 5.99.3 Member Function Documentation

**5.99.3.1** const std::string& [Arc::MCC\\_Status::getExplanation](#) () const

Returns an explanation.

This method returns an explanation of this object.

#### Returns:

An explanation of this object.

**5.99.3.2   [StatusKind](#) Arc::MCC\_Status::getKind () const**

Returns the status kind.

Returns the status kind of this object.

**Returns:**

The status kind of this object.

**5.99.3.3   const std::string& Arc::MCC\_Status::getOrigin () const**

Returns the origin.

This method returns a string specifying the origin [MCC](#) of this object.

**Returns:**

A string specifying the origin [MCC](#) of this object.

**5.99.3.4   bool Arc::MCC\_Status::isOk () const**

Is the status kind ok?

This method returns true if the status kind of this object is STATUS\_OK

**Returns:**

true if kind==STATUS\_OK

**5.99.3.5   Arc::MCC\_Status::operator bool (void) const   [inline]**

Is the status kind ok?

This method returns true if the status kind of this object is STATUS\_OK

**Returns:**

true if kind==STATUS\_OK

**5.99.3.6   Arc::MCC\_Status::operator std::string () const**

Conversion to string.

This operator converts a [MCC\\_Status](#) object to a string.

**5.99.3.7   bool Arc::MCC\_Status::operator! (void) const   [inline]**

not operator

Returns true if the status kind is not OK

**Returns:**

true if kind!=STATUS\_OK

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

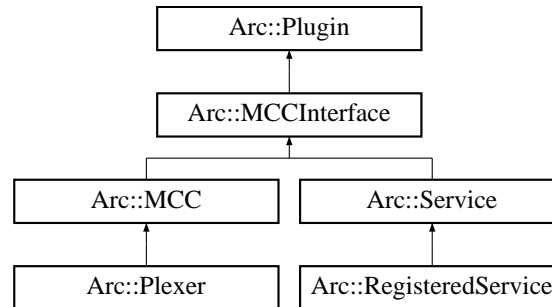
- MCC\_Status.h

## 5.100 Arc::MCCInterface Class Reference

Interface for communication between [MCC](#), [Service](#) and [Plexer](#) objects.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCCInterface::



### Public Member Functions

- virtual [MCC\\_Status](#) process ([Message](#) &request, [Message](#) &response)=0

#### 5.100.1 Detailed Description

Interface for communication between [MCC](#), [Service](#) and [Plexer](#) objects.

The Interface consists of the method [process\(\)](#) which is called by the previous [MCC](#) in the chain. For memory management policies please read the description of the [Message](#) class.

#### 5.100.2 Member Function Documentation

##### 5.100.2.1 virtual [MCC\\_Status](#) Arc::MCCInterface::process ([Message](#) & request, [Message](#) & response) [pure virtual]

Method for processing of requests and responses. This method is called by preceeding [MCC](#) in chain when a request needs to be processed. This method must call similar method of next [MCC](#) in chain unless any failure happens. Result returned by call to next [MCC](#) should be processed and passed back to previous [MCC](#). In case of failure this method is expected to generate valid error response and return it back to previous [MCC](#) without calling the next one.

##### Parameters:

*request* The request that needs to be processed.

*response* A [Message](#) object that will contain the response of the request when the method returns.

##### Returns:

An object representing the status of the call.

Implemented in [Arc::MCC](#), and [Arc::Plexer](#).

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

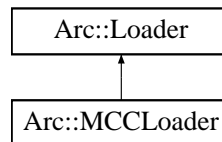
- MCC.h

## 5.101 Arc::MCCLoader Class Reference

Creator of [Message](#) Component Chains ([MCC](#)).

```
#include <MCCLoader.h>
```

Inheritance diagram for Arc::MCCLoader::



### Public Member Functions

- [MCCLoader](#) ([Config](#) &cfg)
- [~MCCLoader](#) ()
- [MCC](#) \* [operator\[\]](#) (const std::string &id)

#### 5.101.1 Detailed Description

Creator of [Message](#) Component Chains ([MCC](#)).

This class processes XML configuration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types [MCC](#), [Service](#) and [Plexer](#). [MCC](#) and [Service](#) are loaded from dynamic libraries. For [Plexer](#) only internal implementation is supported. This object is also a container for loaded componets. All components and chains are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructors are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each call creates labeled link to next component in a chain. 2 step method has an advantage over single step because it allows loops in chains and makes loading procedure more simple. But that also means during short period of time components are only partly configured. Components in such state must produce proper error response if [Message](#) arrives. Note: Current implementation requires all components and links to be labeled. All labels must be unique. Future implementation will be able to assign labels automatically.

#### 5.101.2 Constructor & Destructor Documentation

##### 5.101.2.1 Arc::MCCLoader::MCCLoader ([Config](#) & cfg)

Constructor that takes whole XML configuration and creates component chains

##### 5.101.2.2 Arc::MCCLoader::~~MCCLoader ()

Destructor destroys all components created by constructor

### 5.101.3 Member Function Documentation

#### 5.101.3.1 ]

[MCC](#)\* Arc::MCCLoader::operator[ ] (const std::string & *id*)

Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute

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

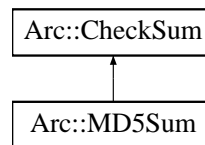
- MCCLoader.h

## 5.102 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

```
#include <CheckSum.h>
```

Inheritance diagram for Arc::MD5Sum::



### Public Member Functions

- virtual void [start](#) (void)
- virtual void [add](#) (void \*buf, unsigned long long int len)
- virtual void [end](#) (void)
- virtual void [result](#) (unsigned char \*&res, unsigned int &len) const
- virtual int [print](#) (char \*buf, int len) const
- virtual void [scan](#) (const char \*buf)
- virtual [operator bool](#) (void) const
- virtual bool [operator!](#) (void) const

### 5.102.1 Detailed Description

Implementation of MD5 checksum.

This class is a specialized class of the [CheckSum](#) class. It provides an implementation of the MD5 message-digest algorithm specified in RFC 1321.

### 5.102.2 Member Function Documentation

#### 5.102.2.1 virtual void Arc::MD5Sum::add (void \* *buf*, unsigned long long int *len*) [virtual]

Add data to be checksummed.

This method calculates the checksum of the passed data chunk, taking into account the previous state of this object.

#### Parameters:

*buf* pointer to data chunk to be checksummed.

*len* size of the data chunk.

Implements [Arc::CheckSum](#).



**5.102.2.2 virtual void Arc::MD5Sum::end (void) [virtual]**

Finalize the checksumming.

This method finalizes the checksum algorithm, that is calculating the final checksum result.

Implements [Arc::Checksum](#).

**5.102.2.3 virtual Arc::MD5Sum::operator bool (void) const [inline, virtual]**

Indicates whether the checksum has been calculated.

Reimplemented from [Arc::Checksum](#).

**5.102.2.4 virtual bool Arc::MD5Sum::operator! (void) const [inline, virtual]**

Indicates whether the checksum has not been calculated.

Reimplemented from [Arc::Checksum](#).

**5.102.2.5 virtual int Arc::MD5Sum::print (char \* *buf*, int *len*) const [virtual]**

Retrieve result of checksum into a string.

The passed string *buf* is filled with result of checksum algorithm in base 16. At most *len* characters is filled into buffer *buf*. The hexadecimal value is prepended with "<algorithm>:", where <algorithm> is one of "cksum", "md5" or "adler32" respectively corresponding to the result from the [CRC32Sum](#), [MD5Sum](#) and Adler32 classes.

**Parameters:**

*buf* pointer to buffer which should be filled with checksum result.

*len* max number of character filled into buffer.

Reimplemented from [Arc::Checksum](#).

**5.102.2.6 virtual void Arc::MD5Sum::result (unsigned char \*& *res*, unsigned int & *len*) const [inline, virtual]**

Retrieve result of checksum as binary blob.

Implements [Arc::Checksum](#).

**5.102.2.7 virtual void Arc::MD5Sum::scan (const char \* *buf*) [virtual]**

Set internal checksum state.

This method sets the internal state to that of the passed textual representation. The format passed to this method must be the same as retrieved from the [Checksum::print](#) method.

**Parameters:**

*buf* string containing textual representation of checksum

See also:

[Checksum::print](#)

Implements [Arc::Checksum](#).

#### **5.102.2.8** `virtual void Arc::MD5Sum::start (void)` [virtual]

Initiate the checksum algorithm.

This method must be called before starting a new checksum calculation.

Implements [Arc::Checksum](#).

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

- CheckSum.h

## 5.103 Arc::Message Class Reference

Object being passed through chain of MCCs.

```
#include <Message.h>
```

### Public Member Functions

- [Message](#) (void)
- [Message](#) ([Message](#) &msg)
- [Message](#) (long msg\_ptr\_addr)
- [~Message](#) (void)
- [Message](#) & [operator=](#) ([Message](#) &msg)
- [MessagePayload](#) \* [Payload](#) (void)
- [MessagePayload](#) \* [Payload](#) ([MessagePayload](#) \*payload)
- [MessageAttributes](#) \* [Attributes](#) (void)
- [MessageAuth](#) \* [Auth](#) (void)
- [MessageContext](#) \* [Context](#) (void)
- [MessageAuthContext](#) \* [AuthContext](#) (void)
- void [Context](#) ([MessageContext](#) \*ctx)
- void [AuthContext](#) ([MessageAuthContext](#) \*auth\_ctx)

### 5.103.1 Detailed Description

Object being passed through chain of MCCs.

An instance of this class refers to objects with main content ([MessagePayload](#)), authentication/authorization information ([MessageAuth](#)) and common purpose attributes ([MessageAttributes](#)). [Message](#) class does not manage pointers to objects and their content. It only serves for grouping those objects. [Message](#) objects are supposed to be processed by MCCs and Services implementing [MCCInterface](#) method process(). All objects constituting content of [Message](#) object are subject to following policies:

1. All objects created inside call to process() method using new command must be explicitly destroyed within same call using delete command with following exceptions. a) Objects which are assigned to 'response' [Message](#). b) Objects whose management is completely acquired by objects assigned to 'response' [Message](#).
2. All objects not created inside call to process() method are not explicitly destroyed within that call with following exception. a) Objects which are part of 'response' Method returned from call to next's process() method. Unless those objects are passed further to calling process(), of course.
3. It is not allowed to make 'response' point to same objects as 'request' does on entry to process() method. That is needed to avoid double destruction of same object. (Note: if in a future such need arises it may be solved by storing additional flags in [Message](#) object).
4. It is allowed to change content of pointers of 'request' [Message](#). Calling process() method must not rely on that object to stay intact.
5. Called process() method should either fill 'response' [Message](#) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

## 5.103.2 Constructor & Destructor Documentation

### 5.103.2.1 `Arc::Message::Message (void)` `[inline]`

Dummy constructor

### 5.103.2.2 `Arc::Message::Message (Message & msg)` `[inline]`

Copy constructor. Ensures shallow copy.

### 5.103.2.3 `Arc::Message::Message (long msg_ptr_addr)`

Copy constructor. Used by language bindings

### 5.103.2.4 `Arc::Message::~~Message (void)` `[inline]`

Destructor does not affect referred objects except those created internally

## 5.103.3 Member Function Documentation

### 5.103.3.1 `MessageAttributes*` `Arc::Message::Attributes (void)` `[inline]`

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

### 5.103.3.2 `MessageAuth*` `Arc::Message::Auth (void)` `[inline]`

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

### 5.103.3.3 `void Arc::Message::AuthContext (MessageAuthContext * auth_ctx)` `[inline]`

Assigns auth\* context object

### 5.103.3.4 `MessageAuthContext*` `Arc::Message::AuthContext (void)` `[inline]`

Returns a pointer to the current auth\* context object or creates it if no object has been assigned.

### 5.103.3.5 `void Arc::Message::Context (MessageContext * ctx)` `[inline]`

Assigns message context object

### 5.103.3.6 `MessageContext*` `Arc::Message::Context (void)` `[inline]`

Returns a pointer to the current context object or creates it if no object has been assigned. Last case should happen only if first [MCC](#) in a chain is connectionless like one implementing UDP protocol.

**5.103.3.7** `Message& Arc::Message::operator= (Message & msg)` [inline]

Assignment. Ensures shallow copy.

**5.103.3.8** `MessagePayload* Arc::Message::Payload (MessagePayload * payload)` [inline]

Replaces payload with new one. Returns the old one.

**5.103.3.9** `MessagePayload* Arc::Message::Payload (void)` [inline]

Returns pointer to current payload or NULL if no payload assigned.

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

- Message.h

## 5.104 Arc::MessageAttributes Class Reference

A class for storage of attribute values.

```
#include <MessageAttributes.h>
```

### Public Member Functions

- [MessageAttributes](#) ()
- void [set](#) (const std::string &key, const std::string &value)
- void [add](#) (const std::string &key, const std::string &value)
- void [removeAll](#) (const std::string &key)
- void [remove](#) (const std::string &key, const std::string &value)
- int [count](#) (const std::string &key) const
- const std::string & [get](#) (const std::string &key) const
- [AttributeIterator](#) [getAll](#) (const std::string &key) const
- [AttributeIterator](#) [getAll](#) (void) const

### Protected Attributes

- [AttrMap](#) [attributes\\_](#)

### 5.104.1 Detailed Description

A class for storage of attribute values.

This class is used to store attributes of messages. All attribute keys and their corresponding values are stored as strings. Any key or value that is not a string must thus be represented as a string during storage. Furthermore, an attribute is usually a key-value pair with a unique key, but there may also be multiple such pairs with equal keys.

The key of an attribute is composed by the name of the [Message](#) Chain Component ([MCC](#)) which produce it and the name of the attribute itself with a colon (:) in between, i.e. MCC\_Name:Attribute\_Name. For example, the key of the "Content-Length" attribute of the HTTP [MCC](#) is thus "HTTP:Content-Length".

There are also "global attributes", which may be produced by different MCCs depending on the configuration. The keys of such attributes are NOT prefixed by the name of the producing [MCC](#). Before any new global attribute is introduced, it must be agreed upon by the core development team and added below. The global attributes decided so far are:

- `Request-URI` Identifies the service to which the message shall be sent. This attribute is produced by e.g. the HTTP [MCC](#) and used by the plexer for routing the message to the appropriate service.

### 5.104.2 Constructor & Destructor Documentation

#### 5.104.2.1 Arc::MessageAttributes::MessageAttributes ()

The default constructor.

This is the default constructor of the [MessageAttributes](#) class. It constructs an empty object that initially contains no attributes.

### 5.104.3 Member Function Documentation

#### 5.104.3.1 void Arc::MessageAttributes::add (const std::string & *key*, const std::string & *value*)

Adds a value to an attribute.

This method adds a new value to an attribute. Any previous value will be preserved, i.e. the attribute may become multiple valued.

**Parameters:**

*key* The key of the attribute.

*value* The (new) value of the attribute.

#### 5.104.3.2 int Arc::MessageAttributes::count (const std::string & *key*) const

Returns the number of values of an attribute.

Returns the number of values of an attribute that matches a certain key.

**Parameters:**

*key* The key of the attribute for which to count values.

**Returns:**

The number of values that corresponds to the key.

#### 5.104.3.3 const std::string& Arc::MessageAttributes::get (const std::string & *key*) const

Returns the value of a single-valued attribute.

This method returns the value of a single-valued attribute. If the attribute is not single valued (i.e. there is no such attribute or it is a multiple-valued attribute) an empty string is returned.

**Parameters:**

*key* The key of the attribute for which to return the value.

**Returns:**

The value of the attribute.

#### 5.104.3.4 [AttributeIterator](#) Arc::MessageAttributes::getAll (void) const

Access all value and attributes.

#### 5.104.3.5 [AttributeIterator](#) Arc::MessageAttributes::getAll (const std::string & *key*) const

Access the value(s) of an attribute.

This method returns an [AttributeIterator](#) that can be used to access the values of an attribute.

**Parameters:**

*key* The key of the attribute for which to return the values.

**Returns:**

An [AttributeIterator](#) for access of the values of the attribute.

**5.104.3.6 void Arc::MessageAttributes::remove (const std::string & key, const std::string & value)**

Removes one value of an attribute.

This method removes a certain value from the attribute that matches a certain key.

**Parameters:**

*key* The key of the attribute from which the value shall be removed.

*value* The value to remove.

**5.104.3.7 void Arc::MessageAttributes::removeAll (const std::string & key)**

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

**Parameters:**

*key* The key of the attributes to remove.

**5.104.3.8 void Arc::MessageAttributes::set (const std::string & key, const std::string & value)**

Sets a unique value of an attribute.

This method removes any previous value of an attribute and sets the new value as the only value.

**Parameters:**

*key* The key of the attribute.

*value* The (new) value of the attribute.

**5.104.4 Field Documentation****5.104.4.1 [AttrMap Arc::MessageAttributes::attributes\\_](#) [protected]**

Internal storage of attributes.

An AttrMap (multimap) in which all attributes (key-value pairs) are stored.

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

- MessageAttributes.h

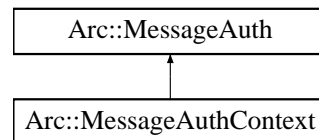


## 5.105 Arc::MessageAuth Class Reference

Contains authenticity information, authorization tokens and decisions.

```
#include <MessageAuth.h>
```

Inheritance diagram for Arc::MessageAuth::



### Public Member Functions

- void [set](#) (const std::string &key, [SecAttr](#) \*value)
- void [remove](#) (const std::string &key)
- [SecAttr](#) \* [get](#) (const std::string &key)
- [SecAttr](#) \* [operator\[\]](#) (const std::string &key)
- bool [Export](#) ([SecAttrFormat](#) format, [XMLNode](#) &val) const
- [MessageAuth](#) \* [Filter](#) (const std::list< std::string > &selected\_keys, const std::list< std::string > &rejected\_keys)

### 5.105.1 Detailed Description

Contains authenticity information, authorization tokens and decisions.

This class only supports string keys and [SecAttr](#) values.

### 5.105.2 Member Function Documentation

#### 5.105.2.1 bool Arc::MessageAuth::Export ([SecAttrFormat](#) format, [XMLNode](#) & val) const

Returns properly catenated attributes in specified format.

Content of XML node at is replaced with generated information if XML tree is empty. If tree at is not empty then [Export\(\)](#) tries to merge generated information to already existing like everything would be generated inside same [Export\(\)](#) method. If does not represent valid node then new XML tree is created.

#### 5.105.2.2 [MessageAuth](#)\* Arc::MessageAuth::Filter (const std::list< std::string > & selected\_keys, const std::list< std::string > & rejected\_keys)

Creates new instance of [MessageAuth](#) with attributes filtered.

In new instance all attributes with keys listed in are removed. If is not empty only corresponding attributes are transferred to new instance. Created instance does not own refered attributes. Hence parent instance must not be deleted as long as this one is in use.

#### 5.105.2.3 [SecAttr](#)\* Arc::MessageAuth::get (const std::string & key)

Retrieves reference to security attribute stored under specified key.

**5.105.2.4**   `]`

[SecAttr](#)\* `Arc::MessageAuth::operator[]` (`const std::string & key`)   `[inline]`

Same as [MessageAuth::get](#).

**5.105.2.5**   `void Arc::MessageAuth::remove` (`const std::string & key`)

Deletes security attribute stored under specified key.

**5.105.2.6**   `void Arc::MessageAuth::set` (`const std::string & key`, [SecAttr](#) \* `value`)

Adds/overwrites security attribute stored under specified key.

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

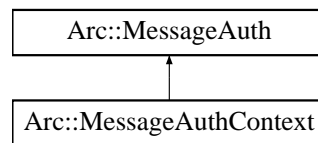
- `MessageAuth.h`

## 5.106 Arc::MessageAuthContext Class Reference

Handler for content of message auth\* context.

```
#include <Message.h>
```

Inheritance diagram for Arc::MessageAuthContext::



### 5.106.1 Detailed Description

Handler for content of message auth\* context.

This class is a container for authorization and authentication information. It gets associated with [Message](#) object usually by first [MCC](#) in a chain and is kept as long as connection persists.

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

- Message.h

## 5.107 Arc::MessageContext Class Reference

Handler for content of message context.

```
#include <Message.h>
```

### Public Member Functions

- void [Add](#) (const std::string &name, [MessageContextElement](#) \*element)

#### 5.107.1 Detailed Description

Handler for content of message context.

This class is a container for objects derived from [MessageContextElement](#). It gets associated with [Message](#) object usually by first [MCC](#) in a chain and is kept as long as connection persists.

#### 5.107.2 Member Function Documentation

##### 5.107.2.1 void Arc::MessageContext::Add (const std::string & name, [MessageContextElement](#) \* element)

Provided element is taken over by this class. It is remembered by it and destroyed when this class is destroyed.

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

- Message.h

## 5.108 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

```
#include <Message.h>
```

### 5.108.1 Detailed Description

Top class for elements contained in message context.

Objects of classes inherited with this one may be stored in [MessageContext](#) container.

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

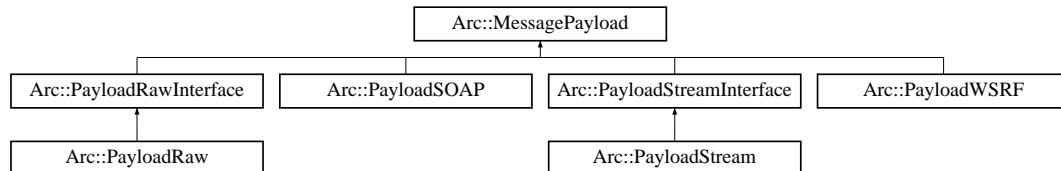
- Message.h

## 5.109 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

```
#include <Message.h>
```

Inheritance diagram for Arc::MessagePayload::



### 5.109.1 Detailed Description

Base class for content of message passed through chain.

It's not intended to be used directly. Instead functional classes must be derived from it.

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

- Message.h

## 5.110 Arc::ModuleDesc Class Reference

Description of loadable module.

```
#include <Plugin.h>
```

### 5.110.1 Detailed Description

Description of loadable module.

This class is used for reports

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

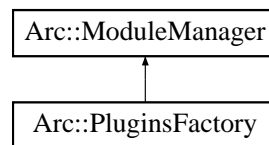
- Plugin.h

## 5.111 Arc::ModuleManager Class Reference

Manager of shared libraries.

```
#include <ModuleManager.h>
```

Inheritance diagram for Arc::ModuleManager::



### Public Member Functions

- [ModuleManager](#) ([XMLNode](#) cfg)
- Glib::Module \* [load](#) (const std::string &name, bool probe)
- std::string [find](#) (const std::string &name)
- Glib::Module \* [reload](#) (Glib::Module \*module)
- void [use](#) (Glib::Module \*module)
- void [unuse](#) (Glib::Module \*module)
- std::string [findLocation](#) (const std::string &name)
- bool [makePersistent](#) (Glib::Module \*module)
- bool [makePersistent](#) (const std::string &name)
- void [setCfg](#) ([XMLNode](#) cfg)

### Protected Member Functions

- void [unload](#) (Glib::Module \*module)
- void [unload](#) (const std::string &name)

#### 5.111.1 Detailed Description

Manager of shared libraries.

This class loads shared libraries/modules. There supposed to be created one instance of it per executable. In such circumstances it would cache handles to loaded modules and not load them multiple times.

#### 5.111.2 Constructor & Destructor Documentation

##### 5.111.2.1 Arc::ModuleManager::ModuleManager ([XMLNode](#) cfg)

Constructor. It is supposed to process correponding configuration subtree and tune module loading parameters accordingly.



### 5.111.3 Member Function Documentation

#### 5.111.3.1 `std::string Arc::ModuleManager::find (const std::string & name)`

Finds loadable module by 'name' looking in same places as `load()` does, but does not load it.

#### 5.111.3.2 `std::string Arc::ModuleManager::findLocation (const std::string & name)`

Finds shared library corresponding to module 'name' and returns path to it

#### 5.111.3.3 `Glib::Module* Arc::ModuleManager::load (const std::string & name, bool probe)`

Finds module 'name' in cache or loads corresponding loadable module

#### 5.111.3.4 `bool Arc::ModuleManager::makePersistent (const std::string & name)`

Make sure this module is never unloaded. Even if `unload()` is called.

#### 5.111.3.5 `bool Arc::ModuleManager::makePersistent (Glib::Module * module)`

Make sure this module is never unloaded. Even if `unload()` is called. Call to this method does not affect how other methods are behaving. Just loaded module stays in memory after all unloading procedures.

#### 5.111.3.6 `Glib::Module* Arc::ModuleManager::reload (Glib::Module * module)`

Reload module previously loaded in probe mode. New module is loaded with all symbols resolved and old module handler is unloaded. In case of error old module is not unloaded.

#### 5.111.3.7 `void Arc::ModuleManager::setCfg (XMLNode cfg)`

Input the configuration subtree, and trigger the module loading (do almost the same as the Constructor). This method is designed for ClassLoader to adopt the singleton pattern.

#### 5.111.3.8 `void Arc::ModuleManager::unload (const std::string & name)` [protected]

Unload module by its name

#### 5.111.3.9 `void Arc::ModuleManager::unload (Glib::Module * module)` [protected]

Unload module by its identifier. Decreases load counter and unloads module when it reaches 0.

#### 5.111.3.10 `void Arc::ModuleManager::unuse (Glib::Module * module)`

Decrease usage count till it reaches 0. This call does not unload module. Usage counter is only for preventing unexpected unload. Unloading is done by `unload()` methods and by destructor if usage counter is zero.

**5.111.3.11 void Arc::ModuleManager::use (Glib::Module \* *module*)**

Increase usage count of loaded module. It is intended to be called by plugins or other code which needs prevent module to be unloaded while its code is running. Must be accompanied by `unuse` when module is not needed.

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

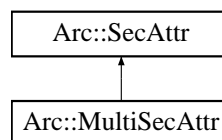
- ModuleManager.h

## 5.112 Arc::MultiSecAttr Class Reference

Container of multiple [SecAttr](#) attributes.

```
#include <SecAttr.h>
```

Inheritance diagram for Arc::MultiSecAttr::



### Public Member Functions

- virtual [operator bool](#) () const
- virtual bool [Export](#) ([SecAttrFormat](#) format, [XMLNode](#) &val) const

#### 5.112.1 Detailed Description

Container of multiple [SecAttr](#) attributes.

This class combines multiple attributes. It's export/import methods catenate results of underlying objects. Primary meaning of this class is to serve as base for classes implementing multi level hierarchical tree-like descriptions of user identity. It may also be used for collecting information of same source or kind. Like all information extracted from X509 certificate.

#### 5.112.2 Member Function Documentation

**5.112.2.1** virtual bool Arc::MultiSecAttr::Export ([SecAttrFormat](#) format, [XMLNode](#) & val) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented from [Arc::SecAttr](#).

**5.112.2.2** virtual Arc::MultiSecAttr::operator bool () const [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented from [Arc::SecAttr](#).

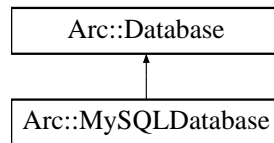
The documentation for this class was generated from the following file:

- SecAttr.h

## 5.113 Arc::MySQLDatabase Class Reference

```
#include <MysqlWrapper.h>
```

Inheritance diagram for Arc::MySQLDatabase::



### Public Member Functions

- virtual bool [connect](#) (std::string &dbname, std::string &user, std::string &password)
- virtual bool [isconnected](#) () const
- virtual void [close](#) ()
- virtual bool [enable\\_ssl](#) (const std::string &keyfile="", const std::string &certfile="", const std::string &cafile="", const std::string &capath="")
- virtual bool [shutdown](#) ()

### 5.113.1 Detailed Description

Implement the database accessing interface in [DBInterface.h](#) by using mysql client library for accessing mysql database

### 5.113.2 Member Function Documentation

#### 5.113.2.1 virtual void Arc::MySQLDatabase::close () [virtual]

Close the connection with database server

Implements [Arc::Database](#).

#### 5.113.2.2 virtual bool Arc::MySQLDatabase::connect (std::string & dbname, std::string & user, std::string & password) [virtual]

Do connection with database server

##### Parameters:

**dbname** The database name which will be used.

**user** The username which will be used to access database.

**password** The password which will be used to access database.

Implements [Arc::Database](#).

**5.113.2.3** `virtual bool Arc::MySQLDatabase::enable_ssl (const std::string & keyfile = "", const std::string & certfile = "", const std::string & cafile = "", const std::string & capath = "")` [virtual]

Enable ssl communication for the connection

**Parameters:**

*keyfile* The location of key file.

*certfile* The location of certificate file.

*cafile* The location of ca file.

*capath* The location of ca directory

Implements [Arc::Database](#).

**5.113.2.4** `virtual bool Arc::MySQLDatabase::isconnected () const` [inline, virtual]

Get the connection status

Implements [Arc::Database](#).

**5.113.2.5** `virtual bool Arc::MySQLDatabase::shutdown ()` [virtual]

Ask database server to shutdown

Implements [Arc::Database](#).

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

- MysqlWrapper.h

## 5.114 Arc::OAuthConsumer Class Reference

```
#include <OAuthConsumer.h>
```

### Public Member Functions

- [OAuthConsumer](#) (const MCCCConfig *cfg*, const [URL](#) *url*, std::list< std::string > *idp\_stack*)
- [MCC\\_Status parseDN](#) (std::string \**dn*)
- [MCC\\_Status approveCSR](#) (const std::string *approve\_page*)
- [MCC\\_Status pushCSR](#) (const std::string *b64\_pub\_key*, const std::string *pub\_key\_hash*, std::string \**approve\_page*)
- [MCC\\_Status storeCert](#) (const std::string *cert\_path*, const std::string *auth\_token*, const std::string *b64\_dn*)

### Protected Member Functions

- [MCC\\_Status processLogin](#) (const std::string *username*="", const std::string *password*="")

#### 5.114.1 Detailed Description

The OAuth functionality depends on the availability of the liboauth C-bindings library

#### 5.114.2 Constructor & Destructor Documentation

##### 5.114.2.1 Arc::OAuthConsumer::OAuthConsumer (const MCCCConfig *cfg*, const [URL](#) *url*, std::list< std::string > *idp\_stack*)

Construct an OAuth consumer with *url* as service provider. *idp\_name* is currently ignored, since the idp to which the SAML2 redirect will take place is presently a hardcoded value on the SAML2 SP side. This is expected to change in the future.

#### 5.114.3 Member Function Documentation

##### 5.114.3.1 [MCC\\_Status](#) Arc::OAuthConsumer::approveCSR (const std::string *approve\_page*)

Unsupported placeholder function until Confusa supports OAuth.

##### 5.114.3.2 [MCC\\_Status](#) Arc::OAuthConsumer::parseDN (std::string \* *dn*)

Unsupported placeholder function until Confusa supports OAuth.

##### 5.114.3.3 [MCC\\_Status](#) Arc::OAuthConsumer::processLogin (const std::string *username* = "", const std::string *password* = "") [protected]

Main function performing all the OAuth login steps. Username and password will be ignored.

#### 5.114.3.4 [MCC\\_Status](#) Arc::OAuthConsumer::pushCSR (const std::string *b64\_pub\_key*, const std::string *pub\_key\_hash*, std::string \* *approve\_page*)

Unsupported placeholder function until Confusa supports OAuth.

#### 5.114.3.5 [MCC\\_Status](#) Arc::OAuthConsumer::storeCert (const std::string *cert\_path*, const std::string *auth\_token*, const std::string *b64\_dn*)

Unsupported placeholder function until Confusa supports OAuth.

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

- OAuthConsumer.h

## 5.115 Arc::PathIterator Class Reference

Class to iterate through elements of path.

```
#include <URL.h>
```

### Public Member Functions

- [PathIterator](#) (const std::string &path, bool end=false)
- [PathIterator](#) & [operator++](#) ()
- [PathIterator](#) & [operator--](#) ()
- [operator bool](#) () const
- std::string [operator \\*](#) () const
- std::string [Rest](#) () const

### 5.115.1 Detailed Description

Class to iterate through elements of path.

### 5.115.2 Constructor & Destructor Documentation

#### 5.115.2.1 Arc::PathIterator::PathIterator (const std::string &path, bool end = false)

Constructor accepts path and stores it internally. If end is set to false iterator is pointing at first element in path. Otherwise selected element is one before last.

### 5.115.3 Member Function Documentation

#### 5.115.3.1 std::string Arc::PathIterator::operator \* () const

Returns part of initial path from first till and including current

#### 5.115.3.2 Arc::PathIterator::operator bool () const

Return false when iterator moved outside path elements

#### 5.115.3.3 [PathIterator](#)& Arc::PathIterator::operator++ ()

Advances iterator to point at next path element

#### 5.115.3.4 [PathIterator](#)& Arc::PathIterator::operator-- ()

Moves iterator to element before current



#### 5.115.3.5 std::string Arc::PathIterator::Rest () const

Returns part of initial path from one after current till end

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

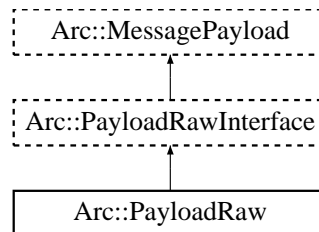
- URL.h

## 5.116 Arc::PayloadRaw Class Reference

Raw byte multi-buffer.

```
#include <PayloadRaw.h>
```

Inheritance diagram for Arc::PayloadRaw::



### Public Member Functions

- [PayloadRaw](#) (void)
- virtual [~PayloadRaw](#) (void)
- virtual [Size\\_t Size](#) (void) const
- virtual [char \\* Buffer](#) (unsigned int num=0)
- virtual [Size\\_t BufferSize](#) (unsigned int num=0) const
- virtual [Size\\_t BufferPos](#) (unsigned int num=0) const

### 5.116.1 Detailed Description

Raw byte multi-buffer.

This is implementation of [PayloadRawInterface](#). Buffers are memory blocks logically placed one after another.

### 5.116.2 Constructor & Destructor Documentation

#### 5.116.2.1 Arc::PayloadRaw::PayloadRaw (void) [inline]

Constructor. Created object contains no buffers.

#### 5.116.2.2 virtual Arc::PayloadRaw::~~PayloadRaw (void) [virtual]

Destructor. Frees allocated buffers.

### 5.116.3 Member Function Documentation

#### 5.116.3.1 virtual char\* Arc::PayloadRaw::Buffer (unsigned int *num* = 0) [virtual]

Returns pointer to num'th buffer

Implements [Arc::PayloadRawInterface](#).

**5.116.3.2 virtual Size\_t Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const** [virtual]

Returns position of num'th buffer

Implements [Arc::PayloadRawInterface](#).

**5.116.3.3 virtual Size\_t Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const** [virtual]

Returns length of num'th buffer

Implements [Arc::PayloadRawInterface](#).

**5.116.3.4 virtual Size\_t Arc::PayloadRaw::Size (void) const** [virtual]

Returns logical size of whole structure.

Implements [Arc::PayloadRawInterface](#).

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

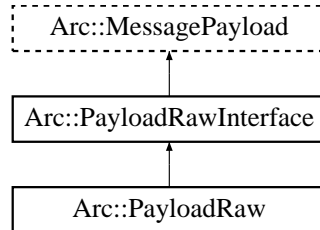
- PayloadRaw.h

## 5.117 Arc::PayloadRawInterface Class Reference

Random Access Payload for [Message](#) objects.

```
#include <PayloadRaw.h>
```

Inheritance diagram for Arc::PayloadRawInterface::



### Public Member Functions

- virtual char [operator\[ \]](#) (Size\_t pos) const =0
- virtual char \* [Content](#) (Size\_t pos=-1)=0
- virtual Size\_t [Size](#) (void) const =0
- virtual char \* [Insert](#) (Size\_t pos=0, Size\_t size=0)=0
- virtual char \* [Insert](#) (const char \*s, Size\_t pos=0, Size\_t size=-1)=0
- virtual char \* [Buffer](#) (unsigned int num)=0
- virtual Size\_t [BufferSize](#) (unsigned int num) const =0
- virtual Size\_t [BufferPos](#) (unsigned int num) const =0
- virtual bool [Truncate](#) (Size\_t size)=0

#### 5.117.1 Detailed Description

Random Access Payload for [Message](#) objects.

This class is a virtual interface for managing [Message](#) payload with arbitrarily accessible content. Inheriting classes are supposed to implement memory-resident or memory-mapped content made of optionally multiple chunks/buffers. Every buffer has own size and offset. This class is purely virtual.

#### 5.117.2 Member Function Documentation

**5.117.2.1** `virtual char* Arc::PayloadRawInterface::Buffer (unsigned int num)` [pure virtual]

Returns pointer to num'th buffer

Implemented in [Arc::PayloadRaw](#).

**5.117.2.2** `virtual Size_t Arc::PayloadRawInterface::BufferPos (unsigned int num) const` [pure virtual]

Returns position of num'th buffer

Implemented in [Arc::PayloadRaw](#).

**5.117.2.3** `virtual Size_t Arc::PayloadRawInterface::BufferSize (unsigned int num) const` [pure virtual]

Returns length of *num*'th buffer

Implemented in [Arc::PayloadRaw](#).

**5.117.2.4** `virtual char* Arc::PayloadRawInterface::Content (Size_t pos = -1)` [pure virtual]

Get pointer to buffer content at global position '*pos*'. By default to beginning of main buffer whatever that means.

**5.117.2.5** `virtual char* Arc::PayloadRawInterface::Insert (const char * s, Size_t pos = 0, Size_t size = -1)` [pure virtual]

Create new buffer at global position '*pos*' of size '*size*'. Created buffer is filled with content of memory at '*s*'. If '*size*' is negative content at '*s*' is expected to be null-terminated.

**5.117.2.6** `virtual char* Arc::PayloadRawInterface::Insert (Size_t pos = 0, Size_t size = 0)` [pure virtual]

Create new buffer at global position '*pos*' of size '*size*'.

**5.117.2.7** ]

`virtual char Arc::PayloadRawInterface::operator[] (Size_t pos) const` [pure virtual]

Returns content of byte at specified position. Specified position '*pos*' is treated as global one and goes through all buffers placed one after another.

**5.117.2.8** `virtual Size_t Arc::PayloadRawInterface::Size (void) const` [pure virtual]

Returns logical size of whole structure.

Implemented in [Arc::PayloadRaw](#).

**5.117.2.9** `virtual bool Arc::PayloadRawInterface::Truncate (Size_t size)` [pure virtual]

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

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

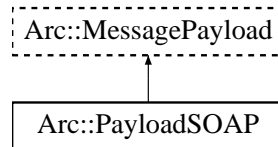
- PayloadRaw.h

## 5.118 Arc::PayloadSOAP Class Reference

Payload of [Message](#) with SOAP content.

```
#include <PayloadSOAP.h>
```

Inheritance diagram for Arc::PayloadSOAP::



### Public Member Functions

- [PayloadSOAP](#) (const NS &ns, bool fault=false)
- [PayloadSOAP](#) (const SOAPEnvelope &soap)
- [PayloadSOAP](#) (const [MessagePayload](#) &source)

### 5.118.1 Detailed Description

Payload of [Message](#) with SOAP content.

This class combines [MessagePayload](#) with SOAPEnvelope to make it possible to pass SOAP messages through [MCC](#) chain.

### 5.118.2 Constructor & Destructor Documentation

#### 5.118.2.1 Arc::PayloadSOAP::PayloadSOAP (const NS & ns, bool *fault* = false)

Constructor - creates new [Message](#) payload

#### 5.118.2.2 Arc::PayloadSOAP::PayloadSOAP (const SOAPEnvelope & soap)

Constructor - creates [Message](#) payload from SOAP document. Provided SOAP document is copied to new object.

#### 5.118.2.3 Arc::PayloadSOAP::PayloadSOAP (const [MessagePayload](#) & source)

Constructor - creates SOAP message from payload. [PayloadRawInterface](#) and derived classes are supported.

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

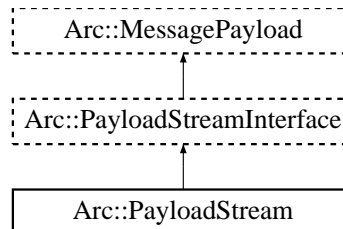
- PayloadSOAP.h

## 5.119 Arc::PayloadStream Class Reference

POSIX handle as Payload.

```
#include <PayloadStream.h>
```

Inheritance diagram for Arc::PayloadStream::



### Public Member Functions

- [PayloadStream](#) (int h=-1)
- virtual [~PayloadStream](#) (void)
- virtual bool [Get](#) (char \*buf, int &size)
- virtual bool [Get](#) (std::string &buf)
- virtual std::string [Get](#) (void)
- virtual bool [Put](#) (const std::string &buf)
- virtual bool [Put](#) (const char \*buf)
- virtual [operator bool](#) (void)
- virtual bool [operator!](#) (void)
- virtual int [Timeout](#) (void) const
- virtual void [Timeout](#) (int to)
- virtual Size\_t [Pos](#) (void) const
- virtual Size\_t [Size](#) (void) const
- virtual Size\_t [Limit](#) (void) const

### Protected Attributes

- int [handle\\_](#)
- bool [seekable\\_](#)

#### 5.119.1 Detailed Description

POSIX handle as Payload.

This is an implemetation of [PayloadStreamInterface](#) for generic POSIX handle.

#### 5.119.2 Constructor & Destructor Documentation

##### 5.119.2.1 Arc::PayloadStream::PayloadStream (int h = -1)

Constructor. Attaches to already open handle. Handle is not managed by this class and must be closed by external code.

**5.119.2.2 virtual Arc::PayloadStream::~~PayloadStream (void) [inline, virtual]**

Destructor.

**5.119.3 Member Function Documentation****5.119.3.1 virtual std::string Arc::PayloadStream::Get (void) [inline, virtual]**

Read as many as possible (sane amount) of bytes.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.2 virtual bool Arc::PayloadStream::Get (std::string & buf) [virtual]**

Read as many as possible (sane amount) of bytes into buf.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.3 virtual bool Arc::PayloadStream::Get (char \* buf, int & size) [virtual]**

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.4 virtual Size\_t Arc::PayloadStream::Limit (void) const [inline, virtual]**

Returns position at which stream reading will stop if supported. That may be not same as [Size\(\)](#) if instance is meant to provide access to only part of underlying object.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.5 virtual Arc::PayloadStream::operator bool (void) [inline, virtual]**

Returns true if stream is valid.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.6 virtual bool Arc::PayloadStream::operator! (void) [inline, virtual]**

Returns true if stream is invalid.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.7 virtual Size\_t Arc::PayloadStream::Pos (void) const [inline, virtual]**

Returns current position in stream if supported.

Implements [Arc::PayloadStreamInterface](#).



**5.119.3.8 virtual bool Arc::PayloadStream::Put (const char \* *buf*)** [inline, virtual]

Push null terminated information from 'buf' into stream. Returns true on success.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.9 virtual bool Arc::PayloadStream::Put (const std::string & *buf*)** [inline, virtual]

Push information from 'buf' into stream. Returns true on success.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.10 virtual Size\_t Arc::PayloadStream::Size (void) const** [inline, virtual]

Returns size of underlying object if supported.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.11 virtual void Arc::PayloadStream::Timeout (int *to*)** [inline, virtual]

Set current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implements [Arc::PayloadStreamInterface](#).

**5.119.3.12 virtual int Arc::PayloadStream::Timeout (void) const** [inline, virtual]

Query current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implements [Arc::PayloadStreamInterface](#).

**5.119.4 Field Documentation****5.119.4.1 int [Arc::PayloadStream::handle\\_](#)** [protected]

Timeout for read/write operations

**5.119.4.2 bool [Arc::PayloadStream::seekable\\_](#)** [protected]

Handle for operations

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

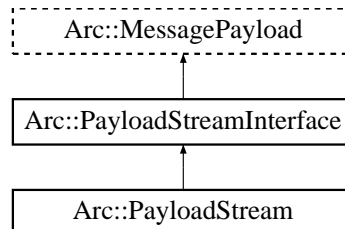
- [PayloadStream.h](#)

## 5.120 Arc::PayloadStreamInterface Class Reference

Stream-like Payload for [Message](#) object.

```
#include <PayloadStream.h>
```

Inheritance diagram for Arc::PayloadStreamInterface::



### Public Member Functions

- virtual bool [Get](#) (char \*buf, int &size)=0
- virtual bool [Get](#) (std::string &buf)=0
- virtual std::string [Get](#) (void)=0
- virtual bool [Put](#) (const char \*buf, Size\_t size)=0
- virtual bool [Put](#) (const std::string &buf)=0
- virtual bool [Put](#) (const char \*buf)=0
- virtual [operator bool](#) (void)=0
- virtual bool [operator!](#) (void)=0
- virtual int [Timeout](#) (void) const =0
- virtual void [Timeout](#) (int to)=0
- virtual Size\_t [Pos](#) (void) const =0
- virtual Size\_t [Size](#) (void) const =0
- virtual Size\_t [Limit](#) (void) const =0

### 5.120.1 Detailed Description

Stream-like Payload for [Message](#) object.

This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through [MCC](#) chain as payload of [Message](#). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

### 5.120.2 Member Function Documentation

#### 5.120.2.1 virtual std::string Arc::PayloadStreamInterface::Get (void) [pure virtual]

Read as many as possible (sane amount) of bytes.

Implemented in [Arc::PayloadStream](#).

**5.120.2.2 virtual bool Arc::PayloadStreamInterface::Get (std::string & buf) [pure virtual]**

Read as many as possible (sane amount) of bytes into buf.

Implemented in [Arc::PayloadStream](#).

**5.120.2.3 virtual bool Arc::PayloadStreamInterface::Get (char \* buf, int & size) [pure virtual]**

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in [Arc::PayloadStream](#).

**5.120.2.4 virtual Size\_t Arc::PayloadStreamInterface::Limit (void) const [pure virtual]**

Returns position at which stream reading will stop if supported. That may be not same as [Size\(\)](#) if instance is meant to provide access to only part of underlying object.

Implemented in [Arc::PayloadStream](#).

**5.120.2.5 virtual Arc::PayloadStreamInterface::operator bool (void) [pure virtual]**

Returns true if stream is valid.

Implemented in [Arc::PayloadStream](#).

**5.120.2.6 virtual bool Arc::PayloadStreamInterface::operator! (void) [pure virtual]**

Returns true if stream is invalid.

Implemented in [Arc::PayloadStream](#).

**5.120.2.7 virtual Size\_t Arc::PayloadStreamInterface::Pos (void) const [pure virtual]**

Returns current position in stream if supported.

Implemented in [Arc::PayloadStream](#).

**5.120.2.8 virtual bool Arc::PayloadStreamInterface::Put (const char \* buf) [pure virtual]**

Push null terminated information from 'buf' into stream. Returns true on success.

Implemented in [Arc::PayloadStream](#).

**5.120.2.9 virtual bool Arc::PayloadStreamInterface::Put (const std::string & buf) [pure virtual]**

Push information from 'buf' into stream. Returns true on success.

Implemented in [Arc::PayloadStream](#).

**5.120.2.10** `virtual bool Arc::PayloadStreamInterface::Put (const char * buf, Size_t size)` [pure virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

**5.120.2.11** `virtual Size_t Arc::PayloadStreamInterface::Size (void) const` [pure virtual]

Returns size of underlying object if supported.

Implemented in [Arc::PayloadStream](#).

**5.120.2.12** `virtual void Arc::PayloadStreamInterface::Timeout (int to)` [pure virtual]

Set current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implemented in [Arc::PayloadStream](#).

**5.120.2.13** `virtual int Arc::PayloadStreamInterface::Timeout (void) const` [pure virtual]

Query current timeout for [Get\(\)](#) and [Put\(\)](#) operations.

Implemented in [Arc::PayloadStream](#).

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

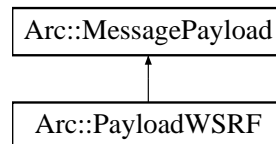
- [PayloadStream.h](#)

## 5.121 Arc::PayloadWSRF Class Reference

This class combines [MessagePayload](#) with [WSRF](#).

```
#include <PayloadWSRF.h>
```

Inheritance diagram for Arc::PayloadWSRF::



### Public Member Functions

- [PayloadWSRF](#) (const SOAPEnvelope &soap)
- [PayloadWSRF](#) ([WSRF](#) &wsrp)
- [PayloadWSRF](#) (const [MessagePayload](#) &source)

#### 5.121.1 Detailed Description

This class combines [MessagePayload](#) with [WSRF](#).

It's intention is to make it possible to pass [WSRF](#) messages through [MCC](#) chain as one more Payload type.

#### 5.121.2 Constructor & Destructor Documentation

##### 5.121.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & soap)

Constructor - creates [Message](#) payload from SOAP message. Returns invalid [WSRF](#) if SOAP does not represent WS-ResourceProperties

##### 5.121.2.2 Arc::PayloadWSRF::PayloadWSRF ([WSRF](#) &wsrp)

Constructor - creates [Message](#) payload with acquired [WSRF](#) message. [WSRF](#) message will be destroyed by destructor of this object.

##### 5.121.2.3 Arc::PayloadWSRF::PayloadWSRF (const [MessagePayload](#) & source)

Constructor - creates [WSRF](#) message from payload. All classes derived from SOAPEnvelope are supported.

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

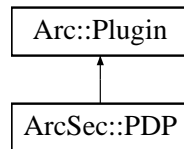
- PayloadWSRF.h

## 5.122 ArcSec::PDP Class Reference

Base class for [Policy](#) Decision Point plugins.

```
#include <PDP.h>
```

Inheritance diagram for ArcSec::PDP::



### 5.122.1 Detailed Description

Base class for [Policy](#) Decision Point plugins.

This virtual class defines method `isPermitted()` which processes security related information/attributes in `Message` and makes security decision - permit (true) or deny (false). Configuration of [PDP](#) is consumed during creation of instance through XML subtree fed to constructor.

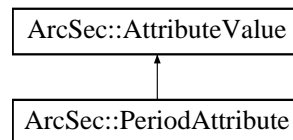
The documentation for this class was generated from the following file:

- `PDP.h`

## 5.123 ArcSec::PeriodAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::PeriodAttribute::



### Public Member Functions

- virtual bool [equal](#) ([AttributeValue](#) \*other, bool check\_id=true)
- virtual std::string [encode](#) ()
- virtual std::string [getType](#) ()
- virtual std::string [getId](#) ()

#### 5.123.1 Detailed Description

Formate: datetime"/"duration datetime"/"datetime duration"/"datetime

#### 5.123.2 Member Function Documentation

##### 5.123.2.1 virtual std::string ArcSec::PeriodAttribute::encode () [virtual]

encode the value in a string format

Implements [ArcSec::AttributeValue](#).

##### 5.123.2.2 virtual bool ArcSec::PeriodAttribute::equal ([AttributeValue](#) \* other, bool check\_id = true) [virtual]

Evaluate whether "this" equale to the parameter value

Implements [ArcSec::AttributeValue](#).

##### 5.123.2.3 virtual std::string ArcSec::PeriodAttribute::getId () [inline, virtual]

Get the AttributeId of the <Attribute>

Implements [ArcSec::AttributeValue](#).

##### 5.123.2.4 virtual std::string ArcSec::PeriodAttribute::getType () [inline, virtual]

Get the DataType of the <Attribute>

Implements [ArcSec::AttributeValue](#).

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

- DateTimeAttribute.h

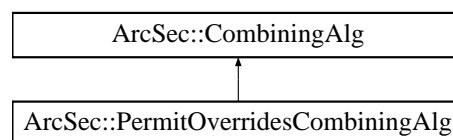


## 5.124 ArcSec::PermitOverridesCombiningAlg Class Reference

Implement the "Permit-Overrides" algorithm.

```
#include <PermitOverridesAlg.h>
```

Inheritance diagram for ArcSec::PermitOverridesCombiningAlg::



### Public Member Functions

- virtual Result [combine](#) (EvaluationCtx \*ctx, std::list< [Policy](#) \* > policies)
- virtual const std::string & [getalgId](#) (void) const

#### 5.124.1 Detailed Description

Implement the "Permit-Overrides" algorithm.

Permit-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "permit" result from any policy, then stops scanning and gives "permit" as result, otherwise gives "deny".

#### 5.124.2 Member Function Documentation

**5.124.2.1** virtual Result ArcSec::PermitOverridesCombiningAlg::combine ([EvaluationCtx](#) \* ctx, std::list< [Policy](#) \* > *policies*) [virtual]

If there is one policy which return positive evaluation result, then omit the other policies and return DECISION\_PERMIT

##### Parameters:

- ctx* This object contains request information which will be used to evaluated against policy.
- policies* This is a container which contains policy objects.

##### Returns:

The combined result according to the algorithm.

Implements [ArcSec::CombiningAlg](#).

**5.124.2.2** virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId (void) const [inline, virtual]

Get the identifier

Implements [ArcSec::CombiningAlg](#).

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

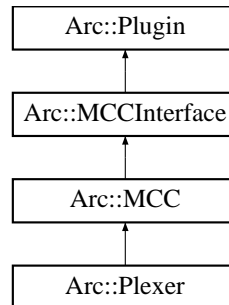
- PermitOverridesAlg.h

## 5.125 Arc::Plexer Class Reference

The [Plexer](#) class, used for routing messages to services.

```
#include <Plexer.h>
```

Inheritance diagram for Arc::Plexer::



### Public Member Functions

- [Plexer](#) ([Config](#) \*cfg, [PluginArgument](#) \*arg)
- virtual [~Plexer](#) ()
- virtual void [Next](#) ([MCCInterface](#) \*next, const std::string &label)
- virtual [MCC\\_Status process](#) ([Message](#) &request, [Message](#) &response)

### Static Public Attributes

- static [Logger](#) logger

#### 5.125.1 Detailed Description

The [Plexer](#) class, used for routing messages to services.

This is the [Plexer](#) class. Its purpose is to route incoming messages to appropriate Services and [MCC](#) chains.

#### 5.125.2 Constructor & Destructor Documentation

##### 5.125.2.1 Arc::Plexer::Plexer ([Config](#) \* cfg, [PluginArgument](#) \* arg)

The constructor.

This is the constructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

##### 5.125.2.2 virtual Arc::Plexer::~~Plexer () [virtual]

The destructor.

This is the destructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

### 5.125.3 Member Function Documentation

#### 5.125.3.1 `virtual void Arc::Plexer::Next (MCCInterface * next, const std::string & label)` [virtual]

Add reference to next [MCC](#) in chain.

This method is called by [Loader](#) for every potentially labeled link to next component which implements [MCCInterface](#). If next is set NULL corresponding link is removed.

Reimplemented from [Arc::MCC](#).

#### 5.125.3.2 `virtual MCC\_Status Arc::Plexer::process (Message & request, Message & response)` [virtual]

Route request messages to appropriate services.

Routes the request message to the appropriate service. Routing is based on the path part of value of the ENDPOINT attribute. Routed message is assigned following attributes: PLEXER:PATTERN - matched pattern, PLEXER:EXTENSION - last unmatched part of ENDPOINT path.

Reimplemented from [Arc::MCC](#).

### 5.125.4 Field Documentation

#### 5.125.4.1 `Logger Arc::Plexer::logger` [static]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented from [Arc::MCC](#).

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

- [Plexer.h](#)

## 5.126 Arc::PlexerEntry Class Reference

A pair of label (regex) and pointer to [MCC](#).

```
#include <Plexer.h>
```

### 5.126.1 Detailed Description

A pair of label (regex) and pointer to [MCC](#).

A helper class that stores a label (regex) and a pointer to a service.

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

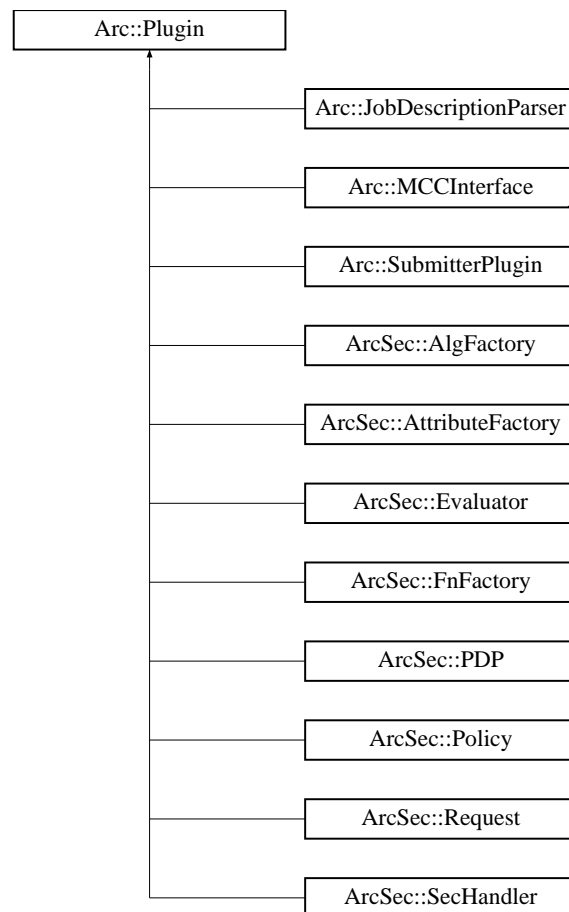
- Plexer.h

## 5.127 Arc::Plugin Class Reference

Base class for loadable ARC components.

```
#include <Plugin.h>
```

Inheritance diagram for Arc::Plugin::



### Protected Member Functions

- [Plugin](#) ([PluginArgument](#) \*arg)
- [Plugin](#) (const [Plugin](#) &obj)

#### 5.127.1 Detailed Description

Base class for loadable ARC components.

All classes representing loadable ARC components must be either descendants of this class or be wrapped by its offspring.

## 5.127.2 Constructor & Destructor Documentation

### 5.127.2.1 Arc::Plugin::Plugin (PluginArgument \* *arg*) [protected]

Main constructor for creating new plugin object.

### 5.127.2.2 Arc::Plugin::Plugin (const Plugin & *obj*) [protected]

Constructor to be used if plugin want to copy itself.

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

- Plugin.h

## 5.128 Arc::PluginArgument Class Reference

Base class for passing arguments to loadable ARC components.

```
#include <Plugin.h>
```

### Public Member Functions

- [PluginsFactory](#) \* [get\\_factory](#) (void)
- [Glib::Module](#) \* [get\\_module](#) (void)

### 5.128.1 Detailed Description

Base class for passing arguments to loadable ARC components.

During its creation constructor function of ARC loadable component expects instance of class inherited from this one or wrapped in it. Then dynamic type casting is used for obtaining class of expected kind.

### 5.128.2 Member Function Documentation

#### 5.128.2.1 [PluginsFactory](#)\* Arc::PluginArgument::get\_factory (void)

Returns pointer to factory which instantiated plugin.

Because factory usually destroys/unloads plugins in its destructor it should be safe to keep this pointer inside plugin for later use. But one must always check.

#### 5.128.2.2 [Glib::Module](#)\* Arc::PluginArgument::get\_module (void)

Returns pointer to loadable module/library which contains plugin.

Corresponding factory keeps list of modules till itself is destroyed. So it should be safe to keep that pointer. But care must be taken if module contains persistent plugins. Such modules stay in memory after factory is destroyed. So it is advisable to use obtained pointer only in constructor function of plugin.

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

- [Plugin.h](#)



## 5.129 Arc::PluginDesc Class Reference

Description of plugin.

```
#include <Plugin.h>
```

### 5.129.1 Detailed Description

Description of plugin.

This class is used for reports

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

- Plugin.h

## 5.130 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

```
#include <Plugin.h>
```

### 5.130.1 Detailed Description

Description of ARC lodable component.

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

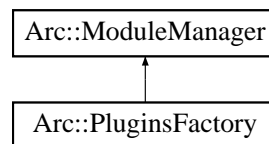
- Plugin.h

## 5.131 Arc::PluginsFactory Class Reference

Generic ARC plugins loader.

```
#include <Plugin.h>
```

Inheritance diagram for Arc::PluginsFactory::



### Public Member Functions

- [PluginsFactory](#) ([XMLNode](#) cfg)
- void [TryLoad](#) (bool v)
- bool [load](#) (const std::string &name)
- bool [scan](#) (const std::string &name, [ModuleDesc](#) &desc)
- void [report](#) (std::list< [ModuleDesc](#) > &descs)

### Static Public Member Functions

- static void [FilterByKind](#) (const std::string &kind, std::list< [ModuleDesc](#) > &descs)

#### 5.131.1 Detailed Description

Generic ARC plugins loader.

The instance of this class provides functionality of loading pluggable ARC components stored in shared libraries. For more information please check HED documentation. This class is thread-safe - its methods are protected from simultaneous use from multiple threads. Current thread protection implementation is suboptimal and will be revised in future.

#### 5.131.2 Constructor & Destructor Documentation

##### 5.131.2.1 Arc::PluginsFactory::PluginsFactory ([XMLNode](#) cfg)

Constructor - accepts configuration (not yet used) meant to tune loading of modules.

#### 5.131.3 Member Function Documentation

##### 5.131.3.1 static void Arc::PluginsFactory::FilterByKind (const std::string & *kind*, std::list< [ModuleDesc](#) > & *descs*) [static]

Filter list of modules by kind.

**5.131.3.2 bool Arc::PluginsFactory::load (const std::string & name)**

These methods load module named lib'name' and check if it contains ARC plugin(s) of specified 'kind' and 'name'. If there are no specified plugins or module does not contain any ARC plugins it is unloaded. All loaded plugins are also registered in internal list of this instance of [PluginsFactory](#) class. Returns true if any plugin was loaded.

**5.131.3.3 void Arc::PluginsFactory::report (std::list< [ModuleDesc](#) > & desc)**

Provides information about currently loaded modules and plugins.

**5.131.3.4 bool Arc::PluginsFactory::scan (const std::string & name, [ModuleDesc](#) & desc)**

Collect information about plugins stored in module(s) with specified names. Returns true if any of specified modules has plugins.

**5.131.3.5 void Arc::PluginsFactory::TryLoad (bool v) [inline]**

Specifies if loadable module may be loaded while looking for analyzing its content. If set to false only \*.apd files are checked. Modules without corresponding \*.apd will be ignored. Default is true;

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

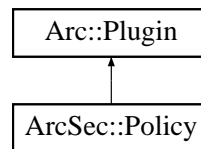
- Plugin.h

## 5.132 ArcSec::Policy Class Reference

Interface for containing and processing different types of policy.

```
#include <Policy.h>
```

Inheritance diagram for ArcSec::Policy::



### Public Member Functions

- [Policy](#) ([Arc::PluginArgument](#) \*parg)
- [Policy](#) (const [Arc::XMLNode](#), [Arc::PluginArgument](#) \*parg)
- [Policy](#) (const [Arc::XMLNode](#), [EvaluatorContext](#) \*, [Arc::PluginArgument](#) \*parg)
- virtual [operator bool](#) (void) const =0
- virtual [MatchResult](#) [match](#) ([EvaluationCtx](#) \*) =0
- virtual [Result](#) [eval](#) ([EvaluationCtx](#) \*) =0
- virtual void [addPolicy](#) ([Policy](#) \*pl)
- virtual void [setEvaluatorContext](#) ([EvaluatorContext](#) \*)
- virtual void [make\\_policy](#) ()
- virtual std::string [getEffect](#) () const =0
- virtual [EvalResult](#) & [getEvalResult](#) () =0
- virtual void [setEvalResult](#) ([EvalResult](#) &res) =0
- virtual const char \* [getEvalName](#) () const =0
- virtual const char \* [getName](#) () const =0

### 5.132.1 Detailed Description

Interface for containing and processing different types of policy.

Basically, each policy object is a container which includes a few elements e.g., [ArcPolicySet](#) objects includes a few [ArcPolicy](#) objects; [ArcPolicy](#) object includes a few [ArcRule](#) objects. There is logical relationship between [ArcRules](#) or [ArcPolicies](#), which is called combining algorithm. According to algorithm, evaluation results from the elements are combined, and then the combined evaluation result is returned to the up-level.

### 5.132.2 Constructor & Destructor Documentation

#### 5.132.2.1 ArcSec::Policy::Policy ([Arc::PluginArgument](#) \*parg) [inline]

Template constructor - creates empty policy.

### 5.132.2.2 `ArcSec::Policy::Policy (const Arc::XMLNode, Arc::PluginArgument * parg)` `[inline]`

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid.

### 5.132.2.3 `ArcSec::Policy::Policy (const Arc::XMLNode, EvaluatorContext *, Arc::PluginArgument * parg)` `[inline]`

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid. This constructor is based on the policy node and i the [EvaluatorContext](#) which includes the factory objects for combining algorithm and function

## 5.132.3 Member Function Documentation

### 5.132.3.1 `virtual void ArcSec::Policy::addPolicy (Policy * pl)` `[inline, virtual]`

Add a policy element to into "this" object

### 5.132.3.2 `virtual Result ArcSec::Policy::eval (EvaluationCtx *)` `[pure virtual]`

Evaluate policy For the <Rule> of [Arc](#), only get the "Effect" from rules; For the <Policy> of [Arc](#), combine the evaluation result from <Rule>; For the <Rule> of XACML, evaluate the <Condition> node by using information from request, and use the "Effect" attribute of <Rule>; For the <Policy> of XACML, combine the evaluation result from <Rule>

### 5.132.3.3 `virtual std::string ArcSec::Policy::getEffect () const` `[pure virtual]`

Get the "Effect" attribute

### 5.132.3.4 `virtual const char* ArcSec::Policy::getEvalName () const` `[pure virtual]`

Get the name of [Evaluator](#) which can evaluate this policy

### 5.132.3.5 `virtual EvalResult& ArcSec::Policy::getEvalResult ()` `[pure virtual]`

Get evaluation result

### 5.132.3.6 `virtual const char* ArcSec::Policy::getName () const` `[pure virtual]`

Get the name of this policy

### 5.132.3.7 `virtual void ArcSec::Policy::make_policy ()` `[inline, virtual]`

Parse XMLNode, and construct the low-level Rule object

**5.132.3.8** `virtual MatchResult ArcSec::Policy::match (EvaluationCtx *)` [pure virtual]

Evaluate whether the two targets to be evaluated match to each other.

**5.132.3.9** `virtual ArcSec::Policy::operator bool (void) const` [pure virtual]

Returns true is object is valid.

**5.132.3.10** `virtual void ArcSec::Policy::setEvalResult (EvalResult & res)` [pure virtual]

Set eveluation result

**5.132.3.11** `virtual void ArcSec::Policy::setEvaluatorContext (EvaluatorContext *)` [inline, virtual]

Set [Evaluator](#) Context for the usage in creating low-level policy object

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

- Policy.h

## 5.133 ArcSec::PolicyParser Class Reference

A interface which will isolate the policy object from actual policy storage (files, urls, database).

```
#include <PolicyParser.h>
```

### Public Member Functions

- virtual [Policy](#) \* [parsePolicy](#) (const [Source](#) &source, std::string policyclassname, [EvaluatorContext](#) \*ctx)

#### 5.133.1 Detailed Description

A interface which will isolate the policy object from actual policy storage (files, urls, database).

Parse the policy from policy source (e.g. files, urls, database, etc.).

#### 5.133.2 Member Function Documentation

**5.133.2.1** virtual [Policy](#)\* ArcSec::PolicyParser::parsePolicy (const [Source](#) & *source*, std::string *policyclassname*, [EvaluatorContext](#) \* *ctx*) [virtual]

Parse policy

##### Parameters:

- source* location of the policy
- policyclassname* name of the policy for ClassLoader
- ctx* [EvaluatorContext](#) which includes the \*\*Factory

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

- PolicyParser.h



## 5.134 ArcSec::PolicyStore Class Reference

Storage place for policy objects.

```
#include <PolicyStore.h>
```

### Public Member Functions

- [PolicyStore](#) (const std::string &alg, const std::string &policyclassname, [EvaluatorContext](#) \*ctx)

### Data Structures

- class [PolicyElement](#)

#### 5.134.1 Detailed Description

Storage place for policy objects.

#### 5.134.2 Constructor & Destructor Documentation

##### 5.134.2.1 ArcSec::PolicyStore::PolicyStore (const std::string & *alg*, const std::string & *policyclassname*, [EvaluatorContext](#) \* *ctx*)

Creates policy store with specified combing algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

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

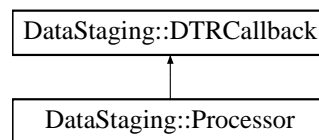
- PolicyStore.h

## 5.135 DataStaging::Processor Class Reference

The [Processor](#) performs pre- and post-transfer operations.

```
#include <Processor.h>
```

Inheritance diagram for DataStaging::Processor::



### Public Member Functions

- [Processor](#) ()
- [~Processor](#) ()
- void [start](#) (void)
- void [stop](#) (void)
- virtual void [receiveDTR](#) ([DTR\\_ptr](#) dtr)

### Data Structures

- class **BulkThreadArgument**  
*Class used to pass information to spawned thread (for bulk operations).*
- class **ThreadArgument**  
*Class used to pass information to spawned thread.*

#### 5.135.1 Detailed Description

The [Processor](#) performs pre- and post-transfer operations.

The [Processor](#) takes care of everything that should happen before and after a transfer takes place. Calling [receiveDTR\(\)](#) spawns a thread to perform the required operation depending on the [DTR](#) state.

#### 5.135.2 Constructor & Destructor Documentation

##### 5.135.2.1 DataStaging::Processor::Processor () [inline]

Constructor.

##### 5.135.2.2 DataStaging::Processor::~~Processor () [inline]

Destructor waits for all active threads to stop.

### 5.135.3 Member Function Documentation

#### 5.135.3.1 virtual void DataStaging::Processor::receiveDTR ([DTR\\_ptr](#) *dtr*) [virtual]

Send a [DTR](#) to the [Processor](#).

The [DTR](#) is sent to the [Processor](#) through this method when some long-latency processing is to be performed, eg contacting a remote service. The [Processor](#) spawns a thread to do the processing, and then returns. The thread notifies the scheduler when it is finished.

Implements [DataStaging::DTRCallback](#).

#### 5.135.3.2 void DataStaging::Processor::start (void)

Start [Processor](#).

This method actually does nothing. It is here only to make all classes of data staging to look alike. But it is better to call it before starting to use object because it may do something in the future.

#### 5.135.3.3 void DataStaging::Processor::stop (void)

Stop [Processor](#).

This method sends waits for all started threads to end and exits. Since threads a short-lived it is better to wait rather than interrupt them.

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

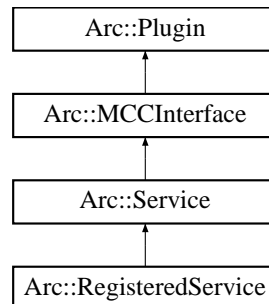
- [Processor.h](#)

## 5.136 Arc::RegisteredService Class Reference

[RegisteredService](#) - extension of [Service](#) performing self-registration.

```
#include <RegisteredService.h>
```

Inheritance diagram for Arc::RegisteredService::



### Public Member Functions

- [RegisteredService](#) ([Config](#) \*, [PluginArgument](#) \*)

#### 5.136.1 Detailed Description

[RegisteredService](#) - extension of [Service](#) performing self-registration.

#### 5.136.2 Constructor & Destructor Documentation

##### 5.136.2.1 Arc::RegisteredService::RegisteredService ([Config](#) \*, [PluginArgument](#) \*)

Example contructor - Server takes at least it's configuration subtree

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

- `RegisteredService.h`

## 5.137 Arc::RegularExpression Class Reference

A regular expression class.

```
#include <ArcRegex.h>
```

### Public Member Functions

- [RegularExpression](#) ()
- [RegularExpression](#) (std::string pattern)
- [RegularExpression](#) (const [RegularExpression](#) &regex)
- [~RegularExpression](#) ()
- [RegularExpression](#) & [operator=](#) (const [RegularExpression](#) &regex)
- bool [isOk](#) ()
- bool [hasPattern](#) (std::string str)
- bool [match](#) (const std::string &str) const
- bool [match](#) (const std::string &str, std::list< std::string > &unmatched, std::list< std::string > &matched) const
- std::string [getPattern](#) () const

### 5.137.1 Detailed Description

A regular expression class.

This class is a wrapper around the functions provided in regex.h.

### 5.137.2 Constructor & Destructor Documentation

#### 5.137.2.1 Arc::RegularExpression::RegularExpression () [inline]

default constructor

#### 5.137.2.2 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a regex from a pattern string.

#### 5.137.2.3 Arc::RegularExpression::RegularExpression (const [RegularExpression](#) & *regex*)

Copy constructor.

#### 5.137.2.4 Arc::RegularExpression::~~RegularExpression ()

Destructor.

### 5.137.3 Member Function Documentation

#### 5.137.3.1 std::string Arc::RegularExpression::getPattern () const

Returns pattern.

**5.137.3.2 bool Arc::RegularExpression::hasPattern (std::string *str*)**

Returns true if this regex has the pattern provided.

**5.137.3.3 bool Arc::RegularExpression::isOk ()**

Returns true if the pattern of this regex is ok.

**5.137.3.4 bool Arc::RegularExpression::match (const std::string & *str*, std::list< std::string > & *unmatched*, std::list< std::string > & *matched*) const**

Returns true if this regex matches the string provided.

Unmatched parts of the string are stored in 'unmatched'. Matched parts of the string are stored in 'matched'. The first entry in matched is the string that matched the regex, and the following entries are parenthesised elements of the regex

**5.137.3.5 bool Arc::RegularExpression::match (const std::string & *str*) const**

Returns true if this regex matches whole string provided.

**5.137.3.6 [RegularExpression](#)& Arc::RegularExpression::operator= (const [RegularExpression](#) & *regex*)**

Assignment operator.

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

- ArcRegex.h

## 5.138 Arc::RemoteLoggingType Class Reference

Remote logging.

```
#include <JobDescription.h>
```

### Data Fields

- std::string [ServiceType](#)
- [URL Location](#)
- bool [optional](#)

### 5.138.1 Detailed Description

Remote logging.

This class is used to specify a service which should be used to report logging information to, such as job resource usage.

### 5.138.2 Field Documentation

#### 5.138.2.1 [URL Arc::RemoteLoggingType::Location](#)

[URL](#) of logging service.

The Location [URL](#) specifies the [URL](#) of the service which job logging information should be sent to.

#### 5.138.2.2 [bool Arc::RemoteLoggingType::optional](#)

Requirement satisfaction switch.

The optional boolean specifies whether the requirement specified in the particular object is mandatory for job execution, or whether it be ignored.

#### 5.138.2.3 [std::string Arc::RemoteLoggingType::ServiceType](#)

Type of logging service.

The ServiceType string specifies the type of logging service. Some examples are "SGAS" (<http://www.sgas.se>) and "APEL" (<https://wiki.ege.eu/wiki/APEL>), however please refer to the particular execution service for a list of supported logging service types.

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

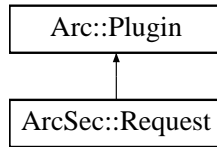
- JobDescription.h

## 5.139 ArcSec::Request Class Reference

Base class/Interface for request, includes a container for RequestItems and some operations.

```
#include <Request.h>
```

Inheritance diagram for ArcSec::Request::



### Public Member Functions

- virtual ReqItemList [getRequestItems](#) () const
- virtual void [setRequestItems](#) (ReqItemList)
- virtual void [addRequestItem](#) (Attrs &, Attrs &, Attrs &, Attrs &)
- virtual void [setAttributeFactory](#) (AttributeFactory \*attributefactory)=0
- virtual void [make\\_request](#) ()=0
- virtual const char \* [getEvalName](#) () const =0
- virtual const char \* [getName](#) () const =0
- [Request](#) (Arc::PluginArgument \*parg)
- [Request](#) (const Source &, Arc::PluginArgument \*parg)

### 5.139.1 Detailed Description

Base class/Interface for request, includes a container for RequestItems and some operations.

A [Request](#) object can has a few <subjects, actions, objects> tuples, i.e. [RequestItem](#) The [Request](#) class and any customized class which inherit from it, should be loadable, which means these classes can be dynamically loaded according to the configuration information, see the example configuration below: <Service name="pdp.service" id="pdp\_service"> <pdp:PDPCfg> <.....> <pdp:[Request](#) name="arc.request" /> <.....> </pdp:PDPCfg> </Service>

There can be different types of subclass which inherit [Request](#), such like XACMLRequest, ArcRequest, GACLRequest

### 5.139.2 Constructor & Destructor Documentation

#### 5.139.2.1 ArcSec::Request::Request (Arc::PluginArgument \*parg) [inline]

Default constructor

#### 5.139.2.2 ArcSec::Request::Request (const Source &, Arc::PluginArgument \*parg) [inline]

Constructor: Parse request information from a xml stucture in memory



### 5.139.3 Member Function Documentation

**5.139.3.1** `virtual void ArcSec::Request::addRequestItem (Attrs &, Attrs &, Attrs &, Attrs &)`  
[inline, virtual]

Add request tuple from non-XMLNode

**5.139.3.2** `virtual const char* ArcSec::Request::getEvalName () const` [pure virtual]

Get the name of corresponding evaluator

**5.139.3.3** `virtual const char* ArcSec::Request::getName () const` [pure virtual]

Get the name of this request

**5.139.3.4** `virtual ReqItemList ArcSec::Request::getRequestItems () const` [inline, virtual]

Get all the [RequestItem](#) inside [RequestItem](#) container

**5.139.3.5** `virtual void ArcSec::Request::make_request ()` [pure virtual]

Create the objects included in [Request](#) according to the node attached to the [Request](#) object

**5.139.3.6** `virtual void ArcSec::Request::setAttributeFactory (AttributeFactory * attributefactory)`  
[pure virtual]

Set the attribute factory for the usage of [Request](#)

**5.139.3.7** `virtual void ArcSec::Request::setRequestItems (ReqItemList)` [inline, virtual]

Set the content of the container

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

- [Request.h](#)

## 5.140 ArcSec::RequestAttribute Class Reference

Wrapper which includes [AttributeValue](#) object which is generated according to date type of one specif node in Request.xml.

```
#include <RequestAttribute.h>
```

### Public Member Functions

- [RequestAttribute](#) ([Arc::XMLNode](#) &node, [AttributeFactory](#) \*attrfactory)
- [RequestAttribute](#) & [duplicate](#) ([RequestAttribute](#) &)

### 5.140.1 Detailed Description

Wrapper which includes [AttributeValue](#) object which is generated according to date type of one specif node in Request.xml.

### 5.140.2 Constructor & Destructor Documentation

#### 5.140.2.1 ArcSec::RequestAttribute::RequestAttribute ([Arc::XMLNode](#) & node, [AttributeFactory](#) \* attrfactory)

Constructor - create attribute value object according to the "Type" in the node <Attribute attributeid="urn:arc:subject:voms-attribute" type="string">urn:mace:shibboleth:examples</Attribute>

### 5.140.3 Member Function Documentation

#### 5.140.3.1 [RequestAttribute](#)& ArcSec::RequestAttribute::duplicate ([RequestAttribute](#) &)

Duplicate the parameter into "this"

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

- RequestAttribute.h

## 5.141 ArcSec::RequestItem Class Reference

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

```
#include <RequestItem.h>
```

### Public Member Functions

- [RequestItem](#) ([Arc::XMLNode](#) &, [AttributeFactory](#) \*)

#### 5.141.1 Detailed Description

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

#### 5.141.2 Constructor & Destructor Documentation

##### 5.141.2.1 ArcSec::RequestItem::RequestItem ([Arc::XMLNode](#) &, [AttributeFactory](#) \*) [inline]

Constructor

##### Parameters:

*node* The XMLNode structure of the request item

*attributefactory* The [AttributeFactory](#) which will be used to generate [RequestAttribute](#)

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

- RequestItem.h

## 5.142 ArcSec::Response Class Reference

Container for the evaluation results.

```
#include <Response.h>
```

### 5.142.1 Detailed Description

Container for the evaluation results.

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

- Response.h

## 5.143 ArcSec::ResponseItem Class Reference

Evaluation result concerning one RequestTuple.

```
#include <Response.h>
```

### 5.143.1 Detailed Description

Evaluation result concerning one RequestTuple.

Include the RequestTuple, related XMLNode, the set of policy objects which give positive evaluation result, and the related XMLNode

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

- Response.h

## 5.144 Arc::Run Class Reference

```
#include <Run.h>
```

### Public Member Functions

- [Run](#) (const std::string &cmdline)
- [Run](#) (const std::list< std::string > &argv)
- [~Run](#) (void)
- [operator bool](#) (void)
- [bool operator!](#) (void)
- [bool Start](#) (void)
- [bool Wait](#) (int timeout)
- [bool Wait](#) (void)
- [int Result](#) (void)
- [bool Running](#) (void)
- [Time RunTime](#) (void)
- [Time ExitTime](#) (void)
- [int ReadStdout](#) (int timeout, char \*buf, int size)
- [int ReadStderr](#) (int timeout, char \*buf, int size)
- [int WriteStdin](#) (int timeout, const char \*buf, int size)
- [void AssignStdout](#) (std::string &str)
- [void AssignStderr](#) (std::string &str)
- [void AssignStdin](#) (std::string &str)
- [void KeepStdout](#) (bool keep=true)
- [void KeepStderr](#) (bool keep=true)
- [void KeepStdin](#) (bool keep=true)
- [void CloseStdout](#) (void)
- [void CloseStderr](#) (void)
- [void CloseStdin](#) (void)
- [void AssignWorkingDirectory](#) (std::string &wd)
- [void Kill](#) (int timeout)
- [void Abandon](#) (void)

### Static Public Member Functions

- static void [AfterFork](#) (void)

#### 5.144.1 Detailed Description

This class runs external executable. It is possible to read/write it's standard handles or to redirect then to std::string elements.

#### 5.144.2 Constructor & Destructor Documentation

##### 5.144.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

### 5.144.2.2 Arc::Run::Run (const std::list< std::string > & argv)

Constructor preapres object to run executable and arguments specified in argv

### 5.144.2.3 Arc::Run::~~Run (void)

Destructor kills running executable and releases associated resources

## 5.144.3 Member Function Documentation

### 5.144.3.1 void Arc::Run::Abandon (void)

Detach this object from running process. After calling this method instance is not associated with external process anymore. As result destructor will not kill process.

### 5.144.3.2 static void Arc::Run::AfterFork (void) [static]

Call this method after fork() in child cporocess. It will reinitialize internal structures for new environment. Do not call it in any other case than defined.

### 5.144.3.3 void Arc::Run::AssignStderr (std::string & str)

Associate stderr handle of executable with string. This method must be called before [Start\(\)](#). str object must be valid as long as this object exists.

### 5.144.3.4 void Arc::Run::AssignStdin (std::string & str)

Associate stdin handle of executable with string. This method must be called before [Start\(\)](#). str object must be valid as long as this object exists.

### 5.144.3.5 void Arc::Run::AssignStdout (std::string & str)

Associate stdout handle of executable with string. This method must be called before [Start\(\)](#). str object must be valid as long as this object exists.

### 5.144.3.6 void Arc::Run::AssignWorkingDirectory (std::string & wd) [inline]

Assign working direcotry of the running process

### 5.144.3.7 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

### 5.144.3.8 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

**5.144.3.9 void Arc::Run::CloseStdout (void)**

Closes pipe associated with stdout handle

**5.144.3.10 Time Arc::Run::ExitTime (void) [inline]**

Return when executable finished executing.

**5.144.3.11 void Arc::Run::KeepStderr (bool *keep* = true)**

Keep stderr same as parent's if keep = true

**5.144.3.12 void Arc::Run::KeepStdin (bool *keep* = true)**

Keep stdin same as parent's if keep = true

**5.144.3.13 void Arc::Run::KeepStdout (bool *keep* = true)**

Keep stdout same as parent's if keep = true

**5.144.3.14 void Arc::Run::Kill (int *timeout*)**

Kill running executable. First soft kill signal (SIGTERM) is sent to executable. If after timeout seconds executable is still running it's killed completely. Curenly this method does not work for Windows OS

**5.144.3.15 Arc::Run::operator bool (void) [inline]**

Returns true if object is valid

**5.144.3.16 bool Arc::Run::operator! (void) [inline]**

Returns true if object is invalid

**5.144.3.17 int Arc::Run::ReadStderr (int *timeout*, char \* *buf*, int *size*)**

Read from stderr handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stderr is directed to string. But result is unpredictable. Returns number of read bytes.

**5.144.3.18 int Arc::Run::ReadStdout (int *timeout*, char \* *buf*, int *size*)**

Read from stdout handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdout is directed to string. But result is unpredictable. Returns number of read bytes.



**5.144.3.19** `int Arc::Run::Result (void)` `[inline]`

Returns exit code of execution.

**5.144.3.20** `bool Arc::Run::Running (void)`

Return true if execution is going on.

**5.144.3.21** `Time Arc::Run::RunTime (void)` `[inline]`

Return when executable was started.

**5.144.3.22** `bool Arc::Run::Start (void)`

Starts running executable. This method may be called only once.

**5.144.3.23** `bool Arc::Run::Wait (void)`

Wait till execution finished

**5.144.3.24** `bool Arc::Run::Wait (int timeout)`

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

**5.144.3.25** `int Arc::Run::WriteStdin (int timeout, const char * buf, int size)`

Write to stdin handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdin is directed to string. But result is unpredictable. Returns number of written bytes.

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

- Run.h

## 5.145 Arc::SAMLToken Class Reference

Class for manipulating SAML Token Profile.

```
#include <SAMLToken.h>
```

### Public Types

- enum [SAMLVersion](#)

### Public Member Functions

- [SAMLToken](#) (SOAPEnvelope &soap)
- [SAMLToken](#) (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, [SAMLVersion](#) saml\_version=SAML2, [XMLNode](#) saml\_assertion=[XMLNode](#)())
- [~SAMLToken](#) (void)
- [operator bool](#) (void)
- bool [Authenticate](#) (const std::string &cafile, const std::string &capath)
- bool [Authenticate](#) (void)

#### 5.145.1 Detailed Description

Class for manipulating SAML Token Profile.

This class is for generating/consuming SAML Token profile. See WS-Security SAML Token Profile v1.1 ([www.oasis-open.org/committees/wss](http://www.oasis-open.org/committees/wss)) Currently this class is used by samltoken handler (will appears in src/hed/pdc/samltokensh/) It is not a must to directly called this class. If we need to use SAML Token functionality, we only need to configure the samltoken handler into service and client. Currently, only a minor part of the specification has been implemented.

About how to identify and reference security token for signing message, currently, only the "SAML Assertion Referenced from KeyInfo" (part 3.4.2 of WS-Security SAML Token Profile v1.1 specification) is supported, which means the implementation can only process SAML assertion "referenced from KeyInfo", and also can only generate SAML Token with SAML assertion "referenced from KeyInfo". More complete support need to implement.

About subject confirmation method, the implementation can process "hold-of-key" (part 3.5.1 of WS-Security SAML Token Profile v1.1 specification) subject subject confirmation method.

About SAML version, the implementation can process SAML assertion with SAML version 1.1 and 2.0; can only generate SAML assertion with SAML version 2.0.

In the SAML Token profile, for the hold-of-key subject confirmation method, there are three interaction parts: the attesting entity, the relying party and the issuing authority. In the hold-of-key subject confirmation method, it is the attesting entity's subject identity which will be inserted into the SAML assertion.

Firstly the attesting entity authenticates to issuing authority by using some authentication scheme such as WSS x509 Token profile (Alternatively the username/password authentication scheme or other different authentication scheme can also be used, unless the issuing authority can retrieve the key from a trusted certificate server after firmly establishing the subject's identity under the username/password scheme). So then issuing authority is able to make a definitive statement (sign a SAML assertion) about an act of authentication that has already taken place.

The attesting entity gets the SAML assertion and then signs the soap message together with the assertion by using its private key (the relevant certificate has been authenticated by issuing authority, and its relevant

public key has been put into SubjectConfirmation element under saml assertion by issuing authority. Only the actual owner of the saml assertion can do this, as only the subject possesses the private key paired with the public key in the assertion. This establishes an irrefutable connection between the author of the SOAP message and the assertion describing an authentication event.)

The relying party is supposed to trust the issuing authority. When it receives a message from the asserting entity, it will check the saml assertion based on its predetermined trust relationship with the SAML issuing authority, and check the signature of the soap message based on the public key in the saml assertion without directly trust relationship with attesting entity (subject owner).

## 5.145.2 Member Enumeration Documentation

### 5.145.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specification SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

## 5.145.3 Constructor & Destructor Documentation

### 5.145.3.1 Arc::SAMLToken::SAMLToken (SOAPEnvelope & soap)

Constructor. Parse SAML Token information from SOAP header. SAML Token related information is extracted from SOAP header and stored in class variables. And then it the [SAMLToken](#) object will be used for authentication.

#### Parameters:

*soap* The SOAP message which contains the [SAMLToken](#) in the soap header

### 5.145.3.2 Arc::SAMLToken::SAMLToken (SOAPEnvelope & soap, const std::string & certfile, const std::string & keyfile, SAMLVersion saml\_version = SAML2, XMLNode saml\_assertion = XMLNode ( ) )

Constructor. Add SAML Token information into the SOAP header. Generated token contains elements SAML token and signature, and is meant to be used for authentication on the consuming side. This constructor is for a specific SAML Token profile usage, in which the attesting entity signs the SAML assertion for itself (self-sign). This usage implicitly requires that the relying party trust the attesting entity. More general (requires issuing authority) usage will be provided by other constructor. And the under-developing SAML service will be used as the issuing authority.

#### Parameters:

*soap* The SOAP message to which the SAML Token will be inserted.

*certfile* The certificate file.

*keyfile* The key file which will be used to create signature.

*samlversion* The SAML version, only SAML2 is supported currently.

*samlassertion* The SAML assertion got from 3rd party, and used for protecting the SOAP message; If not present, then self-signed assertion will be generated.

#### 5.145.3.3 Arc::SAMLToken::~~SAMLToken (void)

Deconstructor. Nothing to be done except finalizing the xmlsec library.

### 5.145.4 Member Function Documentation

#### 5.145.4.1 bool Arc::SAMLToken::Authenticate (void)

Check signature by using the cert information in soap message

#### 5.145.4.2 bool Arc::SAMLToken::Authenticate (const std::string & *cafile*, const std::string & *capath*)

Check signature by using the trusted certificates It is used by relying parting after calling [SAMLToken\(SOAPEnvelope& soap\)](#) This method will check the SAML assertion based on the trusted certificated specified as parameter *cafile* or *capath*; and also check the signature to soap message (the signature is generated by attesting entity by signing soap body together with SAML assertion) by using the public key inside SAML assestion.

##### Parameters:

*cafile* ca file

*capath* ca directory

#### 5.145.4.3 Arc::SAMLToken::operator bool (void)

Returns true of constructor succeeded

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

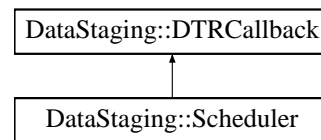
- SAMLToken.h

## 5.146 DataStaging::Scheduler Class Reference

The [Scheduler](#) is the control centre of the data staging framework.

```
#include <Scheduler.h>
```

Inheritance diagram for DataStaging::Scheduler::



### Public Member Functions

- [Scheduler](#) ()
- [~Scheduler](#) ()
- void [SetSlots](#) (int pre\_processor=0, int post\_processor=0, int delivery=0, int emergency=0, int staged\_prepared=0)
- void [AddURLMapping](#) (const [Arc::URL](#) &template\_url, const [Arc::URL](#) &replacement\_url, const [Arc::URL](#) &access\_url=[Arc::URL](#)())
- void [SetURLMapping](#) (const [Arc::URLMap](#) &mapping=[Arc::URLMap](#)())
- void [SetPreferredPattern](#) (const std::string &pattern)
- void [SetTransferSharesConf](#) (const [TransferSharesConf](#) &share\_conf)
- void [SetTransferParameters](#) (const [TransferParameters](#) &params)
- void [SetDeliveryServices](#) (const std::vector< [Arc::URL](#) > &endpoints)
- void [SetRemoteSizeLimit](#) (unsigned long long int limit)
- void [SetDumpLocation](#) (const std::string &location)
- bool [start](#) (void)
- virtual void [receiveDTR](#) ([DTR\\_ptr](#) dtr)
- bool [cancelDTRs](#) (const std::string &jobid)
- bool [stop](#) ()

### 5.146.1 Detailed Description

The [Scheduler](#) is the control centre of the data staging framework.

The [Scheduler](#) manages a global list of DTRs and schedules when they should go into the next state or be sent to other processes. The [DTR](#) priority is used to decide each DTR's position in a queue.

### 5.146.2 Constructor & Destructor Documentation

#### 5.146.2.1 DataStaging::Scheduler::Scheduler ()

Constructor.

#### 5.146.2.2 DataStaging::Scheduler::~~Scheduler () `[inline]`

Destructor calls [stop\(\)](#), which cancels all DTRs and waits for them to complete.

### 5.146.3 Member Function Documentation

**5.146.3.1** `void DataStaging::Scheduler::AddURLMapping (const Arc::URL & template_url, const Arc::URL & replacement_url, const Arc::URL & access_url = Arc::URL\(\))`

Add URL mapping entry.

**5.146.3.2** `bool DataStaging::Scheduler::cancelDTRs (const std::string & jobid)`

Tell the [Scheduler](#) to cancel all the DTRs in the given job description.

**5.146.3.3** `virtual void DataStaging::Scheduler::receiveDTR (DTR\_ptr dtr)` [virtual]

Callback method implemented from [DTRCallback](#).

This method is called by the generator when it wants to pass a [DTR](#) to the scheduler and when other processes send a [DTR](#) back to the scheduler after processing.

Implements [DataStaging::DTRCallback](#).

**5.146.3.4** `void DataStaging::Scheduler::SetDeliveryServices (const std::vector< Arc::URL > & endpoints)`

Set the list of delivery services. [DTR::LOCAL\\_DELIVERY](#) means local Delivery.

**5.146.3.5** `void DataStaging::Scheduler::SetDumpLocation (const std::string & location)`

Set location for periodic dump of [DTR](#) state (only file paths currently supported).

**5.146.3.6** `void DataStaging::Scheduler::SetPreferredPattern (const std::string & pattern)`

Set the preferred pattern.

**5.146.3.7** `void DataStaging::Scheduler::SetRemoteSizeLimit (unsigned long long int limit)`

Set the remote transfer size limit.

**5.146.3.8** `void DataStaging::Scheduler::SetSlots (int pre_processor = 0, int post_processor = 0, int delivery = 0, int emergency = 0, int staged_prepared = 0)`

Set number of slots for processor and delivery stages.

**5.146.3.9** `void DataStaging::Scheduler::SetTransferParameters (const TransferParameters & params)`

Set transfer limits.

**5.146.3.10 void DataStaging::Scheduler::SetTransferSharesConf (const [TransferSharesConf](#) & *share\_conf*)**

Set [TransferShares](#) configuration.

**5.146.3.11 void DataStaging::Scheduler::SetURLMapping (const Arc::URLMap & *mapping* = Arc::URLMap())**

Replace all URL mapping entries.

**5.146.3.12 bool DataStaging::Scheduler::start (void)**

Start scheduling activity.

This method must be called after all configuration parameters are set properly. [Scheduler](#) can be stopped either by calling [stop\(\)](#) method or by destroying its instance.

**5.146.3.13 bool DataStaging::Scheduler::stop ()**

Tell the [Scheduler](#) to shut down all threads and exit.

All active DTRs are cancelled and this method waits until they finish (all DTRs go to CANCELLED state)

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

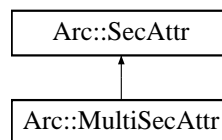
- [Scheduler.h](#)

## 5.147 Arc::SecAttr Class Reference

This is an abstract interface to a security attribute.

```
#include <SecAttr.h>
```

Inheritance diagram for Arc::SecAttr::



### Public Member Functions

- [SecAttr](#) ()
- bool [operator==](#) (const [SecAttr](#) &b) const
- bool [operator!=](#) (const [SecAttr](#) &b) const
- virtual [operator bool](#) () const
- virtual bool [Export](#) ([SecAttrFormat](#) format, std::string &val) const
- virtual bool [Export](#) ([SecAttrFormat](#) format, [XMLNode](#) &val) const
- virtual bool [Import](#) ([SecAttrFormat](#) format, const std::string &val)
- virtual std::string [get](#) (const std::string &id) const
- virtual std::list< std::string > [getAll](#) (const std::string &id) const

### Static Public Attributes

- static [SecAttrFormat](#) ARCAuth
- static [SecAttrFormat](#) XACML
- static [SecAttrFormat](#) SAML
- static [SecAttrFormat](#) GACL

#### 5.147.1 Detailed Description

This is an abstract interface to a security attribute.

This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using `dynamic_cast` operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

#### 5.147.2 Constructor & Destructor Documentation

##### 5.147.2.1 Arc::SecAttr::SecAttr () [inline]

representation for GACL policy



### 5.147.3 Member Function Documentation

**5.147.3.1** `virtual bool Arc::SecAttr::Export (SecAttrFormat format, XMLNode & val) const` [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented in [Arc::MultiSecAttr](#).

**5.147.3.2** `virtual bool Arc::SecAttr::Export (SecAttrFormat format, std::string & val) const` [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute.

**5.147.3.3** `virtual std::string Arc::SecAttr::get (const std::string & id) const` [virtual]

Access to specific item of the security attribute. If there are few items of same id the first one is presented. It is meant to be used for tightly coupled SecHandlers and provides more effective interface than Export.

**5.147.3.4** `virtual std::list<std::string> Arc::SecAttr::getAll (const std::string & id) const` [virtual]

Access to specific items of the security attribute. This method returns all items which have id assigned. It is meant to be used for tightly coupled SecHandlers and provides more effective interface than Export.

**5.147.3.5** `virtual bool Arc::SecAttr::Import (SecAttrFormat format, const std::string & val)` [virtual]

Fills internal structure from external object of specified format. Returns false if failed to do. The usage pattern for this method is not defined and it is provided only to make class symmetric. Hence it's implementation is not required yet.

**5.147.3.6** `virtual Arc::SecAttr::operator bool () const` [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in [Arc::MultiSecAttr](#).

**5.147.3.7** `bool Arc::SecAttr::operator!= (const SecAttr & b) const` [inline]

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

**5.147.3.8** `bool Arc::SecAttr::operator== (const SecAttr & b) const` [inline]

This function should (in inheriting classes) return true if this and b are considered to represent same content. Identifying and restricting the type of b should be done using dynamic\_cast operations. Currently it is not defined how comparison methods to be used. Hence their implementation is not required.

#### 5.147.4 Field Documentation

##### 5.147.4.1 [SecAttrFormat Arc::SecAttr::ARCAuth](#) [static]

own serialization/deserialization format

##### 5.147.4.2 [SecAttrFormat Arc::SecAttr::GACL](#) [static]

suitable for inclusion into SAML structures

##### 5.147.4.3 [SecAttrFormat Arc::SecAttr::SAML](#) [static]

representation for XACML policy

##### 5.147.4.4 [SecAttrFormat Arc::SecAttr::XACML](#) [static]

representation for ARC authorization policy

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

- SecAttr.h

## 5.148 Arc::SecAttrFormat Class Reference

Export/import format.

```
#include <SecAttr.h>
```

### 5.148.1 Detailed Description

Export/import format.

Format is identified by textual identity string. Class description includes basic formats only. That list may be extended.

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

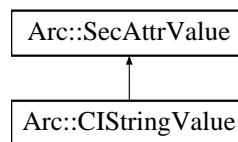
- SecAttr.h

## 5.149 Arc::SecAttrValue Class Reference

This is an abstract interface to a security attribute.

```
#include <SecAttrValue.h>
```

Inheritance diagram for Arc::SecAttrValue::



### Public Member Functions

- `bool operator== (SecAttrValue &b)`
- `bool operator!= (SecAttrValue &b)`
- `virtual operator bool ()`

#### 5.149.1 Detailed Description

This is an abstract interface to a security attribute.

This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using `dynamic_cast` operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

#### 5.149.2 Member Function Documentation

##### 5.149.2.1 `virtual Arc::SecAttrValue::operator bool ()` [virtual]

This function should return false if the value is to be considered null, e g if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in [Arc::CIStrStringValue](#).

##### 5.149.2.2 `bool Arc::SecAttrValue::operator!= (SecAttrValue & b)`

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

##### 5.149.2.3 `bool Arc::SecAttrValue::operator== (SecAttrValue & b)`

This function should (in inheriting classes) return true if this and b are considered to be the same. Identifying and restricting the type of b should be done using `dynamic_cast` operations.

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

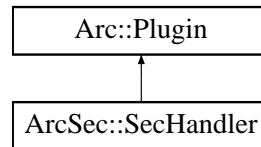
- SecAttrValue.h

## 5.150 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

```
#include <SecHandler.h>
```

Inheritance diagram for ArcSec::SecHandler::



### 5.150.1 Detailed Description

Base class for simple security handling plugins.

This virtual class defines method `Handle()` which processes security related information/attributes in Message and optionally makes security decision. Instances of such classes are normally arranged in chains and are called on incoming and outgoing messages in various MCC and Service plugins. Return value of `Handle()` defines either processing should continue (true) or stop with error (false). Configuration of [SecHandler](#) is consumed during creation of instance through XML subtree fed to constructor.

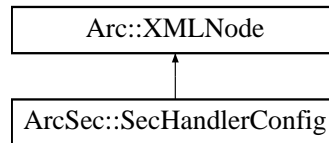
The documentation for this class was generated from the following file:

- SecHandler.h

## 5.151 ArcSec::SecHandlerConfig Class Reference

```
#include <SecHandler.h>
```

Inheritance diagram for ArcSec::SecHandlerConfig::



### 5.151.1 Detailed Description

Helper class to create [Security](#) Handler configuration

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

- SecHandler.h

## 5.152 ArcSec::Security Class Reference

Common stuff used by security related slasses.

```
#include <Security.h>
```

### 5.152.1 Detailed Description

Common stuff used by security related slasses.

This class is just a place where to put common stuff that is used by security related slasses. So far it only contains a logger.

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

- Security.h

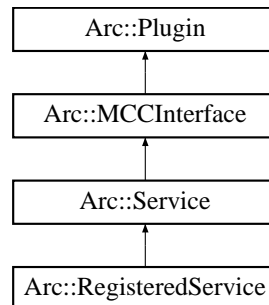


## 5.153 Arc::Service Class Reference

[Service](#) - last component in a [Message](#) Chain.

```
#include <Service.h>
```

Inheritance diagram for Arc::Service::



### Public Member Functions

- [Service](#) ([Config](#) \*, [PluginArgument](#) \*arg)
- virtual void [AddSecHandler](#) ([Config](#) \*cfg, [ArcSec::SecHandler](#) \*sechandler, const std::string &label="")
- virtual bool [RegistrationCollector](#) ([XMLNode](#) &doc)
- virtual std::string [getID](#) ()
- [operator bool](#) () const
- bool [operator!](#) () const

### Protected Member Functions

- bool [ProcessSecHandlers](#) ([Message](#) &message, const std::string &label="") const

### Protected Attributes

- std::map< std::string, std::list< [ArcSec::SecHandler](#) \* > > [sechandlers\\_](#)
- bool [valid](#)

### Static Protected Attributes

- static [Logger](#) [logger](#)

#### 5.153.1 Detailed Description

[Service](#) - last component in a [Message](#) Chain.

This class which defines interface and common functionality for every [Service](#) plugin. Interface is made of method [process\(\)](#) which is called by [Plexer](#) or [MCC](#) class. There is one [Service](#) object created for every service description processed by [Loader](#) class objects. Classes derived from [Service](#) class must implement

`process()` method of [MCCInterface](#). It is up to developer how internal state of service is stored and communicated to other services and external utilities. [Service](#) is free to expect any type of payload passed to it and generate any payload as well. Useful types depend on MCCs in chain which leads to that service. For example if service is expected to be linked to SOAP [MCC](#) it must accept and generate messages with [PayloadSOAP](#) payload. Method `process()` of class derived from [Service](#) class may be called concurrently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in `/src/tests/echo/echo.cpp` of source tree. The way to write client counterpart of corresponding service is undefined yet. For example see `/src/tests/echo/test.cpp`.

### 5.153.2 Constructor & Destructor Documentation

#### 5.153.2.1 `Arc::Service::Service (Config *, PluginArgument * arg)`

Example constructor - Server takes at least its configuration subtree

### 5.153.3 Member Function Documentation

#### 5.153.3.1 `virtual void Arc::Service::AddSecHandler (Config * cfg, ArcSec::SecHandler * sechandler, const std::string & label = "")` [virtual]

Add security components/handlers to this [MCC](#). For more information please see description of [MCC::AddSecHandler](#)

#### 5.153.3.2 `virtual std::string Arc::Service::getID ()` [inline, virtual]

[Service](#) may implement own service identifier gathering method. This method return identifier of service which is used for registering it Information Services.

#### 5.153.3.3 `Arc::Service::operator bool (void) const` [inline]

Returns true if the [Service](#) is valid.

#### 5.153.3.4 `bool Arc::Service::operator! (void) const` [inline]

Returns true if the [Service](#) is not valid.

#### 5.153.3.5 `bool Arc::Service::ProcessSecHandlers (Message & message, const std::string & label = "") const` [protected]

Executes security handlers of specified queue. For more information please see description of [MCC::ProcessSecHandlers](#)

#### 5.153.3.6 `virtual bool Arc::Service::RegistrationCollector (XMLNode & doc)` [virtual]

[Service](#) specific registration collector, used for generate service registrations. In implemented service this method should generate [GLUE2](#) document with part of service description which service wishes to advertise to Information Services.

### 5.153.4 Field Documentation

#### 5.153.4.1 `Logger Arc::Service::logger` [static, protected]

`Logger` object used to print messages generated by this class.

#### 5.153.4.2 `std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::Service::sechandlers_` [protected]

Set of labelled authentication and authorization handlers. `MCC` calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

#### 5.153.4.3 `bool Arc::Service::valid` [protected]

Is service valid? Services which are not valid should set this to false in their constructor.

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

- Service.h

## 5.154 Arc::SimpleCondition Class Reference

Simple triggered condition.

```
#include <Thread.h>
```

### Public Member Functions

- void [lock](#) (void)
- void [unlock](#) (void)
- void [signal](#) (void)
- void [signal\\_nonblock](#) (void)
- void [broadcast](#) (void)
- void [wait](#) (void)
- void [wait\\_nonblock](#) (void)
- bool [wait](#) (int t)
- void [reset](#) (void)

### 5.154.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

### 5.154.2 Member Function Documentation

#### 5.154.2.1 void Arc::SimpleCondition::broadcast (void) [inline]

Signal about condition to all waiting threads. If there are no waiting threads, it works like [signal\(\)](#).

#### 5.154.2.2 void Arc::SimpleCondition::lock (void) [inline]

Acquire semaphor

#### 5.154.2.3 void Arc::SimpleCondition::reset (void) [inline]

Reset object to initial state

#### 5.154.2.4 void Arc::SimpleCondition::signal (void) [inline]

Signal about condition. This overrides [broadcast\(\)](#).

#### 5.154.2.5 void Arc::SimpleCondition::signal\_nonblock (void) [inline]

Signal about condition without using semaphor. Call it *\*only\** with lock acquired.

**5.154.2.6 void Arc::SimpleCondition::unlock (void) [inline]**

Release semaphor

**5.154.2.7 bool Arc::SimpleCondition::wait (int *t*) [inline]**

Wait for condition no longer than *t* milliseconds

**5.154.2.8 void Arc::SimpleCondition::wait (void) [inline]**

Wait for condition

**5.154.2.9 void Arc::SimpleCondition::wait\_nonblock (void) [inline]**

Wait for condition without using semaphor. Call it *\*only\** with lock acquired.

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

- Thread.h

## 5.155 Arc::SimpleCounter Class Reference

```
#include <Thread.h>
```

### Public Member Functions

- virtual int [inc](#) (void)
- virtual int [dec](#) (void)
- virtual int [get](#) (void) const
- virtual int [set](#) (int v)
- virtual void [wait](#) (void) const
- virtual bool [wait](#) (int t) const
- virtual void [forceReset](#) (void)

### 5.155.1 Detailed Description

Thread-safe counter with capability to wait for zero value. It is extendable through re-implementation of virtual methods.

### 5.155.2 Member Function Documentation

#### 5.155.2.1 virtual int Arc::SimpleCounter::dec (void) [virtual]

Decrement value of counter.

Returns new value. Does not go below 0 value.

#### 5.155.2.2 virtual void Arc::SimpleCounter::forceReset (void) [inline, virtual]

This method is meant to be used only after fork.

It resets state of all internal locks and variables.

#### 5.155.2.3 virtual int Arc::SimpleCounter::get (void) const [virtual]

Returns current value of counter.

#### 5.155.2.4 virtual int Arc::SimpleCounter::inc (void) [virtual]

Increment value of counter.

Returns new value.

#### 5.155.2.5 virtual int Arc::SimpleCounter::set (int v) [virtual]

Set value of counter.

Returns new value.

**5.155.2.6 virtual bool Arc::SimpleCounter::wait (int *t*) const** [virtual]

Wait for zero condition no longer than *t* milliseconds.

If *t* is negative - wait forever.

**5.155.2.7 virtual void Arc::SimpleCounter::wait (void) const** [virtual]

Wait for zero condition.

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

- Thread.h

## 5.156 Arc::SOAPMessage Class Reference

[Message](#) restricted to SOAP payload.

```
#include <SOAPMessage.h>
```

### Public Member Functions

- [SOAPMessage](#) (void)
- [SOAPMessage](#) (long msg\_ptr\_addr)
- [SOAPMessage](#) ([Message](#) &msg)
- [~SOAPMessage](#) (void)
- SOAPEnvelope \* [Payload](#) (void)
- void [Payload](#) (SOAPEnvelope \*new\_payload)
- [MessageAttributes](#) \* [Attributes](#) (void)

### 5.156.1 Detailed Description

[Message](#) restricted to SOAP payload.

This is a special [Message](#) intended to be used in language bindings for programming languages which are not flexible enough to support all kinds of Payloads. It is passed through chain of MCCs and works like the [Message](#) but can carry only SOAP content.

### 5.156.2 Constructor & Destructor Documentation

#### 5.156.2.1 Arc::SOAPMessage::SOAPMessage (void) [inline]

Dummy constructor

#### 5.156.2.2 Arc::SOAPMessage::SOAPMessage (long msg\_ptr\_addr)

Copy constructor. Used by language bindings

#### 5.156.2.3 Arc::SOAPMessage::SOAPMessage ([Message](#) & msg)

Copy constructor. Ensures shallow copy.

#### 5.156.2.4 Arc::SOAPMessage::~~SOAPMessage (void)

Destructor does not affect referred objects

### 5.156.3 Member Function Documentation

#### 5.156.3.1 [MessageAttributes](#)\* Arc::SOAPMessage::Attributes (void) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.



**5.156.3.2 void Arc::SOAPMessage::Payload (SOAPEnvelope \* *new\_payload*)**

Replace payload with a COPY of new one

**5.156.3.3 SOAPEnvelope\* Arc::SOAPMessage::Payload (void)**

Returns pointer to current payload or NULL if no payload assigned.

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

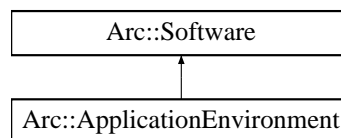
- SOAPMessage.h

## 5.157 Arc::Software Class Reference

Used to represent software (names and version) and comparison.

```
#include <Software.h>
```

Inheritance diagram for Arc::Software::



### Public Types

- typedef bool(Software::\*) [ComparisonOperator](#) (const [Software](#) &) const
- [NOTEQUAL](#) = 0
- [EQUAL](#) = 1
- [GREATERTHAN](#) = 2
- [LESSTHAN](#) = 3
- [GREATERTHANOREQUAL](#) = 4
- [LESSTHANOREQUAL](#) = 5
- enum [ComparisonOperatorEnum](#) {  
[NOTEQUAL](#) = 0, [EQUAL](#) = 1, [GREATERTHAN](#) = 2, [LESSTHAN](#) = 3,  
[GREATERTHANOREQUAL](#) = 4, [LESSTHANOREQUAL](#) = 5 }

### Public Member Functions

- [Software](#) ()
- [Software](#) (const std::string &name\_version)
- [Software](#) (const std::string &name, const std::string &version)
- [Software](#) (const std::string &family, const std::string &name, const std::string &version)
- bool [empty](#) () const
- bool [operator==](#) (const [Software](#) &sw) const
- bool [operator!=](#) (const [Software](#) &sw) const
- bool [operator>](#) (const [Software](#) &sw) const
- bool [operator<](#) (const [Software](#) &sw) const
- bool [operator>=](#) (const [Software](#) &sw) const
- bool [operator<=](#) (const [Software](#) &sw) const
- std::string [operator\(\)](#) () const
- [operator std::string](#) (void) const
- const std::string & [getFamily](#) () const
- const std::string & [getName](#) () const
- const std::string & [getVersion](#) () const

### Static Public Member Functions

- static [ComparisonOperator](#) [convert](#) (const [ComparisonOperatorEnum](#) &co)
- static std::string [toString](#) ([ComparisonOperator](#) co)

## Static Public Attributes

- static const std::string [VERSIONTOKENS](#)

## Friends

- std::ostream & [operator<<](#) (std::ostream &out, const [Software](#) &sw)

### 5.157.1 Detailed Description

Used to represent software (names and version) and comparison.

The [Software](#) class is used to represent the name of a piece of software internally. Generally software are identified by a name and possibly a version number. Some software can also be categorized by type or family (compilers, operating system, etc.). A software object can be compared to other software objects using the comparison operators contained in this class. The basic usage of this class is to test if some specified software requirement ([SoftwareRequirement](#)) are fulfilled, by using the comparability of the class.

Internally the [Software](#) object is represented by a family and name identifier, and the software version is tokenized at the characters defined in VERSIONTOKENS, and stored as a list of tokens.

### 5.157.2 Member Typedef Documentation

#### 5.157.2.1 typedef bool(Software::\*) [Arc::Software::ComparisonOperator](#)(const [Software](#) &) const

Definition of a comparison operator method pointer.

This typedef defines a comparison operator method pointer.

See also:

```
operator==,  
operator!=,  
operator>,  
operator<,  
operator>=,  
operator<=,  
ComparisonOperatorEnum.
```

### 5.157.3 Member Enumeration Documentation

#### 5.157.3.1 enum [Arc::Software::ComparisonOperatorEnum](#)

Comparison operator enum.

The [ComparisonOperatorEnum](#) enumeration is a 1-1 correspondance between the defined comparison method operators ([Software::ComparisonOperator](#)), and can be used in circumstances where method pointers are not supported.

Enumerator:

```
NOTEQUAL see operator!=  
EQUAL see operator==
```

***GREATERTHAN*** see operator>

***LESSTHAN*** see operator<

***GREATERTHANOREQUAL*** see operator>=

***LESSTHANOREQUAL*** see operator<=

## 5.157.4 Constructor & Destructor Documentation

### 5.157.4.1 `Arc::Software::Software ()` `[inline]`

Dummy constructor.

This constructor creates a empty object.

### 5.157.4.2 `Arc::Software::Software (const std::string & name_version)`

Create a [Software](#) object.

Create a [Software](#) object from a single string composed of a name and a version part. The created object will contain a empty family part. The name and version part of the string will be split at the first occurrence of a dash (-) which is followed by a digit (0-9). If the string does not contain such a pattern, the passed string will be taken to be the name and version will be empty.

#### Parameters:

*name\_version* should be a string composed of the name and version of the software to represent.

### 5.157.4.3 `Arc::Software::Software (const std::string & name, const std::string & version)`

Create a [Software](#) object.

Create a [Software](#) object with the specified name and version. The family part will be left empty.

#### Parameters:

*name* the software name to represent.

*version* the software version to represent.

### 5.157.4.4 `Arc::Software::Software (const std::string & family, const std::string & name, const std::string & version)`

Create a [Software](#) object.

Create a [Software](#) object with the specified family, name and version.

#### Parameters:

*family* the software family to represent.

*name* the software name to represent.

*version* the software version to represent.

## 5.157.5 Member Function Documentation

### 5.157.5.1 static [ComparisonOperator](#) Arc::Software::convert (const [ComparisonOperatorEnum](#) & *co*) [static]

Convert a [ComparisonOperatorEnum](#) value to a comparison method pointer.

The passed [ComparisonOperatorEnum](#) will be converted to a comparison method pointer defined by the [Software::ComparisonOperator](#) typedef.

This static method is not defined in language bindings created with Swig, since method pointers are not supported by Swig.

#### Parameters:

*co* a [ComparisonOperatorEnum](#) value.

#### Returns:

A method pointer to a comparison method is returned.

### 5.157.5.2 bool Arc::Software::empty () const [inline]

Indicates whether the object is empty.

#### Returns:

true if the name of this object is empty, otherwise false.

### 5.157.5.3 const std::string& Arc::Software::getFamily () const [inline]

Get family.

#### Returns:

The family the represented software belongs to is returned.

### 5.157.5.4 const std::string& Arc::Software::getName () const [inline]

Get name.

#### Returns:

The name of the represented software is returned.

### 5.157.5.5 const std::string& Arc::Software::getVersion () const [inline]

Get version.

#### Returns:

The version of the represented software is returned.

**5.157.5.6** `Arc::Software::operator std::string (void) const` `[inline]`

Cast to string.

This casting operator behaves exactly as `::operator()` does. The cast is used like `(std::string) <software-object>`.

**See also:**

`operator()`.

**5.157.5.7** `bool Arc::Software::operator!= (const Software & sw) const` `[inline]`

Inequality operator.

The behaviour of the inequality operator is just opposite that of the equality operator (`operator==(())`).

**Parameters:**

`sw` is the RHS `Software` object.

**Returns:**

`true` when the two objects are unequal, otherwise `false`.

**5.157.5.8** `std::string Arc::Software::operator() () const`

Get string representation.

Returns the string representation of this object, which is 'family'-'name'-'version'.

**Returns:**

The string representation of this object is returned.

**See also:**

`operator std::string()`.

**5.157.5.9** `bool Arc::Software::operator< (const Software & sw) const` `[inline]`

Less-than operator.

The behaviour of this less-than operator is equivalent to the greater-than operator (`operator>()`) with the LHS and RHS swapped.

**Parameters:**

`sw` is the RHS object.

**Returns:**

`true` if the LHS is less than the RHS, otherwise `false`.

**See also:**

`operator>()`.

**5.157.5.10** `bool Arc::Software::operator<= (const Software & sw) const` `[inline]`

Less-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (`operator==(())`) or if the LHS is greater than the RHS (`operator>()`).

**Parameters:**

`sw` is the RHS object.

**Returns:**

`true` if the LHS is less than or equal the RHS, otherwise `false`.

**See also:**

`operator==(())`,  
`operator<()`.

**5.157.5.11** `bool Arc::Software::operator== (const Software & sw) const` `[inline]`

Equality operator.

Two `Software` objects are equal only if they are of the same family, have the same name and is of same version. This operator can also be represented by the `Software::EQUAL ComparisonOperatorEnum` value.

**Parameters:**

`sw` is the RHS `Software` object.

**Returns:**

`true` when the two objects equals, otherwise `false`.

**5.157.5.12** `bool Arc::Software::operator> (const Software & sw) const`

Greater-than operator.

For the LHS object to be greater than the RHS object they must first share the same family and name. If the version of the LHS is empty or the LHS and RHS versions equal then LHS is not greater than RHS. If the LHS version is not empty while the RHS is then LHS is greater than RHS. If both versions are non empty and not equal then, the first version token of each object is compared and if they are identical, the two next version tokens will be compared. If not identical, the two tokens will be parsed as integers, and if parsing fails the LHS is not greater than the RHS. If parsing succeeds and the integers equals, the two next tokens will be compared, otherwise the comparison is resolved by the integer comparison.

If the LHS contains more version tokens than the RHS, and the comparison have not been resolved at the point of equal number of tokens, then if the additional tokens contains a token which cannot be parsed to a integer the LHS is not greater than the RHS. If the parsed integer is not 0 then the LHS is greater than the RHS. If the rest of the additional tokens are 0, the LHS is not greater than the RHS.

If the RHS contains more version tokens than the LHS and comparison have not been resolved at the point of equal number of tokens, or simply if comparison have not been resolved at the point of equal number of tokens, then the LHS is not greater than the RHS.

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is greater than the RHS, otherwise `false`.

**5.157.5.13** `bool Arc::Software::operator>= (const Software & sw) const` `[inline]`

Greater-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (`operator==(())`) or if the LHS is greater than the RHS (`operator>()`).

**Parameters:**

*sw* is the RHS object.

**Returns:**

`true` if the LHS is greater than or equal the RHS, otherwise `false`.

**See also:**

`operator==(())`,  
`operator>()`.

**5.157.5.14** `static std::string Arc::Software::toString (ComparisonOperator co)` `[static]`

Convert `Software::ComparisonOperator` to a string.

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig.

**Parameters:**

*co* is a `Software::ComparisonOperator`.

**Returns:**

The string representation of the passed `Software::ComparisonOperator` is returned.

**5.157.6 Friends And Related Function Documentation****5.157.6.1** `std::ostream& operator<< (std::ostream & out, const Software & sw)` `[friend]`

Write `Software` string representation to a `std::ostream`.

Write the string representation of a `Software` object to a `std::ostream`.

**Parameters:**

*out* is a `std::ostream` to write the string representation of the `Software` object to.

*sw* is the `Software` object to write to the `std::ostream`.

**Returns:**

The passed `std::ostream` *out* is returned.



## 5.157.7 Field Documentation

### 5.157.7.1 `const std::string Arc::Software::VERSIONTOKENS` [static]

Tokens used to split version string.

This string constant specifies which tokens will be used to split the version string.

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

- Software.h

## 5.158 Arc::SoftwareRequirement Class Reference

Class used to express and resolve version requirements on software.

```
#include <Software.h>
```

### Public Member Functions

- [SoftwareRequirement](#) ()
- [SoftwareRequirement](#) (const [Software](#) &sw, [Software::ComparisonOperator](#) swComOp)
- [SoftwareRequirement](#) (const [Software](#) &sw, [Software::ComparisonOperatorEnum](#) co=[Software::EQUAL](#))
- [SoftwareRequirement](#) & operator= (const [SoftwareRequirement](#) &sr)
- [SoftwareRequirement](#) (const [SoftwareRequirement](#) &sr)
- void [add](#) (const [Software](#) &sw, [Software::ComparisonOperator](#) swComOp)
- void [add](#) (const [Software](#) &sw, [Software::ComparisonOperatorEnum](#) co)
- bool [isSatisfied](#) (const [Software](#) &sw) const
- bool [isSatisfied](#) (const std::list< [Software](#) > &swList) const
- bool [isSatisfied](#) (const std::list< [ApplicationEnvironment](#) > &swList) const
- bool [selectSoftware](#) (const [Software](#) &sw)
- bool [selectSoftware](#) (const std::list< [Software](#) > &swList)
- bool [selectSoftware](#) (const std::list< [ApplicationEnvironment](#) > &swList)
- bool [isResolved](#) () const
- bool [empty](#) () const
- void [clear](#) ()
- const std::list< [Software](#) > & [getSoftwareList](#) () const
- const std::list< [Software::ComparisonOperator](#) > & [getComparisonOperatorList](#) () const

### 5.158.1 Detailed Description

Class used to express and resolve version requirements on software.

A requirement in this class is defined as a pair composed of a [Software](#) object and either a [Software::ComparisonOperator](#) method pointer or equally a [Software::ComparisonOperatorEnum](#) enum value. A [SoftwareRequirement](#) object can contain multiple of such requirements, and then it can specified if all these requirements should be satisfied, or if it is enough to satisfy only one of them. The requirements can be satisfied by a single [Software](#) object or a list of either [Software](#) or [ApplicationEnvironment](#) objects, by using the method [isSatisfied\(\)](#). This class also contain a number of methods ([selectSoftware\(\)](#)) to select [Software](#) objects which are satisfying the requirements, and in this way resolving requirements.

### 5.158.2 Constructor & Destructor Documentation

#### 5.158.2.1 Arc::SoftwareRequirement::SoftwareRequirement () [inline]

Create a empty [SoftwareRequirement](#) object.

The created [SoftwareRequirement](#) object will contain no requirements.

### 5.158.2.2 Arc::SoftwareRequirement::SoftwareRequirement (const [Software](#) & *sw*, [Software::ComparisonOperator](#) *swComOp*)

Create a [SoftwareRequirement](#) object.

The created [SoftwareRequirement](#) object will contain one requirement specified by the [Software](#) object *sw*, and the [Software::ComparisonOperator](#) *swComOp*.

This constructor is not available in language bindings created by Swig, since method pointers are not supported by Swig, see [SoftwareRequirement\(const Software&, Software::ComparisonOperatorEnum\)](#) instead.

#### Parameters:

*sw* is the [Software](#) object of the requirement to add.

*swComOp* is the [Software::ComparisonOperator](#) of the requirement to add.

### 5.158.2.3 Arc::SoftwareRequirement::SoftwareRequirement (const [Software](#) & *sw*, [Software::ComparisonOperatorEnum](#) *co* = `Software::EQUAL`)

Create a [SoftwareRequirement](#) object.

The created [SoftwareRequirement](#) object will contain one requirement specified by the [Software](#) object *sw*, and the [Software::ComparisonOperatorEnum](#) *co*.

#### Parameters:

*sw* is the [Software](#) object of the requirement to add.

*co* is the [Software::ComparisonOperatorEnum](#) of the requirement to add.

### 5.158.2.4 Arc::SoftwareRequirement::SoftwareRequirement (const [SoftwareRequirement](#) & *sr*) [inline]

Copy constructor.

Create a [SoftwareRequirement](#) object from another [SoftwareRequirement](#) object.

#### Parameters:

*sr* is the [SoftwareRequirement](#) object to make a copy of.

## 5.158.3 Member Function Documentation

### 5.158.3.1 void Arc::SoftwareRequirement::add (const [Software](#) & *sw*, [Software::ComparisonOperatorEnum](#) *co*)

Add a [Software](#) object a corresponding comparison operator to this object.

Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

#### Parameters:

*sw* is the [Software](#) object to add as part of a requirement.

*co* is the [Software::ComparisonOperatorEnum](#) value to add as part of a requirement, the default enum will be [Software::EQUAL](#).

#### 5.158.3.2 void Arc::SoftwareRequirement::add (const [Software](#) & *sw*, [Software::ComparisonOperator](#) *swComOp*)

Add a [Software](#) object a corresponding comparison operator to this object.

Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig, see [add\(const Software&, Software::ComparisonOperatorEnum\)](#) instead.

##### Parameters:

*sw* is the [Software](#) object to add as part of a requirement.

*swComOp* is the [Software::ComparisonOperator](#) method pointer to add as part of a requirement, the default operator will be [Software::operator==\(\(\)\)](#).

#### 5.158.3.3 void Arc::SoftwareRequirement::clear () [inline]

Clear the object.

The requirements in this object will be cleared when invoking this method.

#### 5.158.3.4 bool Arc::SoftwareRequirement::empty () const [inline]

Test if the object is empty.

##### Returns:

`true` if this object do no contain any requirements, otherwise `false`.

#### 5.158.3.5 const std::list<[Software::ComparisonOperator](#)>& Arc::SoftwareRequirement::get-ComparisonOperatorList () const [inline]

Get list of comparison operators.

##### Returns:

The list of internally stored comparison operators is returned.

##### See also:

[Software::ComparisonOperator](#),  
[getSoftwareList](#).

### 5.158.3.6 `const std::list<Software>& Arc::SoftwareRequirement::getSoftwareList () const` `[inline]`

Get list of [Software](#) objects.

#### Returns:

The list of internally stored [Software](#) objects is returned.

#### See also:

[Software](#),  
[getComparisonOperatorList](#).

### 5.158.3.7 `bool Arc::SoftwareRequirement::isResolved () const`

Indicates whether requirements have been resolved or not.

If specified that only one requirement has to be satisfied, then for this object to be resolved it can only contain one requirement and it has use the equal operator ([Software::operator==](#)).

If specified that all requirements has to be satisfied, then for this object to be resolved each requirement must have a [Software](#) object with a unique family/name composition, i.e. no other requirements have a [Software](#) object with the same family/name composition, and each requirement must use the equal operator ([Software::operator==](#)).

If this object has been resolved then `true` is returned when invoking this method, otherwise `false` is returned.

#### Returns:

`true` if this object have been resolved, otherwise `false`.

### 5.158.3.8 `bool Arc::SoftwareRequirement::isSatisfied (const std::list< ApplicationEnvironment > & swList) const`

Test if requirements are satisfied.

This method behaves in exactly the same way as the [isSatisfied\(const Software&\) const](#) method does.

#### Parameters:

*swList* is the list of [ApplicationEnvironment](#) objects which should be used to try satisfy the requirements.

#### Returns:

`true` if requirements are satisfied, otherwise `false`.

#### See also:

[isSatisfied\(const Software&\) const](#),  
[isSatisfied\(const std::list<Software>&\) const](#),  
[selectSoftware\(const std::list<ApplicationEnvironment>&\)](#),  
[isResolved\(\) const](#).

### 5.158.3.9 `bool Arc::SoftwareRequirement::isSatisfied (const std::list< Software > & swList) const` `[inline]`

Test if requirements are satisfied.

Returns `true` if stored requirements are satisfied by software specified in *swList*, otherwise `false` is returned.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single [Software](#) object.

#### Parameters:

*swList* is the list of [Software](#) objects which should be used to try satisfy the requirements.

#### Returns:

`true` if requirements are satisfied, otherwise `false`.

#### See also:

[isSatisfied\(const \[Software\]\(#\)&\) const](#),  
[isSatisfied\(const std::list<\[ApplicationEnvironment\]\(#\)>&\) const](#),  
[selectSoftware\(const std::list<\[Software\]\(#\)>&\)](#),  
[isResolved\(\) const](#).

### 5.158.3.10 `bool Arc::SoftwareRequirement::isSatisfied (const Software & sw) const` `[inline]`

Test if requirements are satisfied.

Returns `true` if the requirements are satisfied by the specified [Software](#) *sw*, otherwise `false` is returned.

#### Parameters:

*sw* is the [Software](#) which should satisfy the requirements.

#### Returns:

`true` if requirements are satisfied, otherwise `false`.

#### See also:

[isSatisfied\(const std::list<\[Software\]\(#\)>&\) const](#),  
[isSatisfied\(const std::list<\[ApplicationEnvironment\]\(#\)>&\) const](#),  
[selectSoftware\(const \[Software\]\(#\)&\)](#),  
[isResolved\(\) const](#).

### 5.158.3.11 `SoftwareRequirement& Arc::SoftwareRequirement::operator= (const SoftwareRequirement & sr)`

Assignment operator.

Set this object equal to that of the passed [SoftwareRequirement](#) object *sr*.

#### Parameters:

*sr* is the [SoftwareRequirement](#) object to set object equal to.

### 5.158.3.12 `bool Arc::SoftwareRequirement::selectSoftware (const std::list< ApplicationEnvironment > & swList)`

Select software.

This method behaves exactly as the `selectSoftware(const std::list<Software>&)` method does.

#### Parameters:

*swList* is a list of [ApplicationEnvironment](#) objects used to satisfy requirements.

#### Returns:

`true` if requirements are satisfied, otherwise `false`.

#### See also:

`selectSoftware(const Software&),`  
`selectSoftware(const std::list<Software>&),`  
`isSatisfied(const std::list<ApplicationEnvironment>&) const,`  
`isResolved() const.`

### 5.158.3.13 `bool Arc::SoftwareRequirement::selectSoftware (const std::list< Software > & swList)`

Select software.

If the passed list of [Software](#) objects *swList* do not satisfy the requirements `false` is returned and this object is not modified. If however the list of [Software](#) objects *swList* do satisfy the requirements `true` is returned and the [Software](#) objects satisfying the requirements will replace these with the equality operator (`Software::operator==`) used as the comparator for the new requirements.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single [Software](#) object and it will replace all these requirements.

#### Parameters:

*swList* is a list of [Software](#) objects used to satisfy requirements.

#### Returns:

`true` if requirements are satisfied, otherwise `false`.

#### See also:

`selectSoftware(const Software&),`  
`selectSoftware(const std::list<ApplicationEnvironment>&),`  
`isSatisfied(const std::list<Software>&) const,`  
`isResolved() const.`

### 5.158.3.14 `bool Arc::SoftwareRequirement::selectSoftware (const Software & sw)` `[inline]`

Select software.

If the passed [Software](#) *sw* do not satisfy the requirements `false` is returned and this object is not modified. If however the [Software](#) object *sw* do satisfy the requirements `true` is returned and the requirements are set to equal the *sw* [Software](#) object.

**Parameters:**

`sw` is the [Software](#) object used to satisfy requirements.

**Returns:**

`true` if requirements are satisfied, otherwise `false`.

**See also:**

[selectSoftware\(const std::list<Software>&\),](#)  
[selectSoftware\(const std::list<ApplicationEnvironment>&\),](#)  
[isSatisfied\(const Software&\) const,](#)  
[isResolved\(\) const.](#)

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

- [Software.h](#)

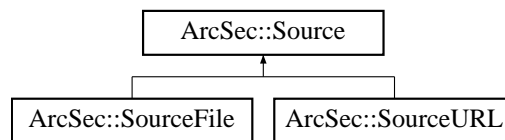


## 5.159 ArcSec::Source Class Reference

Acquires and parses XML document from specified source.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::Source::



### Public Member Functions

- [Source](#) (const [Source](#) &s)
- [Source](#) ([Arc::XMLNode](#) xml)
- [Source](#) (std::istream &stream)
- [Source](#) ([Arc::URL](#) &url)
- [Source](#) (const std::string &str)
- [Arc::XMLNode Get](#) (void) const
- [operator bool](#) (void)

### 5.159.1 Detailed Description

Acquires and parses XML document from specified source.

This class is to be used to provide easy way to specify different sources for XML Authorization Policies and Requests.

### 5.159.2 Constructor & Destructor Documentation

#### 5.159.2.1 ArcSec::Source::Source (const [Source](#) & s) [inline]

Copy constructor.

Use this constructor only for temporary objects. Parsed XML document is still owned by copied source and hence lifetime of created object should not exceed that of copied one.

#### 5.159.2.2 ArcSec::Source::Source ([Arc::XMLNode](#) xml)

Use XML subtree referred by xml.

There is no copy of xml made. Hence lifetime of this object should not exceed that of xml.

#### 5.159.2.3 ArcSec::Source::Source (std::istream & stream)

Read XML document from stream and parse it.

#### 5.159.2.4 `ArcSec::Source::Source` ([Arc::URL](#) & *url*)

Fetch XML document from specified url and parse it.

This constructor is not implemented yet.

#### 5.159.2.5 `ArcSec::Source::Source` (`const std::string & str`)

Read XML document from string.

### 5.159.3 Member Function Documentation

#### 5.159.3.1 [Arc::XMLNode](#) `ArcSec::Source::Get` (`void`) `const` `[inline]`

Get reference to parsed document.

#### 5.159.3.2 `ArcSec::Source::operator bool` (`void`) `[inline]`

Returns true if valid document is available.

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

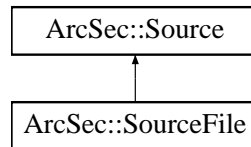
- Source.h

## 5.160 ArcSec::SourceFile Class Reference

Convenience class for obtaining XML document from file.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::SourceFile::



### Public Member Functions

- [SourceFile](#) (const [SourceFile](#) &s)
- [SourceFile](#) (const char \*name)
- [SourceFile](#) (const std::string &name)

#### 5.160.1 Detailed Description

Convenience class for obtaining XML document from file.

#### 5.160.2 Constructor & Destructor Documentation

##### 5.160.2.1 ArcSec::SourceFile::SourceFile (const [SourceFile](#) & s) [inline]

See corresponding constructor of [Source](#) class.

##### 5.160.2.2 ArcSec::SourceFile::SourceFile (const char \* name)

Read XML document from file named name and store it.

##### 5.160.2.3 ArcSec::SourceFile::SourceFile (const std::string & name)

Read XML document from file named name and store it.

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

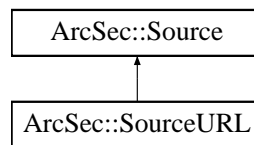
- Source.h

## 5.161 ArcSec::SourceURL Class Reference

Convenience class for obtaining XML document from remote URL.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::SourceURL::



### Public Member Functions

- [SourceURL](#) (const [SourceURL](#) &s)
- [SourceURL](#) (const char \*url)
- [SourceURL](#) (const std::string &url)

#### 5.161.1 Detailed Description

Convenience class for obtaining XML document from remote URL.

#### 5.161.2 Constructor & Destructor Documentation

##### 5.161.2.1 ArcSec::SourceURL::SourceURL (const [SourceURL](#) & s) [inline]

See corresponding constructor of [Source](#) class.

##### 5.161.2.2 ArcSec::SourceURL::SourceURL (const char \* url)

Read XML document from URL url and store it.

##### 5.161.2.3 ArcSec::SourceURL::SourceURL (const std::string & url)

Read XML document from URL url and store it.

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

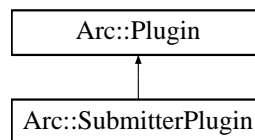
- Source.h

## 5.162 Arc::SubmitterPlugin Class Reference

Base class for the SubmitterPlugins.

```
#include <SubmitterPlugin.h>
```

Inheritance diagram for Arc::SubmitterPlugin::



### Public Member Functions

- virtual bool [Submit](#) (const std::list< [JobDescription](#) > &jobdesc, const [ExecutionTarget](#) &et, EntityConsumer< [Job](#) > &jc, std::list< const [JobDescription](#) \* > &notSubmitted)=0
- virtual bool [Migrate](#) (const [URL](#) &jobid, const [JobDescription](#) &jobdesc, const [ExecutionTarget](#) &et, bool forcemigration, [Job](#) &job)

### 5.162.1 Detailed Description

Base class for the SubmitterPlugins.

[SubmitterPlugin](#) is the base class for Grid middleware specialized [SubmitterPlugin](#) objects. The class submits job(s) to the computing resource it represents and uploads (needed by the job) local input files.

### 5.162.2 Member Function Documentation

- 5.162.2.1** virtual bool Arc::SubmitterPlugin::Migrate (const [URL](#) &jobid, const [JobDescription](#) &jobdesc, const [ExecutionTarget](#) &et, bool forcemigration, [Job](#) &job) [virtual]

Migrate job.

This virtual method should be overridden by plugins which should be capable of migrating jobs. The active job which should be migrated is pointed to by the [URL](#) jobid, and is represented by the [JobDescription](#) jobdesc. The forcemigration boolean specifies if the migration should succeed if the active job cannot be terminated. The protected method AddJob can be used to save job information. This method should return the [URL](#) of the migrated job. In case migration fails an empty [URL](#) should be returned.

- 5.162.2.2** virtual bool Arc::SubmitterPlugin::Submit (const std::list< [JobDescription](#) > &jobdesc, const [ExecutionTarget](#) &et, EntityConsumer< [Job](#) > &jc, std::list< const [JobDescription](#) \* > &notSubmitted) [pure virtual]

Submit job.

This virtual method should be overridden by plugins which should be capable of submitting jobs, defined in the [JobDescription](#) jobdesc, to the [ExecutionTarget](#) et. The protected convenience method AddJob can be used to save job information. This method should return the [URL](#) of the submitted job. In case submission fails an empty [URL](#) should be returned.

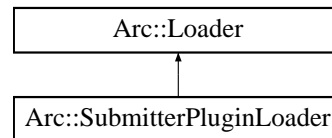
The documentation for this class was generated from the following file:

- SubmitterPlugin.h

## 5.163 Arc::SubmitterPluginLoader Class Reference

```
#include <SubmitterPlugin.h>
```

Inheritance diagram for Arc::SubmitterPluginLoader::



### Public Member Functions

- [SubmitterPluginLoader](#) ()
- [~SubmitterPluginLoader](#) ()
- [SubmitterPlugin](#) \* [load](#) (const std::string &name, const [UserConfig](#) &usercfg)

### 5.163.1 Detailed Description

Class responsible for loading [SubmitterPlugin](#) plugins The [SubmitterPlugin](#) objects returned by a [SubmitterPluginLoader](#) must not be used after the [SubmitterPluginLoader](#) goes out of scope.

### 5.163.2 Constructor & Destructor Documentation

#### 5.163.2.1 Arc::SubmitterPluginLoader::SubmitterPluginLoader ()

Constructor Creates a new [SubmitterPluginLoader](#).

#### 5.163.2.2 Arc::SubmitterPluginLoader::~~SubmitterPluginLoader ()

Destructor Calling the destructor destroys all [SubmitterPlugins](#) loaded by the [SubmitterPluginLoader](#) instance.

### 5.163.3 Member Function Documentation

#### 5.163.3.1 [SubmitterPlugin](#)\* Arc::SubmitterPluginLoader::load (const std::string & name, const [UserConfig](#) & usercfg)

Load a new [SubmitterPlugin](#)

##### Parameters:

*name* The name of the [SubmitterPlugin](#) to load.

*usercfg* The [UserConfig](#) object for the new [SubmitterPlugin](#).

##### Returns:

A pointer to the new [SubmitterPlugin](#) (NULL on error).

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

- SubmitterPlugin.h



## 5.164 Arc::ThreadDataItem Class Reference

Base class for per-thread object.

```
#include <Thread.h>
```

### Public Member Functions

- [ThreadDataItem](#) (void)
- [ThreadDataItem](#) (std::string &key)
- [ThreadDataItem](#) (const std::string &key)
- void [Attach](#) (std::string &key)
- void [Attach](#) (const std::string &key)
- virtual void [Dup](#) (void)

### Static Public Member Functions

- static [ThreadDataItem](#) \* [Get](#) (const std::string &key)

#### 5.164.1 Detailed Description

Base class for per-thread object.

Classes inherited from this one are attached to current thread under specified key and destroyed only when thread ends or object is replaced by another one with same key.

#### 5.164.2 Constructor & Destructor Documentation

##### 5.164.2.1 Arc::ThreadDataItem::ThreadDataItem (void)

Dummy constructor which does nothing. To make object usable one of Attach(...) methods must be used.

##### 5.164.2.2 Arc::ThreadDataItem::ThreadDataItem (std::string & key)

Creates instance and attaches it to current thread under key. If supplied key is empty random one is generated and stored in key variable.

##### 5.164.2.3 Arc::ThreadDataItem::ThreadDataItem (const std::string & key)

Creates instance and attaches it to current thread under key.

#### 5.164.3 Member Function Documentation

##### 5.164.3.1 void Arc::ThreadDataItem::Attach (const std::string & key)

Attaches object to current thread under key. This method must be used only if object was created using dummy constructor.

**5.164.3.2 void Arc::ThreadDataItem::Attach (std::string & *key*)**

Attaches object to current thread under key. If supplied key is empty random one is generated and stored in key variable. This method must be used only if object was created using dummy constructor.

**5.164.3.3 virtual void Arc::ThreadDataItem::Dup (void) [virtual]**

Creates copy of object. This method is called when new thread is created from current thread. It is called in new thread, so new object - if created - gets attached to new thread. If object is not meant to be inherited by new threads then this method should do nothing.

**5.164.3.4 static ThreadDataItem\* Arc::ThreadDataItem::Get (const std::string & *key*)**  
[static]

Retrieves object attached to thread under key. Returns NULL if no such object.

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

- Thread.h

## 5.165 Arc::ThreadedPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction and mutiple references.

```
#include <Thread.h>
```

### Public Member Functions

- T & [operator \\*](#) (void) const
- T \* [operator →](#) (void) const
- [operator bool](#) (void) const
- bool [operator!](#) (void) const
- bool [operator==](#) (const [ThreadedPointer](#) &p) const
- bool [operator!=](#) (const [ThreadedPointer](#) &p) const
- bool [operator<](#) (const [ThreadedPointer](#) &p) const
- T \* [Ptr](#) (void) const
- T \* [Release](#) (void)
- unsigned int [Holders](#) (void)
- unsigned int [WaitOutRange](#) (unsigned int minThr, unsigned int maxThr)
- unsigned int [WaitOutRange](#) (unsigned int minThr, unsigned int maxThr, int timeout)
- unsigned int [WaitInRange](#) (unsigned int minThr, unsigned int maxThr)
- unsigned int [WaitInRange](#) (unsigned int minThr, unsigned int maxThr, int timeout)

### 5.165.1 Detailed Description

```
template<typename T> class Arc::ThreadedPointer< T >
```

Wrapper for pointer with automatic destruction and mutiple references.

See for [CountedPointer](#) for description. Differently from [CountedPointer](#) this class provides thread safe destruction of refered object. But the instance of [ThreadedPointer](#) itself is not thread safe. Hence it is advisable to use different instances in different threads.

### 5.165.2 Member Function Documentation

**5.165.2.1** `template<typename T> unsigned int Arc::ThreadedPointer< T >::Holders (void)`  
[inline]

Returns number of [ThreadedPointer](#) instances refering to underlying object.

**5.165.2.2** `template<typename T> T& Arc::ThreadedPointer< T >::operator * (void) const`  
[inline]

For refering wrapped object.

**5.165.2.3** `template<typename T> Arc::ThreadedPointer< T >::operator bool (void) const`  
[inline]

Returns false if pointer is NULL and true otherwise.

**5.165.2.4** `template<typename T> bool Arc::ThreadedPointer< T >::operator! (void) const`  
`[inline]`

Returns true if pointer is NULL and false otherwise.

**5.165.2.5** `template<typename T> bool Arc::ThreadedPointer< T >::operator!= (const`  
`ThreadedPointer< T > & p) const` `[inline]`

Returns true if pointers are not equal.

**5.165.2.6** `template<typename T> T* Arc::ThreadedPointer< T >::operator → (void) const`  
`[inline]`

For refering wrapped object.

**5.165.2.7** `template<typename T> bool Arc::ThreadedPointer< T >::operator< (const`  
`ThreadedPointer< T > & p) const` `[inline]`

Comparison operator.

**5.165.2.8** `template<typename T> bool Arc::ThreadedPointer< T >::operator== (const`  
`ThreadedPointer< T > & p) const` `[inline]`

Returns true if pointers are equal.

**5.165.2.9** `template<typename T> T* Arc::ThreadedPointer< T >::Ptr (void) const` `[inline]`

Cast to original pointer.

**5.165.2.10** `template<typename T> T* Arc::ThreadedPointer< T >::Release (void)` `[inline]`

Release refered object so that it can be passed to other container.

After [Release\(\)](#) is called refered object is will not be destroyed automatically anymore.

**5.165.2.11** `template<typename T> unsigned int Arc::ThreadedPointer< T >::WaitInRange`  
`(unsigned int minThr, unsigned int maxThr, int timeout)` `[inline]`

Waits till number of [ThreadedPointer](#) instances  $\geq$  minThr and  $\leq$  maxThr.

Waits no longer than timeout milliseconds. If timeout is negative - wait forever. Returns current number of instances.

**5.165.2.12** `template<typename T> unsigned int Arc::ThreadedPointer< T >::WaitInRange`  
`(unsigned int minThr, unsigned int maxThr)` `[inline]`

Waits till number of [ThreadedPointer](#) instances  $\geq$  minThr and  $\leq$  maxThr.

**5.165.2.13** `template<typename T> unsigned int Arc::ThreadedPointer< T >::WaitOutOfRange  
(unsigned int minThr, unsigned int maxThr, int timeout)` `[inline]`

Waits till number of [ThreadedPointer](#) instances  $\leq$  minThr or  $\geq$  maxThr.

Waits no longer than timeout milliseconds. If timeout is negative - wait forever. Returns current number of instances.

**5.165.2.14** `template<typename T> unsigned int Arc::ThreadedPointer< T >::WaitOutOfRange  
(unsigned int minThr, unsigned int maxThr)` `[inline]`

Waits till number of [ThreadedPointer](#) instances  $\leq$  minThr or  $\geq$  maxThr.

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

- Thread.h

## 5.166 Arc::ThreadedPointerBase Class Reference

Helper class for [ThreadedPointer](#).

```
#include <Thread.h>
```

### 5.166.1 Detailed Description

Helper class for [ThreadedPointer](#).

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

- Thread.h

## 5.167 Arc::ThreadRegistry Class Reference

```
#include <Thread.h>
```

### Public Member Functions

- void [RegisterThread](#) (void)
- void [UnregisterThread](#) (void)
- bool [WaitOrCancel](#) (int timeout)
- bool [WaitForExit](#) (int timeout=-1)

### 5.167.1 Detailed Description

This class is a set of conditions, mutexes, etc. conveniently exposed to monitor running child threads and to wait till they exit. There are no protections against race conditions. So use it carefully.

### 5.167.2 Member Function Documentation

#### 5.167.2.1 void Arc::ThreadRegistry::RegisterThread (void)

Register thread as started/starting into this instance.

#### 5.167.2.2 void Arc::ThreadRegistry::UnregisterThread (void)

Report thread as exited.

#### 5.167.2.3 bool Arc::ThreadRegistry::WaitForExit (int *timeout* = -1)

Wait for registered threads to exit. Leave after timeout milliseconds if failed. Returns true if all registered threads reported their exit.

#### 5.167.2.4 bool Arc::ThreadRegistry::WaitOrCancel (int *timeout*)

Wait for timeout milliseconds or cancel request. Returns true if cancel request received.

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

- Thread.h

## 5.168 Arc::Time Class Reference

A class for storing and manipulating times.

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

### Public Member Functions

- [Time](#) ()
- [Time](#) (time\_t)
- [Time](#) (time\_t time, uint32\_t nanosec)
- [Time](#) (const std::string &)
- [Time](#) & [operator=](#) (time\_t)
- [Time](#) & [operator=](#) (const [Time](#) &)
- [Time](#) & [operator=](#) (const char \*)
- [Time](#) & [operator=](#) (const std::string &)
- void [SetTime](#) (time\_t)
- void [SetTime](#) (time\_t time, uint32\_t nanosec)
- time\_t [GetTime](#) () const
- [operator std::string](#) () const
- std::string [str](#) (const [TimeFormat](#) &=time\_format) const
- bool [operator<](#) (const [Time](#) &) const
- bool [operator>](#) (const [Time](#) &) const
- bool [operator<=](#) (const [Time](#) &) const
- bool [operator>=](#) (const [Time](#) &) const
- bool [operator==](#) (const [Time](#) &) const
- bool [operator!=](#) (const [Time](#) &) const
- [Time](#) [operator+](#) (const Period &) const
- [Time](#) [operator-](#) (const Period &) const
- Period [operator-](#) (const [Time](#) &) const

### Static Public Member Functions

- static void [SetFormat](#) (const [TimeFormat](#) &)
- static [TimeFormat](#) [GetFormat](#) ()

#### 5.168.1 Detailed Description

A class for storing and manipulating times.

#### 5.168.2 Constructor & Destructor Documentation

##### 5.168.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

##### 5.168.2.2 Arc::Time::Time (time\_t)

Constructor that takes a time\_t variable and stores it.



### 5.168.2.3 Arc::Time::Time (time\_t *time*, uint32\_t *nanosec*)

Constructor that takes a fine grained time variables and stores them.

### 5.168.2.4 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time\_t.

## 5.168.3 Member Function Documentation

### 5.168.3.1 static TimeFormat Arc::Time::GetFormat () [static]

Gets the default format for time strings.

### 5.168.3.2 time\_t Arc::Time::GetTime () const

gets the time

### 5.168.3.3 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

### 5.168.3.4 bool Arc::Time::operator!= (const Time &) const

Comparing two Time objects.

### 5.168.3.5 Time Arc::Time::operator+ (const Period &) const

Adding Time object with Period object.

### 5.168.3.6 Period Arc::Time::operator- (const Time &) const

Subtracting Time object from the other Time object.

### 5.168.3.7 Time Arc::Time::operator- (const Period &) const

Subtracting Period object from Time object.

### 5.168.3.8 bool Arc::Time::operator< (const Time &) const

Comparing two Time objects.

### 5.168.3.9 bool Arc::Time::operator<= (const Time &) const

Comparing two Time objects.

**5.168.3.10** `Time& Arc::Time::operator= (const std::string &)`

Assignment operator from a string.

**5.168.3.11** `Time& Arc::Time::operator= (const char *)`

Assignment operator from a char pointer.

**5.168.3.12** `Time& Arc::Time::operator= (const Time &)`

Assignment operator from a `Time`.

**5.168.3.13** `Time& Arc::Time::operator= (time_t)`

Assignment operator from a `time_t`.

**5.168.3.14** `bool Arc::Time::operator== (const Time &) const`

Comparing two `Time` objects.

**5.168.3.15** `bool Arc::Time::operator> (const Time &) const`

Comparing two `Time` objects.

**5.168.3.16** `bool Arc::Time::operator>= (const Time &) const`

Comparing two `Time` objects.

**5.168.3.17** `static void Arc::Time::SetFormat (const TimeFormat &) [static]`

Sets the default format for time strings.

**5.168.3.18** `void Arc::Time::SetTime (time_t time, uint32_t nanosec)`

sets the fine grained time

**5.168.3.19** `void Arc::Time::SetTime (time_t)`

sets the time

**5.168.3.20** `std::string Arc::Time::str (const TimeFormat & = time_format) const`

Returns a string representation of the time, using the specified format.

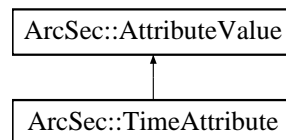
The documentation for this class was generated from the following file:

- `DateTime.h`

## 5.169 ArcSec::TimeAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::TimeAttribute::



### Public Member Functions

- virtual bool [equal](#) ([AttributeValue](#) \*other, bool check\_id=true)
- virtual std::string [encode](#) ()
- virtual std::string [getType](#) ()
- virtual std::string [getId](#) ()

#### 5.169.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

#### 5.169.2 Member Function Documentation

##### 5.169.2.1 virtual std::string ArcSec::TimeAttribute::encode () [virtual]

encode the value in a string format

Implements [ArcSec::AttributeValue](#).

##### 5.169.2.2 virtual bool ArcSec::TimeAttribute::equal ([AttributeValue](#) \* other, bool check\_id = true) [virtual]

Evaluate whether "this" equale to the parameter value

Implements [ArcSec::AttributeValue](#).

##### 5.169.2.3 virtual std::string ArcSec::TimeAttribute::getId () [inline, virtual]

Get the AttributeId of the <Attribute>

Implements [ArcSec::AttributeValue](#).

##### 5.169.2.4 virtual std::string ArcSec::TimeAttribute::getType () [inline, virtual]

Get the DataType of the <Attribute>

Implements [ArcSec::AttributeValue](#).

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

- `DateTimeAttribute.h`

## 5.170 DataStaging::TransferParameters Class Reference

```
#include <DTR.h>
```

### Public Member Functions

- [TransferParameters \(\)](#)

### Data Fields

- unsigned long long int [min\\_average\\_bandwidth](#)
- unsigned int [max\\_inactivity\\_time](#)
- unsigned long long int [min\\_current\\_bandwidth](#)
- unsigned int [averaging\\_time](#)

#### 5.170.1 Detailed Description

Represents limits and properties of a [DTR](#) transfer. These generally apply to all DTRs.

#### 5.170.2 Constructor & Destructor Documentation

##### 5.170.2.1 DataStaging::TransferParameters::TransferParameters () [inline]

Constructor. Initialises all values to zero.

#### 5.170.3 Field Documentation

##### 5.170.3.1 unsigned int DataStaging::TransferParameters::averaging\_time

The time over which to average the calculation of min\_curr\_bandwidth.

##### 5.170.3.2 unsigned int DataStaging::TransferParameters::max\_inactivity\_time

Maximum inactivity time in sec - if transfer stops for longer than this time it should be killed

##### 5.170.3.3 unsigned long long int DataStaging::TransferParameters::min\_average\_bandwidth

Minimum average bandwidth in bytes/sec - if the average bandwidth used drops below this level the transfer should be killed

##### 5.170.3.4 unsigned long long int DataStaging::TransferParameters::min\_current\_bandwidth

Minimum current bandwidth - if bandwidth averaged over averaging\_time is less than minimum the transfer should be killed (allows transfers which slow down to be killed quicker)

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

- [DTR.h](#)

## 5.171 DataStaging::TransferShares Class Reference

[TransferShares](#) is used to implement fair-sharing and priorities.

```
#include <TransferShares.h>
```

### Public Member Functions

- [TransferShares](#) ()
- [TransferShares](#) (const [TransferSharesConf](#) &shares\_conf)
- [~TransferShares](#) ()
- [TransferShares](#) (const [TransferShares](#) &shares)
- [TransferShares](#) & operator= (const [TransferShares](#) &shares)
- void [set\\_shares\\_conf](#) (const [TransferSharesConf](#) &share\_conf)
- void [calculate\\_shares](#) (int TotalNumberOfSlots)
- void [increase\\_transfer\\_share](#) (const std::string &ShareToIncrease)
- void [decrease\\_transfer\\_share](#) (const std::string &ShareToDecrease)
- void [decrease\\_number\\_of\\_slots](#) (const std::string &ShareToDecrease)
- bool [can\\_start](#) (const std::string &ShareToStart)
- std::map< std::string, int > [active\\_shares](#) () const

### 5.171.1 Detailed Description

[TransferShares](#) is used to implement fair-sharing and priorities.

[TransferShares](#) defines the algorithm used to prioritise and share transfers among different users or groups. Configuration information on the share type and reference shares is held in a [TransferSharesConf](#) instance. The [Scheduler](#) uses [TransferShares](#) to determine which DTRs in the queue for each process go first. The calculation is based on the configuration and the currently active shares (the DTRs already in the process). [can\\_start\(\)](#) is the method called by the [Scheduler](#) to determine whether a particular share has an available slot in the process.

### 5.171.2 Constructor & Destructor Documentation

#### 5.171.2.1 DataStaging::TransferShares::TransferShares () [inline]

Create a new [TransferShares](#) with default configuration.

#### 5.171.2.2 DataStaging::TransferShares::TransferShares (const [TransferSharesConf](#) & shares\_conf)

Create a new [TransferShares](#) with given configuration.

#### 5.171.2.3 DataStaging::TransferShares::~~TransferShares () [inline]

Empty destructor.

#### 5.171.2.4 DataStaging::TransferShares::TransferShares (const [TransferShares](#) & shares)

Copy constructor must be defined because SimpleCondition cannot be copied.

### 5.171.3 Member Function Documentation

#### 5.171.3.1 `std::map<std::string, int> DataStaging::TransferShares::active_shares () const`

Returns the map of active shares.

#### 5.171.3.2 `void DataStaging::TransferShares::calculate_shares (int TotalNumberOfSlots)`

Calculate how many slots to assign to each active share.

This method is called each time the [Scheduler](#) loops to calculate the number of slots to assign to each share, based on the current number of active shares and the shares' relative priorities.

#### 5.171.3.3 `bool DataStaging::TransferShares::can_start (const std::string & ShareToStart)`

Returns true if there is a slot available for the given share.

#### 5.171.3.4 `void DataStaging::TransferShares::decrease_number_of_slots (const std::string & ShareToDecrease)`

Decrease by one the number of slots available to the given share.

Called when there is a slot already used by this share to reduce the number available.

#### 5.171.3.5 `void DataStaging::TransferShares::decrease_transfer_share (const std::string & ShareToDecrease)`

Decrease by one the active count for the given share.

Called when a completed [DTR](#) leaves the queue.

#### 5.171.3.6 `void DataStaging::TransferShares::increase_transfer_share (const std::string & ShareToIncrease)`

Increase by one the active count for the given share.

Called when a new [DTR](#) enters the queue.

#### 5.171.3.7 `TransferShares& DataStaging::TransferShares::operator= (const TransferShares & shares)`

Assignment operator must be defined because SimpleCondition cannot be copied.

#### 5.171.3.8 `void DataStaging::TransferShares::set_shares_conf (const TransferSharesConf & share_conf)`

Set a new configuration, if a new reference share gets added for example.

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

- TransferShares.h

## 5.172 DataStaging::TransferSharesConf Class Reference

[TransferSharesConf](#) describes the configuration of [TransferShares](#).

```
#include <TransferShares.h>
```

### Public Types

- [USER](#)
- [VO](#)
- [GROUP](#)
- [ROLE](#)
- [NONE](#)
- enum [ShareType](#) {  
    [USER](#), [VO](#), [GROUP](#), [ROLE](#),  
    [NONE](#) }

### Public Member Functions

- [TransferSharesConf](#) (const std::string &type, const std::map< std::string, int > &ref\_shares)
- [TransferSharesConf](#) ()
- void [set\\_share\\_type](#) (const std::string &type)
- void [set\\_reference\\_share](#) (const std::string &RefShare, int Priority)
- void [set\\_reference\\_shares](#) (const std::map< std::string, int > &shares)
- bool [is\\_configured](#) (const std::string &ShareToCheck)
- int [get\\_basic\\_priority](#) (const std::string &ShareToCheck)
- std::string [conf](#) () const
- std::string [extract\\_share\\_info](#) ([DTR\\_ptr](#) DTRToExtract)

### 5.172.1 Detailed Description

[TransferSharesConf](#) describes the configuration of [TransferShares](#).

It allows reference shares to be defined with certain priorities. An instance of this class is used when creating a [TransferShares](#) object.

### 5.172.2 Member Enumeration Documentation

#### 5.172.2.1 enum [DataStaging::TransferSharesConf::ShareType](#)

The criterion for assigning a share to a [DTR](#).

#### Enumerator:

- USER** Shares are defined per DN of the user's proxy.
- VO** Shares are defined per VOMS VO of the user's proxy.
- GROUP** Shares are defined per VOMS group of the user's proxy.
- ROLE** Shares are defined per VOMS role of the user's proxy.
- NONE** No share criterion - all DTRs will be assigned to a single share.



### 5.172.3 Constructor & Destructor Documentation

#### 5.172.3.1 DataStaging::TransferSharesConf::TransferSharesConf (const std::string & *type*, const std::map< std::string, int > & *ref\_shares*)

Construct a new [TransferSharesConf](#) with given share type and reference shares.

#### 5.172.3.2 DataStaging::TransferSharesConf::TransferSharesConf ()

Construct a new [TransferSharesConf](#) with no defined shares or policy.

### 5.172.4 Member Function Documentation

#### 5.172.4.1 std::string DataStaging::TransferSharesConf::conf () const

Return human-readable configuration of shares.

#### 5.172.4.2 std::string DataStaging::TransferSharesConf::extract\_share\_info ([DTR\\_ptr](#) *DTRToExtract*)

Get the name of the share the [DTR](#) should be assigned to and the proxy type.

#### 5.172.4.3 int DataStaging::TransferSharesConf::get\_basic\_priority (const std::string & *ShareToCheck*)

Get the priority of this share.

#### 5.172.4.4 bool DataStaging::TransferSharesConf::is\_configured (const std::string & *ShareToCheck*)

Returns true if the given share is a reference share.

#### 5.172.4.5 void DataStaging::TransferSharesConf::set\_reference\_share (const std::string & *RefShare*, int *Priority*)

Add a reference share.

#### 5.172.4.6 void DataStaging::TransferSharesConf::set\_reference\_shares (const std::map< std::string, int > & *shares*)

Set reference shares.

#### 5.172.4.7 void DataStaging::TransferSharesConf::set\_share\_type (const std::string & *type*)

Set the share type.

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

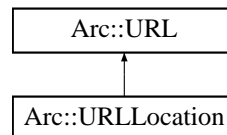
- TransferShares.h

## 5.173 Arc::URL Class Reference

Class to hold general URLs.

```
#include <URL.h>
```

Inheritance diagram for Arc::URL::



### Public Types

- enum [Scope](#)

### Public Member Functions

- [URL](#) ()
- [URL](#) (const std::string &url)
- virtual [~URL](#) ()
- const std::string & [Protocol](#) () const
- void [ChangeProtocol](#) (const std::string &newprot)
- bool [IsSecureProtocol](#) () const
- const std::string & [Username](#) () const
- const std::string & [Passwd](#) () const
- const std::string & [Host](#) () const
- void [ChangeHost](#) (const std::string &newhost)
- int [Port](#) () const
- void [ChangePort](#) (int newport)
- const std::string & [Path](#) () const
- std::string [FullPath](#) () const
- std::string [FullPathURIEncoded](#) () const
- void [ChangePath](#) (const std::string &newpath)
- void [ChangeFullPath](#) (const std::string &newpath)
- const std::map< std::string, std::string > & [HTTPOptions](#) () const
- const std::string & [HTTPOption](#) (const std::string &option, const std::string &undefined="") const
- bool [AddHTTPOption](#) (const std::string &option, const std::string &value, bool overwrite=true)
- void [RemoveHTTPOption](#) (const std::string &option)
- const std::list< std::string > & [LDAPAttributes](#) () const
- void [AddLDAPAttribute](#) (const std::string &attribute)
- [Scope](#) [LDAPScope](#) () const
- void [ChangeLDAPScope](#) (const [Scope](#) newscope)
- const std::string & [LDAPFilter](#) () const
- void [ChangeLDAPFilter](#) (const std::string &newfilter)
- const std::map< std::string, std::string > & [Options](#) () const
- const std::string & [Option](#) (const std::string &option, const std::string &undefined="") const
- const std::map< std::string, std::string > & [MetaDataOptions](#) () const

- const std::string & [MetaDataOption](#) (const std::string &option, const std::string &undefined="") const
- bool [AddOption](#) (const std::string &option, const std::string &value, bool overwrite=true)
- bool [AddOption](#) (const std::string &option, bool overwrite=true)
- void [AddMetaDataOption](#) (const std::string &option, const std::string &value, bool overwrite=true)
- void [AddLocation](#) (const [URLLocation](#) &location)
- const std::list< [URLLocation](#) > & [Locations](#) () const
- const std::map< std::string, std::string > & [CommonLocOptions](#) () const
- const std::string & [CommonLocOption](#) (const std::string &option, const std::string &undefined="") const
- void [RemoveOption](#) (const std::string &option)
- void [RemoveMetaDataOption](#) (const std::string &option)
- virtual std::string [str](#) () const
- virtual std::string [plainstr](#) () const
- virtual std::string [fullstr](#) () const
- virtual std::string [ConnectionURL](#) () const
- bool [operator<](#) (const [URL](#) &url) const
- bool [operator==](#) (const [URL](#) &url) const
- [operator bool](#) () const
- bool [StringMatches](#) (const std::string &str) const
- std::map< std::string, std::string > [ParseOptions](#) (const std::string &optstring, char separator)

## Static Public Member Functions

- static std::string [OptionString](#) (const std::map< std::string, std::string > &options, char separator)

## Protected Member Functions

- void [ParsePath](#) (void)

## Static Protected Member Functions

- static std::string [BaseDN2Path](#) (const std::string &)
- static std::string [Path2BaseDN](#) (const std::string &)

## Protected Attributes

- std::string [protocol](#)
- std::string [username](#)
- std::string [passwd](#)
- std::string [host](#)
- bool [ip6addr](#)
- int [port](#)
- std::string [path](#)
- std::map< std::string, std::string > [httpoptions](#)
- std::map< std::string, std::string > [metadataoptions](#)
- std::list< std::string > [ldapattributes](#)
- [Scope](#) [ldapscope](#)
- std::string [ldapfilter](#)

- `std::map< std::string, std::string >` [urloptions](#)
- `std::list< URLLocation >` [locations](#)
- `std::map< std::string, std::string >` [commonlocoptions](#)
- `bool` [valid](#)

## Friends

- `std::ostream &` [operator<<](#) (`std::ostream &out`, `const URL &u`)

### 5.173.1 Detailed Description

Class to hold general URLs.

The [URL](#) is split into protocol, hostname, port and path. This class tries to follow RFC 3986 for splitting URLs, at least for protocol + host part. It also accepts local file paths which are converted to `file:path`. The usual system dependent file paths are supported. Relative paths are converted to absolute paths by prepending them with current working directory path. A file path can't start from `#` symbol. If the string representation of [URL](#) starts from `'@'` then it is treated as path to a file containing a list of URLs.

A [URL](#) is parsed in the following way:

```
[protocol:][/[username:passwd@][host][:port]][;urloptions[;...]][/path[?httpoption[&...]][;metadataoption[;...]]]
```

The 'protocol' and 'host' parts are treated as case-insensitive and to avoid confusion are converted to lowercase in constructor. Note that 'path' is always converted to absolute path in constructor. The meaning of 'absolute' may depend upon [URL](#) type. For generic [URL](#) and local POSIX file paths that means path starts from `/` like

```
/path/to/file
```

For Windows paths absolute path may look like

```
C:\path\to\file
```

It is important to note that path still can be empty. For referencing local file using absolute path on POSIX filesystem one may use either

```
file:///path/to/file or file:/path/to/file
```

Relative path will look like

```
file:to/file
```

For local Windows files possible URLs are

```
file:C:\path\to\file or file:to\file
```

URLs representing LDAP resources have different structure of options following 'path' part:

```
ldap://host[:port][;urloptions[;...]][/path[?attributes[?scope[?filter]]]]
```

For LDAP URLs paths are converted from `/key1=value1/.../keyN=valueN` notation to `keyN=valueN,...,key1=value1` and hence path does not contain leading `/`. If LDAP [URL](#) initially had path in second notation leading `/` is treated as separator only and is stripped.

URLs of indexing services optionally may have locations specified before 'host' part

```
protocol://[location[:location[;...]]@][host][:port]...
```

The structure of 'location' element is protocol specific.

## 5.173.2 Member Enumeration Documentation

### 5.173.2.1 enum [Arc::URL::Scope](#)

Scope for LDAP URLs

## 5.173.3 Constructor & Destructor Documentation

### 5.173.3.1 [Arc::URL::URL](#) ()

Empty constructor. Necessary when the class is part of another class and the like.

### 5.173.3.2 [Arc::URL::URL](#) (const std::string & *url*)

Constructs a new [URL](#) from a string representation.

### 5.173.3.3 virtual [Arc::URL::~~URL](#) () [virtual]

[URL](#) Destructor

## 5.173.4 Member Function Documentation

### 5.173.4.1 bool [Arc::URL::AddHTTPOption](#) (const std::string & *option*, const std::string & *value*, bool *overwrite* = true)

Adds a HTTP option with the given value. Returns false if overwrite is false and option already exists, true otherwise.

### 5.173.4.2 void [Arc::URL::AddLDAPAttribute](#) (const std::string & *attribute*)

Adds an LDAP attribute.

### 5.173.4.3 void [Arc::URL::AddLocation](#) (const [URLLocation](#) & *location*)

Adds a Location

### 5.173.4.4 void [Arc::URL::AddMetaDataOption](#) (const std::string & *option*, const std::string & *value*, bool *overwrite* = true)

Adds a metadata option

### 5.173.4.5 bool [Arc::URL::AddOption](#) (const std::string & *option*, bool *overwrite* = true)

Adds a [URL](#) option where option has the format "name=value". Returns false if overwrite is true and option already exists or if option does not have the correct format. Returns true otherwise.

**5.173.4.6** `bool Arc::URL::AddOption (const std::string & option, const std::string & value, bool overwrite = true)`

Adds a [URL](#) option with the given value. Returns false if overwrite is false and option already exists, true otherwise. Note that some compilers may interpret AddOption("name", "value") as a call to AddOption(string, bool) so it is recommended to use explicit string types when calling this method.

**5.173.4.7** `static std::string Arc::URL::BaseDN2Path (const std::string &) [static, protected]`

a private method that converts an ldap basedn to a path.

**5.173.4.8** `void Arc::URL::ChangeFullPath (const std::string & newpath)`

Changes the path of the [URL](#) and all options attached.

**5.173.4.9** `void Arc::URL::ChangeHost (const std::string & newhost)`

Changes the hostname of the [URL](#).

**5.173.4.10** `void Arc::URL::ChangeLDAPFilter (const std::string & newfilter)`

Changes the LDAP filter.

**5.173.4.11** `void Arc::URL::ChangeLDAPScope (const Scope newscope)`

Changes the LDAP scope.

**5.173.4.12** `void Arc::URL::ChangePath (const std::string & newpath)`

Changes the path of the [URL](#).

**5.173.4.13** `void Arc::URL::ChangePort (int newport)`

Changes the port of the [URL](#).

**5.173.4.14** `void Arc::URL::ChangeProtocol (const std::string & newprot)`

Changes the protocol of the [URL](#).

**5.173.4.15** `const std::string& Arc::URL::CommonLocOption (const std::string & option, const std::string & undefined = "") const`

Returns the value of a common location option.

#### Parameters:

*option* The option whose value is returned.

*undefined* This value is returned if the common location option is not defined.

**5.173.4.16** `const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const`

Returns the common location options if any.

**5.173.4.17** `virtual std::string Arc::URL::ConnectionURL () const` [virtual]

Returns a string representation with protocol, host and port only

**5.173.4.18** `std::string Arc::URL::FullPath () const`

Returns the path of the [URL](#) with all options attached.

**5.173.4.19** `std::string Arc::URL::FullPathURIEncoded () const`

Returns the path and all options, URI-encoded according to RFC 3986. Forward slashes (‘/’) in the path are not encoded but are encoded in the options.

**5.173.4.20** `virtual std::string Arc::URL::fullstr () const` [virtual]

Returns a string representation including options and locations

Reimplemented in [Arc::URLLocation](#).

**5.173.4.21** `const std::string& Arc::URL::Host () const`

Returns the hostname of the [URL](#).

**5.173.4.22** `const std::string& Arc::URL::HTTPOption (const std::string & option, const std::string & undefined = "") const`

Returns the value of an HTTP option.

**Parameters:**

*option* The option whose value is returned.

*undefined* This value is returned if the HTTP option is not defined.

**5.173.4.23** `const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const`

Returns HTTP options if any.

**5.173.4.24** `bool Arc::URL::IsSecureProtocol () const`

Indicates whether the protocol is secure or not.



**5.173.4.25** `const std::list<std::string>& Arc::URL::LDAPAttributes () const`

Returns the LDAP attributes if any.

**5.173.4.26** `const std::string& Arc::URL::LDAPFilter () const`

Returns the LDAP filter.

**5.173.4.27** `Scope Arc::URL::LDAPScope () const`

Returns the LDAP scope.

**5.173.4.28** `const std::list<URLLocation>& Arc::URL::Locations () const`

Returns the locations if any.

**5.173.4.29** `const std::string& Arc::URL::MetaDataOption (const std::string & option, const std::string & undefined = "") const`

Returns the value of a metadata option.

**Parameters:**

*option* The option whose value is returned.

*undefined* This value is returned if the metadata option is not defined.

**5.173.4.30** `const std::map<std::string, std::string>& Arc::URL::MetaDataOptions () const`

Returns metadata options if any.

**5.173.4.31** `Arc::URL::operator bool () const`

Check if instance holds valid [URL](#)

**5.173.4.32** `bool Arc::URL::operator< (const URL & url) const`

Compares one [URL](#) to another

**5.173.4.33** `bool Arc::URL::operator== (const URL & url) const`

Is one [URL](#) equal to another?

**5.173.4.34** `const std::string& Arc::URL::Option (const std::string & option, const std::string & undefined = "") const`

Returns the value of a [URL](#) option.

**Parameters:**

*option* The option whose value is returned.

*undefined* This value is returned if the [URL](#) option is not defined.

**5.173.4.35** `const std::map<std::string, std::string>& Arc::URL::Options () const`

Returns [URL](#) options if any.

**5.173.4.36** `static std::string Arc::URL::OptionString (const std::map< std::string, std::string > & options, char separator) [static]`

Returns a string representation of the options given in the options map

**5.173.4.37** `std::map<std::string, std::string> Arc::URL::ParseOptions (const std::string & optstring, char separator)`

Parse a string of options separated by separator into an attribute->value map

**5.173.4.38** `void Arc::URL::ParsePath (void) [protected]`

Convenience method for splitting schema specific part into path and options

**5.173.4.39** `const std::string& Arc::URL::Passwd () const`

Returns the password of the [URL](#).

**5.173.4.40** `const std::string& Arc::URL::Path () const`

Returns the path of the [URL](#).

**5.173.4.41** `static std::string Arc::URL::Path2BaseDN (const std::string &) [static, protected]`

a private method that converts an ldap path to a basedn.

**5.173.4.42** `virtual std::string Arc::URL::plainstr () const [virtual]`

Returns a string representation of the [URL](#) without any options

**5.173.4.43** `int Arc::URL::Port () const`

Returns the port of the [URL](#).

**5.173.4.44** `const std::string& Arc::URL::Protocol () const`

Returns the protocol of the [URL](#).

**5.173.4.45 void Arc::URL::RemoveHTTPOption (const std::string & *option*)**

Removes a HTTP option if exists.

**Parameters:**

*option* The option to remove.

**5.173.4.46 void Arc::URL::RemoveMetaDataOption (const std::string & *option*)**

Remove a metadata option if exists.

**Parameters:**

*option* The option to remove.

**5.173.4.47 void Arc::URL::RemoveOption (const std::string & *option*)**

Removes a [URL](#) option if exists.

**Parameters:**

*option* The option to remove.

**5.173.4.48 virtual std::string Arc::URL::str () const** [virtual]

Returns a string representation of the [URL](#) including meta-options.

Reimplemented in [Arc::URLLocation](#).

**5.173.4.49 bool Arc::URL::StringMatches (const std::string & *str*) const**

Returns true if string matches url.

**5.173.4.50 const std::string& Arc::URL::Username () const**

Returns the username of the [URL](#).

**5.173.5 Friends And Related Function Documentation****5.173.5.1 std::ostream& operator<< (std::ostream & *out*, const [URL](#) & *u*)** [friend]

Overloaded operator << to print a [URL](#).

**5.173.6 Field Documentation****5.173.6.1 std::map<std::string, std::string> [Arc::URL::commonlocoptions](#)** [protected]

common location options for index server URLs.

**5.173.6.2** `std::string` [`Arc::URL::host`](#) [protected]

hostname of the url.

**5.173.6.3** `std::map<std::string, std::string>` [`Arc::URL::httpoptions`](#) [protected]

HTTP options of the url.

**5.173.6.4** `bool` [`Arc::URL::ip6addr`](#) [protected]

if host is IPv6 numerical address notation.

**5.173.6.5** `std::list<std::string>` [`Arc::URL::ldapattributes`](#) [protected]

LDAP attributes of the url.

**5.173.6.6** `std::string` [`Arc::URL::ldapfilter`](#) [protected]

LDAP filter of the url.

**5.173.6.7** `Scope` [`Arc::URL::ldapscope`](#) [protected]

LDAP scope of the url.

**5.173.6.8** `std::list<URLLocation>` [`Arc::URL::locations`](#) [protected]

locations for index server URLs.

**5.173.6.9** `std::map<std::string, std::string>` [`Arc::URL::metadataoptions`](#) [protected]

Meta data options

**5.173.6.10** `std::string` [`Arc::URL::passwd`](#) [protected]

password of the url.

**5.173.6.11** `std::string` [`Arc::URL::path`](#) [protected]

the url path.

**5.173.6.12** `int` [`Arc::URL::port`](#) [protected]

portnumber of the url.

**5.173.6.13** `std::string` [Arc::URL::protocol](#) [protected]

the url protocol.

**5.173.6.14** `std::map<std::string, std::string>` [Arc::URL::urloptions](#) [protected]

options of the url.

**5.173.6.15** `std::string` [Arc::URL::username](#) [protected]

username of the url.

**5.173.6.16** `bool` [Arc::URL::valid](#) [protected]

flag to describe validity of [URL](#)

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

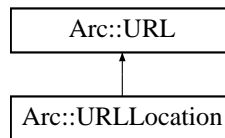
- URL.h

## 5.174 Arc::URLLocation Class Reference

Class to hold a resolved [URL](#) location.

```
#include <URL.h>
```

Inheritance diagram for Arc::URLLocation::



### Public Member Functions

- [URLLocation](#) (const std::string &url="")
- [URLLocation](#) (const std::string &url, const std::string &name)
- [URLLocation](#) (const [URL](#) &url)
- [URLLocation](#) (const [URL](#) &url, const std::string &name)
- [URLLocation](#) (const std::map< std::string, std::string > &options, const std::string &name)
- virtual ~[URLLocation](#) ()
- const std::string & [Name](#) () const
- virtual std::string [str](#) () const
- virtual std::string [fullstr](#) () const

### Protected Attributes

- std::string [name](#)

#### 5.174.1 Detailed Description

Class to hold a resolved [URL](#) location.

It is specific to file indexing service registrations.

#### 5.174.2 Constructor & Destructor Documentation

##### 5.174.2.1 Arc::URLLocation::URLLocation (const std::string & url = " ")

Creates a [URLLocation](#) from a string representaion.

##### 5.174.2.2 Arc::URLLocation::URLLocation (const std::string & url, const std::string & name)

Creates a [URLLocation](#) from a string representaion and a name.

##### 5.174.2.3 Arc::URLLocation::URLLocation (const [URL](#) & url)

Creates a [URLLocation](#) from a [URL](#).

**5.174.2.4 Arc::URLLocation::URLLocation (const [URL](#) & *url*, const std::string & *name*)**

Creates a [URLLocation](#) from a [URL](#) and a name.

**5.174.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)**

Creates a [URLLocation](#) from options and a name.

**5.174.2.6 virtual Arc::URLLocation::~~URLLocation () [virtual]**

[URLLocation](#) destructor.

**5.174.3 Member Function Documentation****5.174.3.1 virtual std::string Arc::URLLocation::fullstr () const [virtual]**

Returns a string representation including options and locations

Reimplemented from [Arc::URL](#).

**5.174.3.2 const std::string& Arc::URLLocation::Name () const**

Returns the [URLLocation](#) name.

**5.174.3.3 virtual std::string Arc::URLLocation::str () const [virtual]**

Returns a string representation of the [URLLocation](#).

Reimplemented from [Arc::URL](#).

**5.174.4 Field Documentation****5.174.4.1 std::string [Arc::URLLocation::name](#) [protected]**

the [URLLocation](#) name as registered in the indexing service.

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

- [URL.h](#)

## 5.175 Arc::UserConfig Class Reference

User configuration class

```
#include <UserConfig.h>
```

### Public Member Functions

- [UserConfig](#) ([initializeCredentialsType](#) initializeCredentials=[initializeCredentialsType](#)())
- [UserConfig](#) (const std::string &conffile, [initializeCredentialsType](#) initializeCredentials=[initializeCredentialsType](#)(), bool loadSysConfig=true)
- [UserConfig](#) (const std::string &conffile, const std::string &jfile, [initializeCredentialsType](#) initializeCredentials=[initializeCredentialsType](#)(), bool loadSysConfig=true)
- [UserConfig](#) (const long int &ptraddr)
- bool [InitializeCredentials](#) ([initializeCredentialsType](#) initializeCredentials)
- bool [CredentialsFound](#) () const
- bool [LoadConfigurationFile](#) (const std::string &conffile, bool ignoreJobListFile=true)
- bool [SaveToFile](#) (const std::string &filename) const
- void [ApplyToConfig](#) ([BaseConfig](#) &ccfg) const
- [operator bool](#) () const
- bool [operator!](#) () const
- bool [JobListFile](#) (const std::string &path)
- const std::string & [JobListFile](#) () const
- bool [Timeout](#) (int newTimeout)
- int [Timeout](#) () const
- bool [Verbosity](#) (const std::string &newVerbosity)
- const std::string & [Verbosity](#) () const
- bool [Broker](#) (const std::string &name)
- bool [Broker](#) (const std::string &name, const std::string &argument)
- const std::pair< std::string, std::string > & [Broker](#) () const
- bool [Bartender](#) (const std::vector< [URL](#) > &urls)
- void [AddBartender](#) (const [URL](#) &url)
- const std::vector< [URL](#) > & [Bartender](#) () const
- bool [VOMSESPath](#) (const std::string &path)
- const std::string & [VOMSESPath](#) ()
- bool [UserName](#) (const std::string &name)
- const std::string & [UserName](#) () const
- bool [Password](#) (const std::string &newPassword)
- const std::string & [Password](#) () const
- bool [ProxyPath](#) (const std::string &newProxyPath)
- const std::string & [ProxyPath](#) () const
- bool [CertificatePath](#) (const std::string &newCertificatePath)
- const std::string & [CertificatePath](#) () const
- bool [KeyPath](#) (const std::string &newKeyPath)
- const std::string & [KeyPath](#) () const
- bool [KeyPassword](#) (const std::string &newKeyPassword)
- const std::string & [KeyPassword](#) () const
- bool [KeySize](#) (int newKeySize)
- int [KeySize](#) () const
- bool [CACertificatePath](#) (const std::string &newCACertificatePath)



- const std::string & [CACertificatePath](#) () const
- bool [CACertificatesDirectory](#) (const std::string &newCACertificatesDirectory)
- const std::string & [CACertificatesDirectory](#) () const
- bool [CertificateLifeTime](#) (const Period &newCertificateLifeTime)
- const Period & [CertificateLifeTime](#) () const
- bool [SLCS](#) (const [URL](#) &newSLCS)
- const [URL](#) & [SLCS](#) () const
- bool [StoreDirectory](#) (const std::string &newStoreDirectory)
- const std::string & [StoreDirectory](#) () const
- bool [JobDownloadDirectory](#) (const std::string &newDownloadDirectory)
- const std::string & [JobDownloadDirectory](#) () const
- bool [IdPName](#) (const std::string &name)
- const std::string & [IdPName](#) () const
- bool [OverlayFile](#) (const std::string &path)
- const std::string & [OverlayFile](#) () const
- bool [UtilsDirPath](#) (const std::string &dir)
- const std::string & [UtilsDirPath](#) () const
- void [SetUser](#) (const User &u)
- const User & [GetUser](#) () const

## Static Public Attributes

- static const std::string [ARCUSERDIRECTORY](#)
- static const std::string [SYSCONFIG](#)
- static const std::string [SYSCONFIGARCLOC](#)
- static const std::string [DEFAULTCONFIG](#)
- static const std::string [EXAMPLECONFIG](#)
- static const int [DEFAULT\\_TIMEOUT](#) = 20
- static const std::string [DEFAULT\\_BROKER](#)

### 5.175.1 Detailed Description

User configuration class

This class provides a container for a selection of various attributes/parameters which can be configured to needs of the user, and can be read by implementing instances or programs. The class can be used in two ways. One can create a object from a configuration file, or simply set the desired attributes by using the setter method, associated with every setable attribute. The list of attributes which can be configured in this class are:

- certificatepath / [CertificatePath\(const std::string&\)](#)
- keypath / [KeyPath\(const std::string&\)](#)
- proxypath / [ProxyPath\(const std::string&\)](#)
- cacertificatesdirectory / [CACertificatesDirectory\(const std::string&\)](#)
- cacertificatepath / [CACertificatePath\(const std::string&\)](#)
- timeout / [Timeout\(int\)](#)
- joblist / [JobListFile\(const std::string&\)](#)

- verbosity / [Verbosity\(const std::string&\)](#)
- brokername / [Broker\(const std::string&\)](#) or [Broker\(const std::string&, const std::string&\)](#)
- brokerarguments / [Broker\(const std::string&\)](#) or [Broker\(const std::string&, const std::string&\)](#)
- bartender / [Bartender\(const std::list<URL>&\)](#)
- vomsserverpath / [VOMSESPath\(const std::string&\)](#)
- username / [UserName\(const std::string&\)](#)
- password / [Password\(const std::string&\)](#)
- keypassword / [KeyPassword\(const std::string&\)](#)
- keysize / [KeySize\(int\)](#)
- certificatelifetime / [CertificateLifeTime\(const Period&\)](#)
- slcs / [SLCS\(const URL&\)](#)
- storedirectory / [StoreDirectory\(const std::string&\)](#)
- jobdownloaddirectory / [JobDownloadDirectory\(const std::string&\)](#)
- idpname / [IdPName\(const std::string&\)](#)

where the first term is the name of the attribute used in the configuration file, and the second term is the associated setter method (for more information about a given attribute see the description of the setter method).

The configuration file should have a INI-style format and the `IniConfig` class will thus be used to parse the file. The above mentioned attributes should be placed in the common section. Another section is also valid in the configuration file, which is the alias section. Here it is possible to define aliases representing one or multiple services. These aliases can be used in the `AddServices(const std::list<std::string>&, ServiceType)` and `AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)` methods.

The `UserConfig` class also provides a method `InitializeCredentials()` for locating user credentials by searching in different standard locations. The `CredentialsFound()` method can be used to test if locating the credentials succeeded.

## 5.175.2 Constructor & Destructor Documentation

### 5.175.2.1 `Arc::UserConfig::UserConfig(initializeCredentialsType initializeCredentials = initializeCredentialsType())`

Create a `UserConfig` object.

The `UserConfig` object created by this constructor initializes only default values, and if specified by the `initializeCredentials` boolean credentials will be tried initialized using the `InitializeCredentials()` method. The object is only non-valid if initialization of credentials fails which can be checked with the operator `bool()` method.

#### Parameters:

*initializeCredentials* is a optional boolean indicating if the `InitializeCredentials()` method should be invoked, the default is `true`.

See also:

[InitializeCredentials\(\)](#)  
operator bool()

**5.175.2.2 Arc::UserConfig::UserConfig (const std::string & *conf*file, [initializeCredentialsType](#) *initializeCredentials* = [initializeCredentialsType](#) (), bool *loadSysConfig* = true)**

Create a [UserConfig](#) object.

The [UserConfig](#) object created by this constructor will, if specified by the *loadSysConfig* boolean, first try to load the system configuration file by invoking the [LoadConfigurationFile\(\)](#) method, and if this fails a WARNING is reported. Then the configuration file passed will be tried loaded using the before mentioned method, and if this fails an ERROR is reported, and the created object will be non-valid. Note that if the passed file path is empty the example configuration will be tried copied to the default configuration file path specified by DEFAULTCONFIG. If the example file cannot be copied one or more WARNING messages will be reported and no configuration will be loaded. If loading the configurations file succeeded and if *initializeCredentials* is true then credentials will be initialized using the [InitializeCredentials\(\)](#) method, and if no valid credentials are found the created object will be non-valid.

**Parameters:**

*conf*file is the path to a INI-configuration file.

*initializeCredentials* is a boolean indicating if credentials should be initialized, the default is true.

*loadSysConfig* is a boolean indicating if the system configuration file should be loaded aswell, the default is true.

See also:

[LoadConfigurationFile\(const std::string&, bool\)](#)  
[InitializeCredentials\(\)](#)  
operator bool()  
SYSCONFIG  
EXAMPLECONFIG

**5.175.2.3 Arc::UserConfig::UserConfig (const std::string & *conf*file, const std::string & *jfile*, [initializeCredentialsType](#) *initializeCredentials* = [initializeCredentialsType](#) (), bool *loadSysConfig* = true)**

Create a [UserConfig](#) object.

The [UserConfig](#) object created by this constructor does only differ from the UserConfig(const std::string&, bool, bool) constructor in that it is possible to pass the path of the job list file directly to this constructor. If the job list file *joblistfile* is empty, the behaviour of this constructor is exactly the same as the before mentioned, otherwise the job list file will be initilized by invoking the setter method [JobListFile\(const std::string&\)](#). If it fails the created object will be non-valid, otherwise the specified configuration file *conf*file will be loaded with the *ignoreJobListFile* argument set to true.

**Parameters:**

*conf*file is the path to a INI-configuration file

*jfile* is the path to a (non-)existing job list file.

*initializeCredentials* is a boolean indicating if credentials should be initialized, the default is `true`.

*loadSysConfig* is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

**See also:**

[JobListFile\(const std::string&\)](#)  
[LoadConfigurationFile\(const std::string&, bool\)](#)  
[InitializeCredentials\(\)](#)  
[operator bool\(\)](#)

#### 5.175.2.4 Arc::UserConfig::UserConfig (const long int & ptraddr)

Language binding constructor.

The passed long int should be a pointer address to a [UserConfig](#) object, and this address is then casted into this [UserConfig](#) object.

**Parameters:**

*ptraddr* is an memory address to a [UserConfig](#) object.

### 5.175.3 Member Function Documentation

#### 5.175.3.1 void Arc::UserConfig::AddBartender (const URL & url) [inline]

Set bartenders, used to contact Chelonia.

Takes as input a Bartender [URL](#) and adds this to the list of bartenders.

**Parameters:**

*url* is a [URL](#) to be added to the list of bartenders.

**See also:**

[Bartender\(const std::list<URL>&\)](#)  
[Bartender\(\) const](#)

#### 5.175.3.2 void Arc::UserConfig::ApplyToConfig (BaseConfig & ccfg) const

Apply credentials to [BaseConfig](#).

This methods sets the [BaseConfig](#) credentials to the credentials contained in this object. It also passes user defined configuration overlay if any.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\)](#)  
[BaseConfig](#)

**Parameters:**

*ccfg* a [BaseConfig](#) object which will configured with the credentials of this object.

**5.175.3.3** `const std::vector<URL>& Arc::UserConfig::Bartender () const` [inline]

Get bartenders.

Returns a list of Bartender URLs

**Returns:**

The list of bartender [URL](#) objects is returned.

**See also:**

[Bartender\(const std::list<URL>&\)](#)  
[AddBartender\(const URL&\)](#)

**5.175.3.4** `bool Arc::UserConfig::Bartender (const std::vector< URL > & urls)` [inline]

Set bartenders, used to contact Chelonia.

Takes as input a vector of Bartender URLs.

The attribute associated with this setter method is 'bartender'.

**Parameters:**

*urls* is a list of [URL](#) object to be set as bartenders.

**Returns:**

This method always returns `true`.

**See also:**

[AddBartender\(const URL&\)](#)  
[Bartender\(\) const](#)

**5.175.3.5** `const std::pair<std::string, std::string>& Arc::UserConfig::Broker () const`  
[inline]

Get the broker and corresponding arguments.

The returned pair contains the broker name as the first component and the argument as the second.

**See also:**

[Broker\(const std::string&\)](#)  
[Broker\(const std::string&, const std::string&\)](#)  
[DEFAULT\\_BROKER](#)

**5.175.3.6** `bool Arc::UserConfig::Broker (const std::string & name, const std::string & argument)`  
[inline]

Set broker to use in target matching.

As opposed to the [Broker\(const std::string&\)](#) method this method sets broker name and arguments directly from the passed two arguments.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

**Parameters:**

*name* is the name of the broker.  
*argument* is the arguments of the broker.

**Returns:**

This method always returns `true`.

**See also:**

[Broker](#)  
[Broker\(const std::string&\)](#)  
[Broker\(\) const](#)  
[DEFAULT\\_BROKER](#)

**5.175.3.7 bool Arc::UserConfig::Broker (const std::string & name)**

Set broker to use in target matching.

The string passed to this method should be in the format:

`< name > [:< argument >]`

where the `<name>` is the name of the broker and cannot contain any `'.'`, and the optional `<argument>` should contain arguments which should be passed to the broker.

Two attributes are associated with this setter method `'brokername'` and `'brokerarguments'`.

**Parameters:**

*name* the broker name and argument specified in the format given above.

**Returns:**

This method always returns `true`.

**See also:**

[Broker](#)  
[Broker\(const std::string&, const std::string&\)](#)  
[Broker\(\) const](#)  
[DEFAULT\\_BROKER](#)

**5.175.3.8 const std::string& Arc::UserConfig::CACertificatePath () const [inline]**

Get path to CA-certificate.

Retrieve the path to the file containing CA-certificate. This configuration parameter is deprecated.

**Returns:**

The path to the CA-certificate is returned.

**See also:**

[CACertificatePath\(const std::string&\)](#)

### 5.175.3.9 `bool Arc::UserConfig::CACertificatePath (const std::string & newCACertificatePath)` [inline]

Set CA-certificate path.

The path to the file containing CA-certificate will be set when calling this method. This configuration parameter is deprecated - use CACertificatesDirectory instead. Only arcslcs uses it.

The attribute associated with this setter method is 'cacertificatepath'.

#### Parameters:

*newCACertificatePath* is the path to the CA-certificate.

#### Returns:

This method always returns `true`.

#### See also:

[CACertificatePath\(\) const](#)

### 5.175.3.10 `const std::string& Arc::UserConfig::CACertificatesDirectory () const` [inline]

Get path to CA-certificate directory.

Retrieve the path to the CA-certificate directory.

#### Returns:

The path to the CA-certificate directory is returned.

#### See also:

[InitializeCredentials\(\)](#)

[CredentialsFound\(\) const](#)

[CACertificatesDirectory\(const std::string&\)](#)

### 5.175.3.11 `bool Arc::UserConfig::CACertificatesDirectory (const std::string & newCACertificatesDirectory)` [inline]

Set path to CA-certificate directory.

The path to the directory containing CA-certificates will be set when calling this method. Note that the [InitializeCredentials\(\)](#) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'cacertificatesdirectory'.

#### Parameters:

*newCACertificatesDirectory* is the path to the CA-certificate directory.

#### Returns:

This method always returns `true`.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\) const](#)  
[CACertificatesDirectory\(\) const](#)

**5.175.3.12 const Period& Arc::UserConfig::CertificateLifeTime () const [inline]**

Get certificate life time.

Gets lifetime of user certificate which will be obtained from Short Lived Credentials [Service](#).

**Returns:**

The certificate life time is returned as a Period object.

**See also:**

[CertificateLifeTime\(const Period&\)](#)

**5.175.3.13 bool Arc::UserConfig::CertificateLifeTime (const Period & *newCertificateLifeTime*) [inline]**

Set certificate life time.

Sets lifetime of user certificate which will be obtained from Short Lived Credentials [Service](#).

The attribute associated with this setter method is 'certificatelifetime'.

**Parameters:**

*newCertificateLifeTime* is the life time of a certificate, as a Period object.

**Returns:**

This method always returns `true`.

**See also:**

[CertificateLifeTime\(\) const](#)

**5.175.3.14 const std::string& Arc::UserConfig::CertificatePath () const [inline]**

Get path to certificate.

The path to the cerficate is returned when invoking this method.

**Returns:**

The certificate path is returned.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\) const](#)  
[CertificatePath\(const std::string&\)](#)  
[KeyPath\(\) const](#)



### 5.175.3.15 `bool Arc::UserConfig::CertificatePath (const std::string & newCertificatePath)` [inline]

Set path to certificate.

The path to user certificate will be set by this method. The path to the corresponding key can be set with the [KeyPath\(const std::string&\)](#) method. Note that the [InitializeCredentials\(\)](#) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'certificatepath'.

#### Parameters:

*newCertificatePath* is the path to the new certificate.

#### Returns:

This method always returns `true`.

#### See also:

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\) const](#)  
[CertificatePath\(\) const](#)  
[KeyPath\(const std::string&\)](#)

### 5.175.3.16 `bool Arc::UserConfig::CredentialsFound () const` [inline]

Validate credential location.

Valid credentials consists of a combination of a path to existing CA-certificate directory and either a path to existing proxy or a path to existing user key/certificate pair. If valid credentials are found this method returns `true`, otherwise `false` is returned.

#### Returns:

`true` if valid credentials are found, otherwise `false`.

#### See also:

[InitializeCredentials\(\)](#)

### 5.175.3.17 `const User& Arc::UserConfig::GetUser () const` [inline]

Get User for filesystem access.

#### Returns:

The user identity to use for file system access

#### See also:

[SetUser\(const User&\)](#)

**5.175.3.18** `const std::string& Arc::UserConfig::IdPName () const` `[inline]`

Get IdP name.

Gets Identity Provider name (Shibboleth) to which user belongs.

**Returns:**

The IdP name

**See also:**

[IdPName\(const std::string&\)](#)

**5.175.3.19** `bool Arc::UserConfig::IdPName (const std::string & name)` `[inline]`

Set IdP name.

Sets Identity Provider name (Shibboleth) to which user belongs. It is used for contacting Short Lived Certificate [Service](#).

The attribute associated with this setter method is 'idpname'.

**Parameters:**

*name* is the new IdP name.

**Returns:**

This method always returns `true`.

**See also:****5.175.3.20** `bool Arc::UserConfig::InitializeCredentials (initializeCredentialsType initializeCredentials)`

Initialize user credentials.

The location of the user credentials will be tried located when calling this method and stored internally when found. The method searches in different locations. Depending on value of `initializeCredentials` this method behaves differently. Following is an explanation for `RequireCredentials`. For less strict values see information below. First the user proxy or the user key/certificate pair is tried located in the following order:

- Proxy path specified by the environment variable `X509_USER_PROXY`. If value is set and corresponding file does not exist it considered to be an error and no other locations are tried. If found no more proxy paths are tried.
- Current proxy path as passed to the constructor, explicitly set using the setter method [ProxyPath\(const std::string&\)](#) or read from configuration by constructor or `LoadConfiguartionFile()` method. If value is set and corresponding file does not exist it considered to be an error and no other locations are tried. If found no more proxy paths are tried.
- Proxy path made of `x509up_u` token concatenated with the user numerical ID located in the OS temporary directory. It is NOT an error if corresponding file does not exist and processing continues.

- Key/certificate paths specified by the environment variables X509\_USER\_KEY and X509\_USER\_CERT. If values are set and corresponding files do not exist it is considered to be an error and no other locations are tried. Error message is suppressed if proxy was previously found.
- Current key/certificate paths passed to the constructor or explicitly set using the setter methods [KeyPath\(const std::string&\)](#) and [CertificatePath\(const std::string&\)](#) or read from configuration by constructor or LoadConfigurationFile() method. If values are set and corresponding files do not exist it is an error and no other locations are tried. Error message is suppressed if proxy was previously found.
- Key/certificate paths ~/.arc/usercert.pem and ~/.arc/userkey.pem respectively are tried. It is not an error if not found.
- Key/certificate paths ~/.globus/usercert.pem and ~/.globus/userkey.pem respectively are tried. It is not an error if not found.
- Key/certificate paths created by concatenation of ARC installation location and /etc/arc/usercert.pem and /etc/arc/userkey.pem respectively are tried. It is not an error if not found.
- Key/certificate located in current working directory are tried.
- If neither proxy nor key/certificate files are found this is considered to be an error.

Along with the proxy and key/certificate pair, the path of the directory containing CA certificates is also located. The presence of directory will be checked in the following order and first found is accepted:

- Path specified by the X509\_CERT\_DIR environment variable. It is an error if value is set and directory does not exist.
- Current path explicitly specified by using the setter method [CACertificatesDirectory\(\)](#) or read from configuration by constructor or LoadConfigurationFile() method. It is an error if value is set and directory does not exist.
- Path ~/.globus/certificates. It is not an error if it does not exist.
- Path created by concatenating the ARC installation location and /etc/certificates. It is not an error if it does not exist.
- Path created by concatenating the ARC installation location and /share/certificates. It is not an error if it does not exist.
- Path /etc/grid-security/certificates.

It is an error if none of the directories above exist.

In case of initializeCredentials == TryCredentials method behaves same way like in case RequireCredentials except it does not report errors through its [Logger](#) object and does not return false.

If NotTryCredentials is used method does not check for presence of credentials. It behaves like if corresponding files are always present.

And in case of SkipCredentials method does nothing.

All options with SkipCA\* prefix behaves similar to those without prefix except the path of the directory containing CA certificates is completely ignored.

**See also:**

[CredentialsFound\(\)](#)  
[ProxyPath\(const std::string&\)](#)

[KeyPath\(const std::string&\)](#)  
[CertificatePath\(const std::string&\)](#)  
[CACertificatesDirectory\(const std::string&\)](#)

#### **5.175.3.21** `const std::string& Arc::UserConfig::JobDownloadDirectory () const` [inline]

Get download directory.

returns directory which will be used to download the job directory using arcget command.

The attribute associated with the method is 'jobdownloadaddirectory'.

##### **Returns:**

This method returns the job download directory.

##### **See also:**

#### **5.175.3.22** `bool Arc::UserConfig::JobDownloadDirectory (const std::string & newDownloadDirectory)` [inline]

Set download directory.

Sets directory which will be used to download the job directory using arcget command.

The attribute associated with this setter method is 'jobdownloadaddirectory'.

##### **Parameters:**

*newDownloadDirectory* is the path to the download directory.

##### **Returns:**

This method always returns `true`.

##### **See also:**

#### **5.175.3.23** `const std::string& Arc::UserConfig::JobListFile () const` [inline]

Get a reference to the path of the job list file.

The job list file is used to store and fetch information about submitted computing jobs to computing services. This method will return the path to the specified job list file.

##### **Returns:**

The path to the job list file is returned.

##### **See also:**

[JobListFile\(const std::string&\)](#)

#### 5.175.3.24 bool Arc::UserConfig::JobListFile (const std::string & path)

Set path to job list file.

The method takes a path to a file which will be used as the job list file for storing and reading job information. If the specified path *path* does not exist a empty job list file will be tried created. If creating the job list file in any way fails *false* will be returned and a ERROR message will be reported. Otherwise *true* is returned. If the directory containing the file does not exist, it will be tried created. The method will also return *false* if the file is not a regular file.

The attribute associated with this setter method is 'joblist'.

##### Parameters:

*path* the path to the job list file.

##### Returns:

If the job list file is a regular file or if it can be created *true* is returned, otherwise *false* is returned.

##### See also:

[JobListFile\(\) const](#)

#### 5.175.3.25 const std::string& Arc::UserConfig::KeyPassword () const [inline]

Get password for generated key.

Get password to be used to encode private key of credentials obtained from Short Lived Credentials [Service](#).

##### Returns:

The key password is returned.

##### See also:

[KeyPassword\(const std::string&\)](#)  
[KeyPath\(\) const](#)  
[KeySize\(\) const](#)

#### 5.175.3.26 bool Arc::UserConfig::KeyPassword (const std::string & newKeyPassword) [inline]

Set password for generated key.

Set password to be used to encode private key of credentials obtained from Short Lived Credentials [Service](#).

The attribute associated with this setter method is 'keypassword'.

##### Parameters:

*newKeyPassword* is the new password to the key.

##### Returns:

This method always returns `true`.

**See also:**

[KeyPassword\(\) const](#)  
[KeyPath\(const std::string&\)](#)  
[KeySize\(int\)](#)

**5.175.3.27** `const std::string& Arc::UserConfig::KeyPath () const` [inline]

Get path to key.

The path to the key is returned when invoking this method.

**Returns:**

The path to the user key is returned.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\) const](#)  
[KeyPath\(const std::string&\)](#)  
[CertificatePath\(\) const](#)  
[KeyPassword\(\) const](#)  
[KeySize\(\) const](#)

**5.175.3.28** `bool Arc::UserConfig::KeyPath (const std::string & newKeyPath)` [inline]

Set path to key.

The path to user key will be set by this method. The path to the corresponding certificate can be set with the [CertificatePath\(const std::string&\)](#) method. Note that the [InitializeCredentials\(\)](#) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'keypath'.

**Parameters:**

*newKeyPath* is the path to the new key.

**Returns:**

This method always returns `true`.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\) const](#)  
[KeyPath\(\) const](#)  
[CertificatePath\(const std::string&\)](#)  
[KeyPassword\(const std::string&\)](#)  
[KeySize\(int\)](#)

**5.175.3.29 int Arc::UserConfig::KeySize () const** [inline]

Get key size.

Get size/strengt of private key of credentials obtained from Short Lived Credentials [Service](#).

**Returns:**

The key size, as an integer, is returned.

**See also:**

[KeySize\(int\)](#)

[KeyPath\(\) const](#)

[KeyPassword\(\) const](#)

**5.175.3.30 bool Arc::UserConfig::KeySize (int *newKeySize*)** [inline]

Set key size.

Set size/strengt of private key of credentials obtained from Short Lived Credentials [Service](#).

The attribute associated with this setter method is 'keysize'.

**Parameters:**

*newKeySize* is the size, an an integer, of the key.

**Returns:**

This method always returns `true`.

**See also:**

[KeySize\(\) const](#)

[KeyPath\(const std::string&\)](#)

[KeyPassword\(const std::string&\)](#)

**5.175.3.31 bool Arc::UserConfig::LoadConfigurationFile (const std::string & *confFile*, bool *ignoreJobListFile* = `true`)**

Load specified configuration file.

The configuration file passed is parsed by this method by using the IniConfig class. If the parsing is unsuccessful a WARNING is reported.

The format of the configuration file should follow that of INI, and every attribute present in the file is only allowed once, if otherwise a WARNING will be reported. The file can contain at most two sections, one named common and the other name alias. If other sections exist a WARNING will be reported. Only the following attributes is allowed in the common section of the configuration file:

- certificatepath ([CertificatePath\(const std::string&\)](#))
- keypath ([KeyPath\(const std::string&\)](#))
- proxypath ([ProxyPath\(const std::string&\)](#))

- `cacertificatesdirectory` ([CACertificatesDirectory\(const std::string&\)](#))
- `cacertificatepath` ([CACertificatePath\(const std::string&\)](#))
- `timeout` ([Timeout\(int\)](#))
- `joblist` ([JobListFile\(const std::string&\)](#))
- `verbosity` ([Verbosity\(const std::string&\)](#))
- `brokername` ([Broker\(const std::string&\)](#) or [Broker\(const std::string&, const std::string&\)](#))
- `brokerarguments` ([Broker\(const std::string&\)](#) or [Broker\(const std::string&, const std::string&\)](#))
- `bartender` ([Bartender\(const std::list<URL>&\)](#))
- `vomsserverpath` ([VOMSEPath\(const std::string&\)](#))
- `username` ([UserName\(const std::string&\)](#))
- `password` ([Password\(const std::string&\)](#))
- `keypassword` ([KeyPassword\(const std::string&\)](#))
- `keysize` ([KeySize\(int\)](#))
- `certificatelifetime` ([CertificateLifeTime\(const Period&\)](#))
- `slcs` ([SLCS\(const URL&\)](#))
- `storedirectory` ([StoreDirectory\(const std::string&\)](#))
- `jobdownloadaddirectory` ([JobDownloadDirectory\(const std::string&\)](#))
- `idpname` ([IdPName\(const std::string&\)](#))

where the method in parentheses is the associated setter method. If other attributes exist in the common section a WARNING will be reported for each of these attributes. In the alias section aliases can be defined, and should represent a selection of services. The alias can then be referred to by input to the `AddServices(const std::list<std::string>&, ServiceType)` and `AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)` methods. An alias can not contain any of the characters `'`, `:`, `'`, `:`, `'` or `\t` and should be defined as follows:

`< alias_name >=< service_type >:< flavour >:< service_url > | < alias_ref > [...]`

where `<alias_name>` is the name of the defined alias, `<service_type>` is the service type in lower case, `<flavour>` is the type of middleware plugin to use, `<service_url>` is the [URL](#) which should be used to contact the service and `<alias_ref>` is another defined alias. The parsed aliases will be stored internally and resolved when needed. If a alias already exist, and another alias with the same name is parsed then this other alias will overwrite the existing alias.

#### Parameters:

***conffile*** is the path to the configuration file.

***ignoreJobListFile*** is a optional boolean which indicates whether the joblistfile attribute in the configuration file should be ignored. Default is to ignore it (`true`).

#### Returns:

If loading the configuration file succeeds `true` is returned, otherwise `false` is returned.

#### See also:

[SaveToFile\(\)](#)



**5.175.3.32 Arc::UserConfig::operator bool (void) const** [inline]

Check for validity.

The validity of an object created from this class can be checked using this casting operator. An object is valid if the constructor did not encounter any errors.

**See also:**

[operator!\(\)](#)

**5.175.3.33 bool Arc::UserConfig::operator! (void) const** [inline]

Check for non-validity.

See operator bool() for a description.

**See also:**

operator bool()

**5.175.3.34 const std::string& Arc::UserConfig::OverlayFile () const** [inline]

Get path to configuration overlay file.

**Returns:**

The overlay file path

**See also:**

[OverlayFile\(const std::string&\)](#)

**5.175.3.35 bool Arc::UserConfig::OverlayFile (const std::string & *path*)** [inline]

Set path to configuration overlay file.

Content of specified file is a backdoor to configuration XML generated from information stored in this class. The content of file is passed to [BaseConfig](#) class in `ApplyToConfig(BaseConfig&)` then merged with internal configuration XML representation. This feature is meant for quick prototyping/testing/tuning of functionality without rewriting code. It is meant for developers and most users won't need it.

The attribute associated with this setter method is 'overlayfile'.

**Parameters:**

*path* is the new overlay file path.

**Returns:**

This method always returns `true`.

**See also:**

**5.175.3.36** `const std::string& Arc::UserConfig::Password () const` `[inline]`

Get password.

Get password which is used for requesting credentials from Short Lived Credentials [Service](#).

**Returns:**

The password is returned.

**See also:**

[Password\(const std::string&\)](#)

**5.175.3.37** `bool Arc::UserConfig::Password (const std::string & newPassword)` `[inline]`

Set password.

Set password which is used for requesting credentials from Short Lived Credentials [Service](#).

The attribute associated with this setter method is 'password'.

**Parameters:**

*newPassword* is the new password to set.

**Returns:**

This method always returns true.

**See also:**

[Password\(\) const](#)

**5.175.3.38** `const std::string& Arc::UserConfig::ProxyPath () const` `[inline]`

Get path to user proxy.

Retrieve path to user proxy.

**Returns:**

Returns the path to the user proxy.

**See also:**

[ProxyPath\(const std::string&\)](#)

**5.175.3.39** `bool Arc::UserConfig::ProxyPath (const std::string & newProxyPath)` `[inline]`

Set path to user proxy.

This method will set the path of the user proxy. Note that the [InitializeCredentials\(\)](#) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'proxypath'

**Parameters:**

*newProxyPath* is the path to a user proxy.

**Returns:**

This method always returns `true`.

**See also:**

[InitializeCredentials\(\)](#)  
[CredentialsFound\(\)](#)  
[ProxyPath\(\) const](#)

**5.175.3.40 bool Arc::UserConfig::SaveToFile (const std::string &filename) const**

Save to INI file.

This method will save the object data as a INI file. The saved file can be loaded with the LoadConfigurationFile method.

**Parameters:**

*filename* the name of the file which the data will be saved to.

**Returns:**

`false` if unable to get handle on file, otherwise `true` is returned.

**See also:**

[LoadConfigurationFile\(\)](#)

**5.175.3.41 void Arc::UserConfig::SetUser (const User &u) [inline]**

Set User for filesystem access.

Sometimes it is desirable to use the identity of another user when accessing the filesystem. This user can be specified through this method. By default this user is the same as the user running the process.

**Parameters:**

*u* User identity to use

**5.175.3.42 const URL& Arc::UserConfig::SLCS () const [inline]**

Get the [URL](#) to the Short Lived Certificate [Service](#) (SLCS).

**Returns:**

The SLCS is returned.

**See also:**

[SLCS\(const URL&\)](#)

**5.175.3.43** `bool Arc::UserConfig::SLCS (const URL & newSLCS)` `[inline]`

Set the [URL](#) to the Short Lived Certificate [Service](#) (SLCS).

The attribute associated with this setter method is 'slcs'.

**Parameters:**

*newSLCS* is the [URL](#) to the SLCS

**Returns:**

This method always returns `true`.

**See also:**

[SLCS\(\)](#) `const`

**5.175.3.44** `const std::string& Arc::UserConfig::StoreDirectory () const` `[inline]`

Get store directory.

Sets directory which is used to store credentials obtained from Short Lived [Credential](#) Service.

**Returns:**

The path to the store directory is returned.

**See also:**

[StoreDirectory\(const std::string&\)](#)

**5.175.3.45** `bool Arc::UserConfig::StoreDirectory (const std::string & newStoreDirectory)`  
`[inline]`

Set store directory.

Sets directory which will be used to store credentials obtained from Short Lived [Credential](#) Service.

The attribute associated with this setter method is 'storedirectory'.

**Parameters:**

*newStoreDirectory* is the path to the store directory.

**Returns:**

This method always returns `true`.

**See also:**

**5.175.3.46 int Arc::UserConfig::Timeout () const** [inline]

Get timeout.

Returns the timeout in seconds.

**Returns:**

timeout in seconds.

**See also:**

[Timeout\(int\)](#)  
[DEFAULT\\_TIMEOUT](#)

**5.175.3.47 bool Arc::UserConfig::Timeout (int *newTimeout*)**

Set timeout.

When communicating with a service the timeout specifies how long, in seconds, the communicating instance should wait for a response. If the response have not been recieved before this period in time, the connection is typically dropped, and an error will be reported.

This method will set the timeout to the specified integer. If the passed integer is less than or equal to 0 then `false` is returned and the timeout will not be set, otherwise `true` is returned and the timeout will be set to the new value.

The attribute associated with this setter method is 'timeout'.

**Parameters:**

*newTimeout* the new timeout value in seconds.

**Returns:**

`false` in case *newTimeout* <= 0, otherwise `true`.

**See also:**

[Timeout\(\) const](#)  
[DEFAULT\\_TIMEOUT](#)

**5.175.3.48 const std::string& Arc::UserConfig::UserName () const** [inline]

Get user-name.

Get username which is used for requesting credentials from Short Lived Credentials [Service](#).

**Returns:**

The username is returned.

**See also:**

[UserName\(const std::string&\)](#)

**5.175.3.49** `bool Arc::UserConfig::UserName (const std::string & name)` `[inline]`

Set user-name for SLCS.

Set username which is used for requesting credentials from Short Lived Credentials [Service](#).

The attribute associated with this setter method is 'username'.

**Parameters:**

*name* is the name of the user.

**Returns:**

This method always return true.

**See also:**

[UserName\(\) const](#)

**5.175.3.50** `const std::string& Arc::UserConfig::UtilsDirPath () const` `[inline]`

Get path to directory storing utility files for DataPoints.

**Returns:**

The utils dir path

**See also:**

[UtilsDirPath\(const std::string&\)](#)

**5.175.3.51** `bool Arc::UserConfig::UtilsDirPath (const std::string & dir)`

Set path to directory storing utility files for DataPoints.

Some DataPoints can store information on remote services in local files. This method sets the path to the directory containing these files. For example arc\* tools set it to ARCUSERDIRECTORY and A-REX sets it to the control directory. The directory is created if it does not exist.

**Parameters:**

*path* is the new utils dir path.

**Returns:**

This method always returns `true`.

**5.175.3.52** `const std::string& Arc::UserConfig::Verbosity () const` `[inline]`

Get the user selected level of verbosity.

The string representation of the verbosity level specified by the user is returned when calling this method. If the user have not specified the verbosity level the empty string will be referenced.

**Returns:**

the verbosity level, or empty if it has not been set.

**See also:**

[Verbosity\(const std::string&\)](#)

**5.175.3.53 bool Arc::UserConfig::Verbosity (const std::string & newVerbosity)**

Set verbosity.

The verbosity will be set when invoking this method. If the string passed cannot be parsed into a corresponding LogLevel, using the function a WARNING is reported and `false` is returned, otherwise `true` is returned.

The attribute associated with this setter method is 'verbosity'.

**Returns:**

`true` in case the verbosity could be set to a allowed LogLevel, otherwise `false`.

**See also:**

[Verbosity\(\) const](#)

**5.175.3.54 const std::string& Arc::UserConfig::VOMSESPath ()**

Get path to file containing VOMS configuration.

Get path to file which contains list of VOMS services and associated configuration parameters.

**Returns:**

The path to VOMS configuration file is returned.

**See also:**

[VOMSESPath\(const std::string&\)](#)

**5.175.3.55 bool Arc::UserConfig::VOMSESPath (const std::string & path) [inline]**

Set path to file containing VOMS configuration.

Set path to file which contains list of VOMS services and associated configuration parameters needed to contact those services. It is used by arcproxy.

The attribute associated with this setter method is 'vomsserverpath'.

**Parameters:**

*path* the path to VOMS configuration file

**Returns:**

This method always return true.

**See also:**

[VOMSESPath\(\) const](#)

## 5.175.4 Field Documentation

### 5.175.4.1 `const std::string Arc::UserConfig::ARCUSERDIRECTORY` [static]

Path to ARC user home directory.

The *ARCUSERDIRECTORY* variable is the path to the ARC home directory of the current user. This path is created using the `User::Home()` method.

**See also:**

`User::Home()`

### 5.175.4.2 `const std::string Arc::UserConfig::DEFAULT_BROKER` [static]

Default broker.

The *DEFAULT\_BROKER* specifies the name of the broker which should be used in case no broker is explicitly chosen.

**See also:**

`Broker`  
`Broker(const std::string&)`  
`Broker(const std::string&, const std::string&)`  
`Broker() const`

### 5.175.4.3 `const int Arc::UserConfig::DEFAULT_TIMEOUT = 20` [static]

Default timeout in seconds.

The *DEFAULT\_TIMEOUT* specifies interval which will be used in case no timeout interval have been explicitly specified. For a description about timeout see `Timeout(int)`.

**See also:**

`Timeout(int)`  
`Timeout() const`

### 5.175.4.4 `const std::string Arc::UserConfig::DEFAULTCONFIG` [static]

Path to default configuration file.

The *DEFAULTCONFIG* variable is the path to the default configuration file used in case no configuration file have been specified. The path is created from the *ARCUSERDIRECTORY* object.

### 5.175.4.5 `const std::string Arc::UserConfig::EXAMPLECONFIG` [static]

Path to example configuration.

The *EXAMPLECONFIG* variable is the path to the example configuration file.



**5.175.4.6** `const std::string Arc::UserConfig::SYSCONFIG` `[static]`

Path to system configuration.

The *SYSCONFIG* variable is the path to the system configuration file. This variable is only equal to *SYSCONFIGARCLOC* if ARC is installed in the root (highly unlikely).

**5.175.4.7** `const std::string Arc::UserConfig::SYSCONFIGARCLOC` `[static]`

Path to system configuration at ARC location.

The *SYSCONFIGARCLOC* variable is the path to the system configuration file which reside at the ARC installation location.

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

- UserConfig.h

## 5.176 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security according to Username Token Profile.

```
#include <UsernameToken.h>
```

### Public Types

- enum [PasswordType](#)

### Public Member Functions

- [UsernameToken](#) (SOAPEnvelope &soap)
- [UsernameToken](#) (SOAPEnvelope &soap, const std::string &username, const std::string &password, const std::string &uid, [PasswordType](#) pwdtype)
- [UsernameToken](#) (SOAPEnvelope &soap, const std::string &username, const std::string &id, bool mac, int iteration)
- [operator bool](#) (void)
- std::string [Username](#) (void)
- bool [Authenticate](#) (const std::string &password, std::string &derived\_key)
- bool [Authenticate](#) (std::istream &password, std::string &derived\_key)

### 5.176.1 Detailed Description

Interface for manipulation of WS-Security according to Username Token Profile.

### 5.176.2 Member Enumeration Documentation

#### 5.176.2.1 enum [Arc::UsernameToken::PasswordType](#)

SOAP header element

### 5.176.3 Constructor & Destructor Documentation

#### 5.176.3.1 [Arc::UsernameToken::UsernameToken](#) (SOAPEnvelope & *soap*)

Link to existing SOAP header and parse Username Token information. Username Token related information is extracted from SOAP header and stored in class variables.

#### 5.176.3.2 [Arc::UsernameToken::UsernameToken](#) (SOAPEnvelope & *soap*, const std::string & *username*, const std::string & *password*, const std::string & *uid*, [PasswordType](#) *pwdtype*)

Add Username Token information into the SOAP header. Generated token contains elements Username and Password and is meant to be used for authentication.

#### Parameters:

*soap* the SOAP message

*username* <wsse:Username>...</wsse:Username> - if empty it is entered interactively from stdin

*password* <wsse:Password Type="...">...</wsse:Password> - if empty it is entered interactively from stdin

*uid* <wsse:UsernameToken wsu:ID="...">

*pwdtype* <wsse:Password Type="...">...</wsse:Password>

### 5.176.3.3 Arc::UsernameToken::UsernameToken (SOAPEnvelope & soap, const std::string & username, const std::string & id, bool mac, int iteration)

Add Username Token information into the SOAP header. Generated token contains elements Username and Salt and is meant to be used for deriving Key Derivation.

#### Parameters:

*soap* the SOAP message

*username* <wsse:Username>...</wsse:Username>

*mac* if derived key is meant to be used for [Message](#) Authentication Code

*iteration* <wsse11:Iteration>...</wsse11:Iteration>

## 5.176.4 Member Function Documentation

### 5.176.4.1 bool Arc::UsernameToken::Authenticate (std::istream & password, std::string & derived\_key)

Checks parsed token against password stored in specified stream. If token is meant to be used for deriving a key then key is returned in derived\_key

### 5.176.4.2 bool Arc::UsernameToken::Authenticate (const std::string & password, std::string & derived\_key)

Checks parsed/generated token against specified password. If token is meant to be used for deriving a key then key is returned in derived\_key. In that case authentication is performed outside of [UsernameToken](#) class using obtained derived\_key.

### 5.176.4.3 Arc::UsernameToken::operator bool (void)

Returns true of constructor succeeded

### 5.176.4.4 std::string Arc::UsernameToken::Username (void)

Returns username associated with this instance

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

- UsernameToken.h

## 5.177 Arc::UserSwitch Class Reference

```
#include <User.h>
```

### 5.177.1 Detailed Description

If this class is created user identity is switched to provided uid and gid. Due to internal lock there will be only one valid instance of this class. Any attempt to create another instance will block till first one is destroyed. If uid and gid are set to 0 then user identity is not switched. But lock is applied anyway. The lock has dual purpose. First and most important is to protect communication with underlying operating system which may depend on user identity. For that it is advisable for code which talks to operating system to acquire valid instance of this class. Care must be taken for not to hold that instance too long cause that may block other code in multithreaded environment. Other purpose of this lock is to provide workaround for glibc bug in `__nptl_setxid`. That bug causes lockup of `seteuid()` function if racing with fork. To avoid this problem the lock mentioned above is used by [Run](#) class while spawning new process.

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

- User.h

## 5.178 Arc::VOMSTrustList Class Reference

```
#include <VOMSUtil.h>
```

### Public Member Functions

- [VOMSTrustList](#) (const std::vector< std::string > &encoded\_list)
- [VOMSTrustList](#) (const std::vector< VOMSTrustChain > &chains, const std::vector< VOMSTrustRegex > &regexs)
- VOMSTrustChain & [AddChain](#) (const VOMSTrustChain &chain)
- VOMSTrustChain & [AddChain](#) (void)
- [RegularExpression](#) & [AddRegex](#) (const VOMSTrustRegex &reg)

### 5.178.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

### 5.178.2 Constructor & Destructor Documentation

#### 5.178.2.1 Arc::VOMSTrustList::VOMSTrustList (const std::vector< std::string > & encoded\_list)

Creates chain lists and regexps from plain list. List is made of chunks delimited by elements containing pattern "NEXT CHAIN". Each chunk with more than one element is converted into one instance of VOMSTrustChain. Chunks with single element are converted to VOMSTrustChain if element does not have special symbols. Otherwise it is treated as regular expression. Those symbols are '^','\$' and '\*'. Trusted chains can be configured in two ways: one way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>—NEXT CHAIN—</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> the other way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> each chunk is supposed to contain a suit of DN of trusted certificate chain, in which the first DN is the DN of the certificate (cert0) which is used to sign the Attribute Certificate (AC), the second DN is the DN of the issuer certificate(cert1) which is used to sign cert0. So if there are one or more intermediate issuers, then there should be 3 or more than 3 DNs in this chunk (considering cert0 and the root certificate, plus the intermediate certificate) .

#### 5.178.2.2 Arc::VOMSTrustList::VOMSTrustList (const std::vector< VOMSTrustChain > & chains, const std::vector< VOMSTrustRegex > & regexs)

Creates chain lists and regexps from those specified in arguments. See [AddChain\(\)](#) and [AddRegex\(\)](#) for more information.

### 5.178.3 Member Function Documentation

#### 5.178.3.1 VOMSTrustChain& Arc::VOMSTrustList::AddChain (void)

Adds empty chain of trusted DNs to list.

#### 5.178.3.2 VOMSTrustChain& Arc::VOMSTrustList::AddChain (const VOMSTrustChain & *chain*)

Adds chain of trusted DNs to list. During verification each signature of AC is checked against all stored chains. DNs of chain of certificate used for signing AC are compared against DNs stored in these chains one by one. If needed DN of issuer of last certificate is checked too. Comparison succeeds if DNs in at least one stored chain are same as those in certificate chain. Comparison stops when all DNs in stored chain are compared. If there are more DNs in stored chain than in certificate chain then comparison fails. Empty stored list matches any certificate chain. Taking into account that certificate chains are verified down to trusted CA anyway, having more than one DN in stored chain seems to be useless. But such feature may be found useful by some very strict sysadmins. ??? IMO, DN list here is not only for authentication, it is also kind of ACL, which means the AC consumer only trusts those DNs which issues AC.

#### 5.178.3.3 [RegularExpression](#)& Arc::VOMSTrustList::AddRegex (const VOMSTrustRegex & *reg*)

Adds regular expression to list. During verification each signature of AC is checked against all stored regular expressions. DN of signing certificate must match at least one of stored regular expressions.

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

- VOMSUtil.h

## 5.179 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

```
#include <WSA.h>
```

### Public Member Functions

- [WSAEndpointReference](#) ([XMLNode](#) epr)
- [WSAEndpointReference](#) (const [WSAEndpointReference](#) &wsa)
- [WSAEndpointReference](#) (const std::string &address)
- [WSAEndpointReference](#) (void)
- [~WSAEndpointReference](#) (void)
- std::string [Address](#) (void) const
- bool [hasAddress](#) (void) const
- void [Address](#) (const std::string &uri)
- [WSAEndpointReference](#) & [operator=](#) (const std::string &address)
- [XMLNode](#) [ReferenceParameters](#) (void)
- [XMLNode](#) [MetaData](#) (void)
- [operator XMLNode](#) (void)

### 5.179.1 Detailed Description

Interface for manipulation of WS-Adressing Endpoint Reference.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

### 5.179.2 Constructor & Destructor Documentation

#### 5.179.2.1 Arc::WSAEndpointReference::WSAEndpointReference ([XMLNode](#) epr)

Linking to existing EPR in XML tree

#### 5.179.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const [WSAEndpointReference](#) & wsa)

Copy constructor

#### 5.179.2.3 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & address)

Creating independent EPR - not implemented

#### 5.179.2.4 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

**5.179.2.5 Arc::WSAEndpointReference::~~WSAEndpointReference (void)**

Destructor. All empty elements of EPR XML are destroyed here too

**5.179.3 Member Function Documentation****5.179.3.1 void Arc::WSAEndpointReference::Address (const std::string & uri)**

Assigns new Address value. If EPR had no Address element it is created.

**5.179.3.2 std::string Arc::WSAEndpointReference::Address (void) const**

Returns Address ([URL](#)) encoded in EPR

**5.179.3.3 bool Arc::WSAEndpointReference::hasAddress (void) const**

Returns true if Address is defined

**5.179.3.4 [XMLNode](#) Arc::WSAEndpointReference::MetaData (void)**

Access to MetaData element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no MetaData element it is created.

**5.179.3.5 Arc::WSAEndpointReference::operator [XMLNode](#) (void)**

Returns reference to EPR top XML node

**5.179.3.6 [WSAEndpointReference&](#) Arc::WSAEndpointReference::operator= (const std::string & address)**

Same as Address(uri)

**5.179.3.7 [XMLNode](#) Arc::WSAEndpointReference::ReferenceParameters (void)**

Access to ReferenceParameters element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no ReferenceParameters element it is created.

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

- WSA.h



## 5.180 Arc::WSAHeader Class Reference

Interface for manipulation WS-Addressing information in SOAP header.

```
#include <WSA.h>
```

### Public Member Functions

- [WSAHeader](#) (SOAPEnvelope &soap)
- [WSAHeader](#) (const std::string &action)
- std::string [To](#) (void) const
- bool [hasTo](#) (void) const
- void [To](#) (const std::string &uri)
- [WSAEndpointReference From](#) (void)
- [WSAEndpointReference ReplyTo](#) (void)
- [WSAEndpointReference FaultTo](#) (void)
- std::string [Action](#) (void) const
- bool [hasAction](#) (void) const
- void [Action](#) (const std::string &uri)
- std::string [MessageID](#) (void) const
- bool [hasMessageID](#) (void) const
- void [MessageID](#) (const std::string &uri)
- std::string [RelatesTo](#) (void) const
- bool [hasRelatesTo](#) (void) const
- void [RelatesTo](#) (const std::string &uri)
- std::string [RelationshipType](#) (void) const
- bool [hasRelationshipType](#) (void) const
- void [RelationshipType](#) (const std::string &uri)
- [XMLNode ReferenceParameter](#) (int n)
- [XMLNode ReferenceParameter](#) (const std::string &name)
- [XMLNode NewReferenceParameter](#) (const std::string &name)
- operator [XMLNode](#) (void)

### Static Public Member Functions

- static bool [Check](#) (SOAPEnvelope &soap)

### Protected Attributes

- bool [header\\_allocated\\_](#)

#### 5.180.1 Detailed Description

Interface for manipulation WS-Addressing information in SOAP header.

It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

## 5.180.2 Constructor & Destructor Documentation

### 5.180.2.1 `Arc::WSAHeader::WSAHeader (SOAPEnvelope & soap)`

Linking to a header of existing SOAP message

### 5.180.2.2 `Arc::WSAHeader::WSAHeader (const std::string & action)`

Creating independent SOAP header - not implemented

## 5.180.3 Member Function Documentation

### 5.180.3.1 `void Arc::WSAHeader::Action (const std::string & uri)`

Set content of Action element of SOAP Header. If such element does not exist it's created.

### 5.180.3.2 `std::string Arc::WSAHeader::Action (void) const`

Returns content of Action element of SOAP Header.

### 5.180.3.3 `static bool Arc::WSAHeader::Check (SOAPEnvelope & soap) [static]`

Tells if specified SOAP message has WSA header

### 5.180.3.4 [WSAEndpointReference](#) `Arc::WSAHeader::FaultTo (void)`

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

### 5.180.3.5 [WSAEndpointReference](#) `Arc::WSAHeader::From (void)`

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

### 5.180.3.6 `bool Arc::WSAHeader::hasAction (void) const`

Returns true if Action element is defined.

### 5.180.3.7 `bool Arc::WSAHeader::hasMessageID (void) const`

Returns true if MessageID element is defined.

### 5.180.3.8 `bool Arc::WSAHeader::hasRelatesTo (void) const`

Returns true if RelatesTo element is defined.

**5.180.3.9 bool Arc::WSAHeader::hasRelationshipType (void) const**

Returns true if RelationshipType element is defined.

**5.180.3.10 bool Arc::WSAHeader::hasTo (void) const**

Returns true if To element is defined.

**5.180.3.11 void Arc::WSAHeader::MessageID (const std::string & uri)**

Set content of MessageID element of SOAP Header. If such element does not exist it's created.

**5.180.3.12 std::string Arc::WSAHeader::MessageID (void) const**

Returns content of MessageID element of SOAP Header.

**5.180.3.13 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & name)**

Creates new ReferenceParameter element with specified name. Returns reference to created element.

**5.180.3.14 Arc::WSAHeader::operator XMLNode (void)**

Returns reference to SOAP Header - not implemented

**5.180.3.15 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & name)**

Returns first ReferenceParameter element with specified name

**5.180.3.16 XMLNode Arc::WSAHeader::ReferenceParameter (int n)**

Return n-th ReferenceParameter element

**5.180.3.17 void Arc::WSAHeader::RelatesTo (const std::string & uri)**

Set content of RelatesTo element of SOAP Header. If such element does not exist it's created.

**5.180.3.18 std::string Arc::WSAHeader::RelatesTo (void) const**

Returns content of RelatesTo element of SOAP Header.

**5.180.3.19 void Arc::WSAHeader::RelationshipType (const std::string & uri)**

Set content of RelationshipType element of SOAP Header. If such element does not exist it's created.

**5.180.3.20** `std::string Arc::WSAHeader::RelationshipType (void) const`

Returns content of RelationshipType element of SOAP Header.

**5.180.3.21** `WSAEndpointReference Arc::WSAHeader::ReplyTo (void)`

Returns ReplyTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

**5.180.3.22** `void Arc::WSAHeader::To (const std::string & uri)`

Set content of To element of SOAP Header. If such element does not exist it's created.

**5.180.3.23** `std::string Arc::WSAHeader::To (void) const`

Returns content of To element of SOAP Header.

**5.180.4 Field Documentation****5.180.4.1** `bool Arc::WSAHeader::header_allocated_ [protected]`

SOAP header element

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

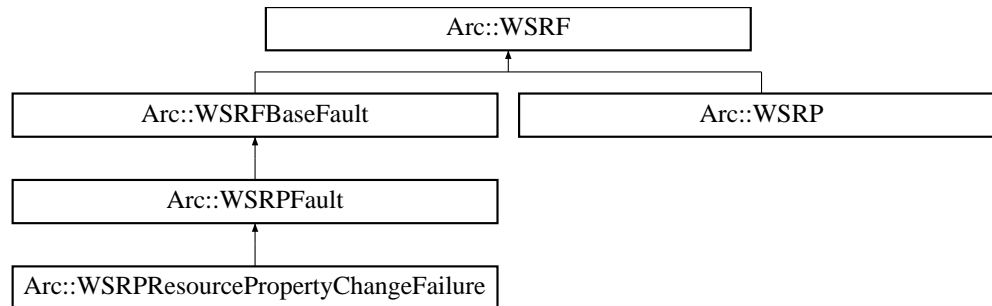
- WSA.h

## 5.181 Arc::WSRF Class Reference

Base class for every [WSRF](#) message.

```
#include <WSRF.h>
```

Inheritance diagram for Arc::WSRF::



### Public Member Functions

- [WSRF](#) (SOAPEnvelope &soap, const std::string &action="")
- [WSRF](#) (bool fault=false, const std::string &action="")
- virtual SOAPEnvelope & [SOAP](#) (void)
- virtual [operator bool](#) (void)

### Protected Member Functions

- void [set\\_namespaces](#) (void)

### Protected Attributes

- bool [allocated\\_](#)
- bool [valid\\_](#)

### 5.181.1 Detailed Description

Base class for every [WSRF](#) message.

This class is not intended to be used directly. Use it like reference while passing through unknown [WSRF](#) message or use classes derived from it.

### 5.181.2 Constructor & Destructor Documentation

#### 5.181.2.1 Arc::WSRF::WSRF (SOAPEnvelope & soap, const std::string & action = "")

Constructor - creates object out of supplied SOAP tree.

**5.181.2.2** `Arc::WSRF::WSRF (bool fault = false, const std::string & action = "")`

Constructor - creates new [WSRF](#) object

**5.181.3 Member Function Documentation****5.181.3.1** `virtual Arc::WSRF::operator bool (void) [inline, virtual]`

Returns true if instance is valid

**5.181.3.2** `void Arc::WSRF::set_namespaces (void) [protected]`

set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in [Arc::WSRP](#), and [Arc::WSRFBBaseFault](#).

**5.181.3.3** `virtual SOAPEnvelope& Arc::WSRF::SOAP (void) [inline, virtual]`

Direct access to underlying SOAP element

**5.181.4 Field Documentation****5.181.4.1** `bool Arc::WSRF::allocated\_ [protected]`

Associated SOAP message - it's SOAP message after all

**5.181.4.2** `bool Arc::WSRF::valid\_ [protected]`

true if soap\_ needs to be deleted in destructor

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

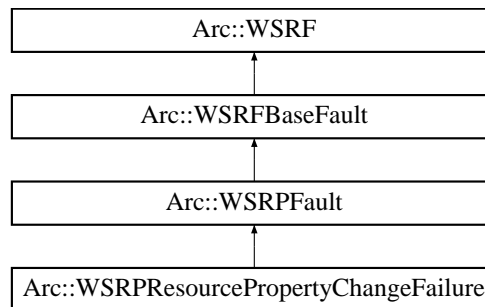
- [WSRF.h](#)

## 5.182 Arc::WSRFBASEFault Class Reference

Base class for [WSRF](#) fault messages.

```
#include <WSRFBASEFault.h>
```

Inheritance diagram for Arc::WSRFBASEFault::



### Public Member Functions

- [WSRFBASEFault](#) (SOAPEnvelope &soap)
- [WSRFBASEFault](#) (const std::string &type)

### Protected Member Functions

- void [set\\_namespaces](#) (void)

### 5.182.1 Detailed Description

Base class for [WSRF](#) fault messages.

Use classes inherited from it for specific faults.

### 5.182.2 Constructor & Destructor Documentation

#### 5.182.2.1 Arc::WSRFBASEFault::WSRFBASEFault (SOAPEnvelope & soap)

Constructor - creates object out of supplied SOAP tree.

#### 5.182.2.2 Arc::WSRFBASEFault::WSRFBASEFault (const std::string & type)

Constructor - creates new [WSRF](#) fault

### 5.182.3 Member Function Documentation

#### 5.182.3.1 void Arc::WSRFBASEFault::set\_namespaces (void) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from [Arc::WSRF](#).

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

- WSRFBaseFault.h

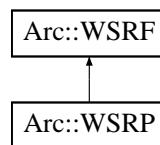


## 5.183 Arc::WSRP Class Reference

Base class for WS-ResourceProperties structures.

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRP::



### Public Member Functions

- [WSRP](#) (bool fault=false, const std::string &action="")
- [WSRP](#) (SOAPEnvelope &soap, const std::string &action="")

### Protected Member Functions

- void [set\\_namespaces](#) (void)

#### 5.183.1 Detailed Description

Base class for WS-ResourceProperties structures.

Inheriting classes implement specific WS-ResourceProperties messages and their properties/elements. Refer to WS-ResourceProperties specifications for things specific to every message.

#### 5.183.2 Constructor & Destructor Documentation

##### 5.183.2.1 Arc::WSRP::WSRP (bool *fault* = false, const std::string & *action* = "")

Constructor - prepares object for creation of new [WSRP](#) request/response/fault

##### 5.183.2.2 Arc::WSRP::WSRP (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree. It does not check if 'soap' represents valid WS-ResourceProperties structure. Actual check for validity of structure has to be done by derived class.

#### 5.183.3 Member Function Documentation

##### 5.183.3.1 void Arc::WSRP::set\_namespaces (void) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

Reimplemented from [Arc::WSRF](#).

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

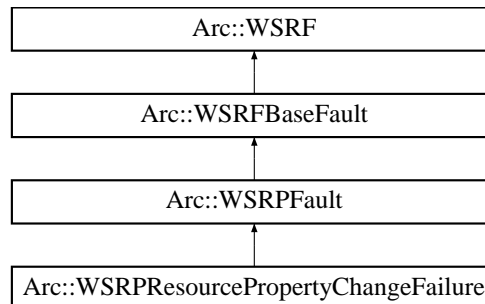
- WSResourceProperties.h

## 5.184 Arc::WSRPFault Class Reference

Base class for WS-ResourceProperties faults.

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRPFault::



### Public Member Functions

- [WSRPFault](#) (SOAPEnvelope &soap)
- [WSRPFault](#) (const std::string &type)

#### 5.184.1 Detailed Description

Base class for WS-ResourceProperties faults.

#### 5.184.2 Constructor & Destructor Documentation

##### 5.184.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & soap)

Constructor - creates object out of supplied SOAP tree.

##### 5.184.2.2 Arc::WSRPFault::WSRPFault (const std::string & type)

Constructor - creates new [WSRP](#) fault

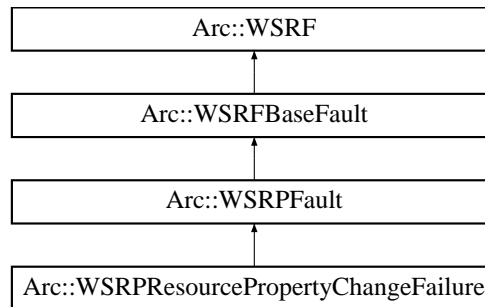
The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 5.185 Arc::WSRPResourcePropertyChangeFailure Class Reference

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::



### Public Member Functions

- [WSRPResourcePropertyChangeFailure](#) (SOAPEnvelope &soap)
- [WSRPResourcePropertyChangeFailure](#) (const std::string &type)

### 5.185.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

### 5.185.2 Constructor & Destructor Documentation

#### 5.185.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & soap) [inline]

Constructor - creates object out of supplied SOAP tree.

#### 5.185.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & type) [inline]

Constructor - creates new [WSRP](#) fault

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

- WSResourceProperties.h

## 5.186 Arc::X509Token Class Reference

Class for manipulating X.509 Token Profile.

```
#include <X509Token.h>
```

### Public Types

- enum [X509TokenType](#)

### Public Member Functions

- [X509Token](#) (SOAPEnvelope &soap, const std::string &keyfile="")
- [X509Token](#) (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, [X509TokenType](#) token\_type=Signature)
- [~X509Token](#) (void)
- [operator bool](#) (void)
- bool [Authenticate](#) (const std::string &cafile, const std::string &capath)
- bool [Authenticate](#) (void)

#### 5.186.1 Detailed Description

Class for manipulating X.509 Token Profile.

This class is for generating/consuming X.509 Token profile. Currently it is used by x509token handler (src/hed/pdc/x509tokensh/) It is not necessary to directly called this class. If we need to use X.509 Token functionality, we only need to configure the x509token handler into service and client.

#### 5.186.2 Member Enumeration Documentation

##### 5.186.2.1 enum [Arc::X509Token::X509TokenType](#)

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constructor.

#### 5.186.3 Constructor & Destructor Documentation

##### 5.186.3.1 [Arc::X509Token::X509Token](#) (SOAPEnvelope & soap, const std::string & keyfile = " ")

Constructor.Parse X509 Token information from SOAP header. X509 Token related information is extracted from SOAP header and stored in class variables. And then it the [X509Token](#) object will be used for authentication if the tokentype is Signature; otherwise if the tokentype is Encryption, the encrypted soap body will be decrypted and replaced by decrypted message. keyfile is only needed when the [X509Token](#) is encryption token

##### 5.186.3.2 [Arc::X509Token::X509Token](#) (SOAPEnvelope & soap, const std::string & certfile, const std::string & keyfile, [X509TokenType](#) token\_type = Signature)

Constructor. Add X509 Token information into the SOAP header. Generated token contains elements X509 token and signature, and is meant to be used for authentication on the consuming side.

**Parameters:**

- soap* The SOAP message to which the X509 Token will be inserted
- certfile* The certificate file which will be used to encrypt the SOAP body (if parameter tokentype is Encryption), or be used as <wsse:BinarySecurityToken/> (if parameter tokentype is Signature).
- keyfile* The key file which will be used to create signature. Not needed when create encryption.
- tokentype* Token type: Signature or Encryption.

**5.186.3.3 Arc::X509Token::~~X509Token (void)**

Deconstructor. Nothing to be done except finalizing the xmlsec library.

**5.186.4 Member Function Documentation****5.186.4.1 bool Arc::X509Token::Authenticate (void)**

Check signature by using the cert information in soap message. Only the signature itself is checked, and it is not guranteed that the certificate which is supposed to check the signature is trusted.

**5.186.4.2 bool Arc::X509Token::Authenticate (const std::string & *cafile*, const std::string & *capath*)**

Check signature by using the certificare information in [X509Token](#) which is parsed by the constructor, and the trusted certificates specified as one of the two parameters. Not only the signature (in the [X509Token](#)) itself is checked, but also the certificate which is supposed to check the signature needs to be trused (which means the certificate is issued by the ca certificate from CA file or CA directory). At least one the the two parameters should be set.

**Parameters:**

- cafile* The CA file
- capath* The CA directory

**Returns:**

true if authentication passes; otherwise false

**5.186.4.3 Arc::X509Token::operator bool (void)**

Returns true of constructor succeeded

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

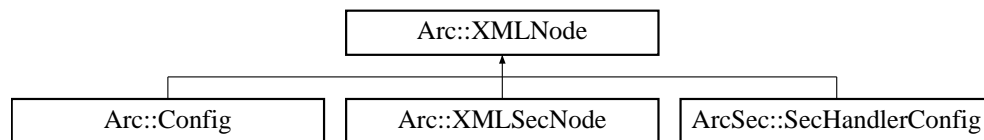
- X509Token.h

## 5.187 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

```
#include <XMLNode.h>
```

Inheritance diagram for Arc::XMLNode::



### Public Member Functions

- [XMLNode](#) (void)
- [XMLNode](#) (const [XMLNode](#) &node)
- [XMLNode](#) (const std::string &xml)
- [XMLNode](#) (const char \*xml, int len=-1)
- [XMLNode](#) (long ptr\_addr)
- [XMLNode](#) (const NS &ns, const char \*name)
- [~XMLNode](#) (void)
- void [New](#) ([XMLNode](#) &node) const
- void [Exchange](#) ([XMLNode](#) &node)
- void [Move](#) ([XMLNode](#) &node)
- void [Swap](#) ([XMLNode](#) &node)
- [operator bool](#) (void) const
- bool [operator!](#) (void) const
- bool [operator==](#) (const [XMLNode](#) &node)
- bool [operator!=](#) (const [XMLNode](#) &node)
- bool [Same](#) (const [XMLNode](#) &node)
- bool [operator==](#) (bool val)
- bool [operator!=](#) (bool val)
- bool [operator==](#) (const std::string &str)
- bool [operator!=](#) (const std::string &str)
- bool [operator==](#) (const char \*str)
- bool [operator!=](#) (const char \*str)
- [XMLNode Child](#) (int n=0)
- [XMLNode operator\[\]](#) (const char \*name) const
- [XMLNode operator\[\]](#) (const std::string &name) const
- [XMLNode operator\[\]](#) (int n) const
- void [operator++](#) (void)
- void [operator--](#) (void)
- int [Size](#) (void) const
- [XMLNode Get](#) (const std::string &name) const
- std::string [Name](#) (void) const
- std::string [Prefix](#) (void) const
- std::string [FullName](#) (void) const
- std::string [Namespace](#) (void) const

- void [Name](#) (const char \*name)
- void [Name](#) (const std::string &name)
- void [GetXML](#) (std::string &out\_xml\_str, bool user\_friendly=false) const
- void [GetXML](#) (std::string &out\_xml\_str, const std::string &encoding, bool user\_friendly=false) const
- void [GetDoc](#) (std::string &out\_xml\_str, bool user\_friendly=false) const
- [operator std::string](#) (void) const
- [XMLNode](#) & [operator=](#) (const char \*content)
- [XMLNode](#) & [operator=](#) (const std::string &content)
- void [Set](#) (const std::string &content)
- [XMLNode](#) & [operator=](#) (const [XMLNode](#) &node)
- [XMLNode](#) [Attribute](#) (int n=0)
- [XMLNode](#) [Attribute](#) (const char \*name)
- [XMLNode](#) [Attribute](#) (const std::string &name)
- [XMLNode](#) [NewAttribute](#) (const char \*name)
- [XMLNode](#) [NewAttribute](#) (const std::string &name)
- int [AttributesSize](#) (void) const
- void [Namespaces](#) (const NS &namespaces, bool keep=false, int recursion=-1)
- NS [Namespaces](#) (void)
- std::string [NamespacePrefix](#) (const char \*urn)
- [XMLNode](#) [NewChild](#) (const char \*name, int n=-1, bool global\_order=false)
- [XMLNode](#) [NewChild](#) (const std::string &name, int n=-1, bool global\_order=false)
- [XMLNode](#) [NewChild](#) (const char \*name, const NS &namespaces, int n=-1, bool global\_order=false)
- [XMLNode](#) [NewChild](#) (const std::string &name, const NS &namespaces, int n=-1, bool global\_order=false)
- [XMLNode](#) [NewChild](#) (const [XMLNode](#) &node, int n=-1, bool global\_order=false)
- void [Replace](#) (const [XMLNode](#) &node)
- void [Destroy](#) (void)
- XMLNodeList [Path](#) (const std::string &path)
- XMLNodeList [XPathLookup](#) (const std::string &xpathExpr, const NS &nsList)
- [XMLNode](#) [GetRoot](#) (void)
- [XMLNode](#) [Parent](#) (void)
- bool [SaveToFile](#) (const std::string &file\_name) const
- bool [SaveToStream](#) (std::ostream &out) const
- bool [ReadFromFile](#) (const std::string &file\_name)
- bool [ReadFromStream](#) (std::istream &in)
- bool [Validate](#) (const std::string &schema\_file, std::string &err\_msg)

## Protected Member Functions

- [XMLNode](#) (xmlNodePtr node)

## Protected Attributes

- bool [is\\_owner\\_](#)
- bool [is\\_temporary\\_](#)



## Friends

- bool [MatchXMLName](#) (const [XMLNode](#) &node1, const [XMLNode](#) &node2)
- bool [MatchXMLName](#) (const [XMLNode](#) &node, const char \*name)
- bool [MatchXMLName](#) (const [XMLNode](#) &node, const std::string &name)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node1, const [XMLNode](#) &node2)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node, const char \*uri)
- bool [MatchXMLNamespace](#) (const [XMLNode](#) &node, const std::string &uri)

### 5.187.1 Detailed Description

Wrapper for LibXML library Tree interface.

This class wraps XML Node, Document and Property/Attribute structures. Each instance serves as pointer to actual LibXML element and provides convenient (for chosen purpose) methods for manipulating it. This class has no special ties to LibXML library and may be easily rewritten for any XML parser which provides interface similar to LibXML Tree. It implements only small subset of XML capabilities, which is probably enough for performing most of useful actions. This class also filters out (usually) useless textual nodes which are often used to make XML documents human-readable.

### 5.187.2 Constructor & Destructor Documentation

#### 5.187.2.1 Arc::XMLNode::XMLNode (xmlNodePtr *node*) [inline, protected]

Private constructor for inherited classes Creates instance and links to existing LibXML structure. Acquired structure is not owned by class instance. If there is need to completely pass control of LibXML document to then instance's `is_owner_` variable has to be set to true.

#### 5.187.2.2 Arc::XMLNode::XMLNode (void) [inline]

Constructor of invalid node Created instance does not point to XML element. All methods are still allowed for such instance but produce no results.

#### 5.187.2.3 Arc::XMLNode::XMLNode (const [XMLNode](#) & *node*) [inline]

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited. Strictly speaking it should be no const here - but that conflicts with C++.

#### 5.187.2.4 Arc::XMLNode::XMLNode (const std::string & *xml*)

Creates XML document structure from textual representation of XML document. Created structure is pointed and owned by constructed instance

#### 5.187.2.5 Arc::XMLNode::XMLNode (const char \* *xml*, int *len* = -1)

Same as previous

**5.187.2.6 Arc::XMLNode::XMLNode (long ptr\_addr)**

Copy constructor. Used by language bindings

**5.187.2.7 Arc::XMLNode::XMLNode (const NS & ns, const char \* name)**

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

**5.187.2.8 Arc::XMLNode::~~XMLNode (void)**

Destructor Also destroys underlying XML document if owned by this instance

**5.187.3 Member Function Documentation****5.187.3.1 XMLNode Arc::XMLNode::Attribute (const std::string & name) [inline]**

Returns XMLNode instance representing first attribute of node with specified by name

**5.187.3.2 XMLNode Arc::XMLNode::Attribute (const char \* name)**

Returns XMLNode instance representing first attribute of node with specified by name

**5.187.3.3 XMLNode Arc::XMLNode::Attribute (int n = 0)**

Returns list of all attributes of node.

Returns XMLNode instance representing n-th attribute of node.

**5.187.3.4 int Arc::XMLNode::AttributesSize (void) const**

Returns number of attributes of node

**5.187.3.5 XMLNode Arc::XMLNode::Child (int n = 0)**

Returns XMLNode instance representing n-th child of XML element. If such does not exist invalid XMLNode instance is returned

**5.187.3.6 void Arc::XMLNode::Destroy (void)**

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation XMLNode instance becomes invalid

**5.187.3.7 void Arc::XMLNode::Exchange (XMLNode & node)**

Exchanges XML (sub)trees. Following combinations are possible If either this or node are referring owned XML tree (top level node) then references are simply exchanged. This operation is fast. If both this

and node are referring to XML (sub)tree of different documents then (sub)trees are exchanged between documents. If both this and node are referring to XML (sub)tree of same document then (sub)trees are moved inside document. The main reason for this method is to provide effective way to insert one XML document inside another. One should take into account that if any of exchanged nodes is top level it must be also owner of document. Otherwise method will fail. If both nodes are top level owners and/or invalid nodes then this method is identical to [Swap\(\)](#).

#### 5.187.3.8 `std::string Arc::XMLNode::FullName (void) const` [inline]

Returns prefix:name of XML node

#### 5.187.3.9 `XMLNode Arc::XMLNode::Get (const std::string & name) const` [inline]

Same as operator[]

#### 5.187.3.10 `void Arc::XMLNode::GetDoc (std::string & out_xml_str, bool user_friendly = false) const`

Fills argument with whole XML document textual representation

#### 5.187.3.11 `XMLNode Arc::XMLNode::GetRoot (void)`

Get the root node from any child node of the tree

#### 5.187.3.12 `void Arc::XMLNode::GetXML (std::string & out_xml_str, const std::string & encoding, bool user_friendly = false) const`

Fills argument with this instance XML subtree textual representation if the XML subtree is corresponding to the encoding format specified in the argument, e.g. utf-8

#### 5.187.3.13 `void Arc::XMLNode::GetXML (std::string & out_xml_str, bool user_friendly = false) const`

Fills argument with this instance XML subtree textual representation

#### 5.187.3.14 `void Arc::XMLNode::Move (XMLNode & node)`

Moves content of this XML (sub)tree to node This operation is similar to New except that XML (sub)tree to referred by this is destroyed. This method is more effective than combination of [New\(\)](#) and [Destroy\(\)](#) because internally it is optimized not to copy data if not needed. The main purpose of this is to effectively extract part of XML document.

#### 5.187.3.15 `void Arc::XMLNode::Name (const std::string & name)` [inline]

Assigns new name to XML node

**5.187.3.16 void Arc::XMLNode::Name (const char \* *name*)**

Assigns new name to XML node

**5.187.3.17 std::string Arc::XMLNode::Name (void) const**

Returns name of XML node

**5.187.3.18 std::string Arc::XMLNode::Namespace (void) const**

Returns namespace URI of XML node

**5.187.3.19 std::string Arc::XMLNode::NamespacePrefix (const char \* *urn*)**

Returns prefix of specified namespace. Empty string if no such namespace.

**5.187.3.20 NS Arc::XMLNode::Namespaces (void)**

Returns namespaces known at this node

**5.187.3.21 void Arc::XMLNode::Namespaces (const NS & *namespaces*, bool *keep* = false, int *recursion* = -1)**

Assigns namespaces of XML document at point specified by this instance. If namespace already exists it gets new prefix. New namespaces are added. It is useful to apply this method to XML being processed in order to refer to it's elements by known prefix. If keep is set to false existing namespace definition residing at this instance and below are removed (default behavior). If recursion is set to positive number then depth of prefix replacement is limited by this number (0 limits it to this node only). For unlimited recursion use -1. If recursion is limited then value of keep is ignored and existing namespaces are always kept.

**5.187.3.22 void Arc::XMLNode::New ([XMLNode](#) & *node*) const**

Creates a copy of XML (sub)tree. If object does not represent whole document - top level document is created. 'node' becomes a pointer owning new XML document.

**5.187.3.23 [XMLNode](#) Arc::XMLNode::NewAttribute (const std::string & *name*) [inline]**

Creates new attribute with specified name.

**5.187.3.24 [XMLNode](#) Arc::XMLNode::NewAttribute (const char \* *name*)**

Creates new attribute with specified name.

**5.187.3.25 [XMLNode](#) Arc::XMLNode::NewChild (const [XMLNode](#) & *node*, int *n* = -1, bool *global\_order* = false)**

Link a copy of supplied XML node as child. Returns instance referring to new child. XML element is a copy of supplied one but not owned by returned instance

**5.187.3.26** [XMLNode](#) Arc::XMLNode::NewChild (const std::string & *name*, const NS & *namespaces*, int *n* = -1, bool *global\_order* = false) [inline]

Same as [NewChild\(const char\\*,const NS&,int,bool\)](#)

**5.187.3.27** [XMLNode](#) Arc::XMLNode::NewChild (const char \* *name*, const NS & *namespaces*, int *n* = -1, bool *global\_order* = false)

Creates new child XML element at specified position with specified name and namespaces. For more information look at [NewChild\(const char\\*,int,bool\)](#)

**5.187.3.28** [XMLNode](#) Arc::XMLNode::NewChild (const std::string & *name*, int *n* = -1, bool *global\_order* = false) [inline]

Same as [NewChild\(const char\\*,int,bool\)](#)

**5.187.3.29** [XMLNode](#) Arc::XMLNode::NewChild (const char \* *name*, int *n* = -1, bool *global\_order* = false)

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name. Returns created node.

**5.187.3.30** Arc::XMLNode::operator bool (void) const [inline]

Returns true if instance points to XML element - valid instance

**5.187.3.31** Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

**5.187.3.32** bool Arc::XMLNode::operator! (void) const [inline]

Returns true if instance does not point to XML element - invalid instance

**5.187.3.33** bool Arc::XMLNode::operator!= (const char \* *str*) [inline]

This operator is needed to avoid ambiguity

**5.187.3.34** bool Arc::XMLNode::operator!= (const std::string & *str*) [inline]

This operator is needed to avoid ambiguity

**5.187.3.35** bool Arc::XMLNode::operator!= (bool *val*) [inline]

This operator is needed to avoid ambiguity

**5.187.3.36** `bool Arc::XMLNode::operator!=(const XMLNode & node)` [inline]

Returns false if 'node' represents same XML element

**5.187.3.37** `void Arc::XMLNode::operator++ (void)`

Convenience operator to switch to next element of same name. If there is no such node this object becomes invalid.

**5.187.3.38** `void Arc::XMLNode::operator-- (void)`

Convenience operator to switch to previous element of same name. If there is no such node this object becomes invalid.

**5.187.3.39** `XMLNode& Arc::XMLNode::operator= (const XMLNode & node)`

Make instance refer to another XML node. Ownership is not inherited. Due to nature of XMLNode there should be no const here, but that does not fit into C++.

**5.187.3.40** `XMLNode& Arc::XMLNode::operator= (const std::string & content)` [inline]

Sets textual content of node. All existing children nodes are discarded.

**5.187.3.41** `XMLNode& Arc::XMLNode::operator= (const char * content)`

Sets textual content of node. All existing children nodes are discarded.

**5.187.3.42** `bool Arc::XMLNode::operator== (const char * str)` [inline]

This operator is needed to avoid ambiguity

**5.187.3.43** `bool Arc::XMLNode::operator== (const std::string & str)` [inline]

This operator is needed to avoid ambiguity

**5.187.3.44** `bool Arc::XMLNode::operator== (bool val)` [inline]

This operator is needed to avoid ambiguity

**5.187.3.45** `bool Arc::XMLNode::operator== (const XMLNode & node)` [inline]

Returns true if 'node' represents same XML element

**5.187.3.46** ]

[XMLNode](#) Arc::XMLNode::operator[ ] (int *n*) const

Returns [XMLNode](#) instance representing *n*-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like `node["name"][5]`. This method should not be marked const because obtaining unrestricted [XMLNode](#) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

**5.187.3.47** ]

[XMLNode](#) Arc::XMLNode::operator[ ] (const std::string & *name*) const [inline]

Similar to previous method

**5.187.3.48** ]

[XMLNode](#) Arc::XMLNode::operator[ ] (const char \* *name*) const

Returns [XMLNode](#) instance representing first child element with specified name. Name may be "namespace\_prefix:name", "namespace\_uri:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid [XMLNode](#) instance is returned. This method should not be marked const because obtaining unrestricted [XMLNode](#) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

**5.187.3.49** [XMLNode](#) Arc::XMLNode::Parent (void)

Get the parent node from any child node of the tree

**5.187.3.50** XMLNodeList Arc::XMLNode::Path (const std::string & *path*)

Collects nodes corresponding to specified path. This is a convenience function to cover common use of XPath but without performance hit. Path is made of `node_name[/node_name[...]]` and is relative to current node. `node_names` are treated in same way as in `operator[ ]`. Returns all nodes which are represented by path.

**5.187.3.51** std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

**5.187.3.52** bool Arc::XMLNode::ReadFromFile (const std::string & *file\_name*)

Read XML document from file and associate it with this node

**5.187.3.53** bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

**5.187.3.54 void Arc::XMLNode::Replace (const XMLNode & node)**

Makes a copy of supplied XML node and makes this instance refer to it

**5.187.3.55 bool Arc::XMLNode::Same (const XMLNode & node) [inline]**

Returns true if 'node' represents same XML element - for bindings

**5.187.3.56 bool Arc::XMLNode::SaveToFile (const std::string & file\_name) const**

Save string representation of node to file

**5.187.3.57 bool Arc::XMLNode::SaveToStream (std::ostream & out) const**

Save string representation of node to stream

**5.187.3.58 void Arc::XMLNode::Set (const std::string & content) [inline]**

Same as operator=. Used for bindings.

**5.187.3.59 int Arc::XMLNode::Size (void) const**

Returns number of children nodes

**5.187.3.60 void Arc::XMLNode::Swap (XMLNode & node)**

Swaps XML (sub)trees to this this and node refer. For XML subtrees this method is not anyhow different then using combination XMLNode tmp=\*this; \*this=node; node=tmp; But in case of either this or node owning XML document ownership is swapped too. And this is a main purpose of Swap() method.

**5.187.3.61 bool Arc::XMLNode::Validate (const std::string & schema\_file, std::string & err\_msg)**

XML schema validation against the schema file defined as argument

**5.187.3.62 XMLNodeList Arc::XMLNode::XPathLookup (const std::string & xpathExpr, const NS & nsList)**

Uses xPath to look up the whole xml structure, Returns a list of XMLNode points. The xpathExpr should be like "//xx:child1/" which indicates the namespace and node that you would like to find; The nsList is the namespace the result should belong to (e.g. xx="uri:test"). Query is run on whole XML document but only the elements belonging to this XML subtree are returned.

**5.187.4 Friends And Related Function Documentation****5.187.4.1 bool MatchXMLName (const XMLNode & node, const std::string & name) [friend]**

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too



**5.187.4.2** `bool MatchXMLName (const XMLNode & node, const char * name)` [friend]

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**5.187.4.3** `bool MatchXMLName (const XMLNode & node1, const XMLNode & node2)`  
[friend]

Returns true if underlying XML elements have same names

**5.187.4.4** `bool MatchXMLNamespace (const XMLNode & node, const std::string & uri)`  
[friend]

Returns true if 'namespace' matches 'node's namespace.

**5.187.4.5** `bool MatchXMLNamespace (const XMLNode & node, const char * uri)` [friend]

Returns true if 'namespace' matches 'node's namespace.

**5.187.4.6** `bool MatchXMLNamespace (const XMLNode & node1, const XMLNode & node2)`  
[friend]

Returns true if underlying XML elements belong to same namespaces

**5.187.5** **Field Documentation****5.187.5.1** `bool Arc::XMLNode::is_owner_` [protected]

If true node is owned by this instance - hence released in destructor. Normally that may be true only for top level node of XML document.

**5.187.5.2** `bool Arc::XMLNode::is_temporary_` [protected]

This variable is for future

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

- XMLNode.h

## 5.188 Arc::XMLNodeContainer Class Reference

```
#include <XMLNode.h>
```

### Public Member Functions

- [XMLNodeContainer](#) (void)
- [XMLNodeContainer](#) (const [XMLNodeContainer](#) &)
- [XMLNodeContainer](#) & [operator=](#) (const [XMLNodeContainer](#) &)
- void [Add](#) (const [XMLNode](#) &)
- void [Add](#) (const std::list< [XMLNode](#) > &)
- void [AddNew](#) (const [XMLNode](#) &)
- void [AddNew](#) (const std::list< [XMLNode](#) > &)
- int [Size](#) (void) const
- [XMLNode](#) [operator\[\]](#) (int)
- std::list< [XMLNode](#) > [Nodes](#) (void)

### 5.188.1 Detailed Description

Container for multiple [XMLNode](#) elements

### 5.188.2 Constructor & Destructor Documentation

#### 5.188.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

#### 5.188.2.2 Arc::XMLNodeContainer::XMLNodeContainer (const [XMLNodeContainer](#) &)

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using [AddNew\(\)](#). Not owning nodes are linked using [Add\(\)](#) method.

### 5.188.3 Member Function Documentation

#### 5.188.3.1 void Arc::XMLNodeContainer::Add (const std::list< [XMLNode](#) > &)

Link multiple XML subtrees to container.

#### 5.188.3.2 void Arc::XMLNodeContainer::Add (const [XMLNode](#) &)

Link XML subtree referred by node to container. XML tree must be available as long as this object is used.

#### 5.188.3.3 void Arc::XMLNodeContainer::AddNew (const std::list< [XMLNode](#) > &)

Copy multiple XML subtrees to container.

**5.188.3.4 void Arc::XMLNodeContainer::AddNew (const XMLNode &)**

Copy XML subtree referenced by node to container. After this operation container refers to independent XML document. This document is deleted when container is destroyed.

**5.188.3.5 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)**

Returns all stored nodes.

**5.188.3.6 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)**

Same as copy constructor with current nodes being deleted first.

**5.188.3.7 ]**

XMLNode Arc::XMLNodeContainer::operator[ ] (int)

Returns n-th node in a store.

**5.188.3.8 int Arc::XMLNodeContainer::Size (void) const**

Return number of refered/stored nodes.

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

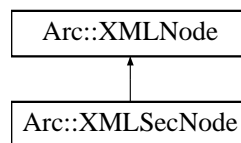
- XMLNode.h

## 5.189 Arc::XMLSecNode Class Reference

Extends [XMLNode](#) class to support XML security operation.

```
#include <XMLSecNode.h>
```

Inheritance diagram for Arc::XMLSecNode::



### Public Member Functions

- [XMLSecNode](#) ([XMLNode](#) &node)
- void [AddSignatureTemplate](#) (const std::string &id\_name, const SignatureMethod sign\_method, const std::string &incl\_namespaces="")
- bool [SignNode](#) (const std::string &privkey\_file, const std::string &cert\_file)
- bool [VerifyNode](#) (const std::string &id\_name, const std::string &ca\_file, const std::string &ca\_path, bool verify\_trusted=true)
- bool [EncryptNode](#) (const std::string &cert\_file, const SymEncryptionType encript\_type)
- bool [DecryptNode](#) (const std::string &privkey\_file, [XMLNode](#) &decrypted\_node)

### 5.189.1 Detailed Description

Extends [XMLNode](#) class to support XML security operation.

All [XMLNode](#) methods are exposed by inheriting from [XMLNode](#). [XMLSecNode](#) itself does not own node, instead it uses the node from the base class [XMLNode](#).

### 5.189.2 Constructor & Destructor Documentation

#### 5.189.2.1 Arc::XMLSecNode::XMLSecNode ([XMLNode](#) & node)

Create a object based on an [XMLNode](#) instance.

### 5.189.3 Member Function Documentation

#### 5.189.3.1 void Arc::XMLSecNode::AddSignatureTemplate (const std::string & id\_name, const SignatureMethod sign\_method, const std::string & incl\_namespaces = "")

Add the signature template for later signing.

#### Parameters:

*id\_name* The identifier name under this node which will be used for the <Signature> to refer to.

*sign\_method* The sign method for signing. Two options now, RSA\_SHA1, DSA\_SHA1

### 5.189.3.2 **bool Arc::XMLSecNode::DecryptNode (const std::string & *privkey\_file*, XMLNode & *decrypted\_node*)**

Decrypt the <xenc:EncryptedData/> under this node, the decrypted node will be output in the second argument of DecryptNode method. And the <xenc:EncryptedData/> under this node will be removed after decryption.

#### Parameters:

*privkey\_file* The private key file, which is used for decrypting  
*decrypted\_node* Output the decrypted node

### 5.189.3.3 **bool Arc::XMLSecNode::EncryptNode (const std::string & *cert\_file*, const SymEncryptionType *encrypt\_type*)**

Encrypt this node, after encryption, this node will be replaced by the encrypted node

#### Parameters:

*cert\_file* The certificate file, the public key parsed from this certificate is used to encrypted the symmetric key, and then the symmetric key is used to encrypted the node  
*encrypt\_type* The encryption type when encrypting the node, four option in SymEncryptionType  
*verify\_trusted* Verify trusted certificates or not. If set to false, then only the signature will be checked (by using the public key from KeyInfo).

### 5.189.3.4 **bool Arc::XMLSecNode::SignNode (const std::string & *privkey\_file*, const std::string & *cert\_file*)**

Sign this node (identified by id\_name).

#### Parameters:

*privkey\_file* The private key file. The private key is used for signing  
*cert\_file* The certificate file. The certificate is used as the <KeyInfo> part of the <Signature>; <Key-Info> will be used for the other end to verify this <Signature>  
*incl\_namespaces* InclusiveNamespaces for Transform in Signature

### 5.189.3.5 **bool Arc::XMLSecNode::VerifyNode (const std::string & *id\_name*, const std::string & *ca\_file*, const std::string & *ca\_path*, bool *verify\_trusted* = true)**

Verify the signature under this node

#### Parameters:

*id\_name* The id of this node, which is used for identifying the node  
*ca\_file* The CA file which used as trusted certificate when verify the certificate in the <KeyInfo> part of <Signature>  
*ca\_path* The CA directory; either ca\_file or ca\_path should be set.

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

- XMLSecNode.h

# Index

- ~AutoPointer
  - Arc::AutoPointer, [66](#)
- ~Counter
  - Arc::Counter, [95](#)
- ~DTR
  - DataStaging::DTR, [151](#)
- ~DTRCallback
  - DataStaging::DTRCallback, [162](#)
- ~DataDelivery
  - DataStaging::DataDelivery, [119](#)
- ~DataDeliveryComm
  - DataStaging::DataDeliveryComm, [123](#)
- ~DataDeliveryLocalComm
  - DataStaging::DataDeliveryLocalComm, [129](#)
- ~Database
  - Arc::Database, [116](#)
- ~IntraProcessCounter
  - Arc::IntraProcessCounter, [217](#)
- ~JobControllerPluginLoader
  - Arc::JobControllerPluginLoader, [227](#)
- ~JobDescriptionParserLoader
  - Arc::JobDescriptionParserLoader, [233](#)
- ~Loader
  - Arc::Loader, [245](#)
- ~Logger
  - Arc::Logger, [252](#)
- ~MCCLoader
  - Arc::MCCLoader, [270](#)
- ~Message
  - Arc::Message, [276](#)
- ~PayloadRaw
  - Arc::PayloadRaw, [298](#)
- ~PayloadStream
  - Arc::PayloadStream, [303](#)
- ~Plexer
  - Arc::Plexer, [315](#)
- ~Processor
  - DataStaging::Processor, [330](#)
- ~RegularExpression
  - Arc::RegularExpression, [333](#)
- ~Run
  - Arc::Run, [343](#)
- ~SAMLToken
  - Arc::SAMLToken, [347](#)
- ~SOAPMessage
  - Arc::SOAPMessage, [368](#)
- ~Scheduler
  - DataStaging::Scheduler, [349](#)
- ~SubmitterPluginLoader
  - Arc::SubmitterPluginLoader, [391](#)
- ~TransferShares
  - DataStaging::TransferShares, [406](#)
- ~URL
  - Arc::URL, [414](#)
- ~URLLocation
  - Arc::URLLocation, [423](#)
- ~WSAEndpointReference
  - Arc::WSAEndpointReference, [455](#)
- ~X509Token
  - Arc::X509Token, [470](#)
- ~XMLNode
  - Arc::XMLNode, [474](#)
- Abandon
  - Arc::Run, [343](#)
- Acquire
  - Arc::DelegationConsumer, [136](#)
  - Arc::InformationContainer, [209](#)
- acquire
  - Arc::FileLock, [197](#)
- Action
  - Arc::WSAHeader, [458](#)
- active\_shares
  - DataStaging::TransferShares, [407](#)
- ActivityOldID
  - Arc::JobIdentificationType, [235](#)
- Add
  - Arc::MessageContext, [284](#)
  - Arc::XMLNodeContainer, [482](#)
  - DataStaging::DataDeliveryCommHandler, [128](#)
- add
  - Arc::Adler32Sum, [50](#)
  - Arc::Checksum, [71](#)
  - Arc::ChecksumAny, [74](#)
  - Arc::CRC32Sum, [102](#)
  - Arc::MD5Sum, [272](#)
  - Arc::MessageAttributes, [279](#)
  - Arc::SoftwareRequirement, [379](#), [380](#)
- add\_dtr

- DataStaging::DTRLList, 166
- add\_problematic\_delivery\_service
  - DataStaging::DTR, 151
- AddBartender
  - Arc::UserConfig, 428
- AddCAdir
  - Arc::BaseConfig, 68
- AddCAFile
  - Arc::BaseConfig, 68
- AddCertExtObj
  - Arc::Credential, 108
- AddCertificate
  - Arc::BaseConfig, 68
- AddChain
  - Arc::VOMSTrustList, 454
- AddConsumer
  - Arc::DelegationContainerSOAP, 139
- addDestination
  - Arc::Logger, 252
- addDestinations
  - Arc::Logger, 252
- AddExtension
  - Arc::Credential, 108
- AddHTTPOption
  - Arc::URL, 414
- AddJob
  - Arc::JobSupervisor, 239
- AddLDAPAttribute
  - Arc::URL, 414
- AddLocation
  - Arc::URL, 414
- AddMetaDataOption
  - Arc::URL, 414
- AddNew
  - Arc::XMLNodeContainer, 482
- AddOption
  - Arc::URL, 414
- AddOverlay
  - Arc::BaseConfig, 68
- AddPluginsPath
  - Arc::BaseConfig, 68
- addPolicy
  - ArcSec::Evaluator, 180
  - ArcSec::Policy, 326
- AddPrivateKey
  - Arc::BaseConfig, 68
- AddProxy
  - Arc::BaseConfig, 68
- AddRegex
  - Arc::VOMSTrustList, 454
- addRegister
  - Arc::InfoRegisters, 207
- addRegistrar
  - Arc::InfoRegisterContainer, 206
- addRequestItem
  - ArcSec::Request, 337
- Address
  - Arc::WSAEndpointReference, 456
- AddSecHandler
  - Arc::ClientSOAP, 81
  - Arc::MCC, 263
  - Arc::Service, 362
- addService
  - Arc::InfoRegisterContainer, 206
  - Arc::InfoRegistrar, 208
- AddSignatureTemplate
  - Arc::XMLSecNode, 484
- AddURLMapping
  - DataStaging::Scheduler, 350
- addVOMSAC
  - Arc, 39
- AfterFork
  - Arc::Run, 343
- all\_jobs
  - DataStaging::DTRLList, 166
- allocated\_
  - Arc::WSRF, 462
- Annotation
  - Arc::JobIdentificationType, 235
- ApplyToConfig
  - Arc::UserConfig, 428
- approveCSR
  - Arc::OAuthConsumer, 294
- Arc, 15
  - addVOMSAC, 39
  - AttrConstIter, 28
  - AttrIter, 28
  - AttrMap, 28
  - booltostr, 35
  - BUSY\_ERROR, 30
  - CanonicalDir, 33
  - ContentFromPayload, 41
  - convert\_to\_rdn, 36
  - CreateThreadFunction, 37
  - createVOMSAC, 38
  - CredentialLogger, 43
  - DirCreate, 32
  - DirDelete, 32
  - EnvLockUnwrap, 38
  - EnvLockUnwrapComplete, 38
  - EnvLockWrap, 37
  - escape\_chars, 37
  - escape\_hex, 29
  - escape\_octal, 29
  - escape\_type, 29
  - ETERNAL, 43
  - FileCopy, 30, 31
  - FileCreate, 31

- FileDelete, 32
- FileLink, 31
- FileRead, 31
- FileReadLink, 31, 32
- FileStat, 31
- final\_xmlsec, 41
- GENERIC\_ERROR, 29
- get\_cert\_str, 41
- get\_key\_from\_certfile, 42
- get\_key\_from\_certstr, 42
- get\_key\_from\_keyfile, 42
- get\_key\_from\_keystr, 42
- get\_node, 43
- get\_plugin\_instance, 28
- get\_token, 36
- getCredentialProperty, 40
- GetEnv, 37
- GUID, 33
- HandleOpenSSLError, 40, 41
- HISTORIC, 43
- init\_xmlsec, 41
- inttostr, 35
- istring\_to\_level, 33
- level\_to\_string, 34
- load\_key\_from\_certfile, 42
- load\_key\_from\_certstr, 42
- load\_key\_from\_keyfile, 42
- load\_trusted\_cert\_file, 42
- load\_trusted\_cert\_str, 42
- load\_trusted\_certs, 42
- LogFormat, 29
- LogLevel, 29
- lower, 35
- MatchXMLName, 38
- MatchXMLNamespace, 38
- old\_level\_to\_level, 34
- OpenSSLInit, 40
- operator<=, 30, 33
- parseVOMSAC, 39, 40
- PARSING\_ERROR, 30
- passphrase\_callback, 41
- plugins\_table\_name, 43
- PROTOCOL\_RECOGNIZED\_ERROR, 30
- ReadURLList, 37
- SESSION\_CLOSE, 30
- SetEnv, 37
- STATUS\_OK, 29
- StatusKind, 29
- StrError, 38
- string, 41
- string\_to\_level, 33
- stringto, 34
- strip, 36
- strtobool, 35
- strtoint, 34
- thread\_stacksize, 43
- TimeFormat, 29
- TimeStamp, 30
- TmpDirCreate, 32
- TmpFileCreate, 32
- tokenize, 36
- tostring, 34
- trim, 36
- unescape\_chars, 37
- UNKNOWN\_SERVICE\_ERROR, 30
- UnsetEnv, 37
- upper, 36
- uri\_encode, 36
- uri\_unencode, 36
- UUID, 33
- VOMSDecode, 40
- WSAFault, 30
- WSAFaultAssign, 41
- WSAFaultExtract, 41
- WSAFaultInvalidAddressingHeader, 30
- WSAFaultUnknown, 30
- Arc::Adler32Sum, 49
  - add, 50
  - end, 50
  - operator bool, 50
  - operator!, 50
  - print, 50
  - result, 50
  - scan, 51
  - start, 51
- Arc::ApplicationEnvironment, 53
- Arc::ArcLocation, 54
- Arc::ArcLocation
  - Get, 54
  - GetPlugins, 54
  - Init, 54
- Arc::ArcVersion, 55
- Arc::AttributeIterator, 58
- Arc::AttributeIterator
  - AttributeIterator, 58
  - current\_, 60
  - end\_, 60
  - hasMore, 59
  - key, 59
  - MessageAttributes, 60
  - operator \*, 59
  - operator++, 59
  - operator->, 60
- Arc::AutoPointer, 66
- Arc::AutoPointer
  - ~AutoPointer, 66
  - AutoPointer, 66
  - operator \*, 66



- operator bool, 66
- operator!, 67
- operator->, 67
- Ptr, 67
- Release, 67
- Arc::BaseConfig, 68
- Arc::BaseConfig
  - AddCAGDir, 68
  - AddCAFile, 68
  - AddCertificate, 68
  - AddOverlay, 68
  - AddPluginsPath, 68
  - AddPrivateKey, 68
  - AddProxy, 68
  - GetOverlay, 69
  - MakeConfig, 69
- Arc::ChainContext, 70
- Arc::ChainContext
  - operator PluginsFactory \*, 70
- Arc::Checksum, 71
- Arc::Checksum
  - add, 71
  - Checksum, 71
  - end, 72
  - operator bool, 72
  - operator!, 72
  - print, 72
  - result, 72
  - scan, 72
  - start, 73
- Arc::ChecksumAny, 74
- Arc::ChecksumAny
  - add, 74
  - end, 74
  - FileChecksum, 75
  - operator bool, 75
  - operator!, 75
  - print, 75
  - result, 75
  - scan, 76
  - start, 76
- Arc::CStringValue, 77
- Arc::CStringValue
  - CStringValue, 77
  - equal, 78
  - operator bool, 78
- Arc::ClientHTTP, 79
- Arc::ClientInterface, 80
- Arc::ClientSOAP, 81
- Arc::ClientSOAP
  - AddSecHandler, 81
  - ClientSOAP, 81
  - GetEntry, 81
  - Load, 82
  - process, 82
- Arc::ClientTCP, 83
- Arc::Config, 86
  - Config, 86, 87
  - getFileName, 87
  - parse, 87
  - print, 87
  - save, 87
  - setFileName, 87
- Arc::ConfusaCertHandler, 88
- Arc::ConfusaCertHandler
  - ConfusaCertHandler, 88
  - createCertRequest, 88
  - getCertRequestB64, 88
- Arc::ConfusaParserUtils, 89
- Arc::ConfusaParserUtils
  - destroy\_doc, 89
  - evaluate\_path, 89
  - extract\_body\_information, 89
  - get\_doc, 89
  - handle\_redirect\_step, 89
  - urlencode, 90
  - urlencode\_params, 90
- Arc::CountedPointer, 91
- Arc::CountedPointer
  - operator \*, 91
  - operator bool, 91
  - operator!, 91
  - operator!=, 91
  - operator->, 92
  - operator<, 92
  - operator==, 92
  - Ptr, 92
  - Release, 92
- Arc::Counter, 93
  - ~Counter, 95
  - cancel, 95
  - changeExcess, 95
  - changeLimit, 95
  - Counter, 95
  - CounterTicket, 99
  - ExpirationReminder, 99
  - extend, 96
  - getCounterTicket, 96
  - getcurrentTime, 96
  - getExcess, 97
  - getExpirationReminder, 97
  - getExpiryTime, 97
  - getLimit, 97
  - getValue, 98
  - IDType, 95
  - reserve, 98
  - setExcess, 98
  - setLimit, 99

- Arc::CounterTicket, 100
- Arc::CounterTicket
  - cancel, 100
  - Counter, 101
  - CounterTicket, 100
  - extend, 101
  - isValid, 101
- Arc::CRC32Sum, 102
  - add, 102
  - end, 102
  - operator bool, 103
  - operator!, 103
  - print, 103
  - result, 103
  - scan, 103
  - start, 104
- Arc::Credential, 105
  - AddCertExtObj, 108
  - AddExtension, 108
  - Credential, 106, 107
  - GenerateEECRequest, 108
  - GenerateRequest, 109
  - GetCAName, 109
  - GetCert, 109
  - GetCertNumofChain, 109
  - GetCertReq, 109
  - GetDN, 109
  - GetEndTime, 109
  - GetExtension, 109
  - getFormat\_BIO, 110
  - GetIdentityName, 110
  - GetIssuerName, 110
  - GetLifeTime, 110
  - GetPrivKey, 110
  - GetProxyPolicy, 110
  - GetPubKey, 110
  - GetStartTime, 110
  - GetType, 110
  - GetVerification, 110
  - InitProxyCertInfo, 110
  - InquireRequest, 111
  - IsCredentialsValid, 111
  - IsValid, 111
  - LogError, 111
  - OutputCertificate, 111
  - OutputCertificateChain, 111
  - OutputPrivatekey, 112
  - OutputPublickey, 112
  - SelfSignEECRequest, 112
  - SetLifeTime, 112
  - SetProxyPolicy, 112
  - SetStartTime, 112
  - SignEECRequest, 112, 113
  - SignRequest, 113
  - STACK\_OF, 113
- Arc::CredentialError, 114
- Arc::CredentialError
  - CredentialError, 114
- Arc::CredentialStore, 115
- Arc::Database, 116
  - ~Database, 116
  - close, 117
  - connect, 117
  - Database, 116
  - enable\_ssl, 117
  - isconnected, 117
  - shutdown, 117
- Arc::DelegationConsumer, 135
- Arc::DelegationConsumer
  - Acquire, 136
  - Backup, 136
  - DelegationConsumer, 135
  - Generate, 136
  - ID, 136
  - LogError, 136
  - Request, 136
  - Restore, 136
- Arc::DelegationConsumerSOAP, 137
- Arc::DelegationConsumerSOAP
  - DelegateCredentialsInit, 137
  - DelegatedToken, 137
  - DelegationConsumerSOAP, 137
  - UpdateCredentials, 138
- Arc::DelegationContainerSOAP, 139
- Arc::DelegationContainerSOAP
  - AddConsumer, 139
  - CheckConsumers, 139
  - context\_lock\_, 140
  - DelegateCredentialsInit, 139
  - DelegatedToken, 140
  - FindConsumer, 140
  - max\_duration\_, 140
  - max\_size\_, 141
  - max\_usage\_, 141
  - QueryConsumer, 140
  - ReleaseConsumer, 140
  - RemoveConsumer, 140
  - TouchConsumer, 140
  - UpdateCredentials, 140
- Arc::DelegationProvider, 142
- Arc::DelegationProvider
  - Delegate, 142
  - DelegationProvider, 142
- Arc::DelegationProviderSOAP, 144
- Arc::DelegationProviderSOAP
  - DelegateCredentialsInit, 145
  - DelegatedToken, 145
  - DelegationProviderSOAP, 144

- ID, 145
- UpdateCredentials, 145
- Arc::ExecutableType, 186
- Arc::ExecutableType
  - Argument, 186
  - Path, 186
  - SuccessExitCode, 186
- Arc::ExecutionTarget, 187
- Arc::ExecutionTarget
  - ExecutionTarget, 187
  - RegisterJobSubmission, 188
  - SaveToStream, 188
- Arc::ExpirationReminder, 189
- Arc::ExpirationReminder
  - Counter, 190
  - getExpiryTime, 189
  - getReservationID, 189
  - operator<, 189
- Arc::FileAccess, 191
- Arc::FileAccess
  - chmod, 192
  - close, 192
  - closedir, 192
  - copy, 192
  - fallocate, 192
  - fstat, 192
  - ftruncate, 192
  - geterno, 192
  - link, 192
  - lseek, 192
  - lstat, 193
  - mkdir, 193
  - mkdirp, 193
  - mkstemp, 193
  - open, 193
  - opendir, 193
  - operator bool, 193
  - operator!, 193
  - ping, 193
  - pread, 193
  - pwrite, 193
  - read, 194
  - readdir, 194
  - readlink, 194
  - remove, 194
  - rmdir, 194
  - rmdirr, 194
  - setuid, 194
  - softlink, 194
  - stat, 194
  - testtune, 194
  - unlink, 194
  - write, 195
- Arc::FileLock, 196
- Arc::FileLock
  - acquire, 197
  - check, 197
  - DEFAULT\_LOCK\_TIMEOUT, 198
  - FileLock, 196
  - getLockSuffix, 197
  - LOCK\_SUFFIX, 198
  - release, 197
- Arc::GLUE2, 202
- ParseExecutionTargets, 202
- Arc::InfoCache, 203
- Arc::InfoCache
  - InfoCache, 203
- Arc::InfoFilter, 204
- Arc::InfoFilter
  - Filter, 204
  - InfoFilter, 204
- Arc::InfoRegister, 205
- Arc::InfoRegisterContainer, 206
- Arc::InfoRegisterContainer
  - addRegistrar, 206
  - addService, 206
  - removeService, 206
- Arc::InfoRegisters, 207
- Arc::InfoRegisters
  - addRegister, 207
  - InfoRegisters, 207
- Arc::InfoRegistrar, 208
- Arc::InfoRegistrar
  - addService, 208
  - registration, 208
  - removeService, 208
- Arc::InformationContainer, 209
- Arc::InformationContainer
  - Acquire, 209
  - Assign, 209
  - doc\_, 210
  - Get, 210
  - InformationContainer, 209
- Arc::InformationInterface, 211
- Arc::InformationInterface
  - Get, 211
  - InformationInterface, 211
  - lock\_, 212
- Arc::InformationRequest, 213
- Arc::InformationRequest
  - InformationRequest, 213
  - SOAP, 213
- Arc::InformationResponse, 214
- Arc::InformationResponse
  - InformationResponse, 214
  - Result, 214
- Arc::initializeCredentialsType, 215
- Arc::IntraProcessCounter, 216

- Arc::IntraProcessCounter
  - ~IntraProcessCounter, 217
  - cancel, 217
  - changeExcess, 217
  - changeLimit, 217
  - extend, 217
  - getExcess, 218
  - getLimit, 218
  - getValue, 218
  - IntraProcessCounter, 216
  - reserve, 218
  - setExcess, 219
  - setLimit, 219
- Arc::Job, 220
  - Job, 220
  - operator=, 221
  - ReadAllJobsFromFile, 221
  - ReadJobIDsFromFile, 221
  - ReadJobsFromFile, 222
  - RemoveJobsFromFile, 223
  - SaveToStream, 223
  - ToXML, 223
  - Update, 224
  - WriteJobIDsToFile, 224
  - WriteJobIDToFile, 224
  - WriteJobsToFile, 225
  - WriteJobsToTruncatedFile, 226
- Arc::JobControllerPluginLoader, 227
- Arc::JobControllerPluginLoader
  - ~JobControllerPluginLoader, 227
  - JobControllerPluginLoader, 227
  - load, 227
- Arc::JobDescription, 229
- Arc::JobDescription
  - GetSourceLanguage, 229
  - OtherAttributes, 231
  - Parse, 229
  - Prepare, 230
  - SaveToStream, 230
  - UnParse, 231
- Arc::JobDescriptionParser, 232
- Arc::JobDescriptionParserLoader, 233
- Arc::JobDescriptionParserLoader
  - ~JobDescriptionParserLoader, 233
  - GetJobDescriptionParsers, 233
  - JobDescriptionParserLoader, 233
  - load, 233
- Arc::JobIdentificationType, 235
- Arc::JobIdentificationType
  - ActivityOldID, 235
  - Annotation, 235
  - Description, 235
  - JobName, 235
  - Type, 236
- Arc::JobState, 237
- Arc::JobState
  - IsFinished, 237
- Arc::JobSupervisor, 238
- Arc::JobSupervisor
  - AddJob, 239
  - Cancel, 239
  - Clean, 239
  - GetJobs, 240
  - JobSupervisor, 238
  - Migrate, 240
  - Renew, 241
  - Resubmit, 241
  - Resume, 242
  - Retrieve, 243
  - Update, 243
- Arc::Loader, 245
  - ~Loader, 245
  - factory\_, 245
  - Loader, 245
- Arc::LogDestination, 246
- Arc::LogDestination
  - log, 246
  - LogDestination, 246
- Arc::LogFile, 248
- Arc::LogFile
  - log, 249
  - LogFile, 248
  - operator bool, 249
  - operator!, 249
  - setBackups, 249
  - setMaxSize, 249
  - setReopen, 249
- Arc::Logger, 251
  - ~Logger, 252
  - addDestination, 252
  - addDestinations, 252
  - deleteDestinations, 252
  - getDestinations, 252
  - getRootLogger, 252
  - getThreshold, 253
  - Logger, 251, 252
  - msg, 253
  - removeDestinations, 253
  - setThreadContext, 253
  - setThreshold, 253
  - setThresholdForDomain, 254
- Arc::LoggerContext, 255
- Arc::LogMessage, 256
- Arc::LogMessage
  - getLevel, 257
  - Logger, 257
  - LogMessage, 256
  - operator<<, 257

- setIdentifier, 257
- Arc::LogStream, 258
- Arc::LogStream
  - log, 259
  - LogStream, 258
- Arc::MCC, 262
  - AddSecHandler, 263
  - logger, 263
  - MCC, 263
  - Next, 263
  - next\_, 264
  - process, 263
  - ProcessSecHandlers, 263
  - sechandlers\_, 264
  - Unlink, 263
- Arc::MCC\_Status, 265
  - getExplanation, 265
  - getKind, 265
  - getOrigin, 266
  - isOk, 266
  - MCC\_Status, 265
  - operator bool, 266
  - operator std::string, 266
  - operator!, 266
- Arc::MCCInterface, 268
  - process, 268
- Arc::MCCLoader, 270
  - ~MCCLoader, 270
  - MCCLoader, 270
  - operator[], 271
- Arc::MD5Sum, 272
  - add, 272
  - end, 272
  - operator bool, 273
  - operator!, 273
  - print, 273
  - result, 273
  - scan, 273
  - start, 274
- Arc::Message, 275
  - ~Message, 276
  - Attributes, 276
  - Auth, 276
  - AuthContext, 276
  - Context, 276
  - Message, 276
  - operator=, 276
  - Payload, 277
- Arc::MessageAttributes, 278
- Arc::MessageAttributes
  - add, 279
  - attributes\_, 280
  - count, 279
  - get, 279
  - getAll, 279
  - MessageAttributes, 278
  - remove, 280
  - removeAll, 280
  - set, 280
- Arc::MessageAuth, 281
- Arc::MessageAuth
  - Export, 281
  - Filter, 281
  - get, 281
  - operator[], 281
  - remove, 282
  - set, 282
- Arc::MessageAuthContext, 283
- Arc::MessageContext, 284
- Arc::MessageContext
  - Add, 284
- Arc::MessageContextElement, 285
- Arc::MessagePayload, 286
- Arc::ModuleDesc, 287
- Arc::ModuleManager, 288
- Arc::ModuleManager
  - find, 289
  - findLocation, 289
  - load, 289
  - makePersistent, 289
  - ModuleManager, 288
  - reload, 289
  - setCfg, 289
  - unload, 289
  - unuse, 289
  - use, 289
- Arc::MultiSecAttr, 291
- Arc::MultiSecAttr
  - Export, 291
  - operator bool, 291
- Arc::MySQLDatabase, 292
- Arc::MySQLDatabase
  - close, 292
  - connect, 292
  - enable\_ssl, 292
  - isconnected, 293
  - shutdown, 293
- Arc::OAuthConsumer, 294
- Arc::OAuthConsumer
  - approveCSR, 294
  - OAuthConsumer, 294
  - parseDN, 294
  - processLogin, 294
  - pushCSR, 294
  - storeCert, 295
- Arc::PathIterator, 296
- Arc::PathIterator
  - operator \*, 296

- operator bool, 296
- operator++, 296
- operator-, 296
- PathIterator, 296
- Rest, 296
- Arc::PayloadRaw, 298
- Arc::PayloadRaw
  - ~PayloadRaw, 298
  - Buffer, 298
  - BufferPos, 298
  - BufferSize, 299
  - PayloadRaw, 298
  - Size, 299
- Arc::PayloadRawInterface, 300
- Arc::PayloadRawInterface
  - Buffer, 300
  - BufferPos, 300
  - BufferSize, 300
  - Content, 301
  - Insert, 301
  - operator[], 301
  - Size, 301
  - Truncate, 301
- Arc::PayloadSOAP, 302
- Arc::PayloadSOAP
  - PayloadSOAP, 302
- Arc::PayloadStream, 303
- Arc::PayloadStream
  - ~PayloadStream, 303
  - Get, 304
  - handle\_, 305
  - Limit, 304
  - operator bool, 304
  - operator!, 304
  - PayloadStream, 303
  - Pos, 304
  - Put, 304, 305
  - seekable\_, 305
  - Size, 305
  - Timeout, 305
- Arc::PayloadStreamInterface, 306
- Arc::PayloadStreamInterface
  - Get, 306, 307
  - Limit, 307
  - operator bool, 307
  - operator!, 307
  - Pos, 307
  - Put, 307
  - Size, 308
  - Timeout, 308
- Arc::PayloadWSRF, 309
- Arc::PayloadWSRF
  - PayloadWSRF, 309
- Arc::Plexer, 315
  - ~Plexer, 315
  - logger, 316
  - Next, 316
  - Plexer, 315
  - process, 316
- Arc::PlexerEntry, 317
- Arc::Plugin, 318
  - Plugin, 319
- Arc::PluginArgument, 320
- Arc::PluginArgument
  - get\_factory, 320
  - get\_module, 320
- Arc::PluginDesc, 321
- Arc::PluginDescriptor, 322
- Arc::PluginsFactory, 323
- Arc::PluginsFactory
  - FilterByKind, 323
  - load, 323
  - PluginsFactory, 323
  - report, 324
  - scan, 324
  - TryLoad, 324
- Arc::RegisteredService, 332
- Arc::RegisteredService
  - RegisteredService, 332
- Arc::RegularExpression, 333
- Arc::RegularExpression
  - ~RegularExpression, 333
  - getPattern, 333
  - hasPattern, 333
  - isOk, 334
  - match, 334
  - operator=, 334
  - RegularExpression, 333
- Arc::RemoteLoggingType, 335
- Arc::RemoteLoggingType
  - Location, 335
  - optional, 335
  - ServiceType, 335
- Arc::Run, 342
  - ~Run, 343
  - Abandon, 343
  - AfterFork, 343
  - AssignStderr, 343
  - AssignStdin, 343
  - AssignStdout, 343
  - AssignWorkingDirectory, 343
  - CloseStderr, 343
  - CloseStdin, 343
  - CloseStdout, 343
  - ExitTime, 344
  - KeepStderr, 344
  - KeepStdin, 344
  - KeepStdout, 344

- Kill, 344
- operator bool, 344
- operator!, 344
- ReadStderr, 344
- ReadStdout, 344
- Result, 344
- Run, 342
- Running, 345
- RunTime, 345
- Start, 345
- Wait, 345
- WriteStdin, 345
- Arc::SAMLToken, 346
  - ~SAMLToken, 347
  - Authenticate, 348
  - operator bool, 348
  - SAMLToken, 347
  - SAMLVersion, 347
- Arc::SecAttr, 352
- Arc::SecAttr
  - ARCAuth, 354
  - Export, 353
  - GACL, 354
  - get, 353
  - getAll, 353
  - Import, 353
  - operator bool, 353
  - operator!=, 353
  - operator==, 353
  - SAML, 354
  - SecAttr, 352
  - XACML, 354
- Arc::SecAttrFormat, 355
- Arc::SecAttrValue, 356
- Arc::SecAttrValue
  - operator bool, 356
  - operator!=, 356
  - operator==, 356
- Arc::Service, 361
  - AddSecHandler, 362
  - getID, 362
  - logger, 363
  - operator bool, 362
  - operator!, 362
  - ProcessSecHandlers, 362
  - RegistrationCollector, 362
  - sechandlers\_, 363
  - Service, 362
  - valid, 363
- Arc::SimpleCondition, 364
- Arc::SimpleCondition
  - broadcast, 364
  - lock, 364
  - reset, 364
  - signal, 364
  - signal\_nonblock, 364
  - unlock, 364
  - wait, 365
  - wait\_nonblock, 365
- Arc::SimpleCounter, 366
- Arc::SimpleCounter
  - dec, 366
  - forceReset, 366
  - get, 366
  - inc, 366
  - set, 366
  - wait, 366, 367
- Arc::SOAPMessage, 368
  - ~SOAPMessage, 368
  - Attributes, 368
  - Payload, 368, 369
  - SOAPMessage, 368
- Arc::Software, 370
  - ComparisonOperator, 371
  - ComparisonOperatorEnum, 371
  - convert, 373
  - empty, 373
  - EQUAL, 371
  - getFamily, 373
  - getName, 373
  - getVersion, 373
  - GREATERTHAN, 371
  - GREATERTHANOREQUAL, 372
  - LESSTHAN, 372
  - LESSTHANOREQUAL, 372
  - NOTEQUAL, 371
  - operator std::string, 373
  - operator!=, 374
  - operator(), 374
  - operator<, 374
  - operator<<, 376
  - operator<=, 374
  - operator==, 375
  - operator>, 375
  - operator>=, 376
  - Software, 372
  - toString, 376
  - VERSIONTOKENS, 377
- Arc::SoftwareRequirement, 378
- Arc::SoftwareRequirement
  - add, 379, 380
  - clear, 380
  - empty, 380
  - getComparisonOperatorList, 380
  - getSoftwareList, 380
  - isResolved, 381
  - isSatisfied, 381, 382
  - operator=, 382

- selectSoftware, 382, 383
  - SoftwareRequirement, 378, 379
- Arc::SubmitterPlugin, 389
- Arc::SubmitterPlugin
  - Migrate, 389
  - Submit, 389
- Arc::SubmitterPluginLoader, 391
- Arc::SubmitterPluginLoader
  - ~SubmitterPluginLoader, 391
  - load, 391
  - SubmitterPluginLoader, 391
- Arc::ThreadDataItem, 393
- Arc::ThreadDataItem
  - Attach, 393
  - Dup, 394
  - Get, 394
  - ThreadDataItem, 393
- Arc::ThreadedPointer, 395
- Arc::ThreadedPointer
  - Holders, 395
  - operator \*, 395
  - operator bool, 395
  - operator!, 395
  - operator!=, 396
  - operator->, 396
  - operator<, 396
  - operator==, 396
  - Ptr, 396
  - Release, 396
  - WaitInRange, 396
  - WaitOutRange, 396, 397
- Arc::ThreadedPointerBase, 398
- Arc::ThreadRegistry, 399
- Arc::ThreadRegistry
  - RegisterThread, 399
  - UnregisterThread, 399
  - WaitForExit, 399
  - WaitOrCancel, 399
- Arc::Time, 400
  - GetFormat, 401
  - GetTime, 401
  - operator std::string, 401
  - operator!=, 401
  - operator+, 401
  - operator-, 401
  - operator<, 401
  - operator<=, 401
  - operator=, 401, 402
  - operator==, 402
  - operator>, 402
  - operator>=, 402
  - SetFormat, 402
  - SetTime, 402
  - str, 402
  - Time, 400, 401
- Arc::URL, 411
  - ~URL, 414
  - AddHTTPOption, 414
  - AddLDAPAttribute, 414
  - AddLocation, 414
  - AddMetaDataOption, 414
  - AddOption, 414
  - BaseDN2Path, 415
  - ChangeFullPath, 415
  - ChangeHost, 415
  - ChangeLDAPFilter, 415
  - ChangeLDAPScope, 415
  - ChangePath, 415
  - ChangePort, 415
  - ChangeProtocol, 415
  - CommonLocOption, 415
  - CommonLocOptions, 416
  - commonlocoptions, 419
  - ConnectionURL, 416
  - FullPath, 416
  - FullPathURIEncoded, 416
  - fullstr, 416
  - Host, 416
  - host, 419
  - HTTPOption, 416
  - HTTPOptions, 416
  - httpoptions, 420
  - ip6addr, 420
  - IsSecureProtocol, 416
  - LDAPAttributes, 416
  - ldapattributes, 420
  - LDAPFilter, 417
  - ldapfilter, 420
  - LDAPScope, 417
  - ldapscope, 420
  - Locations, 417
  - locations, 420
  - MetaDataOption, 417
  - MetaDataOptions, 417
  - metadataoptions, 420
  - operator bool, 417
  - operator<, 417
  - operator<<, 419
  - operator==, 417
  - Option, 417
  - Options, 418
  - OptionString, 418
  - ParseOptions, 418
  - ParsePath, 418
  - Passwd, 418
  - passwd, 420
  - Path, 418
  - path, 420



- Path2BaseDN, 418
- plainstr, 418
- Port, 418
- port, 420
- Protocol, 418
- protocol, 420
- RemoveHTTPOption, 418
- RemoveMetaDataOption, 419
- RemoveOption, 419
- Scope, 414
- str, 419
- StringMatches, 419
- URL, 414
- urloptions, 421
- Username, 419
- username, 421
- valid, 421
- Arc::URLLocation, 422
  - ~URLLocation, 423
  - fullstr, 423
  - Name, 423
  - name, 423
  - str, 423
  - URLLocation, 422, 423
- Arc::UserConfig, 424
- Arc::UserConfig
  - AddBartender, 428
  - ApplyToConfig, 428
  - ARCUSERDIRECTORY, 448
  - Bartender, 428, 429
  - Broker, 429, 430
  - CACertificatePath, 430
  - CACertificatesDirectory, 431
  - CertificateLifeTime, 432
  - CertificatePath, 432
  - CredentialsFound, 433
  - DEFAULT\_BROKER, 448
  - DEFAULT\_TIMEOUT, 448
  - DEFAULTCONFIG, 448
  - EXAMPLECONFIG, 448
  - GetUser, 433
  - IdPName, 433, 434
  - InitializeCredentials, 434
  - JobDownloadDirectory, 436
  - JobListFile, 436
  - KeyPassword, 437
  - KeyPath, 438
  - KeySize, 438, 439
  - LoadConfigurationFile, 439
  - operator bool, 440
  - operator!, 441
  - OverlayFile, 441
  - Password, 441, 442
  - ProxyPath, 442
  - SaveToFile, 443
  - SetUser, 443
  - SLCS, 443
  - StoreDirectory, 444
  - SYSCONFIG, 448
  - SYSCONFIGARCLOC, 449
  - Timeout, 444, 445
  - UserConfig, 426–428
  - UserName, 445
  - UtilsDirPath, 446
  - Verbosity, 446, 447
  - VOMSESPath, 447
- Arc::UsernameToken, 450
- Arc::UsernameToken
  - Authenticate, 451
  - operator bool, 451
  - PasswordType, 450
  - Username, 451
  - UsernameToken, 450, 451
- Arc::UserSwitch, 452
- Arc::VOMSTrustList, 453
- Arc::VOMSTrustList
  - AddChain, 454
  - AddRegex, 454
  - VOMSTrustList, 453
- Arc::WSAEndpointReference, 455
- Arc::WSAEndpointReference
  - ~WSAEndpointReference, 455
  - Address, 456
  - hasAddress, 456
  - MetaData, 456
  - operator XMLNode, 456
  - operator=, 456
  - ReferenceParameters, 456
  - WSAEndpointReference, 455
- Arc::WSAHeader, 457
  - Action, 458
  - Check, 458
  - FaultTo, 458
  - From, 458
  - hasAction, 458
  - hasMessageID, 458
  - hasRelatesTo, 458
  - hasRelationshipType, 458
  - hasTo, 459
  - header\_allocated\_, 460
  - MessageID, 459
  - NewReferenceParameter, 459
  - operator XMLNode, 459
  - ReferenceParameter, 459
  - RelatesTo, 459
  - RelationshipType, 459
  - ReplyTo, 460
  - To, 460

- WSAHeader, 458
- Arc::WSRF, 461
  - allocated\_, 462
  - operator bool, 462
  - set\_namespaces, 462
  - SOAP, 462
  - valid\_, 462
  - WSRF, 461
- Arc::WSRFBBaseFault, 463
- Arc::WSRFBBaseFault
  - set\_namespaces, 463
  - WSRFBBaseFault, 463
- Arc::WSRP, 465
  - set\_namespaces, 465
  - WSRP, 465
- Arc::WSRPFault, 467
  - WSRPFault, 467
- Arc::WSRPResourcePropertyChangeFailure, 468
- Arc::WSRPResourcePropertyChangeFailure
  - WSRPResourcePropertyChangeFailure, 468
- Arc::X509Token, 469
  - ~X509Token, 470
  - Authenticate, 470
  - operator bool, 470
  - X509Token, 469
  - X509TokenType, 469
- Arc::XMLNode, 471
  - ~XMLNode, 474
  - Attribute, 474
  - AttributesSize, 474
  - Child, 474
  - Destroy, 474
  - Exchange, 474
  - FullName, 475
  - Get, 475
  - GetDoc, 475
  - GetRoot, 475
  - GetXML, 475
  - is\_owner\_, 481
  - is\_temporary\_, 481
  - MatchXMLName, 480, 481
  - MatchXMLNamespace, 481
  - Move, 475
  - Name, 475, 476
  - Namespace, 476
  - NamespacePrefix, 476
  - Namespaces, 476
  - New, 476
  - NewAttribute, 476
  - NewChild, 476, 477
  - operator bool, 477
  - operator std::string, 477
  - operator!, 477
  - operator!=", 477
  - operator++, 478
  - operator-, 478
  - operator=, 478
  - operator==, 478
  - operator[], 478, 479
  - Parent, 479
  - Path, 479
  - Prefix, 479
  - ReadFromFile, 479
  - ReadFromStream, 479
  - Replace, 479
  - Same, 480
  - SaveToFile, 480
  - SaveToStream, 480
  - Set, 480
  - Size, 480
  - Swap, 480
  - Validate, 480
  - XMLNode, 473, 474
  - XPathLookup, 480
- Arc::XMLNodeContainer, 482
- Arc::XMLNodeContainer
  - Add, 482
  - AddNew, 482
  - Nodes, 483
  - operator=, 483
  - operator[], 483
  - Size, 483
  - XMLNodeContainer, 482
- Arc::XMLSecNode, 484
- Arc::XMLSecNode
  - AddSignatureTemplate, 484
  - DecryptNode, 484
  - EncryptNode, 485
  - SignNode, 485
  - VerifyNode, 485
  - XMLSecNode, 484
- ARCAuth
  - Arc::SecAttr, 354
- ArcCredential, 44
  - CERT\_TYPE\_CA, 44
  - CERT\_TYPE\_EEC, 44
  - CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY, 45
  - CERT\_TYPE\_GSI\_2\_PROXY, 45
  - CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY, 45
  - CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY, 45
  - CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY, 45
  - CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY, 45
  - CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY, 45

- CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY, 45
- CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY, 45
- CERT\_TYPE\_RFC\_LIMITED\_PROXY, 45
- CERT\_TYPE\_RFC\_RESTRICTED\_PROXY, 45
- ArcCredential
  - certType, 44
- ArcSec::AlgFactory, 52
- ArcSec::AlgFactory
  - createAlg, 52
- ArcSec::Attr, 56
- ArcSec::AttributeFactory, 57
- ArcSec::AttributeProxy, 61
- ArcSec::AttributeProxy
  - getAttribute, 61
- ArcSec::AttributeValue, 62
- ArcSec::AttributeValue
  - encode, 62
  - equal, 62
  - getId, 62
  - getType, 62
- ArcSec::Attrs, 64
- ArcSec::AuthzRequestSection, 65
- ArcSec::CombiningAlg, 84
- ArcSec::CombiningAlg
  - combine, 84
  - getalgId, 84
- ArcSec::DateTimeAttribute, 133
- ArcSec::DateTimeAttribute
  - encode, 133
  - equal, 133
  - getId, 133
  - getType, 133
- ArcSec::DenyOverridesCombiningAlg, 146
- ArcSec::DenyOverridesCombiningAlg
  - combine, 146
  - getalgId, 146
- ArcSec::DurationAttribute, 174
- ArcSec::DurationAttribute
  - encode, 174
  - equal, 174
  - getId, 174
  - getType, 174
- ArcSec::EqualFunction, 176
- ArcSec::EqualFunction
  - evaluate, 176
  - getFunctionName, 176
- ArcSec::EvalResult, 178
- ArcSec::EvaluationCtx, 179
- ArcSec::EvaluationCtx
  - EvaluationCtx, 179
- ArcSec::Evaluator, 180
- ArcSec::Evaluator
  - addPolicy, 180
  - evaluate, 181
  - getAlgFactory, 181
  - getAttrFactory, 181
  - getFnFactory, 182
  - getName, 182
  - setCombiningAlg, 182
- ArcSec::EvaluatorContext, 183
- ArcSec::EvaluatorContext
  - operator AlgFactory \*, 183
  - operator AttributeFactory \*, 183
  - operator FnFactory \*, 183
- ArcSec::EvaluatorLoader, 184
- ArcSec::EvaluatorLoader
  - getEvaluator, 184
  - getPolicy, 184
  - getRequest, 184, 185
- ArcSec::FnFactory, 199
- ArcSec::FnFactory
  - createFn, 199
- ArcSec::Function, 200
- ArcSec::Function
  - evaluate, 200
- ArcSec::MatchFunction, 260
- ArcSec::MatchFunction
  - evaluate, 260
  - getFunctionName, 260
- ArcSec::PDP, 310
- ArcSec::PeriodAttribute, 311
- ArcSec::PeriodAttribute
  - encode, 311
  - equal, 311
  - getId, 311
  - getType, 311
- ArcSec::PermitOverridesCombiningAlg, 313
- ArcSec::PermitOverridesCombiningAlg
  - combine, 313
  - getalgId, 313
- ArcSec::Policy, 325
- ArcSec::Policy
  - addPolicy, 326
  - eval, 326
  - getEffect, 326
  - getEvalName, 326
  - getEvalResult, 326
  - getName, 326
  - make\_policy, 326
  - match, 326
  - operator bool, 327
  - Policy, 325, 326
  - setEvalResult, 327
  - setEvaluatorContext, 327
- ArcSec::PolicyParser, 328

- ArcSec::PolicyParser
  - parsePolicy, 328
- ArcSec::PolicyStore, 329
- ArcSec::PolicyStore
  - PolicyStore, 329
- ArcSec::Request, 336
- ArcSec::Request
  - addRequestItem, 337
  - getEvalName, 337
  - getName, 337
  - getRequestItems, 337
  - make\_request, 337
  - Request, 336
  - setAttributeFactory, 337
  - setRequestItems, 337
- ArcSec::RequestAttribute, 338
- ArcSec::RequestAttribute
  - duplicate, 338
  - RequestAttribute, 338
- ArcSec::RequestItem, 339
- ArcSec::RequestItem
  - RequestItem, 339
- ArcSec::Response, 340
- ArcSec::ResponseItem, 341
- ArcSec::SecHandler, 358
- ArcSec::SecHandlerConfig, 359
- ArcSec::Security, 360
- ArcSec::Source, 385
- ArcSec::Source
  - Get, 386
  - operator bool, 386
  - Source, 385, 386
- ArcSec::SourceFile, 387
- ArcSec::SourceFile
  - SourceFile, 387
- ArcSec::SourceURL, 388
- ArcSec::SourceURL
  - SourceURL, 388
- ArcSec::TimeAttribute, 403
- ArcSec::TimeAttribute
  - encode, 403
  - equal, 403
  - getId, 403
  - getType, 403
- ARCUSERDIRECTORY
  - Arc::UserConfig, 448
- Argument
  - Arc::ExecutableType, 186
- Assign
  - Arc::InformationContainer, 209
- AssignStderr
  - Arc::Run, 343
- AssignStdin
  - Arc::Run, 343
- AssignStdout
  - Arc::Run, 343
- AssignWorkingDirectory
  - Arc::Run, 343
- Attach
  - Arc::ThreadDataItem, 393
- AttrConstIter
  - Arc, 28
- Attribute
  - Arc::XMLNode, 474
- AttributeIterator
  - Arc::AttributeIterator, 58
- Attributes
  - Arc::Message, 276
  - Arc::SOAPMessage, 368
- attributes\_
  - Arc::MessageAttributes, 280
- AttributesSize
  - Arc::XMLNode, 474
- AttrIter
  - Arc, 28
- AttrMap
  - Arc, 28
- Auth
  - Arc::Message, 276
- AuthContext
  - Arc::Message, 276
- Authenticate
  - Arc::SAMLToken, 348
  - Arc::UsernameToken, 451
  - Arc::X509Token, 470
- AutoPointer
  - Arc::AutoPointer, 66
- averaging\_time
  - DataStaging::TransferParameters, 405
- Backup
  - Arc::DelegationConsumer, 136
- Bartender
  - Arc::UserConfig, 428, 429
- BaseDN2Path
  - Arc::URL, 415
- booltostr
  - Arc, 35
- broadcast
  - Arc::SimpleCondition, 364
- Broker
  - Arc::UserConfig, 429, 430
- Buffer
  - Arc::PayloadRaw, 298
  - Arc::PayloadRawInterface, 300
- BufferPos
  - Arc::PayloadRaw, 298
  - Arc::PayloadRawInterface, 300

- BufferSize
  - Arc::PayloadRaw, 299
  - Arc::PayloadRawInterface, 300
- bulk\_possible
  - DataStaging::DTR, 151
- BUSY\_ERROR
  - Arc, 30
- CACertificatePath
  - Arc::UserConfig, 430
- CACertificatesDirectory
  - Arc::UserConfig, 431
- CACHE\_ALREADY\_PRESENT
  - DataStaging, 48
- CACHE\_CHECKED
  - DataStaging::DTRStatus, 170
- cache\_dirs
  - DataStaging::DTRCacheParameters, 160
- CACHE\_DOWNLOADED
  - DataStaging, 48
- CACHE\_ERROR
  - DataStaging::DTRErrorStatus, 164
- CACHE\_LOCKED
  - DataStaging, 48
- CACHE\_NOT\_USED
  - DataStaging, 48
- CACHE\_PROCESSED
  - DataStaging::DTRStatus, 171
- CACHE\_SKIP
  - DataStaging, 48
- CACHE\_WAIT
  - DataStaging::DTRStatus, 170
- CACHEABLE
  - DataStaging, 48
- CacheState
  - DataStaging, 47
- caching\_finished
  - DataStaging::DTRLList, 166
- caching\_started
  - DataStaging::DTRLList, 166
- calculate\_shares
  - DataStaging::TransferShares, 407
- came\_from\_delivery
  - DataStaging::DTR, 151
- came\_from\_generator
  - DataStaging::DTR, 151
- came\_from\_post\_processor
  - DataStaging::DTR, 151
- came\_from\_pre\_processor
  - DataStaging::DTR, 152
- can\_start
  - DataStaging::TransferShares, 407
- Cancel
  - Arc::JobSupervisor, 239
- cancel
  - Arc::Counter, 95
  - Arc::CounterTicket, 100
  - Arc::IntraProcessCounter, 217
- cancel\_requested
  - DataStaging::DTR, 152
- cancelDTR
  - DataStaging::DataDelivery, 119
- cancelDTRs
  - DataStaging::Scheduler, 350
- CANCELLED
  - DataStaging::DTRStatus, 171
- CANCELLED\_FINISHED
  - DataStaging::DTRStatus, 171
- CanonicalDir
  - Arc, 33
- CERT\_TYPE\_CA
  - ArcCredential, 44
- CERT\_TYPE\_EEC
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_GSI\_2\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_RFC\_LIMITED\_PROXY
  - ArcCredential, 45
- CERT\_TYPE\_RFC\_RESTRICTED\_PROXY
  - ArcCredential, 45
- CertificateLifeTime
  - Arc::UserConfig, 432
- CertificatePath
  - Arc::UserConfig, 432
- certType
  - ArcCredential, 44
- changeExcess
  - Arc::Counter, 95
  - Arc::IntraProcessCounter, 217
- ChangeFullPath
  - Arc::URL, 415

- ChangeHost
  - Arc::URL, 415
- ChangeLDAPFilter
  - Arc::URL, 415
- ChangeLDAPScope
  - Arc::URL, 415
- changeLimit
  - Arc::Counter, 95
  - Arc::IntraProcessCounter, 217
- ChangePath
  - Arc::URL, 415
- ChangePort
  - Arc::URL, 415
- ChangeProtocol
  - Arc::URL, 415
- Check
  - Arc::WSAHeader, 458
- check
  - Arc::FileLock, 197
- CHECK\_CACHE
  - DataStaging::DTRStatus, 170
- CheckComm
  - DataStaging::DataDeliveryComm, 123
  - DataStaging::DataDeliveryLocalComm, 129
  - DataStaging::DataDeliveryRemoteComm, 131
- CheckConsumers
  - Arc::DelegationContainerSOAP, 139
- CHECKING\_CACHE
  - DataStaging::DTRStatus, 170
- Checksum
  - Arc::Checksum, 71
- checksum
  - DataStaging::DataDeliveryComm::Status, 126
- Child
  - Arc::XMLNode, 474
- chmod
  - Arc::FileAccess, 192
- CIStrngValue
  - Arc::CIStrngValue, 77
- Clean
  - Arc::JobSupervisor, 239
- clear
  - Arc::SoftwareRequirement, 380
- ClientSOAP
  - Arc::ClientSOAP, 81
- close
  - Arc::Database, 117
  - Arc::FileAccess, 192
  - Arc::MySQLDatabase, 292
- closedir
  - Arc::FileAccess, 192
- CloseStderr
  - Arc::Run, 343
- CloseStdin
  - Arc::Run, 343
- CloseStdout
  - Arc::Run, 343
- combine
  - ArcSec::CombiningAlg, 84
  - ArcSec::DenyOverridesCombiningAlg, 146
  - ArcSec::PermitOverridesCombiningAlg, 313
- CommClosed
  - DataStaging::DataDeliveryComm, 122
- CommExited
  - DataStaging::DataDeliveryComm, 122
- CommFailed
  - DataStaging::DataDeliveryComm, 122
- CommInit
  - DataStaging::DataDeliveryComm, 122
- CommNoError
  - DataStaging::DataDeliveryComm, 122
- CommonLocOption
  - Arc::URL, 415
- CommonLocOptions
  - Arc::URL, 416
- commonlocoptions
  - Arc::URL, 419
- commstatus
  - DataStaging::DataDeliveryComm::Status, 126
- CommStatusType
  - DataStaging::DataDeliveryComm, 122
- CommTimeout
  - DataStaging::DataDeliveryComm, 122
- ComparisonOperator
  - Arc::Software, 371
- ComparisonOperatorEnum
  - Arc::Software, 371
- conf
  - DataStaging::TransferSharesConf, 409
- Config
  - Arc::Config, 86, 87
- ConfusaCertHandler
  - Arc::ConfusaCertHandler, 88
- connect
  - Arc::Database, 117
  - Arc::MySQLDatabase, 292
- connect\_logger
  - DataStaging::DTR, 152
- ConnectionURL
  - Arc::URL, 416
- Content
  - Arc::PayloadRawInterface, 301
- ContentFromPayload
  - Arc, 41
- Context
  - Arc::Message, 276
- context\_lock\_
  - Arc::DelegationContainerSOAP, 140

- convert
  - Arc::Software, 373
- convert\_to\_rdn
  - Arc, 36
- copy
  - Arc::FileAccess, 192
- count
  - Arc::MessageAttributes, 279
- Counter
  - Arc::Counter, 95
  - Arc::CounterTicket, 101
  - Arc::ExpirationReminder, 190
- CounterTicket
  - Arc::Counter, 99
  - Arc::CounterTicket, 100
- createAlg
  - ArcSec::AlgFactory, 52
- createCertRequest
  - Arc::ConfusaCertHandler, 88
- createFn
  - ArcSec::FnFactory, 199
- CreateInstance
  - DataStaging::DataDeliveryComm, 123
- CreateThreadFunction
  - Arc, 37
- createVOMSAC
  - Arc, 38
- Credential
  - Arc::Credential, 106, 107
- CredentialError
  - Arc::CredentialError, 114
- CredentialLogger
  - Arc, 43
- CredentialsFound
  - Arc::UserConfig, 433
- current\_
  - Arc::AttributeIterator, 60
- Database
  - Arc::Database, 116
- DataDelivery
  - DataStaging::DataDelivery, 119
- DataDeliveryComm
  - DataStaging::DataDeliveryComm, 123
- DataDeliveryLocalComm
  - DataStaging::DataDeliveryLocalComm, 129
- DataStaging, 46
  - CACHE\_ALREADY\_PRESENT, 48
  - CACHE\_DOWNLOADED, 48
  - CACHE\_LOCKED, 48
  - CACHE\_NOT\_USED, 48
  - CACHE\_SKIP, 48
  - CACHEABLE, 48
  - NON\_CACHEABLE, 48
- DataStaging
  - CacheState, 47
  - DTR\_ptr, 47
  - DTRLogger, 47
  - ProcessState, 47
  - StagingProcesses, 47
- DataStaging::DataDelivery, 119
- DataStaging::DataDelivery
  - ~DataDelivery, 119
  - cancelDTR, 119
  - DataDelivery, 119
  - receiveDTR, 119
  - SetTransferParameters, 120
  - start, 120
  - stop, 120
- DataStaging::DataDeliveryComm, 121
  - CommClosed, 122
  - CommExited, 122
  - CommFailed, 122
  - CommInit, 122
  - CommNoError, 122
  - CommTimeout, 122
- DataStaging::DataDeliveryComm
  - ~DataDeliveryComm, 123
  - CheckComm, 123
  - CommStatusType, 122
  - CreateInstance, 123
  - DataDeliveryComm, 123
  - dtr\_id, 124
  - GetError, 123
  - GetStatus, 123
  - handler\_, 124
  - lock\_, 124
  - logger\_, 124
  - operator bool, 123
  - operator!, 123
  - PullStatus, 124
  - start\_, 124
  - status\_, 124
  - status\_buf\_, 124
  - status\_pos\_, 124
  - transfer\_params, 125
- DataStaging::DataDeliveryComm::Status, 126
- DataStaging::DataDeliveryComm::Status
  - checksum, 126
  - commstatus, 126
  - error, 126
  - error\_desc, 126
  - error\_location, 126
  - offset, 126
  - size, 127
  - speed, 127
  - status, 127
  - streams, 127



- timestamp\_ns, [127](#)
- timestamp\_s, [127](#)
- transferred, [127](#)
- DataStaging::DataDeliveryCommHandler, [128](#)
- DataStaging::DataDeliveryCommHandler
  - Add, [128](#)
  - getInstance, [128](#)
  - Remove, [128](#)
- DataStaging::DataDeliveryLocalComm, [129](#)
- DataStaging::DataDeliveryLocalComm
  - ~DataDeliveryLocalComm, [129](#)
  - CheckComm, [129](#)
  - DataDeliveryLocalComm, [129](#)
  - operator bool, [129](#)
  - operator!, [130](#)
  - PullStatus, [130](#)
- DataStaging::DataDeliveryRemoteComm, [131](#)
- DataStaging::DataDeliveryRemoteComm
  - CheckComm, [131](#)
  - operator bool, [131](#)
  - operator!, [131](#)
  - PullStatus, [131](#)
- DataStaging::DTR, [148](#)
- DataStaging::DTR
  - ~DTR, [151](#)
  - add\_problematic\_delivery\_service, [151](#)
  - bulk\_possible, [151](#)
  - came\_from\_delivery, [151](#)
  - came\_from\_generator, [151](#)
  - came\_from\_post\_processor, [151](#)
  - came\_from\_pre\_processor, [152](#)
  - cancel\_requested, [152](#)
  - connect\_logger, [152](#)
  - decrease\_tries\_left, [152](#)
  - disconnect\_logger, [152](#)
  - DTR, [151](#)
  - error, [152](#)
  - get\_bulk\_end, [152](#)
  - get\_bulk\_start, [152](#)
  - get\_bytes\_transferred, [152](#)
  - get\_cache\_file, [152](#)
  - get\_cache\_parameters, [152](#)
  - get\_cache\_state, [153](#)
  - get\_callbacks, [153](#)
  - get\_creation\_time, [153](#)
  - get\_delivery\_endpoint, [153](#)
  - get\_destination, [153](#)
  - get\_destination\_str, [153](#)
  - get\_error\_status, [153](#)
  - get\_id, [153](#)
  - get\_initial\_tries, [153](#)
  - get\_local\_user, [153](#)
  - get\_logger, [153](#)
  - get\_mapped\_source, [154](#)
  - get\_modification\_time, [154](#)
  - get\_owner, [154](#)
  - get\_parent\_job\_id, [154](#)
  - get\_priority, [154](#)
  - get\_problematic\_delivery\_services, [154](#)
  - get\_process\_time, [154](#)
  - get\_short\_id, [154](#)
  - get\_source, [154](#)
  - get\_source\_str, [154](#)
  - get\_status, [154](#)
  - get\_sub\_share, [155](#)
  - get\_timeout, [155](#)
  - get\_transfer\_share, [155](#)
  - get\_tries\_left, [155](#)
  - get\_usercfg, [155](#)
  - host\_cert\_for\_remote\_delivery, [155](#)
  - is\_destined\_for\_delivery, [155](#)
  - is\_destined\_for\_post\_processor, [155](#)
  - is\_destined\_for\_pre\_processor, [155](#)
  - is\_force\_registration, [155](#)
  - is\_in\_final\_state, [156](#)
  - is\_replication, [156](#)
  - is\_rfc\_proxy, [156](#)
  - LOCAL\_DELIVERY, [159](#)
  - operator bool, [156](#)
  - operator!, [156](#)
  - push, [156](#)
  - registerCallback, [156](#)
  - reset, [156](#)
  - reset\_error\_status, [156](#)
  - set\_bulk\_end, [156](#)
  - set\_bulk\_start, [157](#)
  - set\_bytes\_transferred, [157](#)
  - set\_cache\_file, [157](#)
  - set\_cache\_parameters, [157](#)
  - set\_cache\_state, [157](#)
  - set\_cancel\_request, [157](#)
  - set\_delivery\_endpoint, [157](#)
  - set\_error\_status, [157](#)
  - set\_force\_registration, [157](#)
  - set\_id, [157](#)
  - set\_mapped\_source, [158](#)
  - set\_priority, [158](#)
  - set\_process\_time, [158](#)
  - set\_replication, [158](#)
  - set\_rfc\_proxy, [158](#)
  - set\_status, [158](#)
  - set\_sub\_share, [158](#)
  - set\_timeout, [158](#)
  - set\_transfer\_share, [158](#)
  - set\_tries\_left, [158](#)
  - suspend, [158](#)
- DataStaging::DTRCacheParameters, [160](#)
- DataStaging::DTRCacheParameters



- cache\_dirs, 160
- drain\_cache\_dirs, 160
- DTRCacheParameters, 160
- remote\_cache\_dirs, 160
- DataStaging::DTRCallback, 162
- DataStaging::DTRCallback
  - ~DTRCallback, 162
  - receiveDTR, 162
- DataStaging::DTRErrorStatus, 163
  - CACHE\_ERROR, 164
  - ERROR\_DESTINATION, 164
  - ERROR\_SOURCE, 164
  - ERROR\_TRANSFER, 164
  - ERROR\_UNKNOWN, 164
  - INTERNAL\_LOGIC\_ERROR, 164
  - INTERNAL\_PROCESS\_ERROR, 164
  - LOCAL\_FILE\_ERROR, 164
  - NO\_ERROR\_LOCATION, 164
  - NONE\_ERROR, 164
  - PERMANENT\_REMOTE\_ERROR, 164
  - SELF\_REPLICATION\_ERROR, 164
  - STAGING\_TIMEOUT\_ERROR, 164
  - TEMPORARY\_REMOTE\_ERROR, 164
  - TRANSFER\_SPEED\_ERROR, 164
- DataStaging::DTRErrorStatus
  - DTRErrorLocation, 164
  - DTRErrorStatus, 164
  - DTRErrorStatusType, 164
  - GetDesc, 165
  - GetErrorLocation, 165
  - GetErrorStatus, 165
  - GetLastErrorState, 165
  - operator!=, 165
  - operator=, 165
  - operator==, 165
- DataStaging::DTRList, 166
- DataStaging::DTRList
  - add\_dtr, 166
  - all\_jobs, 166
  - caching\_finished, 166
  - caching\_started, 166
  - delete\_dtr, 167
  - dumpState, 167
  - empty, 167
  - filter\_dtrs\_by\_job, 167
  - filter\_dtrs\_by\_next\_receiver, 167
  - filter\_dtrs\_by\_owner, 167
  - filter\_dtrs\_by\_status, 167
  - filter\_dtrs\_by\_statuses, 168
  - filter\_pending\_dtrs, 168
  - is\_being\_cached, 168
  - number\_of\_dtrs\_by\_owner, 168
- DataStaging::DTRStatus, 169
  - CACHE\_CHECKED, 170
  - CACHE\_PROCESSED, 171
  - CACHE\_WAIT, 170
  - CANCELLED, 171
  - CANCELLED\_FINISHED, 171
  - CHECK\_CACHE, 170
  - CHECKING\_CACHE, 170
  - DONE, 171
  - ERROR, 171
  - NEW, 170
  - NULL\_STATE, 171
  - PRE\_CLEAN, 171
  - PRE\_CLEARED, 171
  - PRE\_CLEANNING, 171
  - PROCESS\_CACHE, 171
  - PROCESSING\_CACHE, 171
  - QUERY\_REPLICA, 171
  - QUERYING\_REPLICA, 171
  - REGISTER\_REPLICA, 171
  - REGISTERING\_REPLICA, 171
  - RELEASE\_REQUEST, 171
  - RELEASING\_REQUEST, 171
  - REPLICA\_QUERIED, 171
  - REPLICA\_REGISTERED, 171
  - REQUEST\_RELEASED, 171
  - RESOLVE, 170
  - RESOLVED, 170
  - RESOLVING, 170
  - STAGE\_PREPARE, 171
  - STAGED\_PREPARED, 171
  - STAGING\_PREPARING, 171
  - STAGING\_PREPARING\_WAIT, 171
  - TRANSFER, 171
  - TRANSFERRED, 171
  - TRANSFERRING, 171
  - TRANSFERRING\_CANCEL, 171
- DataStaging::DTRStatus
  - DTRStatus, 171
  - DTRStatusType, 170
  - GetDesc, 172
  - GetStatus, 172
  - operator!=, 172
  - operator=, 172
  - operator==, 172
  - ProcessingStates, 172
  - SetDesc, 172
  - StagedStates, 172
  - str, 172
  - ToProcessStates, 173
- DataStaging::Generator, 201
- DataStaging::Generator
  - receiveDTR, 201
  - run, 201
- DataStaging::Processor, 330
- DataStaging::Processor

- ~Processor, 330
- Processor, 330
- receiveDTR, 331
- start, 331
- stop, 331
- DataStaging::Scheduler, 349
- DataStaging::Scheduler
  - ~Scheduler, 349
  - AddURLMapping, 350
  - cancelDTRs, 350
  - receiveDTR, 350
  - Scheduler, 349
  - SetDeliveryServices, 350
  - SetDumpLocation, 350
  - SetPreferredPattern, 350
  - SetRemoteSizeLimit, 350
  - SetSlots, 350
  - SetTransferParameters, 350
  - SetTransferSharesConf, 350
  - SetURLMapping, 351
  - start, 351
  - stop, 351
- DataStaging::TransferParameters, 405
- DataStaging::TransferParameters
  - averaging\_time, 405
  - max\_inactivity\_time, 405
  - min\_average\_bandwidth, 405
  - min\_current\_bandwidth, 405
  - TransferParameters, 405
- DataStaging::TransferShares, 406
- DataStaging::TransferShares
  - ~TransferShares, 406
  - active\_shares, 407
  - calculate\_shares, 407
  - can\_start, 407
  - decrease\_number\_of\_slots, 407
  - decrease\_transfer\_share, 407
  - increase\_transfer\_share, 407
  - operator=, 407
  - set\_shares\_conf, 407
  - TransferShares, 406
- DataStaging::TransferSharesConf, 408
  - GROUP, 408
  - NONE, 408
  - ROLE, 408
  - USER, 408
  - VO, 408
- DataStaging::TransferSharesConf
  - conf, 409
  - extract\_share\_info, 409
  - get\_basic\_priority, 409
  - is\_configured, 409
  - set\_reference\_share, 409
  - set\_reference\_shares, 409
  - set\_share\_type, 409
  - ShareType, 408
  - TransferSharesConf, 409
- dec
  - Arc::SimpleCounter, 366
- decrease\_number\_of\_slots
  - DataStaging::TransferShares, 407
- decrease\_transfer\_share
  - DataStaging::TransferShares, 407
- decrease\_tries\_left
  - DataStaging::DTR, 152
- DecryptNode
  - Arc::XMLSecNode, 484
- DEFAULT\_BROKER
  - Arc::UserConfig, 448
- DEFAULT\_LOCK\_TIMEOUT
  - Arc::FileLock, 198
- DEFAULT\_TIMEOUT
  - Arc::UserConfig, 448
- DEFAULTCONFIG
  - Arc::UserConfig, 448
- Delegate
  - Arc::DelegationProvider, 142
- DelegateCredentialsInit
  - Arc::DelegationConsumerSOAP, 137
  - Arc::DelegationContainerSOAP, 139
  - Arc::DelegationProviderSOAP, 145
- DelegatedToken
  - Arc::DelegationConsumerSOAP, 137
  - Arc::DelegationContainerSOAP, 140
  - Arc::DelegationProviderSOAP, 145
- DelegationConsumer
  - Arc::DelegationConsumer, 135
- DelegationConsumerSOAP
  - Arc::DelegationConsumerSOAP, 137
- DelegationProvider
  - Arc::DelegationProvider, 142
- DelegationProviderSOAP
  - Arc::DelegationProviderSOAP, 144
- delete\_dtr
  - DataStaging::DTRLList, 167
- deleteDestinations
  - Arc::Logger, 252
- Description
  - Arc::JobIdentificationType, 235
- Destroy
  - Arc::XMLNode, 474
- destroy\_doc
  - Arc::ConfusaParserUtils, 89
- DirCreate
  - Arc, 32
- DirDelete
  - Arc, 32
- disconnect\_logger

- DataStaging::DTR, 152
- doc\_
  - Arc::InformationContainer, 210
- DONE
  - DataStaging::DTRStatus, 171
- drain\_cache\_dirs
  - DataStaging::DTRCacheParameters, 160
- DTR
  - DataStaging::DTR, 151
- dtr\_id
  - DataStaging::DataDeliveryComm, 124
- DTR\_ptr
  - DataStaging, 47
- DTRCacheParameters
  - DataStaging::DTRCacheParameters, 160
- DTRErrorLocation
  - DataStaging::DTRErrorStatus, 164
- DTRErrorStatus
  - DataStaging::DTRErrorStatus, 164
- DTRErrorStatusType
  - DataStaging::DTRErrorStatus, 164
- DTRLogger
  - DataStaging, 47
- DTRStatus
  - DataStaging::DTRStatus, 171
- DTRStatusType
  - DataStaging::DTRStatus, 170
- dumpState
  - DataStaging::DTRLList, 167
- Dup
  - Arc::ThreadDataItem, 394
- duplicate
  - ArcSec::RequestAttribute, 338
- empty
  - Arc::Software, 373
  - Arc::SoftwareRequirement, 380
  - DataStaging::DTRLList, 167
- enable\_ssl
  - Arc::Database, 117
  - Arc::MySQLDatabase, 292
- encode
  - ArcSec::AttributeValue, 62
  - ArcSec::DateTimeAttribute, 133
  - ArcSec::DurationAttribute, 174
  - ArcSec::PeriodAttribute, 311
  - ArcSec::TimeAttribute, 403
- EncryptNode
  - Arc::XMLSecNode, 485
- end
  - Arc::Adler32Sum, 50
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 74
  - Arc::CRC32Sum, 102
  - Arc::MD5Sum, 272
- end\_
  - Arc::AttributeIterator, 60
- EnvLockUnwrap
  - Arc, 38
- EnvLockUnwrapComplete
  - Arc, 38
- EnvLockWrap
  - Arc, 37
- EQUAL
  - Arc::Software, 371
- equal
  - Arc::CStringValue, 78
  - ArcSec::AttributeValue, 62
  - ArcSec::DateTimeAttribute, 133
  - ArcSec::DurationAttribute, 174
  - ArcSec::PeriodAttribute, 311
  - ArcSec::TimeAttribute, 403
- ERROR
  - DataStaging::DTRStatus, 171
- error
  - DataStaging::DataDeliveryComm::Status, 126
  - DataStaging::DTR, 152
- error\_desc
  - DataStaging::DataDeliveryComm::Status, 126
- ERROR\_DESTINATION
  - DataStaging::DTRErrorStatus, 164
- error\_location
  - DataStaging::DataDeliveryComm::Status, 126
- ERROR\_SOURCE
  - DataStaging::DTRErrorStatus, 164
- ERROR\_TRANSFER
  - DataStaging::DTRErrorStatus, 164
- ERROR\_UNKNOWN
  - DataStaging::DTRErrorStatus, 164
- escape\_chars
  - Arc, 37
- escape\_hex
  - Arc, 29
- escape\_octal
  - Arc, 29
- escape\_type
  - Arc, 29
- ETERNAL
  - Arc, 43
- eval
  - ArcSec::Policy, 326
- evaluate
  - ArcSec::EqualFunction, 176
  - ArcSec::Evaluator, 181
  - ArcSec::Function, 200
  - ArcSec::MatchFunction, 260
- evaluate\_path
  - Arc::ConfusaParserUtils, 89

- EvaluationCtx
  - ArcSec::EvaluationCtx, 179
- EXAMPLECONFIG
  - Arc::UserConfig, 448
- Exchange
  - Arc::XMLNode, 474
- ExecutionTarget
  - Arc::ExecutionTarget, 187
- ExitTime
  - Arc::Run, 344
- ExpirationReminder
  - Arc::Counter, 99
- Export
  - Arc::MessageAuth, 281
  - Arc::MultiSecAttr, 291
  - Arc::SecAttr, 353
- extend
  - Arc::Counter, 96
  - Arc::CounterTicket, 101
  - Arc::IntraProcessCounter, 217
- extract\_body\_information
  - Arc::ConfusaParserUtils, 89
- extract\_share\_info
  - DataStaging::TransferSharesConf, 409
- factory\_
  - Arc::Loader, 245
- fallocate
  - Arc::FileAccess, 192
- FaultTo
  - Arc::WSAHeader, 458
- FileChecksum
  - Arc::ChecksumAny, 75
- FileCopy
  - Arc, 30, 31
- FileCreate
  - Arc, 31
- FileDelete
  - Arc, 32
- FileLink
  - Arc, 31
- FileLock
  - Arc::FileLock, 196
- FileRead
  - Arc, 31
- FileReadLink
  - Arc, 31, 32
- FileStat
  - Arc, 31
- Filter
  - Arc::InfoFilter, 204
  - Arc::MessageAuth, 281
- filter\_dtrs\_by\_job
  - DataStaging::DTRList, 167
- filter\_dtrs\_by\_next\_receiver
  - DataStaging::DTRList, 167
- filter\_dtrs\_by\_owner
  - DataStaging::DTRList, 167
- filter\_dtrs\_by\_status
  - DataStaging::DTRList, 167
- filter\_dtrs\_by\_statuses
  - DataStaging::DTRList, 168
- filter\_pending\_dtrs
  - DataStaging::DTRList, 168
- FilterByKind
  - Arc::PluginsFactory, 323
- final\_xmlsec
  - Arc, 41
- find
  - Arc::ModuleManager, 289
- FindConsumer
  - Arc::DelegationContainerSOAP, 140
- findLocation
  - Arc::ModuleManager, 289
- forceReset
  - Arc::SimpleCounter, 366
- From
  - Arc::WSAHeader, 458
- fstat
  - Arc::FileAccess, 192
- ftruncate
  - Arc::FileAccess, 192
- FullName
  - Arc::XMLNode, 475
- FullPath
  - Arc::URL, 416
- FullPathURIEncoded
  - Arc::URL, 416
- fullstr
  - Arc::URL, 416
  - Arc::URLLocation, 423
- GACL
  - Arc::SecAttr, 354
- Generate
  - Arc::DelegationConsumer, 136
- GenerateEECRequest
  - Arc::Credential, 108
- GenerateRequest
  - Arc::Credential, 109
- GENERIC\_ERROR
  - Arc, 29
- Get
  - Arc::ArcLocation, 54
  - Arc::InformationContainer, 210
  - Arc::InformationInterface, 211
  - Arc::PayloadStream, 304
  - Arc::PayloadStreamInterface, 306, 307

- Arc::ThreadDataItem, 394
- Arc::XMLNode, 475
- ArcSec::Source, 386
- get
  - Arc::MessageAttributes, 279
  - Arc::MessageAuth, 281
  - Arc::SecAttr, 353
  - Arc::SimpleCounter, 366
- get\_basic\_priority
  - DataStaging::TransferSharesConf, 409
- get\_bulk\_end
  - DataStaging::DTR, 152
- get\_bulk\_start
  - DataStaging::DTR, 152
- get\_bytes\_transferred
  - DataStaging::DTR, 152
- get\_cache\_file
  - DataStaging::DTR, 152
- get\_cache\_parameters
  - DataStaging::DTR, 152
- get\_cache\_state
  - DataStaging::DTR, 153
- get\_callbacks
  - DataStaging::DTR, 153
- get\_cert\_str
  - Arc, 41
- get\_creation\_time
  - DataStaging::DTR, 153
- get\_delivery\_endpoint
  - DataStaging::DTR, 153
- get\_destination
  - DataStaging::DTR, 153
- get\_destination\_str
  - DataStaging::DTR, 153
- get\_doc
  - Arc::ConfusaParserUtils, 89
- get\_error\_status
  - DataStaging::DTR, 153
- get\_factory
  - Arc::PluginArgument, 320
- get\_id
  - DataStaging::DTR, 153
- get\_initial\_tries
  - DataStaging::DTR, 153
- get\_key\_from\_certfile
  - Arc, 42
- get\_key\_from\_certstr
  - Arc, 42
- get\_key\_from\_keyfile
  - Arc, 42
- get\_key\_from\_keystir
  - Arc, 42
- get\_local\_user
  - DataStaging::DTR, 153
- get\_logger
  - DataStaging::DTR, 153
- get\_mapped\_source
  - DataStaging::DTR, 154
- get\_modification\_time
  - DataStaging::DTR, 154
- get\_module
  - Arc::PluginArgument, 320
- get\_node
  - Arc, 43
- get\_owner
  - DataStaging::DTR, 154
- get\_parent\_job\_id
  - DataStaging::DTR, 154
- get\_plugin\_instance
  - Arc, 28
- get\_priority
  - DataStaging::DTR, 154
- get\_problematic\_delivery\_services
  - DataStaging::DTR, 154
- get\_process\_time
  - DataStaging::DTR, 154
- get\_short\_id
  - DataStaging::DTR, 154
- get\_source
  - DataStaging::DTR, 154
- get\_source\_str
  - DataStaging::DTR, 154
- get\_status
  - DataStaging::DTR, 154
- get\_sub\_share
  - DataStaging::DTR, 155
- get\_timeout
  - DataStaging::DTR, 155
- get\_token
  - Arc, 36
- get\_transfer\_share
  - DataStaging::DTR, 155
- get\_tries\_left
  - DataStaging::DTR, 155
- get\_usercfg
  - DataStaging::DTR, 155
- getAlgFactory
  - ArcSec::Evaluator, 181
- getalgId
  - ArcSec::CombiningAlg, 84
  - ArcSec::DenyOverridesCombiningAlg, 146
  - ArcSec::PermitOverridesCombiningAlg, 313
- getAll
  - Arc::MessageAttributes, 279
  - Arc::SecAttr, 353
- getAttrFactory
  - ArcSec::Evaluator, 181
- getAttribute

- ArcSec::AttributeProxy, 61
- GetCAName
  - Arc::Credential, 109
- GetCert
  - Arc::Credential, 109
- GetCertNumofChain
  - Arc::Credential, 109
- GetCertReq
  - Arc::Credential, 109
- getCertRequestB64
  - Arc::ConfusaCertHandler, 88
- getComparisonOperatorList
  - Arc::SoftwareRequirement, 380
- getCounterTicket
  - Arc::Counter, 96
- getCredentialProperty
  - Arc, 40
- getCurrentTime
  - Arc::Counter, 96
- GetDesc
  - DataStaging::DTRErrorStatus, 165
  - DataStaging::DTRStatus, 172
- getDestinations
  - Arc::Logger, 252
- GetDN
  - Arc::Credential, 109
- GetDoc
  - Arc::XMLNode, 475
- getEffect
  - ArcSec::Policy, 326
- GetEndTime
  - Arc::Credential, 109
- GetEntry
  - Arc::ClientSOAP, 81
- GetEnv
  - Arc, 37
- geterrno
  - Arc::FileAccess, 192
- GetError
  - DataStaging::DataDeliveryComm, 123
- GetErrorLocation
  - DataStaging::DTRErrorStatus, 165
- GetErrorStatus
  - DataStaging::DTRErrorStatus, 165
- getEvalName
  - ArcSec::Policy, 326
  - ArcSec::Request, 337
- getEvalResult
  - ArcSec::Policy, 326
- getEvaluator
  - ArcSec::EvaluatorLoader, 184
- getExcess
  - Arc::Counter, 97
  - Arc::IntraProcessCounter, 218
- getExpirationReminder
  - Arc::Counter, 97
- getExpiryTime
  - Arc::Counter, 97
  - Arc::ExpirationReminder, 189
- getExplanation
  - Arc::MCC\_Status, 265
- GetExtension
  - Arc::Credential, 109
- getFamily
  - Arc::Software, 373
- getFileName
  - Arc::Config, 87
- getFnFactory
  - ArcSec::Evaluator, 182
- GetFormat
  - Arc::Time, 401
- getFormat\_BIO
  - Arc::Credential, 110
- getFunctionName
  - ArcSec::EqualFunction, 176
  - ArcSec::MatchFunction, 260
- getID
  - Arc::Service, 362
- getId
  - ArcSec::AttributeValue, 62
  - ArcSec::DateTimeAttribute, 133
  - ArcSec::DurationAttribute, 174
  - ArcSec::PeriodAttribute, 311
  - ArcSec::TimeAttribute, 403
- GetIdentityName
  - Arc::Credential, 110
- getInstance
  - DataStaging::DataDeliveryCommHandler, 128
- GetIssuerName
  - Arc::Credential, 110
- GetJobDescriptionParsers
  - Arc::JobDescriptionParserLoader, 233
- GetJobs
  - Arc::JobSupervisor, 240
- getKind
  - Arc::MCC\_Status, 265
- GetLastErrorState
  - DataStaging::DTRErrorStatus, 165
- getLevel
  - Arc::LogMessage, 257
- GetLifeTime
  - Arc::Credential, 110
- getLimit
  - Arc::Counter, 97
  - Arc::IntraProcessCounter, 218
- getLockSuffix
  - Arc::FileLock, 197

- getName
  - Arc::Software, 373
  - ArcSec::Evaluator, 182
  - ArcSec::Policy, 326
  - ArcSec::Request, 337
- getOrigin
  - Arc::MCC\_Status, 266
- GetOverlay
  - Arc::BaseConfig, 69
- getPattern
  - Arc::RegularExpression, 333
- GetPlugins
  - Arc::ArcLocation, 54
- getPolicy
  - ArcSec::EvaluatorLoader, 184
- GetPrivKey
  - Arc::Credential, 110
- GetProxyPolicy
  - Arc::Credential, 110
- GetPubKey
  - Arc::Credential, 110
- getRequest
  - ArcSec::EvaluatorLoader, 184, 185
- getRequestItems
  - ArcSec::Request, 337
- getReservationID
  - Arc::ExpirationReminder, 189
- GetRoot
  - Arc::XMLNode, 475
- getRootLogger
  - Arc::Logger, 252
- getSoftwareList
  - Arc::SoftwareRequirement, 380
- GetSourceLanguage
  - Arc::JobDescription, 229
- GetStartTime
  - Arc::Credential, 110
- GetStatus
  - DataStaging::DataDeliveryComm, 123
  - DataStaging::DTRStatus, 172
- getThreshold
  - Arc::Logger, 253
- GetTime
  - Arc::Time, 401
- GetType
  - Arc::Credential, 110
- getType
  - ArcSec::AttributeValue, 62
  - ArcSec::DateTimeAttribute, 133
  - ArcSec::DurationAttribute, 174
  - ArcSec::PeriodAttribute, 311
  - ArcSec::TimeAttribute, 403
- GetUser
  - Arc::UserConfig, 433
- getValue
  - Arc::Counter, 98
  - Arc::IntraProcessCounter, 218
- GetVerification
  - Arc::Credential, 110
- getVersion
  - Arc::Software, 373
- GetXML
  - Arc::XMLNode, 475
- GREATERTHAN
  - Arc::Software, 371
- GREATERTHANOREQUAL
  - Arc::Software, 372
- GROUP
  - DataStaging::TransferSharesConf, 408
- GUID
  - Arc, 33
- handle\_
  - Arc::PayloadStream, 305
- handle\_redirect\_step
  - Arc::ConfusaParserUtils, 89
- HandleOpenSSLError
  - Arc, 40, 41
- handler\_
  - DataStaging::DataDeliveryComm, 124
- hasAction
  - Arc::WSAHeader, 458
- hasAddress
  - Arc::WSAEndpointReference, 456
- hasMessageID
  - Arc::WSAHeader, 458
- hasMore
  - Arc::AttributeIterator, 59
- hasPattern
  - Arc::RegularExpression, 333
- hasRelatesTo
  - Arc::WSAHeader, 458
- hasRelationshipType
  - Arc::WSAHeader, 458
- hasTo
  - Arc::WSAHeader, 459
- header\_allocated\_
  - Arc::WSAHeader, 460
- HISTORIC
  - Arc, 43
- Holders
  - Arc::ThreadedPointer, 395
- Host
  - Arc::URL, 416
- host
  - Arc::URL, 419
- host\_cert\_for\_remote\_delivery
  - DataStaging::DTR, 155

- HTTPOption
  - Arc::URL, 416
- HTTPOptions
  - Arc::URL, 416
- httpoptions
  - Arc::URL, 420
- ID
  - Arc::DelegationConsumer, 136
  - Arc::DelegationProviderSOAP, 145
- IdPName
  - Arc::UserConfig, 433, 434
- IDType
  - Arc::Counter, 95
- Import
  - Arc::SecAttr, 353
- inc
  - Arc::SimpleCounter, 366
- increase\_transfer\_share
  - DataStaging::TransferShares, 407
- InfoCache
  - Arc::InfoCache, 203
- InfoFilter
  - Arc::InfoFilter, 204
- InfoRegisters
  - Arc::InfoRegisters, 207
- InformationContainer
  - Arc::InformationContainer, 209
- InformationInterface
  - Arc::InformationInterface, 211
- InformationRequest
  - Arc::InformationRequest, 213
- InformationResponse
  - Arc::InformationResponse, 214
- Init
  - Arc::ArcLocation, 54
- init\_xmlsec
  - Arc, 41
- InitializeCredentials
  - Arc::UserConfig, 434
- InitProxyCertInfo
  - Arc::Credential, 110
- InquireRequest
  - Arc::Credential, 111
- Insert
  - Arc::PayloadRawInterface, 301
- INTERNAL\_LOGIC\_ERROR
  - DataStaging::DTRErrorStatus, 164
- INTERNAL\_PROCESS\_ERROR
  - DataStaging::DTRErrorStatus, 164
- IntraProcessCounter
  - Arc::IntraProcessCounter, 216
- inttostr
  - Arc, 35
- ip6addr
  - Arc::URL, 420
- is\_being\_cached
  - DataStaging::DTRLList, 168
- is\_configured
  - DataStaging::TransferSharesConf, 409
- is\_destined\_for\_delivery
  - DataStaging::DTR, 155
- is\_destined\_for\_post\_processor
  - DataStaging::DTR, 155
- is\_destined\_for\_pre\_processor
  - DataStaging::DTR, 155
- is\_force\_registration
  - DataStaging::DTR, 155
- is\_in\_final\_state
  - DataStaging::DTR, 156
- is\_owner\_
  - Arc::XMLNode, 481
- is\_replication
  - DataStaging::DTR, 156
- is\_rfc\_proxy
  - DataStaging::DTR, 156
- is\_temporary\_
  - Arc::XMLNode, 481
- isconnected
  - Arc::Database, 117
  - Arc::MySQLDatabase, 293
- IsCredentialsValid
  - Arc::Credential, 111
- IsFinished
  - Arc::JobState, 237
- isOk
  - Arc::MCC\_Status, 266
  - Arc::RegularExpression, 334
- isResolved
  - Arc::SoftwareRequirement, 381
- isSatisfied
  - Arc::SoftwareRequirement, 381, 382
- IsSecureProtocol
  - Arc::URL, 416
- istring\_to\_level
  - Arc, 33
- IsValid
  - Arc::Credential, 111
- isValid
  - Arc::CounterTicket, 101
- Job
  - Arc::Job, 220
- JobControllerPluginLoader
  - Arc::JobControllerPluginLoader, 227
- JobDescriptionParserLoader
  - Arc::JobDescriptionParserLoader, 233
- JobDownloadDirectory



- Arc::UserConfig, 436
- JobListFile
  - Arc::UserConfig, 436
- JobName
  - Arc::JobIdentificationType, 235
- JobSupervisor
  - Arc::JobSupervisor, 238
- KeepStderr
  - Arc::Run, 344
- KeepStdin
  - Arc::Run, 344
- KeepStdout
  - Arc::Run, 344
- key
  - Arc::AttributeIterator, 59
- KeyPassword
  - Arc::UserConfig, 437
- KeyPath
  - Arc::UserConfig, 438
- KeySize
  - Arc::UserConfig, 438, 439
- Kill
  - Arc::Run, 344
- LDAPAttributes
  - Arc::URL, 416
- ldapattributes
  - Arc::URL, 420
- LDAPFilter
  - Arc::URL, 417
- ldapfilter
  - Arc::URL, 420
- LDAPScope
  - Arc::URL, 417
- ldapscope
  - Arc::URL, 420
- LESSTHAN
  - Arc::Software, 372
- LESSTHANOREQUAL
  - Arc::Software, 372
- level\_to\_string
  - Arc, 34
- Limit
  - Arc::PayloadStream, 304
  - Arc::PayloadStreamInterface, 307
- link
  - Arc::FileAccess, 192
- Load
  - Arc::ClientSOAP, 82
- load
  - Arc::JobControllerPluginLoader, 227
  - Arc::JobDescriptionParserLoader, 233
  - Arc::ModuleManager, 289
  - Arc::PluginsFactory, 323
  - Arc::SubmitterPluginLoader, 391
  - load\_key\_from\_certfile
    - Arc, 42
  - load\_key\_from\_certstr
    - Arc, 42
  - load\_key\_from\_keyfile
    - Arc, 42
  - load\_trusted\_cert\_file
    - Arc, 42
  - load\_trusted\_cert\_str
    - Arc, 42
  - load\_trusted\_certs
    - Arc, 42
  - LoadConfigurationFile
    - Arc::UserConfig, 439
  - Loader
    - Arc::Loader, 245
  - LOCAL\_DELIVERY
    - DataStaging::DTR, 159
  - LOCAL\_FILE\_ERROR
    - DataStaging::DTRErrorStatus, 164
  - Location
    - Arc::RemoteLoggingType, 335
  - Locations
    - Arc::URL, 417
  - locations
    - Arc::URL, 420
  - lock
    - Arc::SimpleCondition, 364
  - lock\_
    - Arc::InformationInterface, 212
    - DataStaging::DataDeliveryComm, 124
  - LOCK\_SUFFIX
    - Arc::FileLock, 198
  - log
    - Arc::LogDestination, 246
    - Arc::LogFile, 249
    - Arc::LogStream, 259
  - LogDestination
    - Arc::LogDestination, 246
  - LogError
    - Arc::Credential, 111
    - Arc::DelegationConsumer, 136
  - LogFile
    - Arc::LogFile, 248
  - LogFormat
    - Arc, 29
  - Logger
    - Arc::Logger, 251, 252
    - Arc::LogMessage, 257
  - logger
    - Arc::MCC, 263
    - Arc::Plexer, 316

- Arc::Service, 363
- logger\_
  - DataStaging::DataDeliveryComm, 124
- LogLevel
  - Arc, 29
- LogMessage
  - Arc::LogMessage, 256
- LogStream
  - Arc::LogStream, 258
- lower
  - Arc, 35
- lseek
  - Arc::FileAccess, 192
- lstat
  - Arc::FileAccess, 193
- make\_policy
  - ArcSec::Policy, 326
- make\_request
  - ArcSec::Request, 337
- MakeConfig
  - Arc::BaseConfig, 69
- makePersistent
  - Arc::ModuleManager, 289
- match
  - Arc::RegularExpression, 334
  - ArcSec::Policy, 326
- MatchXMLName
  - Arc, 38
  - Arc::XMLNode, 480, 481
- MatchXMLNamespace
  - Arc, 38
  - Arc::XMLNode, 481
- max\_duration\_
  - Arc::DelegationContainerSOAP, 140
- max\_inactivity\_time
  - DataStaging::TransferParameters, 405
- max\_size\_
  - Arc::DelegationContainerSOAP, 141
- max\_usage\_
  - Arc::DelegationContainerSOAP, 141
- MCC
  - Arc::MCC, 263
- MCC\_Status
  - Arc::MCC\_Status, 265
- MCCLoader
  - Arc::MCCLoader, 270
- Message
  - Arc::Message, 276
- MessageAttributes
  - Arc::AttributeIterator, 60
  - Arc::MessageAttributes, 278
- MessageID
  - Arc::WSAHeader, 459
- MetaData
  - Arc::WSAEndpointReference, 456
- MetaDataOption
  - Arc::URL, 417
- MetaDataOptions
  - Arc::URL, 417
- metadataoptions
  - Arc::URL, 420
- Migrate
  - Arc::JobSupervisor, 240
  - Arc::SubmitterPlugin, 389
- min\_average\_bandwidth
  - DataStaging::TransferParameters, 405
- min\_current\_bandwidth
  - DataStaging::TransferParameters, 405
- mkdir
  - Arc::FileAccess, 193
- mkdirp
  - Arc::FileAccess, 193
- mkstemp
  - Arc::FileAccess, 193
- ModuleManager
  - Arc::ModuleManager, 288
- Move
  - Arc::XMLNode, 475
- msg
  - Arc::Logger, 253
- Name
  - Arc::URLLocation, 423
  - Arc::XMLNode, 475, 476
- name
  - Arc::URLLocation, 423
- Namespace
  - Arc::XMLNode, 476
- NamespacePrefix
  - Arc::XMLNode, 476
- Namespaces
  - Arc::XMLNode, 476
- NEW
  - DataStaging::DTRStatus, 170
- New
  - Arc::XMLNode, 476
- NewAttribute
  - Arc::XMLNode, 476
- NewChild
  - Arc::XMLNode, 476, 477
- NewReferenceParameter
  - Arc::WSAHeader, 459
- Next
  - Arc::MCC, 263
  - Arc::Plexer, 316
- next\_
  - Arc::MCC, 264

- NO\_ERROR\_LOCATION
  - DataStaging::DTRErrorStatus, 164
- Nodes
  - Arc::XMLNodeContainer, 483
- NON\_CACHEABLE
  - DataStaging, 48
- NONE
  - DataStaging::TransferSharesConf, 408
- NONE\_ERROR
  - DataStaging::DTRErrorStatus, 164
- NOTEQUAL
  - Arc::Software, 371
- NULL\_STATE
  - DataStaging::DTRStatus, 171
- number\_of\_dtrs\_by\_owner
  - DataStaging::DTRLList, 168
- OAuthConsumer
  - Arc::OAuthConsumer, 294
- offset
  - DataStaging::DataDeliveryComm::Status, 126
- old\_level\_to\_level
  - Arc, 34
- open
  - Arc::FileAccess, 193
- opendir
  - Arc::FileAccess, 193
- OpenSSLInit
  - Arc, 40
- operator \*
  - Arc::AttributeIterator, 59
  - Arc::AutoPointer, 66
  - Arc::CountedPointer, 91
  - Arc::PathIterator, 296
  - Arc::ThreadedPointer, 395
- operator AlgFactory \*
  - ArcSec::EvaluatorContext, 183
- operator AttributeFactory \*
  - ArcSec::EvaluatorContext, 183
- operator bool
  - Arc::Adler32Sum, 50
  - Arc::AutoPointer, 66
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 75
  - Arc::CIStrngValue, 78
  - Arc::CountedPointer, 91
  - Arc::CRC32Sum, 103
  - Arc::FileAccess, 193
  - Arc::LogFile, 249
  - Arc::MCC\_Status, 266
  - Arc::MD5Sum, 273
  - Arc::MultiSecAttr, 291
  - Arc::PathIterator, 296
  - Arc::PayloadStream, 304
  - Arc::PayloadStreamInterface, 307
  - Arc::Run, 344
  - Arc::SAMLToken, 348
  - Arc::SecAttr, 353
  - Arc::SecAttrValue, 356
  - Arc::Service, 362
  - Arc::ThreadedPointer, 395
  - Arc::URL, 417
  - Arc::UserConfig, 440
  - Arc::UsernameToken, 451
  - Arc::WSRF, 462
  - Arc::X509Token, 470
  - Arc::XMLNode, 477
- operator FnFactory \*
  - ArcSec::EvaluatorContext, 183
- operator PluginsFactory \*
  - Arc::ChainContext, 70
- operator std::string
  - Arc::MCC\_Status, 266
  - Arc::Software, 373
  - Arc::Time, 401
  - Arc::XMLNode, 477
- operator XMLNode
  - Arc::WSAEndpointReference, 456
  - Arc::WSAHeader, 459
- operator!
  - Arc::Adler32Sum, 50
  - Arc::AutoPointer, 67
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 75
  - Arc::CountedPointer, 91
  - Arc::CRC32Sum, 103
  - Arc::FileAccess, 193
  - Arc::LogFile, 249
  - Arc::MCC\_Status, 266
  - Arc::MD5Sum, 273
  - Arc::PayloadStream, 304
  - Arc::PayloadStreamInterface, 307
  - Arc::Run, 344
  - Arc::Service, 362
  - Arc::ThreadedPointer, 395
  - Arc::UserConfig, 441
  - Arc::XMLNode, 477
  - DataStaging::DataDeliveryComm, 123
  - DataStaging::DataDeliveryLocalComm, 129
  - DataStaging::DataDeliveryRemoteComm, 131
  - DataStaging::DTR, 156
- operator!=

- Arc::CountedPointer, 91
- Arc::SecAttr, 353
- Arc::SecAttrValue, 356
- Arc::Software, 374
- Arc::ThreadedPointer, 396
- Arc::Time, 401
- Arc::XMLNode, 477
- DataStaging::DTRErrorStatus, 165
- DataStaging::DTRStatus, 172
- operator()
  - Arc::Software, 374
- operator+
  - Arc::Time, 401
- operator++
  - Arc::AttributeIterator, 59
  - Arc::PathIterator, 296
  - Arc::XMLNode, 478
- operator-
  - Arc::Time, 401
- operator-
  - Arc::PathIterator, 296
  - Arc::XMLNode, 478
- operator->
  - Arc::AttributeIterator, 60
  - Arc::AutoPointer, 67
  - Arc::CountedPointer, 92
  - Arc::ThreadedPointer, 396
- operator<
  - Arc::CountedPointer, 92
  - Arc::ExpirationReminder, 189
  - Arc::Software, 374
  - Arc::ThreadedPointer, 396
  - Arc::Time, 401
  - Arc::URL, 417
- operator<<
  - Arc, 30, 33
  - Arc::LogMessage, 257
  - Arc::Software, 376
  - Arc::URL, 419
- operator<=
  - Arc::Software, 374
  - Arc::Time, 401
- operator=
  - Arc::Job, 221
  - Arc::Message, 276
  - Arc::RegularExpression, 334
  - Arc::SoftwareRequirement, 382
  - Arc::Time, 401, 402
  - Arc::WSAEndpointReference, 456
  - Arc::XMLNode, 478
  - Arc::XMLNodeContainer, 483
  - DataStaging::DTRErrorStatus, 165
  - DataStaging::DTRStatus, 172
  - DataStaging::TransferShares, 407
- operator==
  - Arc::CountedPointer, 92
  - Arc::SecAttr, 353
  - Arc::SecAttrValue, 356
  - Arc::Software, 375
  - Arc::ThreadedPointer, 396
  - Arc::Time, 402
  - Arc::URL, 417
  - Arc::XMLNode, 478
  - DataStaging::DTRErrorStatus, 165
  - DataStaging::DTRStatus, 172
- operator>
  - Arc::Software, 375
  - Arc::Time, 402
- operator>=
  - Arc::Software, 376
  - Arc::Time, 402
- operator[]
  - Arc::MCCLoader, 271
  - Arc::MessageAuth, 281
  - Arc::PayloadRawInterface, 301
  - Arc::XMLNode, 478, 479
  - Arc::XMLNodeContainer, 483
- Option
  - Arc::URL, 417
- optional
  - Arc::RemoteLoggingType, 335
- Options
  - Arc::URL, 418
- OptionString
  - Arc::URL, 418
- OtherAttributes
  - Arc::JobDescription, 231
- OutputCertificate
  - Arc::Credential, 111
- OutputCertificateChain
  - Arc::Credential, 111
- OutputPrivateKey
  - Arc::Credential, 112
- OutputPublicKey
  - Arc::Credential, 112
- OverlayFile
  - Arc::UserConfig, 441
- Parent
  - Arc::XMLNode, 479
- Parse
  - Arc::JobDescription, 229
- parse
  - Arc::Config, 87
- parseDN
  - Arc::OAuthConsumer, 294
- ParseExecutionTargets
  - Arc::GLUE2, 202

- ParseOptions
  - Arc::URL, 418
- ParsePath
  - Arc::URL, 418
- parsePolicy
  - ArcSec::PolicyParser, 328
- parseVOMSAC
  - Arc, 39, 40
- PARSING\_ERROR
  - Arc, 30
- passphrase\_callback
  - Arc, 41
- Passwd
  - Arc::URL, 418
- passwd
  - Arc::URL, 420
- Password
  - Arc::UserConfig, 441, 442
- PasswordType
  - Arc::UsernameToken, 450
- Path
  - Arc::ExecutableType, 186
  - Arc::URL, 418
  - Arc::XMLNode, 479
- path
  - Arc::URL, 420
- Path2BaseDN
  - Arc::URL, 418
- PathIterator
  - Arc::PathIterator, 296
- Payload
  - Arc::Message, 277
  - Arc::SOAPMessage, 368, 369
- PayloadRaw
  - Arc::PayloadRaw, 298
- PayloadSOAP
  - Arc::PayloadSOAP, 302
- PayloadStream
  - Arc::PayloadStream, 303
- PayloadWSRF
  - Arc::PayloadWSRF, 309
- PERMANENT\_REMOTE\_ERROR
  - DataStaging::DTRErrorStatus, 164
- ping
  - Arc::FileAccess, 193
- plainstr
  - Arc::URL, 418
- Plexer
  - Arc::Plexer, 315
- Plugin
  - Arc::Plugin, 319
- plugins\_table\_name
  - Arc, 43
- PluginsFactory
  - Arc::PluginsFactory, 323
- Policy
  - ArcSec::Policy, 325, 326
- PolicyStore
  - ArcSec::PolicyStore, 329
- Port
  - Arc::URL, 418
- port
  - Arc::URL, 420
- Pos
  - Arc::PayloadStream, 304
  - Arc::PayloadStreamInterface, 307
- PRE\_CLEAN
  - DataStaging::DTRStatus, 171
- PRE\_CLEARED
  - DataStaging::DTRStatus, 171
- PRE\_CLEANNING
  - DataStaging::DTRStatus, 171
- pread
  - Arc::FileAccess, 193
- Prefix
  - Arc::XMLNode, 479
- Prepare
  - Arc::JobDescription, 230
- print
  - Arc::Adler32Sum, 50
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 75
  - Arc::Config, 87
  - Arc::CRC32Sum, 103
  - Arc::MD5Sum, 273
- process
  - Arc::ClientSOAP, 82
  - Arc::MCC, 263
  - Arc::MCCInterface, 268
  - Arc::Plexer, 316
- PROCESS\_CACHE
  - DataStaging::DTRStatus, 171
- PROCESSING\_CACHE
  - DataStaging::DTRStatus, 171
- ProcessingStates
  - DataStaging::DTRStatus, 172
- processLogin
  - Arc::OAuthConsumer, 294
- Processor
  - DataStaging::Processor, 330
- ProcessSecHandlers
  - Arc::MCC, 263
  - Arc::Service, 362
- ProcessState
  - DataStaging, 47
- Protocol
  - Arc::URL, 418
- protocol

- Arc::URL, 420
- PROTOCOL\_RECOGNIZED\_ERROR
  - Arc, 30
- ProxyPath
  - Arc::UserConfig, 442
- Ptr
  - Arc::AutoPointer, 67
  - Arc::CountedPointer, 92
  - Arc::ThreadedPointer, 396
- PullStatus
  - DataStaging::DataDeliveryComm, 124
  - DataStaging::DataDeliveryLocalComm, 130
  - DataStaging::DataDeliveryRemoteComm, 131
- push
  - DataStaging::DTR, 156
- pushCSR
  - Arc::OAuthConsumer, 294
- Put
  - Arc::PayloadStream, 304, 305
  - Arc::PayloadStreamInterface, 307
- pwrite
  - Arc::FileAccess, 193
- QUERY\_REPLICA
  - DataStaging::DTRStatus, 171
- QueryConsumer
  - Arc::DelegationContainerSOAP, 140
- QUERYING\_REPLICA
  - DataStaging::DTRStatus, 171
- read
  - Arc::FileAccess, 194
- ReadAllJobsFromFile
  - Arc::Job, 221
- readdir
  - Arc::FileAccess, 194
- ReadFromFile
  - Arc::XMLNode, 479
- ReadFromStream
  - Arc::XMLNode, 479
- ReadJobIDsFromFile
  - Arc::Job, 221
- ReadJobsFromFile
  - Arc::Job, 222
- readlink
  - Arc::FileAccess, 194
- ReadStderr
  - Arc::Run, 344
- ReadStdout
  - Arc::Run, 344
- ReadURLList
  - Arc, 37
- receiveDTR
  - DataStaging::DataDelivery, 119
- DataStaging::DTRCallback, 162
- DataStaging::Generator, 201
- DataStaging::Processor, 331
- DataStaging::Scheduler, 350
- ReferenceParameter
  - Arc::WSAHeader, 459
- ReferenceParameters
  - Arc::WSAEndpointReference, 456
- REGISTER\_REPLICA
  - DataStaging::DTRStatus, 171
- registerCallback
  - DataStaging::DTR, 156
- RegisteredService
  - Arc::RegisteredService, 332
- REGISTERING\_REPLICA
  - DataStaging::DTRStatus, 171
- RegisterJobSubmission
  - Arc::ExecutionTarget, 188
- RegisterThread
  - Arc::ThreadRegistry, 399
- registration
  - Arc::InfoRegistrar, 208
- RegistrationCollector
  - Arc::Service, 362
- RegularExpression
  - Arc::RegularExpression, 333
- RelatesTo
  - Arc::WSAHeader, 459
- RelationshipType
  - Arc::WSAHeader, 459
- Release
  - Arc::AutoPointer, 67
  - Arc::CountedPointer, 92
  - Arc::ThreadedPointer, 396
- release
  - Arc::FileLock, 197
- RELEASE\_REQUEST
  - DataStaging::DTRStatus, 171
- ReleaseConsumer
  - Arc::DelegationContainerSOAP, 140
- RELEASING\_REQUEST
  - DataStaging::DTRStatus, 171
- reload
  - Arc::ModuleManager, 289
- remote\_cache\_dirs
  - DataStaging::DTRCacheParameters, 160
- Remove
  - DataStaging::DataDeliveryCommHandler, 128
- remove
  - Arc::FileAccess, 194
  - Arc::MessageAttributes, 280
  - Arc::MessageAuth, 282
- removeAll

- Arc::MessageAttributes, 280
- RemoveConsumer
  - Arc::DelegationContainerSOAP, 140
- removeDestinations
  - Arc::Logger, 253
- RemoveHTTPOption
  - Arc::URL, 418
- RemoveJobsFromFile
  - Arc::Job, 223
- RemoveMetaDataOption
  - Arc::URL, 419
- RemoveOption
  - Arc::URL, 419
- removeService
  - Arc::InfoRegisterContainer, 206
  - Arc::InfoRegistrar, 208
- Renew
  - Arc::JobSupervisor, 241
- Replace
  - Arc::XMLNode, 479
- REPLICA\_QUERIED
  - DataStaging::DTRStatus, 171
- REPLICA\_REGISTERED
  - DataStaging::DTRStatus, 171
- ReplyTo
  - Arc::WSAHeader, 460
- report
  - Arc::PluginsFactory, 324
- Request
  - Arc::DelegationConsumer, 136
  - ArcSec::Request, 336
- REQUEST\_RELEASED
  - DataStaging::DTRStatus, 171
- RequestAttribute
  - ArcSec::RequestAttribute, 338
- RequestItem
  - ArcSec::RequestItem, 339
- reserve
  - Arc::Counter, 98
  - Arc::IntraProcessCounter, 218
- reset
  - Arc::SimpleCondition, 364
  - DataStaging::DTR, 156
- reset\_error\_status
  - DataStaging::DTR, 156
- RESOLVE
  - DataStaging::DTRStatus, 170
- RESOLVED
  - DataStaging::DTRStatus, 170
- RESOLVING
  - DataStaging::DTRStatus, 170
- Rest
  - Arc::PathIterator, 296
- Restore
  - Arc::DelegationConsumer, 136
- Resubmit
  - Arc::JobSupervisor, 241
- Result
  - Arc::InformationResponse, 214
  - Arc::Run, 344
- result
  - Arc::Adler32Sum, 50
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 75
  - Arc::CRC32Sum, 103
  - Arc::MD5Sum, 273
- Resume
  - Arc::JobSupervisor, 242
- Retrieve
  - Arc::JobSupervisor, 243
- rmdir
  - Arc::FileAccess, 194
- rmdirr
  - Arc::FileAccess, 194
- ROLE
  - DataStaging::TransferSharesConf, 408
- Run
  - Arc::Run, 342
- run
  - DataStaging::Generator, 201
- Running
  - Arc::Run, 345
- RunTime
  - Arc::Run, 345
- Same
  - Arc::XMLNode, 480
- SAML
  - Arc::SecAttr, 354
- SAMLTOKEN
  - Arc::SAMLToken, 347
- SAMLVersion
  - Arc::SAMLToken, 347
- save
  - Arc::Config, 87
- SaveToFile
  - Arc::UserConfig, 443
  - Arc::XMLNode, 480
- SaveToStream
  - Arc::ExecutionTarget, 188
  - Arc::Job, 223
  - Arc::JobDescription, 230
  - Arc::XMLNode, 480
- scan
  - Arc::Adler32Sum, 51
  - Arc::Checksum, 72
  - Arc::ChecksumAny, 76
  - Arc::CRC32Sum, 103

- Arc::MD5Sum, 273
  - Arc::PluginsFactory, 324
- Scheduler
  - DataStaging::Scheduler, 349
- Scope
  - Arc::URL, 414
- SecAttr
  - Arc::SecAttr, 352
- sechandlers\_
  - Arc::MCC, 264
  - Arc::Service, 363
- seekable\_
  - Arc::PayloadStream, 305
- selectSoftware
  - Arc::SoftwareRequirement, 382, 383
- SELF\_REPLICATION\_ERROR
  - DataStaging::DTRErrorStatus, 164
- SelfSignEECRequest
  - Arc::Credential, 112
- Service
  - Arc::Service, 362
- ServiceType
  - Arc::RemoteLoggingType, 335
- SESSION\_CLOSE
  - Arc, 30
- Set
  - Arc::XMLNode, 480
- set
  - Arc::MessageAttributes, 280
  - Arc::MessageAuth, 282
  - Arc::SimpleCounter, 366
- set\_bulk\_end
  - DataStaging::DTR, 156
- set\_bulk\_start
  - DataStaging::DTR, 157
- set\_bytes\_transferred
  - DataStaging::DTR, 157
- set\_cache\_file
  - DataStaging::DTR, 157
- set\_cache\_parameters
  - DataStaging::DTR, 157
- set\_cache\_state
  - DataStaging::DTR, 157
- set\_cancel\_request
  - DataStaging::DTR, 157
- set\_delivery\_endpoint
  - DataStaging::DTR, 157
- set\_error\_status
  - DataStaging::DTR, 157
- set\_force\_registration
  - DataStaging::DTR, 157
- set\_id
  - DataStaging::DTR, 157
- set\_mapped\_source
  - DataStaging::DTR, 158
- set\_namespaces
  - Arc::WSRF, 462
  - Arc::WSRFBBaseFault, 463
  - Arc::WSRP, 465
- set\_priority
  - DataStaging::DTR, 158
- set\_process\_time
  - DataStaging::DTR, 158
- set\_reference\_share
  - DataStaging::TransferSharesConf, 409
- set\_reference\_shares
  - DataStaging::TransferSharesConf, 409
- set\_replication
  - DataStaging::DTR, 158
- set\_rfc\_proxy
  - DataStaging::DTR, 158
- set\_share\_type
  - DataStaging::TransferSharesConf, 409
- set\_shares\_conf
  - DataStaging::TransferShares, 407
- set\_status
  - DataStaging::DTR, 158
- set\_sub\_share
  - DataStaging::DTR, 158
- set\_timeout
  - DataStaging::DTR, 158
- set\_transfer\_share
  - DataStaging::DTR, 158
- set\_tries\_left
  - DataStaging::DTR, 158
- setAttributeFactory
  - ArcSec::Request, 337
- setBackups
  - Arc::LogFile, 249
- setCfg
  - Arc::ModuleManager, 289
- setCombiningAlg
  - ArcSec::Evaluator, 182
- SetDeliveryServices
  - DataStaging::Scheduler, 350
- SetDesc
  - DataStaging::DTRStatus, 172
- SetDumpLocation
  - DataStaging::Scheduler, 350
- SetEnv
  - Arc, 37
- setEvalResult
  - ArcSec::Policy, 327
- setEvaluatorContext
  - ArcSec::Policy, 327
- setExcess
  - Arc::Counter, 98
  - Arc::IntraProcessCounter, 219



- setFileName
  - Arc::Config, 87
- SetFormat
  - Arc::Time, 402
- setIdentifier
  - Arc::LogMessage, 257
- SetLifeTime
  - Arc::Credential, 112
- setLimit
  - Arc::Counter, 99
  - Arc::IntraProcessCounter, 219
- setMaxSize
  - Arc::LogFile, 249
- SetPreferredPattern
  - DataStaging::Scheduler, 350
- SetProxyPolicy
  - Arc::Credential, 112
- SetRemoteSizeLimit
  - DataStaging::Scheduler, 350
- setReopen
  - Arc::LogFile, 249
- setRequestItems
  - ArcSec::Request, 337
- SetSlots
  - DataStaging::Scheduler, 350
- SetStartTime
  - Arc::Credential, 112
- setThreadContext
  - Arc::Logger, 253
- setThreshold
  - Arc::Logger, 253
- setThresholdForDomain
  - Arc::Logger, 254
- SetTime
  - Arc::Time, 402
- SetTransferParameters
  - DataStaging::DataDelivery, 120
  - DataStaging::Scheduler, 350
- SetTransferSharesConf
  - DataStaging::Scheduler, 350
- setuid
  - Arc::FileAccess, 194
- SetURLMapping
  - DataStaging::Scheduler, 351
- SetUser
  - Arc::UserConfig, 443
- ShareType
  - DataStaging::TransferSharesConf, 408
- shutdown
  - Arc::Database, 117
  - Arc::MySQLDatabase, 293
- signal
  - Arc::SimpleCondition, 364
- signal\_nonblock
  - Arc::SimpleCondition, 364
- SignEECRequest
  - Arc::Credential, 112, 113
- SignNode
  - Arc::XMLSecNode, 485
- SignRequest
  - Arc::Credential, 113
- Size
  - Arc::PayloadRaw, 299
  - Arc::PayloadRawInterface, 301
  - Arc::PayloadStream, 305
  - Arc::PayloadStreamInterface, 308
  - Arc::XMLNode, 480
  - Arc::XMLNodeContainer, 483
- size
  - DataStaging::DataDeliveryComm::Status, 127
- SLCS
  - Arc::UserConfig, 443
- SOAP
  - Arc::InformationRequest, 213
  - Arc::WSRF, 462
- SOAPMessage
  - Arc::SOAPMessage, 368
- softlink
  - Arc::FileAccess, 194
- Software
  - Arc::Software, 372
- SoftwareRequirement
  - Arc::SoftwareRequirement, 378, 379
- Source
  - ArcSec::Source, 385, 386
- SourceFile
  - ArcSec::SourceFile, 387
- SourceURL
  - ArcSec::SourceURL, 388
- speed
  - DataStaging::DataDeliveryComm::Status, 127
- STACK\_OF
  - Arc::Credential, 113
- STAGE\_PREPARE
  - DataStaging::DTRStatus, 171
- STAGED\_PREPARED
  - DataStaging::DTRStatus, 171
- StagedStates
  - DataStaging::DTRStatus, 172
- STAGING\_PREPARING
  - DataStaging::DTRStatus, 171
- STAGING\_PREPARING\_WAIT
  - DataStaging::DTRStatus, 171
- STAGING\_TIMEOUT\_ERROR
  - DataStaging::DTRErrorStatus, 164
- StagingProcesses
  - DataStaging, 47
- Start

- Arc::Run, 345
- start
  - Arc::Adler32Sum, 51
  - Arc::Checksum, 73
  - Arc::ChecksumAny, 76
  - Arc::CRC32Sum, 104
  - Arc::MD5Sum, 274
  - DataStaging::DataDelivery, 120
  - DataStaging::Processor, 331
  - DataStaging::Scheduler, 351
- start\_
  - DataStaging::DataDeliveryComm, 124
- stat
  - Arc::FileAccess, 194
- status
  - DataStaging::DataDeliveryComm::Status, 127
- status\_
  - DataStaging::DataDeliveryComm, 124
- status\_buf\_
  - DataStaging::DataDeliveryComm, 124
- STATUS\_OK
  - Arc, 29
- status\_pos\_
  - DataStaging::DataDeliveryComm, 124
- StatusKind
  - Arc, 29
- stop
  - DataStaging::DataDelivery, 120
  - DataStaging::Processor, 331
  - DataStaging::Scheduler, 351
- storeCert
  - Arc::OAuthConsumer, 295
- StoreDirectory
  - Arc::UserConfig, 444
- str
  - Arc::Time, 402
  - Arc::URL, 419
  - Arc::URLLocation, 423
  - DataStaging::DTRStatus, 172
- streams
  - DataStaging::DataDeliveryComm::Status, 127
- StrError
  - Arc, 38
- string
  - Arc, 41
- string\_to\_level
  - Arc, 33
- StringMatches
  - Arc::URL, 419
- stringto
  - Arc, 34
- strip
  - Arc, 36
- strtobool
  - Arc, 35
- strtoint
  - Arc, 34
- Submit
  - Arc::SubmitterPlugin, 389
- SubmitterPluginLoader
  - Arc::SubmitterPluginLoader, 391
- SuccessExitCode
  - Arc::ExecutableType, 186
- suspend
  - DataStaging::DTR, 158
- Swap
  - Arc::XMLNode, 480
- SYSCONFIG
  - Arc::UserConfig, 448
- SYSCONFIGARCLOC
  - Arc::UserConfig, 449
- TEMPORARY\_REMOTE\_ERROR
  - DataStaging::DTRErrorStatus, 164
- testtune
  - Arc::FileAccess, 194
- thread\_stacksize
  - Arc, 43
- ThreadDataItem
  - Arc::ThreadDataItem, 393
- Time
  - Arc::Time, 400, 401
- TimeFormat
  - Arc, 29
- Timeout
  - Arc::PayloadStream, 305
  - Arc::PayloadStreamInterface, 308
  - Arc::UserConfig, 444, 445
- TimeStamp
  - Arc, 30
- timestamp\_ns
  - DataStaging::DataDeliveryComm::Status, 127
- timestamp\_s
  - DataStaging::DataDeliveryComm::Status, 127
- TmpDirCreate
  - Arc, 32
- TmpFileCreate
  - Arc, 32
- To
  - Arc::WSAHeader, 460
- tokenize
  - Arc, 36
- ToProcessStates
  - DataStaging::DTRStatus, 173
- toString
  - Arc::Software, 376
- tostring
  - Arc, 34

- TouchConsumer
  - Arc::DelegationContainerSOAP, 140
- ToXML
  - Arc::Job, 223
- TRANSFER
  - DataStaging::DTRStatus, 171
- transfer\_params
  - DataStaging::DataDeliveryComm, 125
- TRANSFER\_SPEED\_ERROR
  - DataStaging::DTRErrorStatus, 164
- TransferParameters
  - DataStaging::TransferParameters, 405
- TRANSFERRED
  - DataStaging::DTRStatus, 171
- transferred
  - DataStaging::DataDeliveryComm::Status, 127
- TRANSFERRING
  - DataStaging::DTRStatus, 171
- TRANSFERRING\_CANCEL
  - DataStaging::DTRStatus, 171
- TransferShares
  - DataStaging::TransferShares, 406
- TransferSharesConf
  - DataStaging::TransferSharesConf, 409
- trim
  - Arc, 36
- Truncate
  - Arc::PayloadRawInterface, 301
- TryLoad
  - Arc::PluginsFactory, 324
- Type
  - Arc::JobIdentificationType, 236
- unescape\_chars
  - Arc, 37
- UNKNOWN\_SERVICE\_ERROR
  - Arc, 30
- Unlink
  - Arc::MCC, 263
- unlink
  - Arc::FileAccess, 194
- unload
  - Arc::ModuleManager, 289
- unlock
  - Arc::SimpleCondition, 364
- UnParse
  - Arc::JobDescription, 231
- UnregisterThread
  - Arc::ThreadRegistry, 399
- UnsetEnv
  - Arc, 37
- unuse
  - Arc::ModuleManager, 289
- Update
  - Arc::Job, 224
  - Arc::JobSupervisor, 243
- UpdateCredentials
  - Arc::DelegationConsumerSOAP, 138
  - Arc::DelegationContainerSOAP, 140
  - Arc::DelegationProviderSOAP, 145
- upper
  - Arc, 36
- uri\_encode
  - Arc, 36
- uri\_unencode
  - Arc, 36
- URL
  - Arc::URL, 414
- urlencode
  - Arc::ConfusaParserUtils, 90
- urlencode\_params
  - Arc::ConfusaParserUtils, 90
- URLLocation
  - Arc::URLLocation, 422, 423
- urloptions
  - Arc::URL, 421
- use
  - Arc::ModuleManager, 289
- USER
  - DataStaging::TransferSharesConf, 408
- UserConfig
  - Arc::UserConfig, 426–428
- UserName
  - Arc::UserConfig, 445
- Username
  - Arc::URL, 419
  - Arc::UsernameToken, 451
- username
  - Arc::URL, 421
- UsernameToken
  - Arc::UsernameToken, 450, 451
- UtilsDirPath
  - Arc::UserConfig, 446
- UUID
  - Arc, 33
- valid
  - Arc::Service, 363
  - Arc::URL, 421
- valid\_
  - Arc::WSRF, 462
- Validate
  - Arc::XMLNode, 480
- Verbosity
  - Arc::UserConfig, 446, 447
- VerifyNode
  - Arc::XMLSecNode, 485
- VERSIONTOKENS

- Arc::Software, [377](#)
- VO
  - DataStaging::TransferSharesConf, [408](#)
- VOMSDecode
  - Arc, [40](#)
- VOMSESPath
  - Arc::UserConfig, [447](#)
- VOMSTrustList
  - Arc::VOMSTrustList, [453](#)
- Wait
  - Arc::Run, [345](#)
- wait
  - Arc::SimpleCondition, [365](#)
  - Arc::SimpleCounter, [366](#), [367](#)
- wait\_nonblock
  - Arc::SimpleCondition, [365](#)
- WaitForExit
  - Arc::ThreadRegistry, [399](#)
- WaitInRange
  - Arc::ThreadedPointer, [396](#)
- WaitOrCancel
  - Arc::ThreadRegistry, [399](#)
- WaitOutRange
  - Arc::ThreadedPointer, [396](#), [397](#)
- write
  - Arc::FileAccess, [195](#)
- WriteJobIDsToFile
  - Arc::Job, [224](#)
- WriteJobIDToFile
  - Arc::Job, [224](#)
- WriteJobsToFile
  - Arc::Job, [225](#)
- WriteJobsToTruncatedFile
  - Arc::Job, [226](#)
- WriteStdin
  - Arc::Run, [345](#)
- WSAEndpointReference
  - Arc::WSAEndpointReference, [455](#)
- WSAFault
  - Arc, [30](#)
- WSAFaultAssign
  - Arc, [41](#)
- WSAFaultExtract
  - Arc, [41](#)
- WSAFaultInvalidAddressingHeader
  - Arc, [30](#)
- WSAFaultUnknown
  - Arc, [30](#)
- WSAHeader
  - Arc::WSAHeader, [458](#)
- WSRF
  - Arc::WSRF, [461](#)
- WSRFBBaseFault
  - Arc::WSRFBBaseFault, [463](#)
- WSRP
  - Arc::WSRP, [465](#)
- WSRPFault
  - Arc::WSRPFault, [467](#)
- WSRPResourcePropertyChangeFailure
  - Arc::WSRPResourcePropertyChangeFailure, [468](#)
- X509Token
  - Arc::X509Token, [469](#)
- X509TokenType
  - Arc::X509Token, [469](#)
- XACML
  - Arc::SecAttr, [354](#)
- XMLNode
  - Arc::XMLNode, [473](#), [474](#)
- XMLNodeContainer
  - Arc::XMLNodeContainer, [482](#)
- XMLSecNode
  - Arc::XMLSecNode, [484](#)
- XPathLookup
  - Arc::XMLNode, [480](#)