# KnowARC

Generated by Doxygen 1.5.5

Tue May 13 11:18:46 2008

# Contents


**1 Namespace Index** **1**

1.1 Namespace List . . . . . 1

**2 Class Index** **3**

2.1 Class Hierarchy . . . . . 3

**3 Class Index** **7**

3.1 Class List . . . . . 7

**4 Namespace Documentation** **11**

4.1 Arc Namespace Reference . . . . . 11

4.2 ArcSec Namespace Reference . . . . . 25

**5 Class Documentation** **31**

5.1 acc_descriptor Struct Reference . . . . . 31

5.2 Arc::ACCFactory Class Reference . . . . . 32

5.3 ArcSec::AlgFactory Class Reference . . . . . 33

5.4 ArcSec::ArcAttributeProxy< TheAttribute > Class Template Reference . . . . . 34

5.5 Arc::ArcLocation Class Reference . . . . . 35

5.6 ArcSec::Attr Struct Reference . . . . . 36

5.7 ArcSec::AttributeFactory Class Reference . . . . . 37

5.8 Arc::AttributeIterator Class Reference . . . . . 38

5.9 ArcSec::AttributeProxy Class Reference . . . . . 41

5.10 ArcSec::AttributeValue Class Reference . . . . . 42

5.11 ArcSec::Attrs Class Reference . . . . . 43

5.12 ArcSec::AuthzRequestSection Struct Reference . . . . . 44

5.13 Arc::BaseConfig Class Reference . . . . . 45

5.14 Arc::ChainContext Class Reference . . . . . 47

5.15 Arc::CheckSum Class Reference . . . . . 49

5.16 Arc::CheckSumAny Class Reference . . . . . 50

5.17 Arc::CIStringValue Class Reference . . . 52
5.18 Arc::ClientSOAP Class Reference . . . 54
5.19 ArcSec::CombiningAlg Class Reference . . . 56
5.20 Arc::Config Class Reference . . . 57
5.21 Arc::Counter Class Reference . . . 59
5.22 Arc::CounterTicket Class Reference . . . 66
5.23 Arc::CRC32Sum Class Reference . . . 68
5.24 Arc::DataBufferPar Class Reference . . . 69
5.25 Arc::DataCache Class Reference . . . 76
5.26 Arc::DataCallback Class Reference . . . 80
5.27 Arc::DataHandle Class Reference . . . 81
5.28 Arc::DataMover Class Reference . . . 82
5.29 Arc::DataPoint Class Reference . . . 86
5.30 Arc::DataPointDirect Class Reference . . . 97
5.31 Arc::DataPointIndex Class Reference . . . 104
5.32 Arc::DataSpeed Class Reference . . . 111
5.33 Arc::DelegationConsumer Class Reference . . . 115
5.34 Arc::DelegationConsumerSOAP Class Reference . . . 117
5.35 Arc::DelegationContainerSOAP Class Reference . . . 119
5.36 Arc::DelegationProvider Class Reference . . . 121
5.37 Arc::DelegationProviderSOAP Class Reference . . . 123
5.38 ArcSec::DenyOverridesCombiningAlg Class Reference . . . 125
5.39 dmc_descriptor Struct Reference . . . 126
5.40 Arc::DMCFactory Class Reference . . . 127
5.41 ArcSec::EqualFunction Class Reference . . . 128
5.42 ArcSec::EvalResult Struct Reference . . . 129
5.43 ArcSec::EvaluationCtx Class Reference . . . 130
5.44 ArcSec::EvaluatorContext Class Reference . . . 131
5.45 Arc::ExpirationReminder Class Reference . . . 132
5.46 Arc::FileInfo Class Reference . . . 134
5.47 ArcSec::FnFactory Class Reference . . . 135
5.48 ArcSec::Function Class Reference . . . 136
5.49 Arc::InfoRegister Class Reference . . . 137
5.50 Arc::InformationContainer Class Reference . . . 138
5.51 Arc::InformationInterface Class Reference . . . 140
5.52 Arc::InformationRequest Class Reference . . . 142

5.53 Arc::InformationResponse Class Reference . . . 144
5.54 Arc::IntraProcessCounter Class Reference . . . 145
5.55 Arc::Loader Class Reference . . . 149
5.56 Arc::loader_descriptor Struct Reference . . . 151
5.57 Arc::LoaderFactory Class Reference . . . 152
5.58 Arc::LogDestination Class Reference . . . 154
5.59 Arc::Logger Class Reference . . . 156
5.60 Arc::LogMessage Class Reference . . . 159
5.61 Arc::LogStream Class Reference . . . 161
5.62 ArcSec::MatchFunction Class Reference . . . 163
5.63 Arc::MCC Class Reference . . . 164
5.64 mcc_descriptor Struct Reference . . . 167
5.65 Arc::MCC_Status Class Reference . . . 168
5.66 Arc::MCCFactory Class Reference . . . 171
5.67 Arc::MCCInterface Class Reference . . . 172
5.68 Arc::MD5Sum Class Reference . . . 173
5.69 Arc::Message Class Reference . . . 174
5.70 Arc::MessageAttributes Class Reference . . . 177
5.71 Arc::MessageAuth Class Reference . . . 180
5.72 Arc::MessageAuthContext Class Reference . . . 182
5.73 Arc::MessageContext Class Reference . . . 183
5.74 Arc::MessageContextElement Class Reference . . . 184
5.75 Arc::MessagePayload Class Reference . . . 185
5.76 Arc::ModuleManager Class Reference . . . 186
5.77 Arc::MultiSecAttr Class Reference . . . 187
5.78 Arc::PayloadRaw Class Reference . . . 189
5.79 Arc::PayloadRawInterface Class Reference . . . 192
5.80 Arc::PayloadSOAP Class Reference . . . 194
5.81 Arc::PayloadStream Class Reference . . . 195
5.82 Arc::PayloadStreamInterface Class Reference . . . 198
5.83 Arc::PayloadWSRF Class Reference . . . 200
5.84 ArcSec::PDP Class Reference . . . 202
5.85 pdp_descriptor Struct Reference . . . 203
5.86 Arc::PDPFactory Class Reference . . . 204
5.87 ArcSec::PermitOverridesCombiningAlg Class Reference . . . 205
5.88 Arc::Plexer Class Reference . . . 206

5.89 Arc::PlexerEntry Class Reference . . . 208
5.90 ArcSec::Policy Class Reference . . . 209
5.91 Arc::RegularExpression Class Reference . . . 211
5.92 ArcSec::Request Class Reference . . . 213
5.93 ArcSec::RequestAttribute Class Reference . . . 215
5.94 ArcSec::RequestItem Class Reference . . . 216
5.95 ArcSec::RequestTuple Class Reference . . . 217
5.96 ArcSec::Response Class Reference . . . 218
5.97 ArcSec::ResponseItem Struct Reference . . . 219
5.98 Arc::Run Class Reference . . . 220
5.99 Arc::SecAttr Class Reference . . . 224
5.100 Arc::SecAttr::Format Class Reference . . . 227
5.101 Arc::SecAttrValue Class Reference . . . 228
5.102 ArcSec::SecHandler Class Reference . . . 230
5.103 sechandler_descriptor Struct Reference . . . 231
5.104 Arc::SecHandlerFactory Class Reference . . . 232
5.105 ArcSec::Security Class Reference . . . 233
5.106 Arc::Service Class Reference . . . 234
5.107 service_descriptor Struct Reference . . . 236
5.108 Arc::ServiceFactory Class Reference . . . 237
5.109 Arc::SimpleCondition Class Reference . . . 238
5.110 Arc::SOAPEnvelope Class Reference . . . 240
5.111 Arc::SOAPFault Class Reference . . . 243
5.112 Arc::SOAPMessage Class Reference . . . 246
5.113 Arc::Time Class Reference . . . 248
5.114 Arc::URL Class Reference . . . 251
5.115 Arc::URLLocation Class Reference . . . 258
5.116 Arc::UsernameToken Class Reference . . . 260
5.117 Arc::WSAEndpointReference Class Reference . . . 262
5.118 Arc::WSAHeader Class Reference . . . 264
5.119 Arc::WSRF Class Reference . . . 267
5.120 Arc::WSRFBaseFault Class Reference . . . 269
5.121 Arc::WSRP Class Reference . . . 271
5.122 Arc::WSRPFault Class Reference . . . 273
5.123 Arc::WSRPResourcePropertyChangeFailure Class Reference . . . 274
5.124 Arc::XMLNode Class Reference . . . 275

5.125 Arc::XMLNodeContainer Class Reference . . . . . 285

# Chapter 1

# Namespace Index

## 1.1 Namespace List

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

Arc . . . . . 11
ArcSec (Interface for policy evaluation. Execute the policy evaluation, based on the request and policy ) . . . . . 25

# Chapter 2

# Class Index

## 2.1 Class Hierarchy

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


acc_descriptor . . . 31
Arc::ArcLocation . . . 35
ArcSec::Attr . . . 36
Arc::AttributeIterator . . . 38
ArcSec::AttributeProxy . . . 41
  ArcSec::ArcAttributeProxy< TheAttribute > . . . 34
ArcSec::AttributeValue . . . 42
ArcSec::Attrs . . . 43
ArcSec::AuthzRequestSection . . . 44
Arc::BaseConfig . . . 45
Arc::ChainContext . . . 47
Arc::CheckSum . . . 49
  Arc::CheckSumAny . . . 50
  Arc::CRC32Sum . . . 68
  Arc::MD5Sum . . . 173
Arc::ClientInterface
  Arc::ClientTCP
    Arc::ClientHTTP
      Arc::ClientSOAP . . . 54
ArcSec::CombiningAlg . . . 56
  ArcSec::DenyOverridesCombiningAlg . . . 125
  ArcSec::PermitOverridesCombiningAlg . . . 205
Arc::Counter . . . 59
  Arc::IntraProcessCounter . . . 145
Arc::CounterTicket . . . 66
Arc::DataBufferPar . . . 69
Arc::DataCallback . . . 80
  Arc::DataCache . . . 76
Arc::DataHandle . . . 81
Arc::DataMover . . . 82
Arc::DataPoint . . . 86
  Arc::DataPointDirect . . . 97

Arc::DataPointIndex . . . 104
Arc::DataSpeed . . . 111
Arc::DelegationConsumer . . . 115
Arc::DelegationConsumerSOAP . . . 117
Arc::DelegationContainerSOAP . . . 119
Arc::DelegationProvider . . . 121
Arc::DelegationProviderSOAP . . . 123
dmc_descriptor . . . 126
ArcSec::EvalResult . . . 129
ArcSec::EvaluationCtx . . . 130
ArcSec::EvaluatorContext . . . 131
Arc::ExpirationReminder . . . 132
Arc::FileInfo . . . 134
ArcSec::Function . . . 136
ArcSec::EqualFunction . . . 128
ArcSec::MatchFunction . . . 163
Arc::InfoRegister . . . 137
Arc::InformationInterface . . . 140
Arc::InformationContainer . . . 138
Arc::InformationRequest . . . 142
Arc::InformationResponse . . . 144
Arc::LoadableClass
ArcSec::AlgFactory . . . 33
ArcSec::AttributeFactory . . . 37
ArcSec::FnFactory . . . 135
ArcSec::Request . . . 213
Arc::Loader . . . 149
Arc::loader_descriptor . . . 151
Arc::LogDestination . . . 154
Arc::LogStream . . . 161
Arc::Logger . . . 156
Arc::LogMessage . . . 159
mcc_descriptor . . . 167
Arc::MCC_Status . . . 168
Arc::MCCInterface . . . 172
Arc::MCC . . . 164
Arc::Plexer . . . 206
Arc::Service . . . 234
Arc::Message . . . 174
Arc::MessageAttributes . . . 177
Arc::MessageAuth . . . 180
Arc::MessageAuthContext . . . 182
Arc::MessageContext . . . 183
Arc::MessageContextElement . . . 184
Arc::MessagePayload . . . 185
Arc::PayloadRawInterface . . . 192
Arc::PayloadRaw . . . 189
Arc::PayloadSOAP . . . 194
Arc::PayloadStreamInterface . . . 198
Arc::PayloadStream . . . 195
Arc::PayloadWSRF . . . 200

Arc::ModuleManager . . . 186
Arc::LoaderFactory . . . 152
Arc::ACCFactory . . . 32
Arc::DMCFactory . . . 127
Arc::MCCFactory . . . 171
Arc::PDPFactory . . . 204
Arc::SecHandlerFactory . . . 232
Arc::ServiceFactory . . . 237
ArcSec::PDP . . . 202
pdp_descriptor . . . 203
Arc::PlexerEntry . . . 208
ArcSec::Policy . . . 209
Arc::RegularExpression . . . 211
ArcSec::RequestAttribute . . . 215
ArcSec::RequestItem . . . 216
ArcSec::RequestTuple . . . 217
ArcSec::Response . . . 218
ArcSec::ResponseItem . . . 219
Arc::Run . . . 220
Arc::SecAttr . . . 224
Arc::MultiSecAttr . . . 187
Arc::SecAttr::Format . . . 227
Arc::SecAttrValue . . . 228
Arc::CIStringValue . . . 52
ArcSec::SecHandler . . . 230
sechandler_descriptor . . . 231
ArcSec::Security . . . 233
service_descriptor . . . 236
Arc::SimpleCondition . . . 238
Arc::SOAPFault . . . 243
Arc::SOAPMessage . . . 246
Arc::Time . . . 248
Arc::URL . . . 251
Arc::URLLocation . . . 258
Arc::UsernameToken . . . 260
Arc::WSAEndpointReference . . . 262
Arc::WSAHeader . . . 264
Arc::WSRF . . . 267
Arc::WSRFBaseFault . . . 269
Arc::WSRPFault . . . 273
Arc::WSRPResourcePropertyChangeFailure . . . 274
Arc::WSRP . . . 271
Arc::XMLNode . . . 275
Arc::Config . . . 57
Arc::SOAPEnvelope . . . 240
Arc::PayloadSOAP . . . 194
Arc::XMLNodeContainer . . . 285

# Chapter 3

# Class Index

## 3.1 Class List

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


acc_descriptor . . . 31
Arc::ACCFactory . . . 32
ArcSec::AlgFactory (Interface for algorithm factory class ) . . . 33
ArcSec::ArcAttributeProxy< TheAttribute > (Arc specific AttributeProxy class, it could be not neccessary since we have the base class ) . . . 34
Arc::ArcLocation (Determines ARC installation location ) . . . 35
ArcSec::Attr (Attr contains a tuple of attribute type and value ) . . . 36
ArcSec::AttributeFactory . . . 37
Arc::AttributeIterator (An iterator class for accessing multiple values of an attribute ) . . . 38
ArcSec::AttributeProxy (Interface for generating the AttributeValue object, it will be used by AttributeFactory ) . . . 41
ArcSec::AttributeValue (Interface for different type of <Attribute>, e.g. StringAttribute ) . . . 42
ArcSec::Attrs (Attrs is a container for one or more Attr ) . . . 43
ArcSec::AuthzRequestSection . . . 44
Arc::BaseConfig . . . 45
Arc::ChainContext (Interface to chain specific functionality ) . . . 47
Arc::CheckSum (Defines interface for variuos checksum manipulations ) . . . 49
Arc::CheckSumAny (Wraper for CheckSum class ) . . . 50
Arc::CIStringValue (This class implements case insensitive strings as security attributes ) . . . 52
Arc::ClientSOAP . . . 54
ArcSec::CombiningAlg (Interface for combining algrithm ) . . . 56
Arc::Config (Configuration element - represents (sub)tree of ARC configuration ) . . . 57
Arc::Counter (A class defining a common interface for counters ) . . . 59
Arc::CounterTicket (A class for "tickets" that correspond to counter reservations ) . . . 66
Arc::CRC32Sum (Implementation of CRC32 checksum ) . . . 68
Arc::DataBufferPar (Represents set of buffers ) . . . 69
Arc::DataCache . . . 76
Arc::DataCallback . . . 80
Arc::DataHandle (This class is a wrapper around the DataPoint class ) . . . 81
Arc::DataMover . . . 82
Arc::DataPoint (This base class is an abstraction of URL ) . . . 86
Arc::DataPointDirect (This is a kind of generalized file handle ) . . . 97

Arc::DataPointIndex (Complements DataPoint with attributes common for Indexing Service URLs ) . . . 104
Arc::DataSpeed (Keeps track of average and instantaneous transfer speed ) . . . 111
Arc::DelegationConsumer . . . 115
Arc::DelegationConsumerSOAP . . . 117
Arc::DelegationContainerSOAP . . . 119
Arc::DelegationProvider . . . 121
Arc::DelegationProviderSOAP . . . 123
ArcSec::DenyOverridesCombiningAlg (Implement the "Deny-Overrides" algorithm ) . . . 125
dmc_descriptor . . . 126
Arc::DMCFactory . . . 127
ArcSec::EqualFunction (Evaluate whether the two values are equal ) . . . 128
ArcSec::EvalResult (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 ) . . . 129
ArcSec::EvaluationCtx (EvaluationCtx, in charge of storing some context information for evaluation, including Request, current time, etc ) . . . 130
ArcSec::EvaluatorContext (Context for evaluator. It includes the factories which will be used to create related objects ) . . . 131
Arc::ExpirationReminder (A class intended for internal use within counters ) . . . 132
Arc::FileInfo (FileInfo stores information about files (metadata) ) . . . 134
ArcSec::FnFactory (Interface for function factory, which is in charge of creating Function object according to function type ) . . . 135
ArcSec::Function (Interface for function, which is in charge of evaluating two AttributeValue ) . . . 136
Arc::InfoRegister (Registration to ISIS interface ) . . . 137
Arc::InformationContainer (Information System document container and processor ) . . . 138
Arc::InformationInterface (Information System message processor ) . . . 140
Arc::InformationRequest (Request for information in InfoSystem ) . . . 142
Arc::InformationResponse (Informational response from InfoSystem ) . . . 144
Arc::IntraProcessCounter (A class for counters used by threads within a single process ) . . . 145
Arc::Loader (Creator of Message Component Chains (MCC) ) . . . 149
Arc::loader_descriptor (Identifier of plugin ) . . . 151
Arc::LoaderFactory (Plugin handler ) . . . 152
Arc::LogDestination (A base class for log destinations ) . . . 154
Arc::Logger (A logger class ) . . . 156
Arc::LogMessage (A class for log messages ) . . . 159
Arc::LogStream (A class for logging to ostreams ) . . . 161
ArcSec::MatchFunction (Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression) ) . . . 163
Arc::MCC (Message Chain Component - base class for every MCC plugin ) . . . 164
mcc_descriptor (Identifier of Message Chain Componet (MCC) plugin ) . . . 167
Arc::MCC_Status (A class for communication of MCC processing results ) . . . 168
Arc::MCCFactory (MCC Plugins handler ) . . . 171
Arc::MCCInterface (Interface for communication between MCC, Service and Plexer objects ) . . . 172
Arc::MD5Sum (Implementation of MD5 checksum ) . . . 173
Arc::Message (Object being passed through chain of MCCs ) . . . 174
Arc::MessageAttributes (A class for storage of attribute values ) . . . 177
Arc::MessageAuth (Contains authencity information, authorization tokens and decisions ) . . . 180
Arc::MessageAuthContext (Handler for content of message auth∗ context ) . . . 182
Arc::MessageContext (Handler for content of message context ) . . . 183
Arc::MessageContextElement (Top class for elements contained in message context ) . . . 184
Arc::MessagePayload (Base class for content of message passed through chain ) . . . 185
Arc::ModuleManager (Manager of shared libraries ) . . . 186
Arc::MultiSecAttr (Container of multiple SecAttr attributes ) . . . 187
Arc::PayloadRaw (Raw byte multi-buffer ) . . . 189

Arc::PayloadRawInterface (Random Access Payload for Message objects ) . . . 192
Arc::PayloadSOAP (Payload of Message with SOAP content ) . . . 194
Arc::PayloadStream (POSIX handle as Payload ) . . . 195
Arc::PayloadStreamInterface (Stream-like Payload for Message object ) . . . 198
Arc::PayloadWSRF (This class combines MessagePayload with WSRF ) . . . 200
ArcSec::PDP (Base class for Policy Decisoion Point plugins ) . . . 202
pdp_descriptor (Identifier of Policy Decision Point (PDP) plugin ) . . . 203
Arc::PDPFactory (PDP Plugins handler ) . . . 204
ArcSec::PermitOverridesCombiningAlg (Implement the "Permit-Overrides" algorithm ) . . . 205
Arc::Plexer (The Plexer class, used for routing messages to services ) . . . 206
Arc::PlexerEntry (A pair of label (regex) and pointer to service ) . . . 208
ArcSec::Policy (Base class for Policy, PolicySet, or Rule ) . . . 209
Arc::RegularExpression (A regular expression class ) . . . 211
ArcSec::Request (Base class/Interface for request, includes a container for RequestItems and some operations ) . . . 213
ArcSec::RequestAttribute (Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml ) . . . 215
ArcSec::RequestItem (Interface for request item container, <subjects, actions, objects, ctxs> tuple ) . . . 216
ArcSec::RequestTuple (RequestTuple, container which includes the ) . . . 217
ArcSec::Response (Container for the evaluation results ) . . . 218
ArcSec::ResponseItem (Evaluation result concerning one RequestTuple ) . . . 219
Arc::Run . . . 220
Arc::SecAttr (This is an abstract interface to a security attribute ) . . . 224
Arc::SecAttr::Format (Export/import format ) . . . 227
Arc::SecAttrValue (This is an abstract interface to a security attribute ) . . . 228
ArcSec::SecHandler (Base class for simple security handling plugins ) . . . 230
sechandler_descriptor (Identifier of SecHandler plugin ) . . . 231
Arc::SecHandlerFactory (SecHandler Plugins handler ) . . . 232
ArcSec::Security (Common stuff used by security related slasses ) . . . 233
Arc::Service (Service - last component in a Message Chain ) . . . 234
service_descriptor (Identifier of Service plugin ) . . . 236
Arc::ServiceFactory (Service Plugins handler ) . . . 237
Arc::SimpleCondition (Simple triggered condition ) . . . 238
Arc::SOAPEnvelope (Extends XMLNode class to support structures of SOAP message ) . . . 240
Arc::SOAPFault (Interface to SOAP Fault message ) . . . 243
Arc::SOAPMessage (Message restricted to SOAP payload ) . . . 246
Arc::Time (A class for storing and manipulating times ) . . . 248
Arc::URL (Class to hold general URL's ) . . . 251
Arc::URLLocation (Class to hold a resolved URL location ) . . . 258
Arc::UsernameToken (Interface for manipulation of WS-Security Username Token Profile ) . . . 260
Arc::WSAEndpointReference (Interface for manipulation of WS-Adressing Endpoint Reference ) 262
Arc::WSAHeader (Interface for manipulation WS-Addressing information in SOAP header ) . . . 264
Arc::WSRF (Base class for every WSRF message ) . . . 267
Arc::WSRFBaseFault (Base class for WSRF fault messages ) . . . 269
Arc::WSRP (Base class for WS-ResourceProperties structures ) . . . 271
Arc::WSRPFault (Base class for WS-ResourceProperties faults ) . . . 273
Arc::WSRPResourcePropertyChangeFailure . . . 274
Arc::XMLNode (Wrapper for LibXML library Tree interface ) . . . 275
Arc::XMLNodeContainer . . . 285

# Chapter 4

# Namespace Documentation

## 4.1 Arc Namespace Reference

### Classes

- class **ACC**
- class **Broker**
- class **ExecutionTarget**
- class **Submitter**
- class **TargetGenerator**
- class **TargetRetriever**
- class Config

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

- class ArcLocation

  *Determines ARC installation location.*

- class RegularExpression

  *A regular expression class.*

- class **Base64**
- 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 IntraProcessCounter

  *A class for counters used by threads within a single process.*

- class **Period**
- class Time

*A class for storing and manipulating times.*

- class **DItem**
- class **DBranch**
- class **DItemString**
- class **PrintFBase**
- class **PrintF**
- class **IString**
- class LogMessage

  *A class for log messages.*

- class LogDestination

  *A base class for log destinations.*

- class LogStream

  *A class for logging to ostreams.*

- class Logger

  *A logger class.*

- class Run
- class SimpleCondition

  *Simple triggered condition.*

- class URL

  *Class to hold general URL's.*

- class URLLocation

  *Class to hold a resolved URL location.*

- class **User**
- class XMLNode

  *Wrapper for LibXML library Tree interface.*

- class XMLNodeContainer
- class **cache_download_handler**
- class CheckSum

  *Defines interface for variuos checksum manipulations.*

- class CRC32Sum

  *Implementation of CRC32 checksum.*

- class MD5Sum

  *Implementation of MD5 checksum.*

- class CheckSumAny

  *Wraper for CheckSum class.*

- class DataBufferPar

  *Represents set of buffers.*

- class DataCache
- class DataCallback
- class DataHandle

  *This class is a wrapper around the DataPoint class.*

- class DataMover
- class DataPoint

  *This base class is an abstraction of URL.*

- class DataPointDirect

  *This is a kind of generalized file handle.*

- class DataPointIndex

  *Complements DataPoint with attributes common for Indexing Service URLs.*

- class DataSpeed

  *Keeps track of average and instantaneous transfer speed.*

- class **DataStatus**
- class **DMC**
- class FileInfo

  *FileInfo stores information about files (metadata).*

- class **URLMap**
- class DelegationConsumer
- class DelegationProvider
- class DelegationConsumerSOAP
- class DelegationProviderSOAP
- class DelegationContainerSOAP
- class **InfoCache**
- class **InfoCacheInterface**
- class InfoRegister

  *Registration to ISIS interface.*

- 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 ACCFactory
- class **ClassLoader**
- class DMCFactory

- class **LoadableClass**
- class Loader

  *Creator of Message Component Chains (MCC).*

- class ChainContext

  *Interface to chain specific functionality.*

- struct loader_descriptor

  *Identifier of plugin.*

- class LoaderFactory

  *Plugin handler.*

- class MCCFactory

  *MCC Plugins handler.*

- class ModuleManager

  *Manager of shared libraries.*

- class PDPFactory

  *PDP Plugins handler.*

- class PlexerEntry

  *A pair of label (regex) and pointer to service.*

- class Plexer

  *The Plexer class, used for routing messages to services.*

- class SecHandlerFactory

  *SecHandler Plugins handler.*

- class ServiceFactory

  *Service Plugins handler.*

- class MCCInterface

  *Interface for communication between MCC, Service and Plexer objects.*

- class MCC

  *Message Chain Component - base class for every MCC plugin.*

- class MCC_Status

  *A class for communication of MCC processing results.*

- 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

  *An iterator class for accessing multiple values of an attribute.*

- class MessageAttributes

  *A class for storage of attribute values.*

- class MessageAuth

  *Contains authencity 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 CIStringValue

  *This class implements case insensitive strings as security attributes.*

- class SecAttrValue

  *This is an abstract interface to a security attribute.*

- 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 SOAPFault

  *Interface to SOAP Fault message.*

- class SOAPEnvelope

  *Extends XMLNode class to support structures of SOAP message.*

- class SOAPMessage

  *Message restricted to SOAP payload.*

- class BaseConfig
- class **ClientInterface**
- class **ClientTCP**
- struct **HTTPClientInfo**
- class **ClientHTTP**
- class ClientSOAP
- class **MCCConfig**
- class **DMCConfig**
- class **ACCConfig**
- class **ClientTool**
- class WSAEndpointReference

  *Interface for manipulation of WS-Adressing Endpoint Reference.*

- class WSAHeader

  *Interface for manipulation WS-Addressing information in SOAP header.*

- class UsernameToken

  *Interface for manipulation of WS-Security Username 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 WSRFBaseFault

  *Base class for WSRF fault messages.*

- class **WSRFResourceUnknownFault**
- class **WSRFResourceUnavailableFault**

## Typedefs

- typedef std::map< std::string, std::string > **NS**
- typedef std::list< Arc::XMLNode > **XMLNodeList**
- typedef std::map< std::string, std::string > **DelegationRestrictions**
- typedef loader_descriptor loader_descriptors [ ]
- typedef std::map< std::string, Glib::Module * > **plugin_cache_t**
- typedef std::multimap< std::string, std::string > AttrMap
- typedef AttrMap::const_iterator AttrConstIter
- typedef AttrMap::iterator AttrIter

## Enumerations

- enum TimeFormat {

  **MDSTime**, **ASCTime**, **UserTime**, **ISOTime**,

  **UTCTime**, **RFC1123Time** }
- enum **PeriodBase** {

  **PeriodMiliseconds**, **PeriodSeconds**, **PeriodMinutes**, **PeriodHours**,

  **PeriodDays**, **PeriodWeeks** }
- enum LogLevel {

  **VERBOSE** = 1, **DEBUG** = 2, **INFO** = 4, **WARNING** = 8,

  **ERROR** = 16, **FATAL** = 32 }

- enum StatusKind {

  **STATUS_UNDEFINED** = 0, 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 {

  **WSAFaultNone**, WSAFaultUnknown, WSAFaultInvalidAddressingHeader, **WSAFaultInvalidAddress**,

  **WSAFaultInvalidEPR**, **WSAFaultInvalidCardinality**, **WSAFaultMissingAddressInEPR**, **WSAFaultDuplicateMessageID**,

  **WSAFaultActionMismatch**, **WSAFaultOnlyAnonymousAddressSupported**, **WSAFaultOnlyNonAnonymousAddressSupported**, **WSAFaultMessageAddressingHeaderRequired**,

  **WSAFaultDestinationUnreachable**, **WSAFaultActionNotSupported**, **WSAFaultEndpointUnavailable** }

## 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())
- void GUID (std::string &guid)
- std::string UUID (void)
- const char * **FindTrans** (const char *p)
- std::ostream & **operator**<< (std::ostream &os, const IString &msg)
- std::ostream & operator<< (std::ostream &os, LogLevel level)
- LogLevel **string_to_level** (const std::string &str)
- template<typename T>
  T stringto (const std::string &s)
- template<typename T>
  bool stringto (const std::string &s, T &t)
- template<typename T>
  std::string tostring (T t, const int width=0, const int precision=0)
- std::string upper (const std::string &s)
- void tokenize (const std::string &str, std::vector< std::string > &tokens, const std::string &delimiters=" ")
- std::string trim (const std::string &str, const char *sep=NULL)
- bool CreateThreadFunction (void(*func)(void *), void *arg)
- std::list< URL > ReadURLList (const URL &urllist)
- std::ostream & **operator**<< (std::ostream &out, const XMLNode &node)
- std::istream & **operator**>> (std::istream &in, XMLNode &node)
- 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)
- int **cache_download_url_start** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &url, const std::string &id, cache_download_handler &handler)

- int **cache_download_file_start** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &fname, const std::string &id, cache_download_handler &handler)
- int **cache_download_url_end** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &url, cache_download_handler &handler, bool success)
- int **cache_find_url** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &url, const std::string &id, std::string &options, std::string &fname)
- int **cache_find_file** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &fname, std::string &url, std::string &options)
- int **cache_release_url** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &url, const std::string &id, bool remove)
- int **cache_release_url** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &id, bool remove)
- int **cache_release_file** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &fname, const std::string &id, bool remove)
- int **cache_invalidate_url** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, const std::string &fname)
- unsigned long long int **cache_clean** (const std::string &cache_path, const std::string &cache_data_path, const Arc::User &cache_user, unsigned long long int size)
- int **cache_claiming_list** (const std::string &cache_path, const std::string &fname, std::list< std::string > &ids)
- int **cache_is_claimed_file** (const std::string &cache_path, const std::string &fname)
- int **cache_files_list** (const std::string &cache_path, const Arc::User &cache_user, std::list< std::string > &files)
- int **cache_history_lists** (const std::string &cache_path, std::list< std::string > &olds, std::list< std::string > &news)
- int **cache_history_remove** (const std::string &cache_path, std::list< std::string > &olds, std::list< std::string > &news)
- int **cache_history** (const std::string &cache_path, bool enable, const Arc::User &cache_user)
- std::string string (StatusKind kind)
- const char * ContentFromPayload (const MessagePayload &payload)
- void WSAFaultAssign (SOAPEnvelope &mesage, WSAFault fid)
- WSAFault WSAFaultExtract (SOAPEnvelope &message)
- WSRF & **CreateWSRP** (SOAPEnvelope &soap)
- WSRF & **CreateWSRFBaseFault** (SOAPEnvelope &soap)

## Variables

- const Glib::TimeVal ETERNAL
- const Glib::TimeVal HISTORIC
- Logger **stringLogger**
- const char * **WSRFBaseFaultAction**

### 4.1.1 Detailed Description

Class for generation of targets

Base class for target retrievers

### 4.1.2 Typedef Documentation

#### 4.1.2.1 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.

#### 4.1.2.2 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.

#### 4.1.2.3 typedef std::multimap<std::string,std::string> Arc::AttrMap

A typefed 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 MesssageAttributes class for internal storage of message attributes, but is not visible externally.

#### 4.1.2.4 typedef loader_descriptor Arc::loader_descriptors[ ]

Elements are detected by presence of element with particular name of loader_descriptors type. That is an array of loader_descriptor or similar elements. To check for end of array use ARC_LOADER_FINAL() macro

### 4.1.3 Enumeration Type Documentation

#### 4.1.3.1 enum Arc::LogLevel

Logging levels.

Logging levels for tagging and filtering log messages.

#### 4.1.3.2 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.

#### 4.1.3.3 enum Arc::TimeFormat

An enumeration that contains the possible textual timeformats.

#### 4.1.3.4 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 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.

#### 4.1.4.2 bool Arc::CreateThreadFunction (void(*)(void *) *func*, void * *arg*)

Helper function to create simple thread.

It takes care of all pecularities og Glib::Thread API. As result it runs function 'func' with argument 'arg' in a separate thread. Returns true on success.

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

This function generates a random identifier which is quite unique as well.

#### 4.1.4.4 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

#### 4.1.4.5 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

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

Returns true if underlying XML elements have same names

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

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

#### 4.1.4.8 bool Arc::MatchXMLNamespace (const XMLNode & *node*, const char * *uri*)

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

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

Returns true if underlying XML elements belong to same namespaces

#### 4.1.4.10 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.

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

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

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

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

#### 4.1.4.13 std::list<URL> Arc::ReadURLList (const URL & *urllist*)

Reads a list of URLs from a file.

#### 4.1.4.14 std::string Arc::string (StatusKind *kind*)

Conversion to string.

Conversion from StatusKind to string.

**Parameters:**

***kind*** The StatusKind to convert.

#### 4.1.4.15 template<typename T> bool Arc::stringto (const std::string & *s*, T & *t*) [inline]

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

#### 4.1.4.16 template<typename T> T Arc::stringto (const std::string & *s*) [inline]

This method converts a string to any type.

References Arc::Logger::msg().

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

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

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

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

#### 4.1.4.19 void Arc::tokenize (const std::string & *str*, std::vector< std::string > & *tokens*, const std::string & *delimiters* = " ")

This method tokenize string.

#### 4.1.4.20 template<typename T> std::string Arc::tostring (T *t*, const int *width* = 0, const int *precision* = 0) [inline]

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

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

This method remove given separatos from the beginig and the end of the string.

#### 4.1.4.22 std::string Arc::upper (const std::string & *s*)

This method converts to upper case of the string.

#### 4.1.4.23 std::string Arc::UUID (void)

This function generates uuid.

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

Makes WS-Addressing fault.

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

#### 4.1.4.25 WSAFault Arc::WSAFaultExtract (SOAPEnvelope & *message*)

Gets WS-addressing fault.

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

### 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.

## 4.2 ArcSec Namespace Reference

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

### Classes

- class AlgFactory

  *Interface for algorithm factory class.*

- class CombiningAlg

  *Interface for combining algrithm.*

- class DenyOverridesCombiningAlg

  *Implement the "Deny-Overrides" algorithm.*

- class PermitOverridesCombiningAlg

  *Implement the "Permit-Overrides" algorithm.*

- class **AnyURIAttribute**
- class AttributeFactory
- class AttributeProxy

  *Interface for generating the AttributeValue object, it will be used by AttributeFactory.*

- class ArcAttributeProxy

  *Arc specific AttributeProxy class, it could be not neccessary since we have the base class.*

- class AttributeValue

  *Interface for different type of <Attribute>, e.g. StringAttribute.*

- class **DateTimeAttribute**
- class **TimeAttribute**
- class **DateAttribute**
- class **DurationAttribute**
- struct **ArcPeriod**
- class **PeriodAttribute**
- class **GenericAttribute**
- class RequestAttribute

  *Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml.*

- class **StringAttribute**
- class **X500NameAttribute**
- class RequestTuple

  *RequestTuple, container which includes the.*

- class EvaluationCtx

  *EvaluationCtx, in charge of storing some context information for evaluation, including Request, current time, etc.*

- class **Evaluator**

- class EvaluatorContext

  *Context for evaluator. It includes the factories which will be used to create related objects.*

- class EqualFunction

  *Evaluate whether the two values are equal.*

- class FnFactory

  *Interface for function factory, which is in charge of creating Function object according to function type.*

- class Function

  *Interface for function, which is in charge of evaluating two AttributeValue.*

- class **InRangeFunction**
- class MatchFunction

  *Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).*

- class Policy

  *Base class for Policy, PolicySet, or Rule.*

- struct Attr

  *Attr contains a tuple of attribute type and value.*

- class Attrs

  *Attrs is a container for one or more Attr.*

- class Request

  *Base class/Interface for request, includes a container for RequestItems and some operations.*

- class RequestItem

  *Interface for request item container, <subjects, actions, objects, ctxs> tuple.*

- struct ResponseItem

  *Evaluation result concerning one RequestTuple.*

- class **ResponseList**
- class Response

  *Container for the evaluation results.*

- struct EvalResult

  *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.*

- struct AuthzRequestSection
- struct **AuthzRequest**
- class **PDPConfigContext**
- class PDP

  *Base class for Policy Decisoion Point plugins.*

- class SecHandler

  *Base class for simple security handling plugins.*

- class Security

  *Common stuff used by security related slasses.*

## Typedefs

- typedef std::map< std::string, CombiningAlg * > **AlgMap**
- typedef std::map< std::string, AttributeProxy * > **AttrProxyMap**
- typedef std::map< std::string, Function * > **FnMap**
- typedef std::list< RequestItem * > ReqItemList
- typedef std::list< RequestAttribute * > Subject
- typedef std::list< RequestAttribute * > **Resource**
- typedef std::list< RequestAttribute * > **Action**
- typedef std::list< RequestAttribute * > **Context**
- typedef std::list< Subject > SubList
- typedef std::list< Resource > **ResList**
- typedef std::list< Action > **ActList**
- typedef std::list< Context > **CtxList**
- typedef std::list< Policy * > **Policies**

## Enumerations

- enum Result { DECISION_PERMIT = 0, DECISION_DENY = 1, DECISION_INDETERMINATE = 2, DECISION_NOT_APPLICABLE = 3 }
- enum MatchResult { MATCH = 0, NO_MATCH = 1, INDETERMINATE = 2 }

### 4.2.1 Detailed Description

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

### 4.2.2 Typedef Documentation

#### 4.2.2.1 typedef std::list<RequestItem*> ArcSec::ReqItemList

ReqItemList is a container for RequestItem objects.

Following is some general structures and classes for storing the request information. In principle, the request structure shoud be in XML format, and also can include a few items

#### 4.2.2.2 typedef std::list<RequestAttribute*> ArcSec::Subject

Attribute containers, which includes a few RequestAttribute objects.

Why do we need such containers? A Subject node could be like below, include a few attributes at the same time: <Subject> <Attribute attributeid="urn:arc:subject:voms-attribute" type="xsd:string">administrator</Attribute> <Attribute attributeid="urn:arc:subject:voms-attribute" type="X500DN">/O=NorduGrid/OU=UIO/CN=admin</Attribute> </Subject> Or only include one attribute: <Subject attributeid="urn:arc:subject:dn"

type="X500DN">/O=NorduGrid/OU=UIO/CN=test</Subject> Or include a few the same types of attributes at the same time: <Subject type="xsd:string"> <Attribute attributeid="urn:arc:subject:voms-attribute">administrator</Attribute> <Attribute attributeid="urn:arc:subject:voms-attribute">/O=NorduGrid/OU=UIO/CN=admin</Attribute> </Subject>

Note, <Subject> (or others) node with more than one <Attribute>s means the <Subject> owns all the included attributes at the same time. e.g. a person with email: abc and DN:/O=XYZ/OU=ABC/CN=theguy and role: administrator However, Parallel <Subject>s inside one SubList (see below about definition if ***List) does not means there is any relationship between these <Subject>s.

Then if there are two examples of <Subject> here: Subject1: <Subject> <Attribute attributeid="urn:arc:subject:voms-attribute" type="xsd:string">administrator</Attribute> <Attribute attributeid="urn:arc:subject:voms-attribute" type="X500DN">/O=NorduGrid/OU=UIO/CN=admin</Attribute> </Subject>

and, Subject2: <Subject attributeid="urn:arc:subject:voms-attribute" type="X500DN">/O=NorduGrid/OU=UIO/CN=test</Subject>

Subject3: <Subject attributeid="urn:arc:subject:voms-attribute" type="xsd:string">administrator</Subject>

the former one will be explained as the <Subject1, Action, Resource, Context> request tuple has two attributes at the same time the later one will be explained as the two <Subject2, Action, Resource, Context>, <Subject3, Action, Resource, Context> independently has one attribute. If we consider the Policy side, a policy snipet example like this: <Rule> <Subjects> <Subject type="X500DN">/O=NorduGrid/OU=UIO/CN=admin</Subject> <Subject type="xsd:string">administrator</Subject> </Subjects> <Resources>.....</Resources> <Actions>.....</Actions> <Conditions>.....</Conditions> </Rule> then all of the Subject1 Subject2 Subject3 will satisfy the <Subjects> in policy. but if the policy snipet is like this: <Rule> <Subjects> <Subject> <SubFraction type="X500DN">/O=NorduGrid/OU=UIO/CN=admin</SubFraction> <SubFraction type="xsd:string">administrator</SubFraction> </Subject> </Subjects> <Resources>.....</Resources> <Actions>.....</Actions> <Conditions>.....</Conditions> </Rule> then only Subject1 can satisfy the <Subjects> in policy.

A complete request item could be like: <RequestItem> <Subject attributeid="urn:arc:subject:dn" type="string">/O=NorduGrid/OU=UIO/CN=test</Subject> <Subject attributeid="urn:arc:subject:voms-attribute" type="xsd:string">administrator</Subject> <Subject> <Attribute attributeid="urn:arc:subject:voms-attribute" type="xsd:string">guest</Attribute> <Attribute attributeid="urn:arc:subject:voms-attribute" type="X500DN">/O=NorduGrid/OU=UIO/CN=anonymous</Attribute> </Subject> <Resource attributeid="urn:arc:resource:file" type="string">file://home/test</Resource> <Action attributeid="urn:arc:action:file-action" type="string">read</Action> <Action attributeid="urn:arc:action:file-action" type="string">copy</Action> <Context attributeid="urn:arc:context:date" type="period">2007-09-10T20:30:20/P1Y1M</Context> </RequestItem>

Here putting a few <Subject>s <Resource>s <Action>s or <Context>s together (inside one RequestItem) is only for the convinient of expression (there is no logical relationship between them). For more than one <<Subject>, <Resource>, <Action>, <Context>> tuples, if there is one element (e.g. <Subject>) which is different to each other, you can put these tuples together by using one tuple <<Subject1>,<Subject2>, <Resource>, <Action>, <Context>> tuple, and don't need to write a few tuples.

#### 4.2.2.3 typedef std::list<Subject> ArcSec::SubList

Containers, which include a few Subject, Resource, Action or Context objects.

### 4.2.3 Enumeration Type Documentation

#### 4.2.3.1 enum ArcSec::MatchResult

Match result.

**Enumerator:**

***MATCH*** Match, the request tuple <Subject, Resource, Action, Context> matches the rule

***NO_MATCH*** No_Match, the request tuple <Subject, Resource, Action, Context> does not match the rule

***INDETERMINATE*** Indeterminate, means that the request tuple <Subject, Resource, Action, Context> matches the rule, but in terms of the other "Condition", the tuple does not match. So far, the Indeterminate has no meaning in the existing code (will never be switched to)

#### 4.2.3.2 enum ArcSec::Result

Evaluation result.

**Enumerator:**

***DECISION_PERMIT*** Permit

***DECISION_DENY*** Deny

***DECISION_INDETERMINATE*** Indeterminate, because of the Indeterminate from the "Matching"

***DECISION_NOT_APPLICABLE*** Not_Applicable, means the the request tuple <Subject, Resource, Action, Context> does not match the rule. So there is no way to get to the "Permit"/"Deny" effect.

# Chapter 5

# Class Documentation

## 5.1 acc_descriptor Struct Reference

`#include <ACCLoader.h>`

### Public Attributes

- const char * **name**
- int **version**
- Arc::ACC *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.1.1 Detailed Description

This structure describes one of the ACCs stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the ACC class.

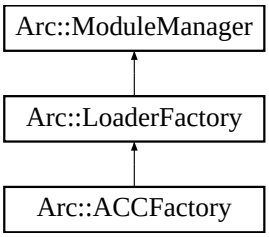
The documentation for this struct was generated from the following file:

- ACCLoader.h

## 5.2 Arc::ACCFactory Class Reference

`#include <ACCFactory.h>`

Inheritance diagram for Arc::ACCFactory::




### Public Member Functions

- ACCFactory (Config *cfg)
- ACC * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- ACC * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- ACC * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

### 5.2.1 Detailed Description

This class handles shared libraries containing ACCs

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Arc::ACCFactory::ACCFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 ACC* Arc::ACCFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of ACC and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created ACC instance.

Reimplemented from Arc::LoaderFactory.

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

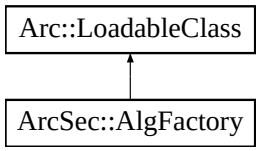
- ACCFactory.h

## 5.3 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

### Protected Attributes

- AlgMap **algmap**

### 5.3.1 Detailed Description

Interface for algorithm factory class.

AlgFactory is in charge of creating CombiningAlg according to the algorithm type

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

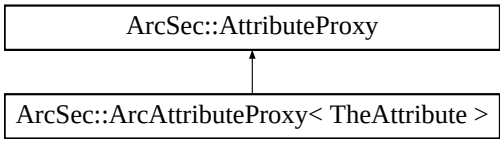
- AlgFactory.h

# 5.4 ArcSec::ArcAttributeProxy< TheAttribute > Class Template Reference

Arc specific AttributeProxy class, it could be not neccessary since we have the base class.

`#include <AttributeProxy.h>`

Inheritance diagram for ArcSec::ArcAttributeProxy< TheAttribute >::




## Public Member Functions

- virtual AttributeValue * getAttribute (const Arc::XMLNode &node)

## 5.4.1 Detailed Description

**template<class TheAttribute> class ArcSec::ArcAttributeProxy< TheAttribute >**

Arc specific AttributeProxy class, it could be not neccessary since we have the base class.

## 5.4.2 Member Function Documentation

### 5.4.2.1 template<class TheAttribute> AttributeValue * ArcSec::ArcAttributeProxy< TheAttribute >::getAttribute (const Arc::XMLNode & *node*) [inline, virtual]

Implementation of getAttribute.

Implements ArcSec::AttributeProxy.

References Arc::XMLNode::Attribute(), and Arc::XMLNode::Child().

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

- AttributeProxy.h

# 5.5 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 ()

### 5.5.1 Detailed Description

Determines ARC installation location.

### 5.5.2 Member Function Documentation

#### 5.5.2.1 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.

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

Returns ARC installation location.

#### 5.5.2.3 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.

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

- ArcLocation.h

## 5.6 ArcSec::Attr Struct Reference

Attr contains a tuple of attribute type and value.

`#include <Request.h>`

### Public Attributes

- std::string **value**
- std::string **type**

### 5.6.1 Detailed Description

Attr contains a tuple of attribute type and value.

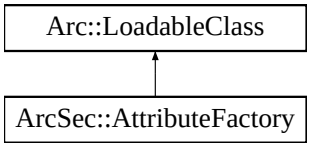
The documentation for this struct was generated from the following file:

- Request.h

## 5.7 ArcSec::AttributeFactory Class Reference

`#include <AttributeFactory.h>`

Inheritance diagram for ArcSec::AttributeFactory::




### Public Member Functions

- virtual AttributeValue * **createValue** (const Arc::XMLNode &node, const std::string &type)=0

### Protected Attributes

- AttrProxyMap **apmap**

### 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

An 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

An 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:

```
Arc::MessageAttributes attributes;
...
for (Arc::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 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.

#### 5.8.3.2 const std::string* Arc::AttributeIterator::operator -> () const

The arrow operator.

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

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

The key of attribute.

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

#### 5.8.3.4 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.

#### 5.8.3.5 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.

#### 5.8.3.6 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.

### 5.8.4 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 Member Data 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:

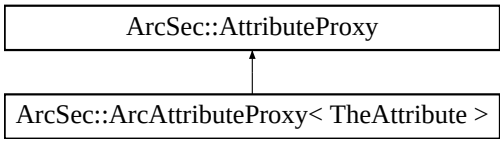
- MessageAttributes.h

## 5.9 ArcSec::AttributeProxy Class Reference

Interface for generating the AttributeValue object, it will be used by AttributeFactory.

`#include <AttributeProxy.h>`

Inheritance diagram for ArcSec::AttributeProxy::




### Public Member Functions

- virtual AttributeValue * **getAttribute** (const Arc::XMLNode &node)=0

### 5.9.1 Detailed Description

Interface for generating 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) is called, in order to generate a specific AttributeValue

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

- AttributeProxy.h

## 5.10 ArcSec::AttributeValue Class Reference

Interface for different type of <Attribute>, e.g. StringAttribute.

`#include <AttributeValue.h>`

Inherited by ArcSec::AnyURIAttribute, ArcSec::DateAttribute, ArcSec::DateTimeAttribute, ArcSec::DurationAttribute, ArcSec::GenericAttribute, ArcSec::PeriodAttribute, ArcSec::StringAttribute, ArcSec::TimeAttribute, and ArcSec::X500NameAttribute.

### Public Member Functions

- virtual bool equal (AttributeValue *value)=0
- virtual std::string encode ()=0
- virtual std::string getType ()=0
- virtual std::string getId ()=0

### 5.10.1 Detailed Description

Interface for different type of <Attribute>, e.g. StringAttribute.

<Attribute> uses different "Type" definition; Each type of <Attribute> will have different approach to compare The "Type" supported so far is: StringAttribute, DateAttribute, TimeAttribute, DurationAttribute, PeriodAttribute, AnyURIAttribute, X500NameAttribute

### 5.10.2 Member Function Documentation

#### 5.10.2.1 virtual bool ArcSec::AttributeValue::equal (AttributeValue * *value*) [pure virtual]

evluate whether "this" equale to the parameter value

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

encode the value in a string format

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

get the type of the <Attribute>

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

get the id of the <Attribute>

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>`

### Public Member Functions

- void **addItem** (Attr attr)
- int **size** ()
- Attr & **getItem** (int n)
- Attr & **operator[ ]** (int n)

### 5.11.1 Detailed Description

Attrs is a container for one or more Attr.

Attrs includes includes methonds 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>`

### Public Attributes

- std::string **value**
- std::string **id**
- std::string **type**
- std::string **issuer**

### 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::BaseConfig Class Reference

`#include <ClientInterface.h>`

Inherited by Arc::ACCConfig, Arc::DMCConfig, and Arc::MCCConfig.

## 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

## Public Attributes

- std::string **key**
- std::string **cert**
- std::string **proxy**
- std::string **cafile**
- std::string **cadir**
- XMLNode **overlay**

## Protected Attributes

- std::list< std::string > **plugin_paths**

### 5.13.1 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.

### 5.13.2 Member Function Documentation

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

Adds non-standard location of plugins

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

Add private key

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

Add certificate

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

Add credentials proxy

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

Add CA file

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

Add CA directory

#### 5.13.2.7 void Arc::BaseConfig::AddOverlay (XMLNode *cfg*)

Add configuration overlay

#### 5.13.2.8 void Arc::BaseConfig::GetOverlay (std::string *fname*)

Read overlay from file

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

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument.

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

- ClientInterface.h

# 5.14 Arc::ChainContext Class Reference

Interface to chain specific functionality.

`#include <Loader.h>`

## Public Member Functions

- operator ServiceFactory * ()
- operator MCCFactory * ()
- operator SecHandlerFactory * ()
- operator PDPFactory * ()

## Friends

- class **Loader**

## 5.14.1 Detailed Description

Interface to chain specific functionality.

Object of this class is associated with every Loader object. It is acecssible 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.

## 5.14.2 Member Function Documentation

### 5.14.2.1 Arc::ChainContext::operator ServiceFactory * () [inline]

Returns associated ServiceFactory object

References Arc::Loader::service_factory.

### 5.14.2.2 Arc::ChainContext::operator MCCFactory * () [inline]

Returns associated MCCFactory object

References Arc::Loader::mcc_factory.

### 5.14.2.3 Arc::ChainContext::operator SecHandlerFactory * () [inline]

Returns associated SecHandlerFactory object

References Arc::Loader::sechandler_factory.

### 5.14.2.4 Arc::ChainContext::operator PDPFactory * () [inline]

Returns associated PDPFactory object

References Arc::Loader::pdp_factory.

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

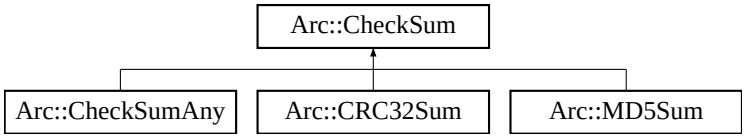
- Loader.h

## 5.15 Arc::CheckSum Class Reference

Defines interface for variuos checksum manipulations.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSum::




### Public Member Functions

- 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

### 5.15.1 Detailed Description

Defines interface for variuos checksum manipulations.

This class is used during data transfers through DataBufferPar class

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

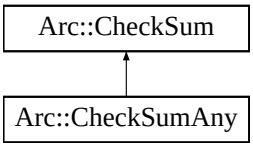
- CheckSum.h

## 5.16 Arc::CheckSumAny Class Reference

Wraper for CheckSum class.

`#include <CheckSum.h>`

Inheritance diagram for Arc::CheckSumAny::




### Public Types

- enum **type** {
  **none**, **unknown**, **undefined**, **cksum**,
  **md5** }

### Public Member Functions

- **CheckSumAny** (CheckSum *c=NULL)
- **CheckSumAny** (type type)
- **CheckSumAny** (const char *type)
- 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
- bool **active** (void)
- type **Type** (void)
- void **operator=** (const char *type)
- bool **operator==** (const char *s)
- bool **operator==** (const CheckSumAny &ck)

### Static Public Member Functions

- static type **Type** (const char *crc)

### 5.16.1 Detailed Description

Wraper for CheckSum class.

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

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

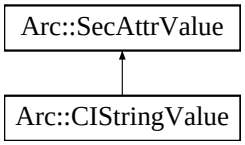
- CheckSum.h

## 5.17 Arc::CIStringValue Class Reference

This class implements case insensitive strings as security attributes.

`#include <CIStringValue.h>`

Inheritance diagram for Arc::CIStringValue::




### Public Member Functions

- CIStringValue ()
- CIStringValue (const char *ss)
- CIStringValue (const std::string &ss)
- virtual operator bool ()

### Protected Member Functions

- virtual bool equal (SecAttrValue &b)

### Protected Attributes

- std::string **s**

### 5.17.1 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.

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 Arc::CIStringValue::CIStringValue ()

Default constructor

#### 5.17.2.2 Arc::CIStringValue::CIStringValue (const char * *ss*)

This is a constructor that takes a string litteral.

#### 5.17.2.3 Arc::CIStringValue::CIStringValue (const std::string & *ss*)

This is a constructor that takes a string object.

## 5.17.3 Member Function Documentation

### 5.17.3.1 virtual Arc::CIStringValue::operator bool () [virtual]

This function returns false if the string is empty or uninitialized

Reimplemented from Arc::SecAttrValue.

### 5.17.3.2 virtual bool Arc::CIStringValue::equal (SecAttrValue & *b*) [protected, virtual]

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

Reimplemented from Arc::SecAttrValue.

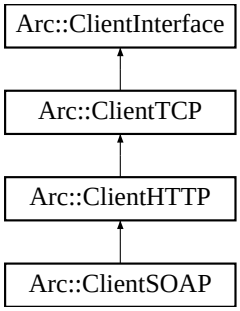
The documentation for this class was generated from the following file:

- CIStringValue.h

## 5.18 Arc::ClientSOAP Class Reference

`#include <ClientInterface.h>`

Inheritance diagram for Arc::ClientSOAP::




### Public Member Functions

- ClientSOAP ()
- **ClientSOAP** (const BaseConfig &cfg, const std::string &host, int port, bool tls, const std::string &path)
- MCC_Status process (PayloadSOAP *request, PayloadSOAP **response)
- MCC_Status process (const std::string &action, PayloadSOAP *request, PayloadSOAP **response)
- MCC * **GetEntry** (void)
- virtual void **Load** (void)

### Protected Attributes

- MCC * **soap_entry**

### 5.18.1 Detailed Description

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

### 5.18.2 Constructor & Destructor Documentation

#### 5.18.2.1 Arc::ClientSOAP::ClientSOAP () `[inline]`

Constructor creates MCC chain and connects to server. cfg - common configuration, host - hostname of remote server, port - TCP port of remote server, tls - true if connection to use HTTPS, false for HTTP, path - internal path of service to be contacted. TODO: use URL.

### 5.18.3 Member Function Documentation

#### 5.18.3.1 MCC_Status Arc::ClientSOAP::process (PayloadSOAP * *request*, PayloadSOAP ** *response*)

Send SOAP request and receive response.

#### 5.18.3.2 MCC_Status Arc::ClientSOAP::process (const std::string & *action*, PayloadSOAP * *request*, PayloadSOAP ** *response*)

Send SOAP request with specified SOAP action and receive response.

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

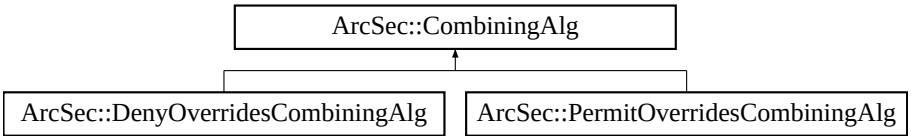
- ClientInterface.h

## 5.19 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 std::string & **getalgId** (void)=0

### 5.19.1 Detailed Description

Interface for combining algrithm.

### 5.19.2 Member Function Documentation

#### 5.19.2.1 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 policy will be called

Implemented in ArcSec::DenyOverridesCombiningAlg, and ArcSec::PermitOverridesCombiningAlg.

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

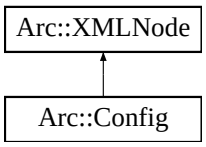
- CombiningAlg.h

# 5.20 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 NS &ns)
- Config (const char *filename)
- Config (const std::string &xml_str)
- Config (Arc::XMLNode xml)
- Config (long cfg_ptr_addr)
- Config (const Config &cfg)
- void print (void)
- void parse (const char *filename)

### 5.20.1 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.

### 5.20.2 Constructor & Destructor Documentation

#### 5.20.2.1 Arc::Config::Config () `[inline]`

Dummy constructor - produces invalid structure

#### 5.20.2.2 Arc::Config::Config (const NS & *ns*) `[inline]`

Creates empty XML tree

#### 5.20.2.3 Arc::Config::Config (const char * *filename*)

Loads configuration document from file 'filename'

#### 5.20.2.4 Arc::Config::Config (const std::string & *xml_str*) [inline]

Parse configuration document from memory

#### 5.20.2.5 Arc::Config::Config (Arc::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.

#### 5.20.2.6 Arc::Config::Config (long *cfg_ptr_addr*)

Copy constructor used by language bindings

#### 5.20.2.7 Arc::Config::Config (const Config & *cfg*)

Copy constructor used by language bindings

### 5.20.3 Member Function Documentation

#### 5.20.3.1 void Arc::Config::print (void)

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

#### 5.20.3.2 void Arc::Config::parse (const char * *filename*)

Parse configuration document from file 'filename'

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

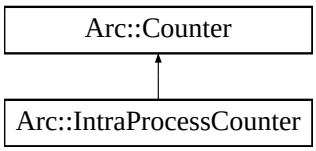
- ArcConfig.h

## 5.21 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

### 5.21.1 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.
Arc::XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
Arc::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 Arc::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(...);
```

## 5.21.2 Member Typedef Documentation

### 5.21.2.1 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 numbesrs (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.

## 5.21.3 Constructor & Destructor Documentation

### 5.21.3.1 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.

### 5.21.3.2 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.

## 5.21.4 Member Function Documentation

### 5.21.4.1 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.

### 5.21.4.2 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.

#### 5.21.4.3 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.

#### 5.21.4.4 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.

#### 5.21.4.5 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.

#### 5.21.4.6 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.

#### 5.21.4.7 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 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.

#### 5.21.4.8 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.

#### 5.21.4.9 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.

#### 5.21.4.10 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.

#### 5.21.4.11 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.

#### 5.21.4.12 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.

#### 5.21.4.13 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 (bot 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.

#### 5.21.4.14 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 (bot 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.

### 5.21.5 Friends And Related Function Documentation

#### 5.21.5.1 friend class CounterTicket [friend]

The CounterTicket class needs to be a friend.

#### 5.21.5.2 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

## 5.22 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

### 5.22.1 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.
Arc::XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
Arc::CounterTicket tick = memory.reserve(2*sizeof(double));
// Use the memory.
double* A=new double[2000000];
doSomething(A);
delete[] A;
// Cancel the reservation.
tick.cancel();
```

### 5.22.2 Constructor & Destructor Documentation

#### 5.22.2.1 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.

### 5.22.3 Member Function Documentation

#### 5.22.3.1 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.

#### 5.22.3.2 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.

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

Cancels a resrvation.

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.

### 5.22.4 Friends And Related Function Documentation

#### 5.22.4.1 friend class Counter [friend]

The Counter class needs to be a friend.

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

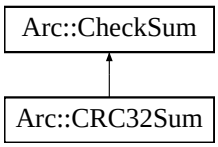
- Counter.h

## 5.23 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
- uint32_t **crc** (void) const

### 5.23.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

## 5.24 Arc::DataBufferPar Class Reference

Represents set of buffers.

`#include <DataBufferPar.h>`

### Public Member Functions

- operator bool ()
- DataBufferPar (unsigned int size=65536, int blocks=3)
- DataBufferPar (CheckSum *cksum, unsigned int size=65536, int blocks=3)
- ~DataBufferPar ()
- bool set (CheckSum *cksum=NULL, unsigned int size=65536, int blocks=3)
- char * operator[ ] (int n)
- bool for_read (int &handle, unsigned int &length, bool wait)
- bool for_read ()
- bool is_read (int handle, unsigned int length, unsigned long long int offset)
- bool is_read (char *buf, unsigned int length, unsigned long long int offset)
- bool for_write (int &handle, unsigned int &length, unsigned long long int &offset, bool wait)
- bool for_write ()
- bool is_written (int handle)
- bool is_written (char *buf)
- bool is_notwritten (int handle)
- bool is_notwritten (char *buf)
- void eof_read (bool v)
- void eof_write (bool v)
- void error_read (bool v)
- void error_write (bool v)
- bool eof_read ()
- bool eof_write ()
- bool error_read ()
- bool error_write ()
- bool error_transfer ()
- bool error ()
- bool wait ()
- bool wait_used ()
- bool checksum_valid ()
- const CheckSum * checksum_object ()
- bool wait_eof_read ()
- bool wait_read ()
- bool wait_eof_write ()
- bool wait_write ()
- bool wait_eof ()
- unsigned long long int eof_position () const
- unsigned int buffer_size ()

### Public Attributes

- DataSpeed speed

## Classes

- struct **buf_desc**

### 5.24.1 Detailed Description

Represents set of buffers.

This class is used used during data transfer using DataPoint classes.

### 5.24.2 Constructor & Destructor Documentation

#### 5.24.2.1 Arc::DataBufferPar::DataBufferPar (unsigned int *size* = `65536`, int *blocks* = `3`)

Contructor

**Parameters:**

***size*** size of every buffer in bytes.

***blocks*** number of buffers.

#### 5.24.2.2 Arc::DataBufferPar::DataBufferPar (CheckSum * *cksum*, unsigned int *size* = `65536`, int *blocks* = `3`)

Contructor

**Parameters:**

***size*** size of every buffer in bytes.

***blocks*** number of buffers.

***cksum*** object which will compute checksum. Should not be destroyed till DataBufferPar itself.

#### 5.24.2.3 Arc::DataBufferPar::~DataBufferPar ()

Destructor.

### 5.24.3 Member Function Documentation

#### 5.24.3.1 Arc::DataBufferPar::operator bool (void) `[inline]`

Check if DataBufferPar object is initialized.

#### 5.24.3.2 bool Arc::DataBufferPar::set (CheckSum * *cksum* = `NULL`, unsigned int *size* = `65536`, int *blocks* = `3`)

Reinitialize buffers with different parameters.

**Parameters:**

***size*** size of every buffer in bytes.

***blocks*** number of buffers.

***cksum*** object which will compute checksum. Should not be destroyed till DataBufferPar itself.

#### 5.24.3.3 char* Arc::DataBufferPar::operator[ ] (int *n*)

Direct access to buffer by number.

#### 5.24.3.4 bool Arc::DataBufferPar::for_read (int & *handle*, unsigned int & *length*, bool *wait*)

Request buffer for READING INTO it.

**Parameters:**

***handle*** returns buffer's number.

***length*** returns size of buffer

***wait*** if true and there are no free buffers, method will wait for one.

**Returns:**

true on success

#### 5.24.3.5 bool Arc::DataBufferPar::for_read ()

Check if there are buffers which can be taken by for_read(). This function checks only for buffers and does not take eof and error conditions into account.

#### 5.24.3.6 bool Arc::DataBufferPar::is_read (int *handle*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

**Parameters:**

***handle*** buffer's number.

***length*** amount of data.

***offset*** offset in stream, file, etc.

#### 5.24.3.7 bool Arc::DataBufferPar::is_read (char * *buf*, unsigned int *length*, unsigned long long int *offset*)

Informs object that data was read into buffer.

**Parameters:**

***buf*** - address of buffer

***length*** amount of data.

***offset*** offset in stream, file, etc.

#### 5.24.3.8 bool Arc::DataBufferPar::for_write (int & *handle*, unsigned int & *length*, unsigned long long int & *offset*, bool *wait*)

Request buffer for WRITING FROM it.

**Parameters:**

***handle*** returns buffer's number.

***length*** returns size of buffer

***wait*** if true and there are no free buffers, method will wait for one.

#### 5.24.3.9 bool Arc::DataBufferPar::for_write ()

Check if there are buffers which can be taken by for_write(). This function checks only for buffers and does not take eof and error conditions into account.

#### 5.24.3.10 bool Arc::DataBufferPar::is_written (int *handle*)

Informs object that data was written from buffer.

**Parameters:**

***handle*** buffer's number.

#### 5.24.3.11 bool Arc::DataBufferPar::is_written (char * *buf*)

Informs object that data was written from buffer.

**Parameters:**

***buf*** - address of buffer

#### 5.24.3.12 bool Arc::DataBufferPar::is_notwritten (int *handle*)

Informs object that data was NOT written from buffer (and releases buffer).

**Parameters:**

***handle*** buffer's number.

#### 5.24.3.13 bool Arc::DataBufferPar::is_notwritten (char * *buf*)

Informs object that data was NOT written from buffer (and releases buffer).

**Parameters:**

***buf*** - address of buffer

#### 5.24.3.14 void Arc::DataBufferPar::eof_read (bool *v*)

Informs object if there will be no more request for 'read' buffers. v true if no more requests.

#### 5.24.3.15 void Arc::DataBufferPar::eof_write (bool *v*)

Informs object if there will be no more request for 'write' buffers. v true if no more requests.

#### 5.24.3.16 void Arc::DataBufferPar::error_read (bool *v*)

Informs object if error accured on 'read' side.

**Parameters:**

*v* true if error.

#### 5.24.3.17 void Arc::DataBufferPar::error_write (bool *v*)

Informs object if error accured on 'write' side.

**Parameters:**

*v* true if error.

#### 5.24.3.18 bool Arc::DataBufferPar::eof_read ()

Returns true if object was informed about end of transfer on 'read' side.

#### 5.24.3.19 bool Arc::DataBufferPar::eof_write ()

Returns true if object was informed about end of transfer on 'write' side.

#### 5.24.3.20 bool Arc::DataBufferPar::error_read ()

Returns true if object was informed about error on 'read' side.

#### 5.24.3.21 bool Arc::DataBufferPar::error_write ()

Returns true if object was informed about error on 'write' side.

#### 5.24.3.22 bool Arc::DataBufferPar::error_transfer ()

Returns true if eror occured inside object.

#### 5.24.3.23 bool Arc::DataBufferPar::error ()

Returns true if object was informed about error or internal error occured.

#### 5.24.3.24 bool Arc::DataBufferPar::wait ()

Wait (max 60 sec.) till any action happens in object. Returns true if action is eof on any side.

#### 5.24.3.25 bool Arc::DataBufferPar::wait_used ()

Wait till there are no more used buffers left in object.

#### 5.24.3.26 bool Arc::DataBufferPar::checksum_valid ()

Returns true if checksum was successfully computed.

#### 5.24.3.27 const CheckSum* Arc::DataBufferPar::checksum_object ()

Returns CheckSum object specified in constructor.

#### 5.24.3.28 bool Arc::DataBufferPar::wait_eof_read ()

Wait till end of transfer happens on 'read' side.

#### 5.24.3.29 bool Arc::DataBufferPar::wait_read ()

Wait till end of transfer or error happens on 'read' side.

#### 5.24.3.30 bool Arc::DataBufferPar::wait_eof_write ()

Wait till end of transfer happens on 'write' side.

#### 5.24.3.31 bool Arc::DataBufferPar::wait_write ()

Wait till end of transfer or error happens on 'write' side.

#### 5.24.3.32 bool Arc::DataBufferPar::wait_eof ()

Wait till end of transfer happens on any side.

#### 5.24.3.33 unsigned long long int Arc::DataBufferPar::eof_position () const [inline]

Returns offset following last piece of data transfered.

#### 5.24.3.34 unsigned int Arc::DataBufferPar::buffer_size ()

Returns size of buffer in object. If not initialized then this number represents size of default buffer.

## 5.24.4 Member Data Documentation

### 5.24.4.1 DataSpeed Arc::DataBufferPar::speed

This object controls transfer speed.

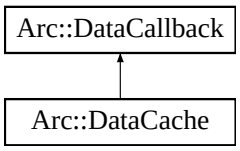
The documentation for this class was generated from the following file:

- DataBufferPar.h

## 5.25 Arc::DataCache Class Reference

`#include <DataCache.h>`

Inheritance diagram for Arc::DataCache::




### Public Types

- enum **file_state_t** { **file_no_error** = 0, **file_download_failed** = 1, **file_not_valid** = 2, **file_keep** = 4 }

### Public Member Functions

- DataCache ()
- DataCache (const std::string &cache_path, const std::string &cache_data_path, const std::string &cache_link_path, const std::string &id, const Arc::User &cache_user)
- DataCache (const DataCache &cache)
- virtual ~DataCache ()
- bool start (const URL &base_url, bool &available)
- const std::string & file () const
- bool stop (int file_state=file_no_error)
- bool link (const std::string &link_path)
- bool link (const std::string &link_path, const Arc::User &user)
- bool copy (const std::string &link_path)
- bool **copy** (const std::string &link_path, const Arc::User &user)
- bool clean (unsigned long long int size=1)
- virtual bool cb (unsigned long long int size)
- operator bool ()
- bool CheckCreated ()
- void SetCreated (Time val)
- Time GetCreated ()
- bool CheckValid ()
- void SetValid (Time val)
- Time GetValid ()

### 5.25.1 Detailed Description

High level interface to cache operations (same functionality :) ) and additional functionality to integrate into grid-manager environment.

## 5.25.2 Constructor & Destructor Documentation

### 5.25.2.1 Arc::DataCache::DataCache ()

Default constructor (non-functional cache).

### 5.25.2.2 Arc::DataCache::DataCache (const std::string & *cache_path*, const std::string & *cache_data_path*, const std::string & *cache_link_path*, const std::string & *id*, const Arc::User & *cache_user*)

Constructor

**Parameters:**

***cache_path*** path to directory with cache info files

***cache_data_path*** path to directory with cache data files

***cache_link_path*** path used to create link in case cache_directory is visible under different name during actual usage

***id*** identifier used to claim files in cache

***cache_user*** owner of cahce (0 for public cache)

### 5.25.2.3 Arc::DataCache::DataCache (const DataCache & *cache*)

Copy constructor.

### 5.25.2.4 virtual Arc::DataCache::~DataCache () [virtual]

and destructor

## 5.25.3 Member Function Documentation

### 5.25.3.1 bool Arc::DataCache::start (const URL & *base_url*, bool & *available*)

Prepare cache for downloading file. On success returns true. This function can block for long time if there is another process downloading same url.

**Parameters:**

***base_url*** url to assign to file in cache (file's identifier)

***available*** contains true on exit if file is already in cache

### 5.25.3.2 const std::string& Arc::DataCache::file () const [inline]

Returns path to file which contains/will contain content of assigned url.

#### 5.25.3.3 bool Arc::DataCache::stop (int *file_state* = `file_no_error`)

This method must be called after file was downloaded or download failed.

**Parameters:**

***failure*** true if download failed

#### 5.25.3.4 bool Arc::DataCache::link (const std::string & *link_path*)

Must be called to create soft-link to cache file or to copy it. It's behavior depends on configuration. All necessary directories will be created. Returns false on error (usually that means soft-link already exists).

**Parameters:**

***link_path*** path for soft-link or new file.

#### 5.25.3.5 bool Arc::DataCache::link (const std::string & *link_path*, const Arc::User & *user*)

**Parameters:**

***user*** set owner of soft-link

#### 5.25.3.6 bool Arc::DataCache::copy (const std::string & *link_path*)

Do same as link() but always create copy.

#### 5.25.3.7 bool Arc::DataCache::clean (unsigned long long int *size* = `1`)

Remove some amount of oldest information from cache. Returns true on success.

**Parameters:**

***size*** amount to be removed (bytes)

#### 5.25.3.8 virtual bool Arc::DataCache::cb (unsigned long long int *size*) `[virtual]`

Callback implementation to clean at least 1 byte.

Reimplemented from Arc::DataCallback.

#### 5.25.3.9 Arc::DataCache::operator bool (void) `[inline]`

Returns true if object is useable.

#### 5.25.3.10 bool Arc::DataCache::CheckCreated () `[inline]`

Check if there is an information about creation time.

#### 5.25.3.11 void Arc::DataCache::SetCreated (Time *val*) [inline]

Set creation time.

**Parameters:**

***val*** creation time

#### 5.25.3.12 Time Arc::DataCache::GetCreated () [inline]

Get creation time.

#### 5.25.3.13 bool Arc::DataCache::CheckValid () [inline]

Check if there is an information about invalidation time.

#### 5.25.3.14 void Arc::DataCache::SetValid (Time *val*) [inline]

Set invalidation time.

**Parameters:**

***val*** validity time

#### 5.25.3.15 Time Arc::DataCache::GetValid () [inline]

Get invalidation time.

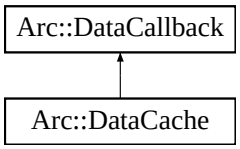
The documentation for this class was generated from the following file:

- DataCache.h

## 5.26 Arc::DataCallback Class Reference

`#include <DataCallback.h>`

Inheritance diagram for Arc::DataCallback::




### Public Member Functions

- virtual bool **cb** (int)
- virtual bool **cb** (unsigned int)
- virtual bool **cb** (long long int)
- virtual bool **cb** (unsigned long long int)

### 5.26.1 Detailed Description

This class is used by DataHandle to report missing space on local filesystem. One of 'cb' functions here will be called if operation initiated by DataHandle::start_reading runs out of disk space.

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

- DataCallback.h

# 5.27 Arc::DataHandle Class Reference

This class is a wrapper around the DataPoint class.

`#include <DataHandle.h>`

## Public Member Functions

- **DataHandle** (const URL &url)
- DataHandle & **operator=** (const URL &url)
- void **Clear** ()
- DataPoint * **operator** → ()
- const DataPoint * **operator** → () const
- DataPoint & **operator*** ()
- const DataPoint & **operator*** () const
- bool **operator!** () const
- **operator bool** () const

### 5.27.1 Detailed Description

This class is a wrapper around the DataPoint class.

It simplifies the construction, use and destruction of DataPoint objects.

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

- DataHandle.h

## 5.28 Arc::DataMover Class Reference

`#include <DataMover.h>`

### Public Types

- typedef void(∗ **callback** )(DataMover ∗, DataStatus, const std::string &, void ∗)

### Public Member Functions

- DataMover ()
- ~DataMover ()
- DataStatus Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const URLMap &map, std::string &failure_description, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- DataStatus Transfer (DataPoint &source, DataPoint &destination, DataCache &cache, const URLMap &map, unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_average_speed, time_t max_inactivity_time, std::string &failure_description, callback cb=NULL, void ∗arg=NULL, const char ∗prefix=NULL)
- DataStatus **Delete** (DataPoint &url, bool errcont=false)
- bool verbose ()
- void verbose (bool)
- void verbose (const std::string &prefix)
- bool retry ()
- void retry (bool)
- void secure (bool)
- void passive (bool)
- void force_to_meta (bool)
- bool checks ()
- void checks (bool v)
- void set_default_min_speed (unsigned long long int min_speed, time_t min_speed_time)
- void set_default_min_average_speed (unsigned long long int min_average_speed)
- void set_default_max_inactivity_time (time_t max_inactivity_time)
- void **set_progress_indicator** (DataSpeed::show_progress_t func=NULL)

### 5.28.1 Detailed Description

A purpose of this class is to provide an interface that moves data between two locations specified by URLs. It's main action is represented by methods DataMover::Transfer. Instance represents only attributes used during transfer.

### 5.28.2 Constructor & Destructor Documentation

#### 5.28.2.1 Arc::DataMover::DataMover ()

Constructor.

#### 5.28.2.2 Arc::DataMover::~DataMover ()

Destructor.

### 5.28.3 Member Function Documentation

#### 5.28.3.1 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, DataCache & *cache*, const URLMap & *map*, std::string & *failure_description*, callback *cb* = NULL, void * *arg* = NULL, const char * *prefix* = NULL)

Initiates transfer from 'source' to 'destination'.

**Parameters:**

***source*** source URL.

***destination*** destination URL.

***cache*** controls caching of downloaded files (if destination url is "file://"). If caching is not needed default constructor DataCache() can be used.

***map*** URL mapping/convertion table (for 'source' URL).

***cb*** if not NULL, transfer is done in separate thread and 'cb' is called after transfer completes/fails.

***arg*** passed to 'cb'.

***prefix*** if 'verbose' is activated this information will be printed before each line representing current transfer status.

#### 5.28.3.2 DataStatus Arc::DataMover::Transfer (DataPoint & *source*, DataPoint & *destination*, DataCache & *cache*, const URLMap & *map*, unsigned long long int *min_speed*, time_t *min_speed_time*, unsigned long long int *min_average_speed*, time_t *max_inactivity_time*, std::string & *failure_description*, callback *cb* = NULL, void * *arg* = NULL, const char * *prefix* = NULL)

Initiates transfer from 'source' to 'destination'.

**Parameters:**

***min_speed*** minimal allowed current speed.

***min_speed_time*** time for which speed should be less than 'min_speed' before transfer fails.

***min_average_speed*** minimal allowed average speed.

***max_inactivity_time*** time for which should be no activity before transfer fails.

#### 5.28.3.3 bool Arc::DataMover::verbose ()

Check if printing information about transfer status is activated.

#### 5.28.3.4 void Arc::DataMover::verbose (bool)

Activate printing information about transfer status.

#### 5.28.3.5 void Arc::DataMover::verbose (const std::string & *prefix*)

Activate printing information about transfer status.

**Parameters:**

***prefix*** use this string if 'prefix' in DataMover::Transfer is NULL.

#### 5.28.3.6 bool Arc::DataMover::retry ()

Check if transfer will be retried in case of failure.

#### 5.28.3.7 void Arc::DataMover::retry (bool)

Set if transfer will be retried in case of failure.

#### 5.28.3.8 void Arc::DataMover::secure (bool)

Set if high level of security (encryption) will be used duirng transfer if available.

#### 5.28.3.9 void Arc::DataMover::passive (bool)

Set if passive transfer should be used for FTP-like transfers.

#### 5.28.3.10 void Arc::DataMover::force_to_meta (bool)

Set if file should be transfered and registered even if such LFN is already registered and source is not one of registered locations.

#### 5.28.3.11 bool Arc::DataMover::checks ()

Check if check for existance of remote file is done before initiating 'reading' and 'writing' operations.

#### 5.28.3.12 void Arc::DataMover::checks (bool *v*)

Set if to make check for existance of remote file (and probably other checks too) before initiating 'reading' and 'writing' operations.

**Parameters:**

***v*** true if allowed (default is true).

#### 5.28.3.13 void Arc::DataMover::set_default_min_speed (unsigned long long int *min_speed*, time_t *min_speed_time*) [inline]

Set minimal allowed transfer speed (default is 0) to 'min_speed'. If speed drops below for time longer than 'min_speed_time' error is raised. For more information see description of DataSpeed class.

#### 5.28.3.14 void Arc::DataMover::set_default_min_average_speed (unsigned long long int *min_average_speed*) [inline]

Set minimal allowed average transfer speed (default is 0 averaged over whole time of transfer. For more information see description of DataSpeed class.

#### 5.28.3.15 void Arc::DataMover::set_default_max_inactivity_time (time_t *max_inactivity_time*) [inline]

Set maximal allowed time for waiting for any data. For more information see description of DataSpeed class.

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

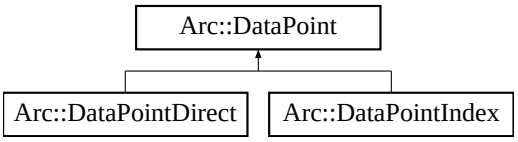
- DataMover.h

## 5.29 Arc::DataPoint Class Reference

This base class is an abstraction of URL.

`#include <DataPoint.h>`

Inheritance diagram for Arc::DataPoint::




### Public Member Functions

- DataPoint (const URL &url)
- virtual ~DataPoint ()
- virtual const URL & GetURL () const
- virtual std::string str () const
- virtual operator bool () const
- virtual bool operator! () const
- virtual DataStatus StartReading (DataBufferPar &buffer)=0
- virtual DataStatus StartWriting (DataBufferPar &buffer, DataCallback *space_cb=NULL)=0
- virtual DataStatus StopReading ()=0
- virtual DataStatus StopWriting ()=0
- virtual DataStatus Check ()=0
- virtual DataStatus Remove ()=0
- virtual DataStatus ListFiles (std::list< FileInfo > &files, bool resolve=true)=0
- virtual void ReadOutOfOrder (bool v)=0
- virtual bool WriteOutOfOrder ()=0
- virtual void SetAdditionalChecks (bool v)=0
- virtual bool GetAdditionalChecks () const =0
- virtual void SetSecure (bool v)=0
- virtual bool GetSecure () const =0
- virtual void Passive (bool v)=0
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)=0
- virtual DataStatus Resolve (bool source)=0
- virtual bool Registered () const =0
- virtual DataStatus PreRegister (bool replication, bool force=false)=0
- virtual DataStatus PostRegister (bool replication)=0
- virtual DataStatus PreUnregister (bool replication)=0
- virtual DataStatus Unregister (bool all)=0
- virtual bool CheckSize () const
- virtual void SetSize (const unsigned long long int val)
- virtual unsigned long long int GetSize () const
- virtual bool CheckCheckSum () const
- virtual void SetCheckSum (const std::string &val)
- virtual const std::string & GetCheckSum () const
- virtual bool CheckCreated () const

- virtual void SetCreated (const Time &val)
- virtual const Time & GetCreated () const
- virtual bool CheckValid () const
- virtual void SetValid (const Time &val)
- virtual const Time & GetValid () const
- virtual unsigned long long int BufSize () const =0
- virtual int BufNum () const =0
- virtual bool Cache () const =0
- virtual bool Local () const =0
- virtual bool **ReadOnly** () const =0
- virtual int GetTries () const
- virtual void SetTries (const int n)
- virtual bool IsIndex () const =0
- virtual bool AcceptsMeta ()=0
- virtual bool ProvidesMeta ()=0
- virtual void SetMeta (const DataPoint &p)
- virtual bool CompareMeta (const DataPoint &p) const
- virtual const URL & CurrentLocation () const =0
- virtual const std::string & CurrentLocationMetadata () const =0
- virtual bool NextLocation ()=0
- virtual bool LocationValid () const =0
- virtual bool HaveLocations () const =0
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)=0
- virtual DataStatus RemoveLocation ()=0
- virtual DataStatus RemoveLocations (const DataPoint &p)=0

## Protected Attributes

- URL **url**
- unsigned long long int **size**
- std::string **checksum**
- Time **created**
- Time **valid**
- int **triesleft**

## Static Protected Attributes

- static Logger **logger**

### 5.29.1 Detailed Description

This base class is an abstraction of URL.

Specializations should be provided for different kind of direct access URLs (`file://`, `ftp://`, gsiftp://, `http://`, `https://`, httpg://, ...) or indexing service URLs (rls://, lfc://, ...). DataPoint provides means to resolve an indexing service URL into multiple URLs and to loop through them.

## 5.29.2 Constructor & Destructor Documentation

### 5.29.2.1 Arc::DataPoint::DataPoint (const URL & *url*)

Constructor requires URL to be provided.

### 5.29.2.2 virtual Arc::DataPoint::~DataPoint () [virtual]

Destructor.

## 5.29.3 Member Function Documentation

### 5.29.3.1 virtual const URL& Arc::DataPoint::GetURL () const [virtual]

Returns the URL that was passed to the constructor.

### 5.29.3.2 virtual std::string Arc::DataPoint::str () const [virtual]

Returns a string representation of the DataPoint.

### 5.29.3.3 virtual Arc::DataPoint::operator bool () const [virtual]

Is DataPoint valid?

### 5.29.3.4 virtual bool Arc::DataPoint::operator! () const [virtual]

Is DataPoint valid?

### 5.29.3.5 virtual DataStatus Arc::DataPoint::StartReading (DataBufferPar & *buffer*) [pure virtual]

Start reading data from URL.

Separate thread to transfer data will be created. No other operation can be performed while reading is in progress.

**Parameters:**

***buffer*** operation will use this buffer to put information into. Should not be destroyed before stop_-reading was called and returned.

Implemented in Arc::DataPointIndex.

### 5.29.3.6 virtual DataStatus Arc::DataPoint::StartWriting (DataBufferPar & *buffer*, DataCallback * ***space_cb*** = NULL) [pure virtual]

Start writing data to URL.

Separate thread to transfer data will be created. No other operation can be performed while writing is in progress.

**Parameters:**

***buffer*** operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

***space_cb*** callback which is called if there is not enough space to store data. May not implemented for all protocols.

Implemented in Arc::DataPointIndex.

#### 5.29.3.7 virtual DataStatus Arc::DataPoint::StopReading () [pure virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in Arc::DataPointIndex.

#### 5.29.3.8 virtual DataStatus Arc::DataPoint::StopWriting () [pure virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in Arc::DataPointIndex.

#### 5.29.3.9 virtual DataStatus Arc::DataPoint::Check () [pure virtual]

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object.

Implemented in Arc::DataPointIndex.

#### 5.29.3.10 virtual DataStatus Arc::DataPoint::Remove () [pure virtual]

Remove/delete object at URL.

Implemented in Arc::DataPointIndex.

#### 5.29.3.11 virtual DataStatus Arc::DataPoint::ListFiles (std::list< FileInfo > & *files*, bool *resolve* = true) [pure virtual]

List file(s).

If the DataPoint represents a directory its contents will be listed.

**Parameters:**

***files*** will contain list of file names and optionally their attributes.

***resolve*** if false, do not try to obtain properties of objects.

#### 5.29.3.12 virtual void Arc::DataPoint::ReadOutOfOrder (bool *v*) [pure virtual]

Allow/disallow DataPoint to produce scattered data during reading∗ operation.

**Parameters:**

*v* true if allowed (default is false).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.13 virtual bool Arc::DataPoint::WriteOutOfOrder () [pure virtual]

Returns true if URL can accept scattered data for ∗writing∗ operation.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.14 virtual void Arc::DataPoint::SetAdditionalChecks (bool *v*) [pure virtual]

Allow/disallow additional checks.

Check for existance of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters:**

*v* true if allowed (default is true).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.15 virtual bool Arc::DataPoint::GetAdditionalChecks () const [pure virtual]

Check if additional checks before will be performed.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.16 virtual void Arc::DataPoint::SetSecure (bool *v*) [pure virtual]

Allow/disallow heavy security during data transfer.

**Parameters:**

*v* true if allowed (default depends on protocol).

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.17 virtual bool Arc::DataPoint::GetSecure () const [pure virtual]

Check if heavy security during data transfer is allowed.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.18 virtual void Arc::DataPoint::Passive (bool *v*) [pure virtual]

Request passive transfers for FTP-like protocols.

**Parameters:**

***true*** to request.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.19 virtual void Arc::DataPoint::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [pure virtual]

Set range of bytes to retrieve.

Default values correspond to whole file.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.20 virtual DataStatus Arc::DataPoint::Resolve (bool *source*) [pure virtual]

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

**Parameters:**

***source*** true if DataPoint object represents source of information.

Implemented in Arc::DataPointDirect.

#### 5.29.3.21 virtual bool Arc::DataPoint::Registered () const [pure virtual]

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.22 virtual DataStatus Arc::DataPoint::PreRegister (bool *replication*, bool *force* = false) [pure virtual]

Index service preregistration.

This function registers the physical location of a file into an indexing service. It should be called ∗before∗ the actual transfer to that location happens.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in the indexing service under same name.

***force*** if true, perform registration of a new file even if it already exists. Should be used to fix failures in Indexing Service.

Implemented in Arc::DataPointDirect.

#### 5.29.3.23 virtual DataStatus Arc::DataPoint::PostRegister (bool *replication*) [pure virtual]

Index Service postregistration.

Used for same purpose as meta_preregister. Should be called after actual transfer of file successfully finished.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in Indexing Service under same name.

Implemented in Arc::DataPointDirect.

#### 5.29.3.24 virtual DataStatus Arc::DataPoint::PreUnregister (bool *replication*) [pure virtual]

Index Service preunregistration.

Should be called if file transfer failed. It removes changes made by PreRegister.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in Indexing Service under same name.

Implemented in Arc::DataPointDirect.

#### 5.29.3.25 virtual DataStatus Arc::DataPoint::Unregister (bool *all*) [pure virtual]

Index Service unregistration.

Remove information about file registered in Indexing Service.

**Parameters:**

***all*** if true, information about file itself is (LFN) is removed. Otherwise only particular physical instance is unregistered.

Implemented in Arc::DataPointDirect.

#### 5.29.3.26 virtual bool Arc::DataPoint::CheckSize () const [virtual]

Check if meta-information 'size' is available.

#### 5.29.3.27 virtual void Arc::DataPoint::SetSize (const unsigned long long int *val*) [virtual]

Set value of meta-information 'size'.

#### 5.29.3.28 virtual unsigned long long int Arc::DataPoint::GetSize () const [virtual]

Get value of meta-information 'size'.

#### 5.29.3.29 virtual bool Arc::DataPoint::CheckCheckSum () const [virtual]

Check if meta-information 'checksum' is available.

#### 5.29.3.30 virtual void Arc::DataPoint::SetCheckSum (const std::string & *val*) [virtual]

Set value of meta-information 'checksum'.

#### 5.29.3.31 virtual const std::string& Arc::DataPoint::GetCheckSum () const [virtual]

Get value of meta-information 'checksum'.

#### 5.29.3.32 virtual bool Arc::DataPoint::CheckCreated () const [virtual]

Check if meta-information 'creation/modification time' is available.

#### 5.29.3.33 virtual void Arc::DataPoint::SetCreated (const Time & *val*) [virtual]

Set value of meta-information 'creation/modification time'.

#### 5.29.3.34 virtual const Time& Arc::DataPoint::GetCreated () const [virtual]

Get value of meta-information 'creation/modification time'.

#### 5.29.3.35 virtual bool Arc::DataPoint::CheckValid () const [virtual]

Check if meta-information 'validity time' is available.

#### 5.29.3.36 virtual void Arc::DataPoint::SetValid (const Time & *val*) [virtual]

Set value of meta-information 'validity time'.

#### 5.29.3.37 virtual const Time& Arc::DataPoint::GetValid () const [virtual]

Get value of meta-information 'validity time'.

#### 5.29.3.38 virtual unsigned long long int Arc::DataPoint::BufSize () const [pure virtual]

Get suggested buffer size for transfers.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.39 virtual int Arc::DataPoint::BufNum () const [pure virtual]

Get suggested number of buffers for transfers.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.40 virtual bool Arc::DataPoint::Cache () const [pure virtual]

Returns true if file is cacheable.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.41 virtual bool Arc::DataPoint::Local () const [pure virtual]

Returns true if file is local, e.g. `file://` urls.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.42 virtual int Arc::DataPoint::GetTries () const [virtual]

Returns number of retries left.

#### 5.29.3.43 virtual void Arc::DataPoint::SetTries (const int *n*) [virtual]

Set number of retries.

Reimplemented in Arc::DataPointIndex.

#### 5.29.3.44 virtual bool Arc::DataPoint::IsIndex () const [pure virtual]

Check if URL is an Indexing Service.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.45 virtual bool Arc::DataPoint::AcceptsMeta () [pure virtual]

If endpoint can have any use from meta information.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.46 virtual bool Arc::DataPoint::ProvidesMeta () [pure virtual]

If endpoint can provide at least some meta information directly.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.47 virtual void Arc::DataPoint::SetMeta (const DataPoint & *p*) [virtual]

Copy meta information from another object.

Already defined values are not overwritten.

**Parameters:**

***p*** object from which information is taken.

#### 5.29.3.48 virtual bool Arc::DataPoint::CompareMeta (const DataPoint & *p*) const [virtual]

Compare meta information from another object.

Undefined values are not used for comparison.

**Parameters:**

***p*** object to which to compare.

#### 5.29.3.49 virtual const URL& Arc::DataPoint::CurrentLocation () const [pure virtual]

Returns current (resolved) URL.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.50 virtual const std::string& Arc::DataPoint::CurrentLocationMetadata () const [pure virtual]

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.51 virtual bool Arc::DataPoint::NextLocation () [pure virtual]

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.52 virtual bool Arc::DataPoint::LocationValid () const [pure virtual]

Returns false if out of retries.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.53 virtual bool Arc::DataPoint::HaveLocations () const [pure virtual]

Returns true if number of resolved URLs is not 0.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.54 virtual DataStatus Arc::DataPoint::AddLocation (const URL & *url*, const std::string & *meta*) [pure virtual]

Add URL to list.

**Parameters:**

***url*** Location URL to add.

***meta*** Location meta information.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.55 virtual DataStatus Arc::DataPoint::RemoveLocation () [pure virtual]

Remove current URL from list.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

#### 5.29.3.56 virtual DataStatus Arc::DataPoint::RemoveLocations (const DataPoint & *p*) [pure virtual]

Remove locations present in another DataPoint object.

Implemented in Arc::DataPointDirect, and Arc::DataPointIndex.

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

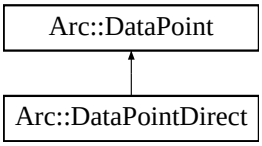
- DataPoint.h

# 5.30 Arc::DataPointDirect Class Reference

This is a kind of generalized file handle.

`#include <DataPointDirect.h>`

Inheritance diagram for Arc::DataPointDirect::




## Public Member Functions

- **DataPointDirect** (const URL &url)
- virtual bool IsIndex () const
- virtual unsigned long long int BufSize () const
- virtual int BufNum () const
- virtual bool Cache () const
- virtual bool Local () const
- virtual bool **ReadOnly** () const
- virtual void ReadOutOfOrder (bool v)
- virtual bool WriteOutOfOrder ()
- virtual void SetAdditionalChecks (bool v)
- virtual bool GetAdditionalChecks () const
- virtual void SetSecure (bool v)
- virtual bool GetSecure () const
- virtual void Passive (bool v)
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)
- virtual DataStatus Resolve (bool source)
- virtual bool Registered () const
- virtual DataStatus PreRegister (bool replication, bool force=false)
- virtual DataStatus PostRegister (bool replication)
- virtual DataStatus PreUnregister (bool replication)
- virtual DataStatus Unregister (bool all)
- virtual bool AcceptsMeta ()
- virtual bool ProvidesMeta ()
- virtual const URL & CurrentLocation () const
- virtual const std::string & CurrentLocationMetadata () const
- virtual bool NextLocation ()
- virtual bool LocationValid () const
- virtual bool HaveLocations () const
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)
- virtual DataStatus RemoveLocation ()
- virtual DataStatus RemoveLocations (const DataPoint &p)

## Protected Attributes

- DataBufferPar * **buffer**
- unsigned long long int **bufsize**
- int **bufnum**
- bool **cache**
- bool **local**
- bool **readonly**
- bool **linkable**
- bool **is_secure**
- bool **force_secure**
- bool **force_passive**
- bool **additional_checks**
- bool **allow_out_of_order**
- unsigned long long int **range_start**
- unsigned long long int **range_end**

### 5.30.1 Detailed Description

This is a kind of generalized file handle.

Differently from file handle it does not support operations read() and write(). Instead it initiates operation and uses object of class DataBufferPar to pass actual data. It also provides other operations like querying parameters of remote object. It is used by higher-level classes DataMove and DataMovePar to provide data transfer service for application.

### 5.30.2 Member Function Documentation

#### 5.30.2.1 virtual bool Arc::DataPointDirect::IsIndex () const [virtual]

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

#### 5.30.2.2 virtual unsigned long long int Arc::DataPointDirect::BufSize () const [virtual]

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

#### 5.30.2.3 virtual int Arc::DataPointDirect::BufNum () const [virtual]

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

#### 5.30.2.4 virtual bool Arc::DataPointDirect::Cache () const [virtual]

Returns true if file is cacheable.

Implements Arc::DataPoint.

#### 5.30.2.5 virtual bool Arc::DataPointDirect::Local () const [virtual]

Returns true if file is local, e.g. file:// urls.

Implements Arc::DataPoint.

#### 5.30.2.6 virtual void Arc::DataPointDirect::ReadOutOfOrder (bool *v*) [virtual]

Allow/disallow DataPoint to produce scattered data during reading∗ operation.

**Parameters:**

*v* true if allowed (default is false).

Implements Arc::DataPoint.

#### 5.30.2.7 virtual bool Arc::DataPointDirect::WriteOutOfOrder () [virtual]

Returns true if URL can accept scattered data for ∗writing∗ operation.

Implements Arc::DataPoint.

#### 5.30.2.8 virtual void Arc::DataPointDirect::SetAdditionalChecks (bool *v*) [virtual]

Allow/disallow additional checks.

Check for existance of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters:**

*v* true if allowed (default is true).

Implements Arc::DataPoint.

#### 5.30.2.9 virtual bool Arc::DataPointDirect::GetAdditionalChecks () const [virtual]

Check if additional checks before will be performed.

Implements Arc::DataPoint.

#### 5.30.2.10 virtual void Arc::DataPointDirect::SetSecure (bool *v*) [virtual]

Allow/disallow heavy security during data transfer.

**Parameters:**

*v* true if allowed (default depends on protocol).

Implements Arc::DataPoint.

#### 5.30.2.11 virtual bool Arc::DataPointDirect::GetSecure () const [virtual]

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

#### 5.30.2.12 virtual void Arc::DataPointDirect::Passive (bool *v*) [virtual]

Request passive transfers for FTP-like protocols.

**Parameters:**

***true*** to request.

Implements Arc::DataPoint.

#### 5.30.2.13 virtual void Arc::DataPointDirect::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [virtual]

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements Arc::DataPoint.

#### 5.30.2.14 virtual DataStatus Arc::DataPointDirect::Resolve (bool *source*) [virtual]

Resolves index service URL into list of ordinary URLs.

Also obtains meta information about the file.

**Parameters:**

***source*** true if DataPoint object represents source of information.

Implements Arc::DataPoint.

#### 5.30.2.15 virtual bool Arc::DataPointDirect::Registered () const [virtual]

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

#### 5.30.2.16 virtual DataStatus Arc::DataPointDirect::PreRegister (bool *replication*, bool *force* = false) [virtual]

Index service preregistration.

This function registers the physical location of a file into an indexing service. It should be called ∗before∗ the actual transfer to that location happens.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in the indexing service under same name.

***force*** if true, perform registration of a new file even if it already exists. Should be used to fix failures in Indexing Service.

Implements Arc::DataPoint.

#### 5.30.2.17 virtual DataStatus Arc::DataPointDirect::PostRegister (bool *replication*) [virtual]

Index Service postregistration.

Used for same purpose as meta_preregister. Should be called after actual transfer of file successfully finished.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in Indexing Service under same name.

Implements Arc::DataPoint.

#### 5.30.2.18 virtual DataStatus Arc::DataPointDirect::PreUnregister (bool *replication*) [virtual]

Index Service preunregistration.

Should be called if file transfer failed. It removes changes made by PreRegister.

**Parameters:**

***replication*** if true, the file is being replicated between two locations registered in Indexing Service under same name.

Implements Arc::DataPoint.

#### 5.30.2.19 virtual DataStatus Arc::DataPointDirect::Unregister (bool *all*) [virtual]

Index Service unregistration.

Remove information about file registered in Indexing Service.

**Parameters:**

***all*** if true, information about file itself is (LFN) is removed. Otherwise only particular physical instance is unregistered.

Implements Arc::DataPoint.

#### 5.30.2.20 virtual bool Arc::DataPointDirect::AcceptsMeta () [virtual]

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

#### 5.30.2.21 virtual bool Arc::DataPointDirect::ProvidesMeta () [virtual]

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

#### 5.30.2.22 virtual const URL& Arc::DataPointDirect::CurrentLocation () const [virtual]

Returns current (resolved) URL.

Implements Arc::DataPoint.

#### 5.30.2.23 virtual const std::string& Arc::DataPointDirect::CurrentLocationMetadata () const [virtual]

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements Arc::DataPoint.

#### 5.30.2.24 virtual bool Arc::DataPointDirect::NextLocation () [virtual]

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements Arc::DataPoint.

#### 5.30.2.25 virtual bool Arc::DataPointDirect::LocationValid () const [virtual]

Returns false if out of retries.

Implements Arc::DataPoint.

#### 5.30.2.26 virtual bool Arc::DataPointDirect::HaveLocations () const [virtual]

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

#### 5.30.2.27 virtual DataStatus Arc::DataPointDirect::AddLocation (const URL & *url*, const std::string & *meta*) [virtual]

Add URL to list.

**Parameters:**

***url*** Location URL to add.

***meta*** Location meta information.

Implements Arc::DataPoint.

#### 5.30.2.28 virtual DataStatus Arc::DataPointDirect::RemoveLocation () [virtual]

Remove current URL from list.

Implements Arc::DataPoint.

#### 5.30.2.29 virtual DataStatus Arc::DataPointDirect::RemoveLocations (const DataPoint & *p*) [virtual]

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

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

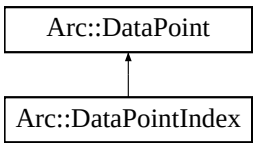
- DataPointDirect.h

## 5.31 Arc::DataPointIndex Class Reference

Complements DataPoint with attributes common for Indexing Service URLs.

`#include <DataPointIndex.h>`

Inheritance diagram for Arc::DataPointIndex::




### Public Member Functions

- **DataPointIndex** (const URL &url)
- virtual const URL & CurrentLocation () const
- virtual const std::string & CurrentLocationMetadata () const
- virtual bool NextLocation ()
- virtual bool LocationValid () const
- virtual bool HaveLocations () const
- virtual DataStatus RemoveLocation ()
- virtual DataStatus RemoveLocations (const DataPoint &p)
- virtual DataStatus AddLocation (const URL &url, const std::string &meta)
- virtual bool IsIndex () const
- virtual bool AcceptsMeta ()
- virtual bool ProvidesMeta ()
- virtual bool Registered () const
- virtual void SetTries (const int n)
- virtual unsigned long long int BufSize () const
- virtual int BufNum () const
- virtual bool Cache () const
- virtual bool Local () const
- virtual bool **ReadOnly** () const
- virtual DataStatus StartReading (DataBufferPar &buffer)
- virtual DataStatus StartWriting (DataBufferPar &buffer, DataCallback *space_cb=NULL)
- virtual DataStatus StopReading ()
- virtual DataStatus StopWriting ()
- virtual DataStatus Check ()
- virtual DataStatus Remove ()
- virtual void ReadOutOfOrder (bool v)
- virtual bool WriteOutOfOrder ()
- virtual void SetAdditionalChecks (bool v)
- virtual bool GetAdditionalChecks () const
- virtual void SetSecure (bool v)
- virtual bool GetSecure () const
- virtual void Passive (bool v)
- virtual void Range (unsigned long long int start=0, unsigned long long int end=0)

## Protected Attributes

- std::list< URLLocation > locations
- std::list< URLLocation >::iterator **location**
- DataHandle **h**
- bool **resolved**
- bool **registered**

### 5.31.1 Detailed Description

Complements DataPoint with attributes common for Indexing Service URLs.

It should never be used directly. Instead inherit from it to provide a class for specific a Indexing Service.

### 5.31.2 Member Function Documentation

#### 5.31.2.1 virtual const URL& Arc::DataPointIndex::CurrentLocation () const [virtual]

Returns current (resolved) URL.

Implements Arc::DataPoint.

#### 5.31.2.2 virtual const std::string& Arc::DataPointIndex::CurrentLocationMetadata () const [virtual]

Returns meta information used to create current URL.

Usage differs between different indexing services.

Implements Arc::DataPoint.

#### 5.31.2.3 virtual bool Arc::DataPointIndex::NextLocation () [virtual]

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements Arc::DataPoint.

#### 5.31.2.4 virtual bool Arc::DataPointIndex::LocationValid () const [virtual]

Returns false if out of retries.

Implements Arc::DataPoint.

#### 5.31.2.5 virtual bool Arc::DataPointIndex::HaveLocations () const [virtual]

Returns true if number of resolved URLs is not 0.

Implements Arc::DataPoint.

#### 5.31.2.6 virtual DataStatus Arc::DataPointIndex::RemoveLocation () [virtual]

Remove current URL from list.

Implements Arc::DataPoint.

#### 5.31.2.7 virtual DataStatus Arc::DataPointIndex::RemoveLocations (const DataPoint & *p*) [virtual]

Remove locations present in another DataPoint object.

Implements Arc::DataPoint.

#### 5.31.2.8 virtual DataStatus Arc::DataPointIndex::AddLocation (const URL & *url*, const std::string & *meta*) [virtual]

Add URL to list.

**Parameters:**

***url*** Location URL to add.

***meta*** Location meta information.

Implements Arc::DataPoint.

#### 5.31.2.9 virtual bool Arc::DataPointIndex::IsIndex () const [virtual]

Check if URL is an Indexing Service.

Implements Arc::DataPoint.

#### 5.31.2.10 virtual bool Arc::DataPointIndex::AcceptsMeta () [virtual]

If endpoint can have any use from meta information.

Implements Arc::DataPoint.

#### 5.31.2.11 virtual bool Arc::DataPointIndex::ProvidesMeta () [virtual]

If endpoint can provide at least some meta information directly.

Implements Arc::DataPoint.

#### 5.31.2.12 virtual bool Arc::DataPointIndex::Registered () const [virtual]

Check if file is registered in Indexing Service.

Proper value is obtainable only after Resolve.

Implements Arc::DataPoint.

#### 5.31.2.13 virtual void Arc::DataPointIndex::SetTries (const int *n*) [virtual]

Set number of retries.

Reimplemented from Arc::DataPoint.

#### 5.31.2.14 virtual unsigned long long int Arc::DataPointIndex::BufSize () const [virtual]

Get suggested buffer size for transfers.

Implements Arc::DataPoint.

#### 5.31.2.15 virtual int Arc::DataPointIndex::BufNum () const [virtual]

Get suggested number of buffers for transfers.

Implements Arc::DataPoint.

#### 5.31.2.16 virtual bool Arc::DataPointIndex::Cache () const [virtual]

Returns true if file is cacheable.

Implements Arc::DataPoint.

#### 5.31.2.17 virtual bool Arc::DataPointIndex::Local () const [virtual]

Returns true if file is local, e.g. `file://` urls.

Implements Arc::DataPoint.

#### 5.31.2.18 virtual DataStatus Arc::DataPointIndex::StartReading (DataBufferPar & *buffer*) [virtual]

Start reading data from URL.

Separate thread to transfer data will be created. No other operation can be performed while reading is in progress.

**Parameters:**

***buffer*** operation will use this buffer to put information into. Should not be destroyed before stop_reading was called and returned.

Implements Arc::DataPoint.

#### 5.31.2.19 virtual DataStatus Arc::DataPointIndex::StartWriting (DataBufferPar & *buffer*, DataCallback * *space_cb* = NULL) [virtual]

Start writing data to URL.

Separate thread to transfer data will be created. No other operation can be performed while writing is in progress.

**Parameters:**

***buffer*** operation will use this buffer to get information from. Should not be destroyed before stop_-writing was called and returned.

***space_cb*** callback which is called if there is not enough space to store data. May not implemented for all protocols.

Implements Arc::DataPoint.

#### 5.31.2.20 virtual DataStatus Arc::DataPointIndex::StopReading () [virtual]

Stop reading.

Must be called after corresponding start_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

#### 5.31.2.21 virtual DataStatus Arc::DataPointIndex::StopWriting () [virtual]

Stop writing.

Must be called after corresponding start_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements Arc::DataPoint.

#### 5.31.2.22 virtual DataStatus Arc::DataPointIndex::Check () [virtual]

Query the DataPoint to check if object is accessible.

If possible this method will also try to provide meta information about the object.

Implements Arc::DataPoint.

#### 5.31.2.23 virtual DataStatus Arc::DataPointIndex::Remove () [virtual]

Remove/delete object at URL.

Implements Arc::DataPoint.

#### 5.31.2.24 virtual void Arc::DataPointIndex::ReadOutOfOrder (bool *v*) [virtual]

Allow/disallow DataPoint to produce scattered data during reading∗ operation.

**Parameters:**

***v*** true if allowed (default is false).

Implements Arc::DataPoint.

#### 5.31.2.25 virtual bool Arc::DataPointIndex::WriteOutOfOrder () [virtual]

Returns true if URL can accept scattered data for ∗writing∗ operation.

Implements Arc::DataPoint.

#### 5.31.2.26 virtual void Arc::DataPointIndex::SetAdditionalChecks (bool *v*) [virtual]

Allow/disallow additional checks.

Check for existance of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters:**

***v*** true if allowed (default is true).

Implements Arc::DataPoint.

#### 5.31.2.27 virtual bool Arc::DataPointIndex::GetAdditionalChecks () const [virtual]

Check if additional checks before will be performed.

Implements Arc::DataPoint.

#### 5.31.2.28 virtual void Arc::DataPointIndex::SetSecure (bool *v*) [virtual]

Allow/disallow heavy security during data transfer.

**Parameters:**

***v*** true if allowed (default depends on protocol).

Implements Arc::DataPoint.

#### 5.31.2.29 virtual bool Arc::DataPointIndex::GetSecure () const [virtual]

Check if heavy security during data transfer is allowed.

Implements Arc::DataPoint.

#### 5.31.2.30 virtual void Arc::DataPointIndex::Passive (bool *v*) [virtual]

Request passive transfers for FTP-like protocols.

**Parameters:**

***true*** to request.

Implements Arc::DataPoint.

#### 5.31.2.31 virtual void Arc::DataPointIndex::Range (unsigned long long int *start* = 0, unsigned long long int *end* = 0) [virtual]

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements Arc::DataPoint.

### 5.31.3 Member Data Documentation

#### 5.31.3.1 std::list<URLLocation> Arc::DataPointIndex::locations [protected]

List of locations at which file can be probably found.

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

- DataPointIndex.h

# 5.32 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

`#include <DataSpeed.h>`

## Public Types

- typedef void(∗ **show_progress_t** )(FILE ∗o, const char ∗s, unsigned int t, unsigned long long int all, unsigned long long int max, double instant, double average)

## Public Member Functions

- DataSpeed (time_t base=DATASPEED_AVERAGING_PERIOD)
- DataSpeed (unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_average_speed, time_t max_inactivity_time, time_t base=DATASPEED_AVERAGING_PERIOD)
- ~DataSpeed (void)
- void verbose (bool val)
- void verbose (const std::string &prefix)
- bool verbose (void)
- void set_min_speed (unsigned long long int min_speed, time_t min_speed_time)
- void set_min_average_speed (unsigned long long int min_average_speed)
- void set_max_inactivity_time (time_t max_inactivity_time)
- void set_base (time_t base_=DATASPEED_AVERAGING_PERIOD)
- void set_max_data (unsigned long long int max=0)
- void set_progress_indicator (show_progress_t func=NULL)
- void reset (void)
- bool transfer (unsigned long long int n=0)
- void hold (bool disable)
- bool min_speed_failure ()
- bool min_average_speed_failure ()
- bool max_inactivity_time_failure ()
- unsigned long long int transfered_size (void)

### 5.32.1 Detailed Description

Keeps track of average and instantaneous transfer speed.

Also detects data transfer inactivity and other transfer timeouts.

### 5.32.2 Constructor & Destructor Documentation

#### 5.32.2.1 Arc::DataSpeed::DataSpeed (time_t *base* = `DATASPEED_AVERAGING_PERIOD`)

Constructor

**Parameters:**

***base*** time period used to average values (default 1 minute).

#### 5.32.2.2 Arc::DataSpeed::DataSpeed (unsigned long long int *min_speed*, time_t *min_speed_time*, unsigned long long int *min_average_speed*, time_t *max_inactivity_time*, time_t *base* = DATASPEED_AVERAGING_PERIOD)

Constructor

**Parameters:**

***base*** time period used to average values (default 1 minute).

***min_speed*** minimal allowed speed (Butes per second). If speed drops and holds below threshold for min_speed_time_ seconds error is triggered.

***min_speed_time***

***min_average_speed_*** minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

***max_inactivity_time*** - if no data is passing for specified amount of time (seconds), error is triggered.

#### 5.32.2.3 Arc::DataSpeed::~DataSpeed (void)

Destructor.

### 5.32.3 Member Function Documentation

#### 5.32.3.1 void Arc::DataSpeed::verbose (bool *val*)

Activate printing information about current time speeds, amount of transfered data.

#### 5.32.3.2 void Arc::DataSpeed::verbose (const std::string & *prefix*)

Print information about current speed and amout of data.

**Parameters:**

***'prefix'*** add this string at the beginning of every string.

#### 5.32.3.3 bool Arc::DataSpeed::verbose (void)

Check if speed information is going to be printed.

#### 5.32.3.4 void Arc::DataSpeed::set_min_speed (unsigned long long int *min_speed*, time_t *min_speed_time*)

Set minimal allowed speed.

**Parameters:**

***min_speed*** minimal allowed speed (Butes per second). If speed drops and holds below threshold for min_speed_time_ seconds error is triggered.

***min_speed_time***

#### 5.32.3.5 void Arc::DataSpeed::set_min_average_speed (unsigned long long int *min_average_speed*)

Set minmal avaerage speed.

**Parameters:**

***min_average_speed_*** minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

#### 5.32.3.6 void Arc::DataSpeed::set_max_inactivity_time (time_t *max_inactivity_time*)

Set inactivity tiemout.

**Parameters:**

***max_inactivity_time*** - if no data is passing for specified amount of time (seconds), error is triggered.

#### 5.32.3.7 void Arc::DataSpeed::set_base (time_t *base_* = DATASPEED_AVERAGING_PERIOD)

Set averaging time period.

**Parameters:**

***base*** time period used to average values (default 1 minute).

#### 5.32.3.8 void Arc::DataSpeed::set_max_data (unsigned long long int *max* = 0)

Set amount of data to be transfered. Used in verbose messages.

**Parameters:**

***max*** amount of data in bytes.

#### 5.32.3.9 void Arc::DataSpeed::set_progress_indicator (show_progress_t *func* = NULL)

Specify which external function will print verbose messages. If not specified internal one is used.

**Parameters:**

***pointer*** to function which prints information.

#### 5.32.3.10 void Arc::DataSpeed::reset (void)

Reset all counters and triggers.

#### 5.32.3.11 bool Arc::DataSpeed::transfer (unsigned long long int *n* = 0)

Inform object, about amount of data has been transfered. All errors are triggered by this method. To make them work application must call this method periodically even with zero value.

**Parameters:**

***n*** amount of data transfered (bytes).

#### 5.32.3.12 void Arc::DataSpeed::hold (bool *disable*)

Turn off speed control.

**Parameters:**

***disable*** true to turn off.

#### 5.32.3.13 bool Arc::DataSpeed::min_speed_failure () [inline]

Check if minimal speed error was triggered.

#### 5.32.3.14 bool Arc::DataSpeed::min_average_speed_failure () [inline]

Check if minimal average speed error was triggered.

#### 5.32.3.15 bool Arc::DataSpeed::max_inactivity_time_failure () [inline]

Check if maximal inactivity time error was triggered.

#### 5.32.3.16 unsigned long long int Arc::DataSpeed::transfered_size (void) [inline]

Returns amount of data this object knows about.

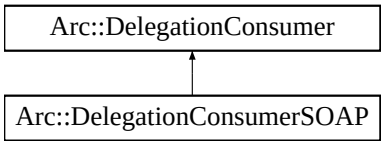
The documentation for this class was generated from the following file:

- DataSpeed.h

# 5.33 Arc::DelegationConsumer Class Reference

`#include <DelegationInterface.h>`

Inheritance diagram for Arc::DelegationConsumer::




## Public Member Functions

- DelegationConsumer (void)
- DelegationConsumer (const std::string &content)
- **operator bool** (void)
- bool **operator!** (void)
- 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)

## Protected Member Functions

- bool Generate (void)
- void LogError (void)

## Protected Attributes

- void * **key_**

### 5.33.1 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.

### 5.33.2 Constructor & Destructor Documentation

#### 5.33.2.1 Arc::DelegationConsumer::DelegationConsumer (void)

Creates object with new private key

#### 5.33.2.2 Arc::DelegationConsumer::DelegationConsumer (const std::string & *content*)

Creates object with provided private key

### 5.33.3 Member Function Documentation

#### 5.33.3.1 bool Arc::DelegationConsumer::Generate (void) [protected]

Private key

#### 5.33.3.2 void Arc::DelegationConsumer::LogError (void) [protected]

Creates private key

#### 5.33.3.3 const std::string& Arc::DelegationConsumer::ID (void)

Return identifier of this object - not implemented

#### 5.33.3.4 bool Arc::DelegationConsumer::Backup (std::string & *content*)

Stores content of this object into a string

#### 5.33.3.5 bool Arc::DelegationConsumer::Restore (const std::string & *content*)

Restores content of object from string

#### 5.33.3.6 bool Arc::DelegationConsumer::Request (std::string & *content*)

Make X509 certificate request from internal private key

#### 5.33.3.7 bool Arc::DelegationConsumer::Acquire (std::string & *content*)

Ads private key into certificates chain in 'content' On exit content contains complete delegated credentials.

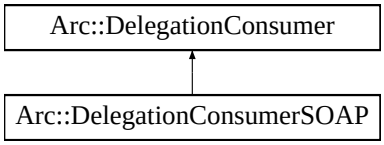
The documentation for this class was generated from the following file:

- DelegationInterface.h

# 5.34 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 DelegatedToken (std::string &credentials, const XMLNode &token)

### 5.34.1 Detailed Description

This class extends DelegationConsumer to support SOAP message exchange. Implements WS interface `http://www.nordugrid.org/schemas/delegation` described in delegation.wsdl.

### 5.34.2 Constructor & Destructor Documentation

#### 5.34.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (void)

Creates object with new private key

#### 5.34.2.2 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP (const std::string & *content*)

Creates object with specified private key

### 5.34.3 Member Function Documentation

#### 5.34.3.1 bool Arc::DelegationConsumerSOAP::DelegateCredentialsInit (const std::string & *id*, const SOAPEnvelope & *in*, SOAPEnvelope & *out*)

Process SOAP message which starts delagation. 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.

#### 5.34.3.2 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 genarated for sending to DelagationProviderSOAP.

#### 5.34.3.3 bool Arc::DelegationConsumerSOAP::DelegatedToken (std::string & *credentials*, const XMLNode & *token*)

Similar to UpdateCredentials but takes only DelegatedToken XML element

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

- DelegationInterface.h

# 5.35 Arc::DelegationContainerSOAP Class Reference

`#include <DelegationInterface.h>`

## Public Member Functions

- bool DelegateCredentialsInit (const SOAPEnvelope &in, SOAPEnvelope &out)
- bool UpdateCredentials (std::string &credentials, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool DelegatedToken (std::string &credentials, const XMLNode &token)

## Protected Attributes

- Glib::Mutex **lock_**
- int max_size_
- int max_duration_
- int max_usage_
- bool context_lock_
- bool restricted_

### 5.35.1 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.

### 5.35.2 Member Function Documentation

#### 5.35.2.1 bool Arc::DelegationContainerSOAP::DelegateCredentialsInit (const SOAPEnvelope & *in*, SOAPEnvelope & *out*)

See DelegationConsumerSOAP::DelegateCredentialsInit

#### 5.35.2.2 bool Arc::DelegationContainerSOAP::UpdateCredentials (std::string & *credentials*, const SOAPEnvelope & *in*, SOAPEnvelope & *out*)

See DelegationConsumerSOAP::UpdateCredentials

#### 5.35.2.3 bool Arc::DelegationContainerSOAP::DelegatedToken (std::string & *credentials*, const XMLNode & *token*)

See DelegationConsumerSOAP::DelegatedToken

### 5.35.3 Member Data Documentation

#### 5.35.3.1 int Arc::DelegationContainerSOAP::max_size_ [protected]

Max. number of delegation consumers

#### 5.35.3.2 int Arc::DelegationContainerSOAP::max_duration_ [protected]

Lifetime of unused delegation consumer

#### 5.35.3.3 int Arc::DelegationContainerSOAP::max_usage_ [protected]

Max. times same delegation consumer may accept credentials

#### 5.35.3.4 bool Arc::DelegationContainerSOAP::context_lock_ [protected]

If true delegation consumer is deleted when connection context is destroyed

#### 5.35.3.5 bool Arc::DelegationContainerSOAP::restricted_ [protected]

If true all delegation phases must be performed by same identity

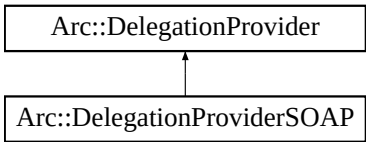
The documentation for this class was generated from the following file:

- DelegationInterface.h

# 5.36 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)
- **operator bool** (void)
- bool **operator!** (void)
- std::string Delegate (const std::string &request, const DelegationRestrictions &restrictions=DelegationRestrictions())

### 5.36.1 Detailed Description

A provider of delegated credentials. During delegation procedure this class generates new credential to be used in proxy/delegated credential.

### 5.36.2 Constructor & Destructor Documentation

#### 5.36.2.1 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.

#### 5.36.2.2 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.

### 5.36.3 Member Function Documentation

#### 5.36.3.1 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 fed into DelegationConsumer::Acquire

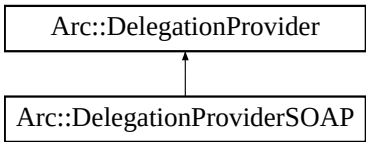
The documentation for this class was generated from the following file:

- DelegationInterface.h

# 5.37 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)
- bool DelegateCredentialsInit (MCCInterface &mcc_interface, MessageContext *context)
- bool DelegateCredentialsInit (MCCInterface &mcc_interface, MessageAttributes *attributes_in, MessageAttributes *attributes_out, MessageContext *context)
- bool UpdateCredentials (MCCInterface &mcc_interface, MessageContext *context)
- bool UpdateCredentials (MCCInterface &mcc_interface, MessageAttributes *attributes_in, MessageAttributes *attributes_out, MessageContext *context)
- bool DelegatedToken (XMLNode &parent)

## Protected Attributes

- std::string **request_**
- std::string **id_**

### 5.37.1 Detailed Description

Extension of DelegationProvider with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

### 5.37.2 Constructor & Destructor Documentation

#### 5.37.2.1 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & *credentials*)

Creates instance from provided credentials. Credentials are used to sign delegated credentials.

#### 5.37.2.2 Arc::DelegationProviderSOAP::DelegationProviderSOAP (const std::string & *cert_file*, const std::string & *key_file*)

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.

### 5.37.3 Member Function Documentation

#### 5.37.3.1 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *mcc_interface*, MessageContext * *context*)

Performs DelegateCredentialsInit SOAP operation. As result request for delegated credentials is received by this instance and stored internally. Call to UpdateCredentials should follow.

#### 5.37.3.2 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit (MCCInterface & *mcc_interface*, MessageAttributes * *attributes_in*, MessageAttributes * *attributes_out*, MessageContext * *context*)

Extended version of DelegateCredentialsInit(MCCInterface&,MessageContext*). Additionally takes attributes for request and response message to make fine control on message processing possible.

#### 5.37.3.3 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageContext * *context*)

Performs UpdateCredentials SOAP operation. This concludes delegation procedure and passes delagated credentials to DelegationConsumerSOAP instance.

#### 5.37.3.4 bool Arc::DelegationProviderSOAP::UpdateCredentials (MCCInterface & *mcc_interface*, MessageAttributes * *attributes_in*, MessageAttributes * *attributes_out*, MessageContext * *context*)

Extended version of UpdateCredentials(MCCInterface&,MessageContext*). Additionally takes attributes for request and response message to make fine control on message processing possible.

#### 5.37.3.5 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.

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

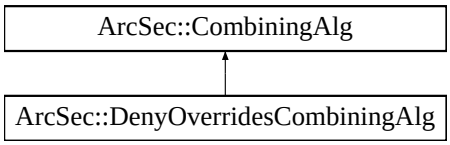
- DelegationInterface.h

# 5.38 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 std::string & **getalgId** (void)

## Static Public Member Functions

- static const std::string & **Identifier** (void)

## 5.38.1 Detailed Description

Implement the "Deny-Overrides" algorithm.

## 5.38.2 Member Function Documentation

### 5.38.2.1 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

Implements ArcSec::CombiningAlg.

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

- DenyOverridesAlg.h

## 5.39 dmc_descriptor Struct Reference

`#include <DMCLoader.h>`

### Public Attributes

- const char * **name**
- int **version**
- Arc::DMC *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.39.1 Detailed Description

This structure describes one of the DMCs stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the DMC class.

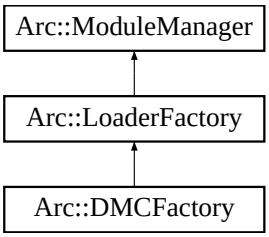
The documentation for this struct was generated from the following file:

- DMCLoader.h

# 5.40 Arc::DMCFactory Class Reference

`#include <DMCFactory.h>`

Inheritance diagram for Arc::DMCFactory::




## Public Member Functions

- DMCFactory (Config *cfg)
- DMC * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- DMC * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- DMC * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

### 5.40.1 Detailed Description

This class handles shared libraries containing DMCs

### 5.40.2 Constructor & Destructor Documentation

#### 5.40.2.1 Arc::DMCFactory::DMCFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

### 5.40.3 Member Function Documentation

#### 5.40.3.1 DMC* Arc::DMCFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of DMC and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created DMC instance.

Reimplemented from Arc::LoaderFactory.

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

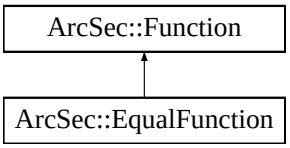
- DMCFactory.h

## 5.41 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

`#include <EqualFunction.h>`

Inheritance diagram for ArcSec::EqualFunction::




### Public Member Functions

- **EqualFunction** (std::string functionName, std::string argumentType)
- virtual bool **evaluate** (AttributeValue *arg0, AttributeValue *arg1)

### Static Public Member Functions

- static std::string getFunctionName (std::string datatype)

### 5.41.1 Detailed Description

Evaluate whether the two values are equal.

### 5.41.2 Member Function Documentation

#### 5.41.2.1 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

## 5.42 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>`

### Public Attributes

- Arc::XMLNode **node**
- std::string **effect**

### 5.42.1 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

## 5.43 ArcSec::EvaluationCtx Class Reference

EvaluationCtx, in charge of storing some context information for evaluation, including Request, current time, etc.

`#include <EvaluationCtx.h>`

### Public Member Functions

- EvaluationCtx (Request *request)
- virtual Request * **getRequest** () const
- virtual void **setRequestItem** (RequestItem *reqit)
- virtual RequestItem * **getRequestItem** () const
- virtual void split ()
- virtual std::list< RequestTuple * > **getRequestTuples** () const
- virtual void **setEvalTuple** (RequestTuple *tuple)
- virtual RequestTuple * **getEvalTuple** () const

### 5.43.1 Detailed Description

EvaluationCtx, in charge of storing some context information for evaluation, including Request, current time, etc.

### 5.43.2 Constructor & Destructor Documentation

#### 5.43.2.1 ArcSec::EvaluationCtx::EvaluationCtx (Request * *request*)

Construct a new EvaluationCtx based on the given request

### 5.43.3 Member Function Documentation

#### 5.43.3.1 virtual void ArcSec::EvaluationCtx::split () `[virtual]`

Convert/split one RequestItem ( one tuple <SubList, ResList, ActList, CtxList>) into a few <Subject, Resource, Action, Context> tuples. The purpose is for evaluation. The evaluator will evaluate each RequestTuple one by one, not the RequestItem because it includes some independent <Subject, Resource, Action, Context>s and the evaluator should deal with them independently.

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

- EvaluationCtx.h

# 5.44 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

- **EvaluatorContext** (Evaluator *evaluator)
- operator AttributeFactory * ()
- operator FnFactory * ()
- operator AlgFactory * ()

## 5.44.1 Detailed Description

Context for evaluator. It includes the factories which will be used to create related objects.

## 5.44.2 Member Function Documentation

### 5.44.2.1 ArcSec::EvaluatorContext::operator AttributeFactory * () [inline]

Returns associated AttributeFactory object

### 5.44.2.2 ArcSec::EvaluatorContext::operator FnFactory * () [inline]

Returns associated FnFactory object

### 5.44.2.3 ArcSec::EvaluatorContext::operator AlgFactory * () [inline]

Returns associated AlgFactory object

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

- Evaluator.h

## 5.45 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

### 5.45.1 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.

### 5.45.2 Member Function Documentation

#### 5.45.2.1 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 allways place the next reservation to expire at the top.

#### 5.45.2.2 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.

#### 5.45.2.3 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.

### 5.45.3 Friends And Related Function Documentation

#### 5.45.3.1 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

## 5.46 Arc::FileInfo Class Reference

FileInfo stores information about files (metadata).

`#include <FileInfo.h>`

### Public Types

- enum **Type** { **file_type_unknown** = 0, **file_type_file** = 1, **file_type_dir** = 2 }

### Public Member Functions

- **FileInfo** (const std::string &name="")
- const std::string & **GetName** () const
- std::string **GetLastName** () const
- const std::list< URL > & **GetURLs** () const
- void **AddURL** (const URL &u)
- bool **CheckSize** () const
- unsigned long long int **GetSize** () const
- void **SetSize** (const unsigned long long int s)
- bool **CheckCheckSum** () const
- const std::string & **GetCheckSum** () const
- void **SetCheckSum** (const std::string &c)
- bool **CheckCreated** () const
- Time **GetCreated** () const
- void **SetCreated** (const Time &t)
- bool **CheckValid** () const
- Time **GetValid** () const
- void **SetValid** (const Time &t)
- bool **CheckType** () const
- Type **GetType** () const
- void **SetType** (const Type t)

### 5.46.1 Detailed Description

FileInfo stores information about files (metadata).

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

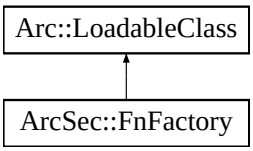
- FileInfo.h

# 5.47 ArcSec::FnFactory Class Reference

Interface for function factory, which is in charge of creating Function object according to function type.

`#include <FnFactory.h>`

Inheritance diagram for ArcSec::FnFactory::




## Public Member Functions

- virtual Function * **createFn** (const std::string &type)=0

## Protected Attributes

- FnMap **fnmap**

## 5.47.1 Detailed Description

Interface for function factory, which is in charge of creating Function object according to function type.

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

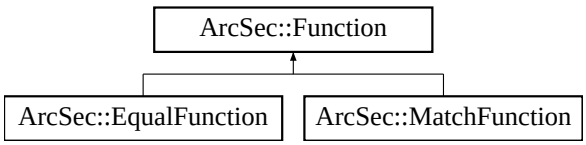
- FnFactory.h

## 5.48 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

- **Function** (std::string, std::string)
- virtual bool **evaluate** (AttributeValue *arg0, AttributeValue *arg1)=0

### 5.48.1 Detailed Description

Interface for function, which is in charge of evaluating two AttributeValue.

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

- Function.h

# 5.49 Arc::InfoRegister Class Reference

Registration to ISIS interface.

`#include <InfoRegister.h>`

## Public Member Functions

- InfoRegister (const std::string &sid, long int reg_period, Arc::Config &cfg)
- void AddUrl (const std::string &url)
- void registration (void)
- void registration_forever (void)

## 5.49.1 Detailed Description

Registration to ISIS interface.

This class provides an interface for service to register itself in Information Indexing Service.

## 5.49.2 Constructor & Destructor Documentation

### 5.49.2.1 Arc::InfoRegister::InfoRegister (const std::string & *sid*, long int *reg_period*, Arc::Config & *cfg*)

Constructor. It takes service identifier (optional), registration frequency in seconds and configuration XML subtree .

## 5.49.3 Member Function Documentation

### 5.49.3.1 void Arc::InfoRegister::AddUrl (const std::string & *url*)

Adds of ISIS service. Specified URLs will all be used during registration process.

### 5.49.3.2 void Arc::InfoRegister::registration (void)

Perform registration. All specified ISIS services are contacted and service specified in constructor is registred.

### 5.49.3.3 void Arc::InfoRegister::registration_forever (void)

Perform registration process in loop. This method calls registration() in loop every reg_period seconds. Never returns so should be run in a separate thread.

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

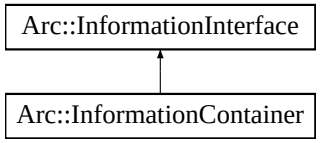
- InfoRegister.h

## 5.50 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 **Release** (void)
- void Assign (XMLNode doc, bool copy=false)

### Protected Member Functions

- virtual void Get (const std::list< std::string > &path, XMLNodeContainer &result)
- virtual void **Get** (XMLNode xpath, XMLNodeContainer &result)

### Protected Attributes

- XMLNode doc_

### 5.50.1 Detailed Description

Information System document container and processor.

This class inherits form InformationInterface and offers container for storing informational XML document.

### 5.50.2 Constructor & Destructor Documentation

#### 5.50.2.1 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.

### 5.50.3 Member Function Documentation

#### 5.50.3.1 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 specidying how to reach requested node(s).

Reimplemented from Arc::InformationInterface.

#### 5.50.3.2 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()

#### 5.50.3.3 void Arc::InformationContainer::Assign (XMLNode *doc*, bool *copy* = `false`)

Replaces internal XML document with . If is true this method makes a copy of for internal use.

### 5.50.4 Member Data Documentation

#### 5.50.4.1 XMLNode Arc::InformationContainer::doc_ `[protected]`

Either link or container of XML document

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

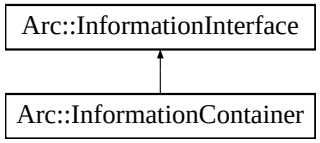
- InformationInterface.h

## 5.51 Arc::InformationInterface Class Reference

Information System message processor.

`#include <InformationInterface.h>`

Inheritance diagram for Arc::InformationInterface::




### Public Member Functions

- InformationInterface (bool safe=true)
- SOAPEnvelope * **Process** (SOAPEnvelope &in)

### Protected Member Functions

- virtual void Get (const std::list< std::string > &path, XMLNodeContainer &result)
- virtual void **Get** (XMLNode xpath, XMLNodeContainer &result)

### Protected Attributes

- Glib::Mutex lock_
- bool **to_lock_**

### 5.51.1 Detailed Description

Information System message processor.

This class provides callback for 2 operations of WS-ResourceProperties and convenient parsing/generation of corresponding SOAP mesages. In a future it may extend range of supported specifications.

### 5.51.2 Constructor & Destructor Documentation

#### 5.51.2.1 Arc::InformationInterface::InformationInterface (bool *safe* = `true`)

Constructor. If 'safe' is true all calls to Get will be locked.

### 5.51.3 Member Function Documentation

#### 5.51.3.1 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 specidying how to reach requested node(s).

Reimplemented in Arc::InformationContainer.

### 5.51.4 Member Data Documentation

#### 5.51.4.1 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

## 5.52 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)
- **operator bool** (void)
- bool **operator!** (void)
- SOAPEnvelope * SOAP (void)

### 5.52.1 Detailed Description

Request for information in InfoSystem.

This is a convenience wrapper creating proper WS-ResourceProperties request targeted InfoSystem interface of service.

### 5.52.2 Constructor & Destructor Documentation

#### 5.52.2.1 Arc::InformationRequest::InformationRequest (void)

Dummy constructor

#### 5.52.2.2 Arc::InformationRequest::InformationRequest (const std::list< std::string > & *path*)

Request for attribute specified by elements of path. Currently only first element is used.

#### 5.52.2.3 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.

#### 5.52.2.4 Arc::InformationRequest::InformationRequest (XMLNode *query*)

Request for attributes specified by XPath query.

### 5.52.3 Member Function Documentation

#### 5.52.3.1 SOAPEnvelope* Arc::InformationRequest::SOAP (void)

Returns generated SOAP message

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

- InformationInterface.h

## 5.53 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

`#include <InformationInterface.h>`

### Public Member Functions

- InformationResponse (SOAPEnvelope &soap)
- **operator bool** (void)
- bool **operator!** (void)
- std::list< XMLNode > Result (void)

### 5.53.1 Detailed Description

Informational response from InfoSystem.

This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

### 5.53.2 Constructor & Destructor Documentation

#### 5.53.2.1 Arc::InformationResponse::InformationResponse (SOAPEnvelope & *soap*)

Constructor parses WS-ResourceProperties ressponse. Provided SOAPEnvelope object must be valid as long as this object is in use.

### 5.53.3 Member Function Documentation

#### 5.53.3.1 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:

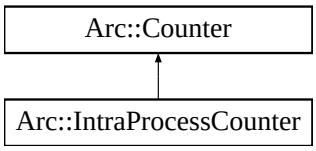
- InformationInterface.h

# 5.54 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)

### 5.54.1 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.

### 5.54.2 Constructor & Destructor Documentation

#### 5.54.2.1 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.

#### 5.54.2.2 virtual Arc::IntraProcessCounter::~IntraProcessCounter () [virtual]

Destructor.

This is the destructor of the IntraProcessCounter class. Does not need to do anything.

### 5.54.3 Member Function Documentation

#### 5.54.3.1 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.

#### 5.54.3.2 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.

#### 5.54.3.3 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.

#### 5.54.3.4 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.

#### 5.54.3.5 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.

#### 5.54.3.6 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.

#### 5.54.3.7 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 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.

#### 5.54.3.8 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.

#### 5.54.3.9 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.

#### 5.54.3.10 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.

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

- IntraProcessCounter.h

## 5.55 Arc::Loader Class Reference

Creator of Message Component Chains (MCC).

`#include <Loader.h>`

### Public Types

- typedef std::map< std::string, MCC * > **mcc_container_t**
- typedef std::map< std::string, Service * > **service_container_t**
- typedef std::map< std::string, ArcSec::SecHandler * > **sechandler_container_t**
- typedef std::map< std::string, DMC * > **dmc_container_t**
- typedef std::map< std::string, ACC * > **acc_container_t**
- typedef std::map< std::string, Plexer * > **plexer_container_t**

### Public Member Functions

- Loader (Config *cfg)
- ~Loader ()
- MCC * operator[ ] (const std::string &id)
- ACC * getACC (const std::string &id)

### Static Public Attributes

- static Logger **logger**

### Friends

- class **ChainContext**

### 5.55.1 Detailed Description

Creator of Message Component Chains (MCC).

This class processes XML configration 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.

## 5.55.2 Constructor & Destructor Documentation

### 5.55.2.1 Arc::Loader::Loader (Config * *cfg*)

Constructor that takes whole XML configuration and creates component chains

### 5.55.2.2 Arc::Loader::~Loader ()

Destructor destroys all components created by constructor

## 5.55.3 Member Function Documentation

### 5.55.3.1 MCC* Arc::Loader::operator[] (const std::string & *id*)

Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute

### 5.55.3.2 ACC* Arc::Loader::getACC (const std::string & *id*)

Access entry ACCs. Those are components exposed for external access using 'entry' attribute
The documentation for this class was generated from the following file:

- Loader.h

# 5.56 Arc::loader_descriptor Struct Reference

Identifier of plugin.

`#include <LoaderFactory.h>`

## Public Attributes

- const char * **name**
- int **version**
- void *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.56.1 Detailed Description

Identifier of plugin.

This structure describes set of elements stored in shared library. It contains name of plugin, version number and pointer to function which creates an instance of object.

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

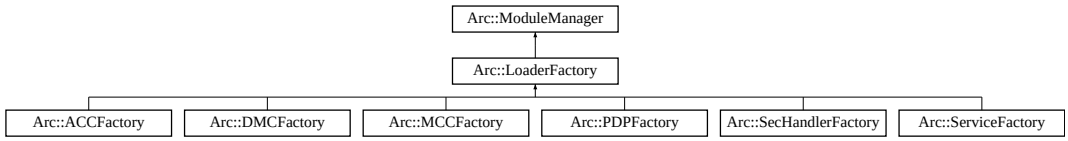
- LoaderFactory.h

## 5.57 Arc::LoaderFactory Class Reference

Plugin handler.

`#include <LoaderFactory.h>`

Inheritance diagram for Arc::LoaderFactory::




### Public Member Functions

- void load_all_instances (const std::string &libname)

### Protected Member Functions

- LoaderFactory (Config *cfg, const std::string &id)
- void * get_instance (const std::string &name, Arc::Config *cfg, Arc::ChainContext *ctx)
- void * **get_instance** (const std::string &name, int version, Arc::Config *cfg, Arc::ChainContext *ctx)
- void * **get_instance** (const std::string &name, int min_version, int max_version, Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.57.1 Detailed Description

Plugin handler.

This class handles shared libraries containing loadable classes

### 5.57.2 Constructor & Destructor Documentation

#### 5.57.2.1 Arc::LoaderFactory::LoaderFactory (Config * *cfg*, const std::string & *id*) [protected]

Constructor - accepts configuration (not yet used) meant to tune loading of modules.

### 5.57.3 Member Function Documentation

#### 5.57.3.1 void* Arc::LoaderFactory::get_instance (const std::string & *name*, Arc::Config * *cfg*, Arc::ChainContext * *ctx*) [protected]

These methods load shared library named lib'name', locates symbol named 'id_' representing descriptor of elements and calls it's constructor function. Supplied configuration tree and context are passed to constructor. Returns created instance. This classes must not be used directly. Inheriting classes must implement it with proper type casting.

Reimplemented in Arc::ACCFactory, Arc::DMCFactory, Arc::MCCFactory, Arc::PDPFactory, Arc::SecHandlerFactory, and Arc::ServiceFactory.

#### 5.57.3.2 void Arc::LoaderFactory::load_all_instances (const std::string & *libname*)

Loads shared library named 'libname' and identifies all elements it provides. Subsequent calls to get_-instance() methods will be able to locate needed elements even if they are not stored in library named after element name.

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

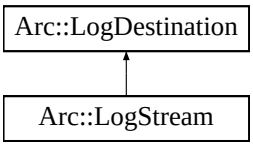
- LoaderFactory.h

## 5.58 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)

### Protected Attributes

- std::string **locale**

### 5.58.1 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.

### 5.58.2 Constructor & Destructor Documentation

#### 5.58.2.1 Arc::LogDestination::LogDestination () `[protected]`

Default constructor.

This destination will use the default locale.

#### 5.58.2.2 Arc::LogDestination::LogDestination (const std::string & *locale*) `[protected]`

Constructor with specific locale.

This destination will use the specified locale.

### 5.58.3 Member Function Documentation

#### 5.58.3.1 virtual void Arc::LogDestination::log (const LogMessage & *message*) [pure virtual]

Logs a LogMessage to this LogDestination.

Implemented in Arc::LogStream.

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

- Logger.h

## 5.59 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)
- void addDestination (LogDestination &destination)
- void removeDestinations (void)
- void setThreshold (LogLevel threshold)
- LogLevel getThreshold () const
- void msg (LogMessage message)
- void msg (LogLevel level, const std::string &str)
- template<class T0>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0)
- template<class T0, class T1>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1)
- template<class T0, class T1, class T2>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2)
- template<class T0, class T1, class T2, class T3>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2, const T3 &t3)
- template<class T0, class T1, class T2, class T3, class T4>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2, const T3 &t3, const T4 &t4)
- template<class T0, class T1, class T2, class T3, class T4, class T5>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2, const T3 &t3, const T4 &t4, const T5 &t5)
- template<class T0, class T1, class T2, class T3, class T4, class T5, class T6>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2, const T3 &t3, const T4 &t4, const T5 &t5, const T6 &t6)
- template<class T0, class T1, class T2, class T3, class T4, class T5, class T6, class T7>
  void **msg** (LogLevel level, const std::string &str, const T0 &t0, const T1 &t1, const T2 &t2, const T3 &t3, const T4 &t4, const T5 &t5, const T6 &t6, const T7 &t7)

### Static Public Member Functions

- static Logger & getRootLogger ()

### 5.59.1 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.

## 5.59.2 Constructor & Destructor Documentation

### 5.59.2.1 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.

### 5.59.2.2 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.

## 5.59.3 Member Function Documentation

### 5.59.3.1 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.

### 5.59.3.2 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 LogDestinatoins 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 iteslf.

### 5.59.3.3 void Arc::Logger::removeDestinations (void)

Removes all LogDestinations.

#### 5.59.3.4 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

#### 5.59.3.5 LogLevel Arc::Logger::getThreshold () const

Returns the threshold.

Returns the threshold.

**Returns:**

The threshold of this Logger.

#### 5.59.3.6 void Arc::Logger::msg (LogMessage *message*)

Sends a LogMessage.

Sends a LogMessage.

**Parameters:**

***The*** LogMessage to send.

Referenced by msg(), and Arc::stringto().

#### 5.59.3.7 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.

References msg().

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

- Logger.h

# 5.60 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)

### 5.60.1 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.

### 5.60.2 Constructor & Destructor Documentation

#### 5.60.2.1 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.

#### 5.60.2.2 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.

### 5.60.3 Member Function Documentation

#### 5.60.3.1 LogLevel Arc::LogMessage::getLevel () const

Returns the level of the LogMessage.

Returns the level of the LogMessage.

**Returns:**

The level of the LogMessage.

#### 5.60.3.2 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.

### 5.60.4 Friends And Related Function Documentation

#### 5.60.4.1 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.

#### 5.60.4.2 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:

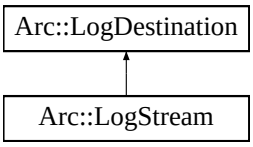
- Logger.h

# 5.61 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)

### 5.61.1 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.

### 5.61.2 Constructor & Destructor Documentation

#### 5.61.2.1 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 erite LogMessages.

#### 5.61.2.2 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.

### 5.61.3 Member Function Documentation

#### 5.61.3.1 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:

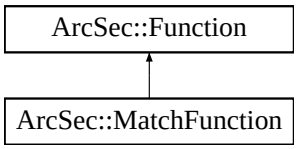
- Logger.h

# 5.62 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

- **MatchFunction** (std::string functionName, std::string argumentType)
- virtual bool **evaluate** (AttributeValue *arg0, AttributeValue *arg1)

## Static Public Member Functions

- static std::string getFunctionName (std::string datatype)

### 5.62.1 Detailed Description

Evaluate whether arg1 (value in regular expression) matched arg0 (lable in regular expression).

### 5.62.2 Member Function Documentation

#### 5.62.2.1 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:

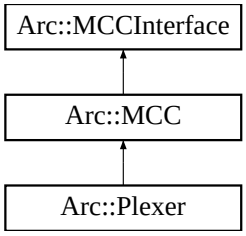
- MatchFunction.h

## 5.63 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 (Arc::Config *)
- virtual void Next (Arc::MCCInterface *next, const std::string &label="")
- virtual void AddSecHandler (Arc::Config *cfg, ArcSec::SecHandler *sechandler, const std::string &label="")
- virtual void Unlink (void)
- virtual Arc::MCC_Status process (Arc::Message &, Arc::Message &)

### Protected Member Functions

- Arc::MCCInterface * **Next** (const std::string &label="")
- bool ProcessSecHandlers (Arc::Message &message, const std::string &label="")

### Protected Attributes

- std::map< std::string, Arc::MCCInterface * > next_
- std::map< std::string, std::list< ArcSec::SecHandler * > > sechandlers_

### Static Protected Attributes

- static Arc::Logger logger

### 5.63.1 Detailed Description

Message Chain Component - base class for every MCC plugin.

This is partialy virtual class which defines interface and common functionality for every MCC plugin needed for managing of component in a chain.

## 5.63.2 Constructor & Destructor Documentation

### 5.63.2.1 Arc::MCC::MCC (Arc::Config *) [inline]

Example contructor - MCC takes at least it's configuration subtree

## 5.63.3 Member Function Documentation

### 5.63.3.1 bool Arc::MCC::ProcessSecHandlers (Arc::Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. Returns true if 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 implemention of MCC.

### 5.63.3.2 virtual void Arc::MCC::Next (Arc::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.

### 5.63.3.3 virtual void Arc::MCC::AddSecHandler (Arc::Config * *cfg*, ArcSec::SecHandler * *sechandler*, const std::string & *label* = "") [virtual]

Add security components/handlers to this MCC. Security handlers are stacked into few queues with each queue identified by it's label. Queue labeled 'incoming' is executed for every 'request' message after message is processes by MCC for service side and before processing on client side. Queue 'outgoing' is run for response message before it is processed by MCC algorithms on service side and after processing on client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

### 5.63.3.4 virtual void Arc::MCC::Unlink (void) [virtual]

Removing all links. Useful for destroying chains.

### 5.63.3.5 virtual Arc::MCC_Status Arc::MCC::process (Arc::Message &, Arc::Message &) [inline, virtual]

Dummy Message processing method. Just a placeholder.

Implements Arc::MCCInterface.

Reimplemented in Arc::Plexer.

### 5.63.4 Member Data Documentation

#### 5.63.4.1 std::map<std::string,Arc::MCCInterface*> Arc::MCC::next_ [protected]

Set of labeled "next" components. Each implemented MCC must call process() metthod of corresponding MCCInterface from this set in own process() method.

#### 5.63.4.2 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::MCC::sechandlers_ [protected]

Set o flabeled 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.

#### 5.63.4.3 Arc::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.

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

- MCC.h

## 5.64 mcc_descriptor Struct Reference

Identifier of Message Chain Componet (MCC) plugin.

`#include <MCCLoader.h>`

### Public Attributes

- const char * **name**
- int **version**
- Arc::MCC *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.64.1 Detailed Description

Identifier of Message Chain Componet (MCC) plugin.

This structure describes one of the MCCs stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the MCC class.

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

- MCCLoader.h

## 5.65 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

### 5.65.1 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.

### 5.65.2 Constructor & Destructor Documentation

#### 5.65.2.1 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.")

### 5.65.3 Member Function Documentation

#### 5.65.3.1 bool Arc::MCC_Status::isOk () const

Is the status kind ok?

This method returns true iff the status kind of this object is STATUS_OK

**Returns:**

true iff kind==STATUS_OK

Referenced by operator bool(), and operator!().

#### 5.65.3.2 StatusKind Arc::MCC_Status::getKind () const

Returns the status kind.

Returns the status kind of this object.

**Returns:**

The status kind of this object.

#### 5.65.3.3 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.

#### 5.65.3.4 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.

#### 5.65.3.5 Arc::MCC_Status::operator std::string () const

Conversion to string.

This operator converts a MCC_Status object to a string.

#### 5.65.3.6 Arc::MCC_Status::operator bool (void) const [inline]

Is the status kind ok?

This method returns true iff the status kind of this object is STATUS_OK

**Returns:**

true iff kind==STATUS_OK

References isOk().

#### 5.65.3.7 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

References isOk().

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

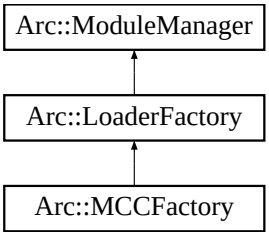
- MCC_Status.h

# 5.66 Arc::MCCFactory Class Reference

MCC Plugins handler.

`#include <MCCFactory.h>`

Inheritance diagram for Arc::MCCFactory::




## Public Member Functions

- MCCFactory (Config *cfg)
- MCC * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- MCC * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- MCC * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

## 5.66.1 Detailed Description

MCC Plugins handler.

This class handles shared libraries containing MCCs

## 5.66.2 Constructor & Destructor Documentation

### 5.66.2.1 Arc::MCCFactory::MCCFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

## 5.66.3 Member Function Documentation

### 5.66.3.1 MCC* Arc::MCCFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of MCC and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created MCC instance.

Reimplemented from Arc::LoaderFactory.

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

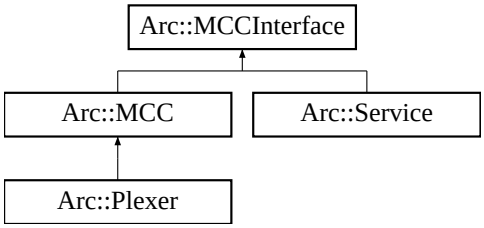
- MCCFactory.h

## 5.67 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 Arc::MCC_Status process (Arc::Message &request, Arc::Message &response)=0

### 5.67.1 Detailed Description

Interface for communication between MCC, Service and Plexer objects.

The Interface is made of method process() which is called by previous MCC in chain. For memory management policies please read description of Message class.

### 5.67.2 Member Function Documentation

#### 5.67.2.1 virtual Arc::MCC_Status Arc::MCCInterface::process (Arc::Message & *request*, Arc::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::Plexer, and Arc::MCC.

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

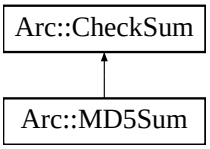
- MCC.h

# 5.68 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

### 5.68.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

## 5.69 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)
- void **Attributes** (MessageAttributes *attr)
- MessageAuth * Auth (void)
- void **Auth** (MessageAuth *auth)
- MessageContext * Context (void)
- MessageAuthContext * AuthContext (void)
- void Context (MessageContext *ctx)
- void AuthContext (MessageAuthContext *auth_ctx)

### 5.69.1 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 explicitely 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 explicitely 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.

## 5.69.2 Constructor & Destructor Documentation

### 5.69.2.1 Arc::Message::Message (void) [inline]

true if auth_ctx_ was created internally Dummy constructor

### 5.69.2.2 Arc::Message::Message (Message & *msg*) [inline]

Copy constructor. Ensures shallow copy.

### 5.69.2.3 Arc::Message::Message (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

### 5.69.2.4 Arc::Message::~Message (void) [inline]

Destructor does not affect refered objects except those created internally

## 5.69.3 Member Function Documentation

### 5.69.3.1 Message& Arc::Message::operator= (Message & *msg*) [inline]

Assignment. Ensures shallow copy.

References attr_, Attributes(), Auth(), auth_, auth_ctx_, AuthContext(), Context(), ctx_, and payload_.

### 5.69.3.2 MessagePayload* Arc::Message::Payload (void) [inline]

Returns pointer to current payload or NULL if no payload assigned.

### 5.69.3.3 MessagePayload* Arc::Message::Payload (MessagePayload * *payload*) [inline]

Replaces payload with new one. Returns the old one.

### 5.69.3.4 MessageAttributes* Arc::Message::Attributes (void) [inline]

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

Referenced by operator=().

### 5.69.3.5 MessageAuth* Arc::Message::Auth (void) [inline]

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

Referenced by operator=().

#### 5.69.3.6 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.

Referenced by operator=().

#### 5.69.3.7 MessageAuthContext* Arc::Message::AuthContext (void) [inline]

Returns a pointer to the current auth* context object or creates it if no object has been assigned.

Referenced by operator=().

#### 5.69.3.8 void Arc::Message::Context (MessageContext * *ctx*) [inline]

Assigns message context object

#### 5.69.3.9 void Arc::Message::AuthContext (MessageAuthContext * *auth_ctx*) [inline]

Assigns auth* context object

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

- Message.h

# 5.70 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_

### 5.70.1 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.

### 5.70.2 Constructor & Destructor Documentation

#### 5.70.2.1 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.

## 5.70.3 Member Function Documentation

### 5.70.3.1 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.

### 5.70.3.2 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.

### 5.70.3.3 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.

### 5.70.3.4 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.

### 5.70.3.5 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.

#### 5.70.3.6 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.

#### 5.70.3.7 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.

#### 5.70.3.8 AttributeIterator Arc::MessageAttributes::getAll (void) const

Access all value and attributes.

### 5.70.4 Member Data Documentation

#### 5.70.4.1 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:

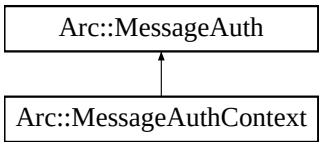
- MessageAttributes.h

## 5.71 Arc::MessageAuth Class Reference

Contains authencity 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 (SecAttr::Format format, XMLNode &val) const
- MessageAuth * Filter (const std::list< std::string > selected_keys, const std::list< std::string > rejected_keys) const

### 5.71.1 Detailed Description

Contains authencity information, authorization tokens and decisions.

This class only supports string keys and SecAttr values.

### 5.71.2 Member Function Documentation

#### 5.71.2.1 void Arc::MessageAuth::set (const std::string & *key*, SecAttr * *value*)

Adds/overwrites security attribute stored under specified key.

#### 5.71.2.2 void Arc::MessageAuth::remove (const std::string & *key*)

Deletes security attribute stored under specified key.

#### 5.71.2.3 SecAttr* Arc::MessageAuth::get (const std::string & *key*)

Retrieves reference to security attribute stored under specified key.

#### 5.71.2.4 SecAttr* Arc::MessageAuth::operator[] (const std::string & *key*) [inline]

Same as MessageAuth::get.

#### 5.71.2.5 bool Arc::MessageAuth::Export (SecAttr::Format *format*, XMLNode & *val*) const

Returns properly catenated attributes in specified format.

#### 5.71.2.6 MessageAuth* Arc::MessageAuth::Filter (const std::list< std::string > *selected_keys*, const std::list< std::string > *rejected_keys*) const

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 transfered 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.

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

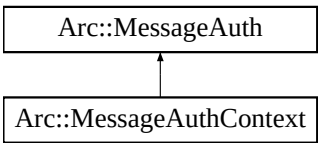
- MessageAuth.h

## 5.72 Arc::MessageAuthContext Class Reference

Handler for content of message auth∗ context.

`#include <Message.h>`

Inheritance diagram for Arc::MessageAuthContext::




### 5.72.1 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

# 5.73 Arc::MessageContext Class Reference

Handler for content of message context.

`#include <Message.h>`

## Public Member Functions

- void Add (const std::string &name, MessageContextElement *element)
- MessageContextElement * **operator[]** (const std::string &id)

## 5.73.1 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.

## 5.73.2 Member Function Documentation

### 5.73.2.1 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

## 5.74 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

`#include <Message.h>`

Inherited by ArcSec::PDPConfigContext.

### 5.74.1 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:

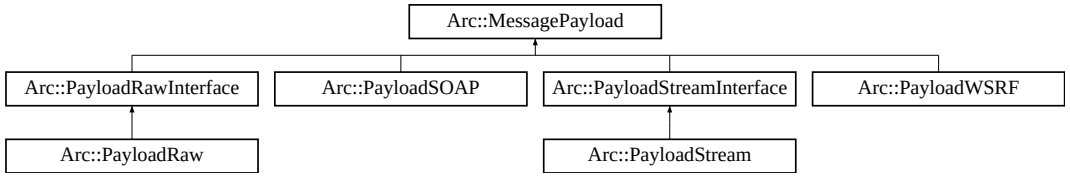
- Message.h

# 5.75 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

`#include <Message.h>`

Inheritance diagram for Arc::MessagePayload::




## 5.75.1 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:

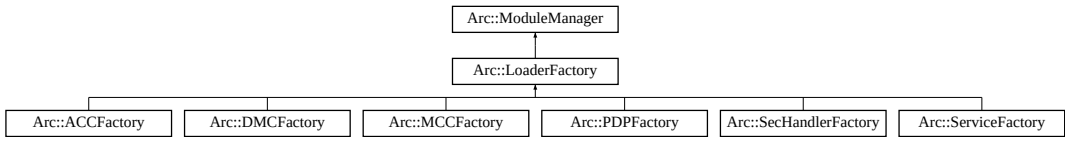
- Message.h

## 5.76 Arc::ModuleManager Class Reference

Manager of shared libraries.

`#include <ModuleManager.h>`

Inheritance diagram for Arc::ModuleManager::




### Public Member Functions

- ModuleManager (Arc::Config *cfg)
- Glib::Module * load (const std::string &name)
- void setCfg (Arc::Config *cfg)

### 5.76.1 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.

### 5.76.2 Constructor & Destructor Documentation

#### 5.76.2.1 Arc::ModuleManager::ModuleManager (Arc::Config * *cfg*)

Cache of handles of loaded modules Constructor. It is supposed to process correponding configuration subtree and tune module loading parameters accordingly. Currently it only sets modulr directory to current one.

### 5.76.3 Member Function Documentation

#### 5.76.3.1 Glib::Module* Arc::ModuleManager::load (const std::string & *name*)

Finds module 'name' in cache or loads corresponding shared library

#### 5.76.3.2 void Arc::ModuleManager::setCfg (Arc::Config * *cfg*)

Input the configuration subtree, and trigger the module loading (do almost the same as the Constructor); It is function desgined for ClassLoader to adopt the singleton pattern

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

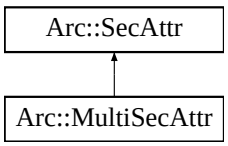
- ModuleManager.h

# 5.77 Arc::MultiSecAttr Class Reference

Container of multiple SecAttr attributes.

`#include <SecAttr.h>`

Inheritance diagram for Arc::MultiSecAttr::




## Public Member Functions

- virtual operator bool ()
- virtual bool Export (Format format, XMLNode &val) const
- virtual bool **Import** (Format format, const XMLNode &val)

## Protected Member Functions

- virtual bool **equal** (const SecAttr &b) const
- virtual bool **Add** (Format format, XMLNode &val)

## Protected Attributes

- std::list< SecAttr * > **attrs_**

### 5.77.1 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.

### 5.77.2 Member Function Documentation

#### 5.77.2.1 virtual Arc::MultiSecAttr::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 from Arc::SecAttr.

#### 5.77.2.2 virtual bool Arc::MultiSecAttr::Export (Format *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.

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

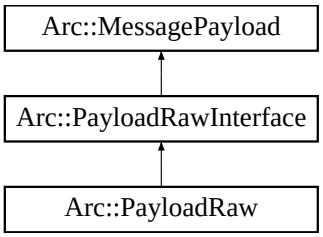
- SecAttr.h

# 5.78 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 char operator[ ] (int pos) const
- virtual char * Content (int pos=-1)
- virtual int Size (void) const
- virtual char * Insert (int pos=0, int size=0)
- virtual char * Insert (const char *s, int pos=0, int size=0)
- virtual char * Buffer (unsigned int num=0)
- virtual int BufferSize (unsigned int num=0) const
- virtual int BufferPos (unsigned int num=0) const
- virtual bool Truncate (unsigned int size)

## Protected Attributes

- int **offset_**
- int **size_**
- std::vector< PayloadRawBuf > **buf_**

### 5.78.1 Detailed Description

Raw byte multi-buffer.

This is implementation of PayloadRawInterface. Buffers are memory blocks logically placed one after another.

### 5.78.2 Constructor & Destructor Documentation

#### 5.78.2.1 Arc::PayloadRaw::PayloadRaw (void) [inline]

List of handled buffers. Constructor. Created object contains no buffers.

#### 5.78.2.2 virtual Arc::PayloadRaw::~PayloadRaw (void) [virtual]

Destructor. Frees allocated buffers.

### 5.78.3 Member Function Documentation

#### 5.78.3.1 virtual char Arc::PayloadRaw::operator[] (int *pos*) const [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.

Implements Arc::PayloadRawInterface.

#### 5.78.3.2 virtual char* Arc::PayloadRaw::Content (int *pos* = -1) [virtual]

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implements Arc::PayloadRawInterface.

#### 5.78.3.3 virtual int Arc::PayloadRaw::Size (void) const [virtual]

Returns logical size of whole structure.

Implements Arc::PayloadRawInterface.

#### 5.78.3.4 virtual char* Arc::PayloadRaw::Insert (int *pos* = 0, int *size* = 0) [virtual]

Create new buffer at global position 'pos' of size 'size'.

Implements Arc::PayloadRawInterface.

#### 5.78.3.5 virtual char* Arc::PayloadRaw::Insert (const char * *s*, int *pos* = 0, int *size* = 0) [virtual]

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is 0 content at 's' is expected to be null-terminated.

Implements Arc::PayloadRawInterface.

#### 5.78.3.6 virtual char* Arc::PayloadRaw::Buffer (unsigned int *num* = 0) [virtual]

Returns pointer to num'th buffer

Implements Arc::PayloadRawInterface.

#### 5.78.3.7 virtual int Arc::PayloadRaw::BufferSize (unsigned int *num* = 0) const [virtual]

Returns length of num'th buffer

Implements Arc::PayloadRawInterface.

#### 5.78.3.8 virtual int Arc::PayloadRaw::BufferPos (unsigned int *num* = 0) const [virtual]

Returns position of num'th buffer

Implements Arc::PayloadRawInterface.

#### 5.78.3.9 virtual bool Arc::PayloadRaw::Truncate (unsigned int *size*) [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.

Implements Arc::PayloadRawInterface.

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

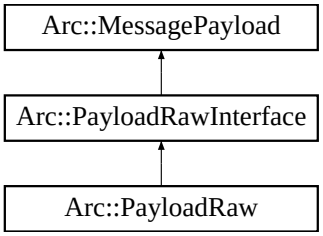
- PayloadRaw.h

## 5.79 Arc::PayloadRawInterface Class Reference

Random Access Payload for Message objects.

`#include <PayloadRaw.h>`

Inheritance diagram for Arc::PayloadRawInterface::




### Public Member Functions

- virtual char operator[ ] (int pos) const =0
- virtual char * Content (int pos=-1)=0
- virtual int Size (void) const =0
- virtual char * Insert (int pos=0, int size=0)=0
- virtual char * Insert (const char *s, int pos=0, int size=0)=0
- virtual char * Buffer (unsigned int num)=0
- virtual int BufferSize (unsigned int num) const =0
- virtual int BufferPos (unsigned int num) const =0
- virtual bool Truncate (unsigned int size)=0

### 5.79.1 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.

### 5.79.2 Member Function Documentation

#### 5.79.2.1 virtual char Arc::PayloadRawInterface::operator[ ] (int *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.

Implemented in Arc::PayloadRaw.

#### 5.79.2.2 virtual char* Arc::PayloadRawInterface::Content (int *pos* = -1) [pure virtual]

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implemented in Arc::PayloadRaw.

#### 5.79.2.3 virtual int Arc::PayloadRawInterface::Size (void) const [pure virtual]

Returns logical size of whole structure.

Implemented in Arc::PayloadRaw.

#### 5.79.2.4 virtual char* Arc::PayloadRawInterface::Insert (int *pos* = 0, int *size* = 0) [pure virtual]

Create new buffer at global position 'pos' of size 'size'.

Implemented in Arc::PayloadRaw.

#### 5.79.2.5 virtual char* Arc::PayloadRawInterface::Insert (const char * *s*, int *pos* = 0, int *size* = 0) [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 0 content at 's' is expected to be null-terminated.

Implemented in Arc::PayloadRaw.

#### 5.79.2.6 virtual char* Arc::PayloadRawInterface::Buffer (unsigned int *num*) [pure virtual]

Returns pointer to num'th buffer

Implemented in Arc::PayloadRaw.

#### 5.79.2.7 virtual int Arc::PayloadRawInterface::BufferSize (unsigned int *num*) const [pure virtual]

Returns length of num'th buffer

Implemented in Arc::PayloadRaw.

#### 5.79.2.8 virtual int Arc::PayloadRawInterface::BufferPos (unsigned int *num*) const [pure virtual]

Returns position of num'th buffer

Implemented in Arc::PayloadRaw.

#### 5.79.2.9 virtual bool Arc::PayloadRawInterface::Truncate (unsigned int *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.

Implemented in Arc::PayloadRaw.

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

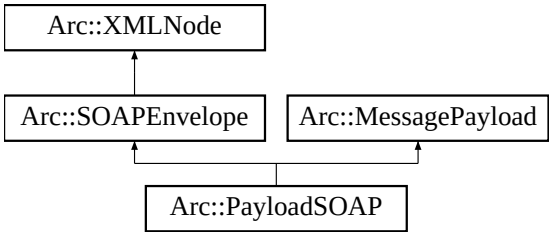
- PayloadRaw.h

## 5.80 Arc::PayloadSOAP Class Reference

Payload of Message with SOAP content.

`#include <PayloadSOAP.h>`

Inheritance diagram for Arc::PayloadSOAP::




### Public Member Functions

- PayloadSOAP (const Arc::NS &ns, bool fault=false)
- PayloadSOAP (const Arc::SOAPEnvelope &soap)
- PayloadSOAP (const Arc::MessagePayload &source)

### 5.80.1 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.

### 5.80.2 Constructor & Destructor Documentation

#### 5.80.2.1 Arc::PayloadSOAP::PayloadSOAP (const Arc::NS & *ns*, bool *fault* = `false`)

Constructor - creates new Message payload

#### 5.80.2.2 Arc::PayloadSOAP::PayloadSOAP (const Arc::SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP document. Provided SOAP document must exist as long as created object exists.

#### 5.80.2.3 Arc::PayloadSOAP::PayloadSOAP (const Arc::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:

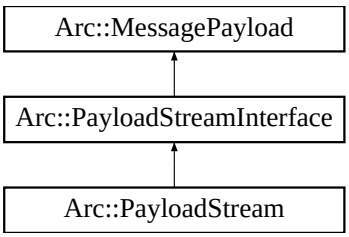
- PayloadSOAP.h

# 5.81 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 char *buf, int size)
- 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 int GetHandle (void)

## Protected Attributes

- int **timeout_**
- int handle_
- bool seekable_

### 5.81.1 Detailed Description

POSIX handle as Payload.

Thsi is an implemetation of PayloadStreamInterface for generic POSIX handle.

### 5.81.2 Constructor & Destructor Documentation

#### 5.81.2.1 Arc::PayloadStream::PayloadStream (int *h* = -1)

true if lseek operation is applicable to open handle Constructor. Attaches to already open handle. Handle is not managed by this class and must be closed by external code.

#### 5.81.2.2 virtual Arc::PayloadStream::~PayloadStream (void) [inline, virtual]

Destructor.

### 5.81.3 Member Function Documentation

#### 5.81.3.1 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.

#### 5.81.3.2 virtual bool Arc::PayloadStream::Get (std::string & *buf*) [virtual]

Read as many as possible (sane amount) of bytes into buf.

Implements Arc::PayloadStreamInterface.

#### 5.81.3.3 virtual std::string Arc::PayloadStream::Get (void) [inline, virtual]

Read as many as possible (sane amount) of bytes.

Implements Arc::PayloadStreamInterface.

#### 5.81.3.4 virtual bool Arc::PayloadStream::Put (const char * *buf*, int *size*) [virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

Referenced by Put().

#### 5.81.3.5 virtual bool Arc::PayloadStream::Put (const std::string & *buf*) [inline, virtual]

Push information from 'buf' into stream. Returns true on success.

Implements Arc::PayloadStreamInterface.

References Put().

#### 5.81.3.6 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.

References Put().

#### 5.81.3.7 virtual Arc::PayloadStream::operator bool (void) [inline, virtual]

Returns true if stream is valid.

Implements Arc::PayloadStreamInterface.

References handle_.

#### 5.81.3.8 virtual bool Arc::PayloadStream::operator! (void) [inline, virtual]

Returns true if stream is invalid.

Implements Arc::PayloadStreamInterface.

References handle_.

#### 5.81.3.9 virtual int Arc::PayloadStream::Timeout (void) const [inline, virtual]

Query current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

#### 5.81.3.10 virtual void Arc::PayloadStream::Timeout (int *to*) [inline, virtual]

Set current timeout for Get() and Put() operations.

Implements Arc::PayloadStreamInterface.

#### 5.81.3.11 virtual int Arc::PayloadStream::GetHandle (void) [inline, virtual]

Returns POSIX handle of the stream. This method is deprecated and will be removed soon. Currently it is only used by Transport Layer Security MCC.

References handle_.

### 5.81.4 Member Data Documentation

#### 5.81.4.1 int Arc::PayloadStream::handle_ [protected]

Timeout for read/write operations

Referenced by GetHandle(), operator bool(), and operator!().

#### 5.81.4.2 bool Arc::PayloadStream::seekable_ [protected]

Handle for operations

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

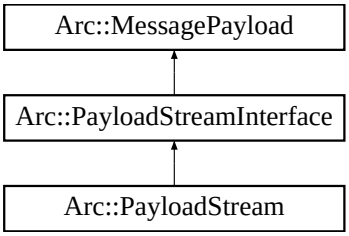
- PayloadStream.h

## 5.82 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, int 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

### 5.82.1 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.

### 5.82.2 Member Function Documentation

#### 5.82.2.1 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.

#### 5.82.2.2 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.

#### 5.82.2.3 virtual std::string Arc::PayloadStreamInterface::Get (void) [pure virtual]

Read as many as possible (sane amount) of bytes.

Implemented in Arc::PayloadStream.

#### 5.82.2.4 virtual bool Arc::PayloadStreamInterface::Put (const char * *buf*, int *size*) [pure virtual]

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implemented in Arc::PayloadStream.

#### 5.82.2.5 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.

#### 5.82.2.6 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.

#### 5.82.2.7 virtual Arc::PayloadStreamInterface::operator bool (void) [pure virtual]

Returns true if stream is valid.

Implemented in Arc::PayloadStream.

#### 5.82.2.8 virtual bool Arc::PayloadStreamInterface::operator! (void) [pure virtual]

Returns true if stream is invalid.

Implemented in Arc::PayloadStream.

#### 5.82.2.9 virtual int Arc::PayloadStreamInterface::Timeout (void) const [pure virtual]

Query current timeout for Get() and Put() operations.

Implemented in Arc::PayloadStream.

#### 5.82.2.10 virtual void Arc::PayloadStreamInterface::Timeout (int *to*) [pure virtual]

Set current timeout for Get() and Put() operations.

Implemented in Arc::PayloadStream.

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

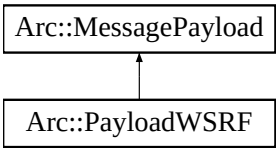
- PayloadStream.h

## 5.83 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)
- **operator WSRF &** (void)
- **operator bool** (void)

### Protected Attributes

- WSRF & **wsrf_**
- bool **owner_**

### 5.83.1 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.

### 5.83.2 Constructor & Destructor Documentation

#### 5.83.2.1 Arc::PayloadWSRF::PayloadWSRF (const SOAPEnvelope & *soap*)

Constructor - creates Message payload from SOAP message. Returns invalid WSRF if SOAP does not represent WS-ResourceProperties

#### 5.83.2.2 Arc::PayloadWSRF::PayloadWSRF (WSRF & *wsrp*)

Constructor - creates Message payload with acquired WSRF message. WSRF message will be destroyed by destructor of this object.

#### 5.83.2.3 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

## 5.84 ArcSec::PDP Class Reference

Base class for Policy Decisoion Point plugins.

`#include <PDP.h>`

### Public Member Functions

- **PDP** (Arc::Config *cfg)
- virtual bool **isPermitted** (Arc::Message *msg)=0
- void **SetId** (std::string &id)
- std::string **GetId** ()

### Protected Attributes

- std::string **id_**

### Static Protected Attributes

- static Arc::Logger **logger**

### 5.84.1 Detailed Description

Base class for Policy Decisoion 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

# 5.85 pdp_descriptor Struct Reference

Identifier of Policy Decision Point (PDP) plugin.

`#include <PDPLoader.h>`

## Public Attributes

- const char * **name**
- int **version**
- ArcSec::PDP *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.85.1 Detailed Description

Identifier of Policy Decision Point (PDP) plugin.

This structure describes one of the PDPs stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the PDP class.

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

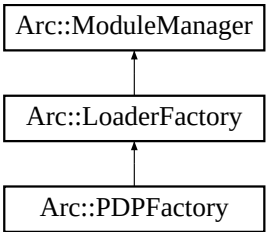
- PDPLoader.h

## 5.86 Arc::PDPFactory Class Reference

PDP Plugins handler.

`#include <PDPFactory.h>`

Inheritance diagram for Arc::PDPFactory::




### Public Member Functions

- PDPFactory (Config *cfg)
- ArcSec::PDP * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- ArcSec::PDP * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- ArcSec::PDP * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

### 5.86.1 Detailed Description

PDP Plugins handler.

This class handles shared libraries containing PDPs

### 5.86.2 Constructor & Destructor Documentation

#### 5.86.2.1 Arc::PDPFactory::PDPFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

### 5.86.3 Member Function Documentation

#### 5.86.3.1 ArcSec::PDP* Arc::PDPFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of PDP and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created PDP instance.

Reimplemented from Arc::LoaderFactory.

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

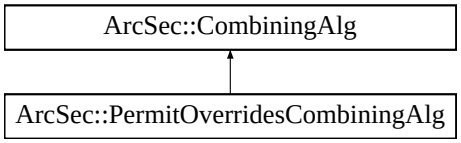
- PDPFactory.h

# 5.87 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 std::string & **getalgId** (void)

## Static Public Member Functions

- static const std::string & **Identifier** (void)

## 5.87.1 Detailed Description

Implement the "Permit-Overrides" algorithm.

## 5.87.2 Member Function Documentation

### 5.87.2.1 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

Implements ArcSec::CombiningAlg.

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

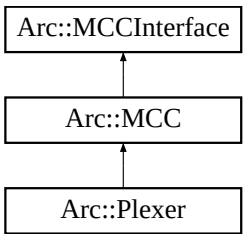
- PermitOverridesAlg.h

## 5.88 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)
- virtual ~Plexer ()
- virtual void Next (MCCInterface *next, const std::string &label)
- virtual MCC_Status process (Message &request, Message &response)

### Static Public Attributes

- static Arc::Logger logger

### 5.88.1 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.

### 5.88.2 Constructor & Destructor Documentation

#### 5.88.2.1 Arc::Plexer::Plexer (Config * *cfg*)

The constructor.

This is the constructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

#### 5.88.2.2 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.

## 5.88.3 Member Function Documentation

### 5.88.3.1 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.

### 5.88.3.2 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.

## 5.88.4 Member Data Documentation

### 5.88.4.1 Arc::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

## 5.89 Arc::PlexerEntry Class Reference

A pair of label (regex) and pointer to service.

`#include <Plexer.h>`

### Friends

- class **Plexer**

### 5.89.1 Detailed Description

A pair of label (regex) and pointer to service.

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

# 5.90 ArcSec::Policy Class Reference

Base class for Policy, PolicySet, or Rule.

`#include <Policy.h>`

## Public Member Functions

- **Policy** (Arc::XMLNode &)
- virtual MatchResult match (EvaluationCtx *ctx)=0
- virtual Result eval (EvaluationCtx *ctx)=0
- virtual void addPolicy (Policy *pl)
- virtual std::string getEffect ()=0
- virtual EvalResult & getEvalResult ()=0

## Protected Attributes

- std::list< Policy * > **subelements**

## Static Protected Attributes

- static Arc::Logger **logger**

### 5.90.1 Detailed Description

Base class for Policy, PolicySet, or Rule.

### 5.90.2 Member Function Documentation

#### 5.90.2.1 virtual MatchResult ArcSec::Policy::match (EvaluationCtx * *ctx*) [pure virtual]

Evaluate whether the two targets to be evaluated match to each other.

As an example for illustration, for the ArcRule, the rule is like this: <Rule ruleid="rule2" effect="Deny"> <Subjects> <Subject type="string">/O=Grid/OU=KnowARC/CN=ANONYMOS</Subject> <Subject type="string">/vo.knowarc/usergroupB</Subject> </Subjects> <Resources type="string"> <Resource>localhost:/home/atlas/</Resource> <Resource>nordugrid.org:/home/atlas/</Resource> </Resources> <Actions type="string"> <Action>read</Action> </Actions> <Conditions> </Rule> the match(ctx) method will check whether the Request (with Arc request schema) satisfies the <Subjects, Resources, Actions, Conditions> tuple.

for the XACML rule, Rule is like this: <Rule ruleid="urn:oasis:names:tc:xacml:2.0:example:ruleid:2" effect="Permit"> <Target> <Resources> <Resource> <ResourceMatch matchid="urn:oasis:names:tc:xacml:1.0:function:string-equal"> <AttributeValue datatype="http://www.w3.org/2001/XMLSchema#string">urn:med:example:schemas:record</AttributeValue> <ResourceAttributeDesignator attributeid="urn:oasis:names:tc:xacml:2.0:resource:target-namespace" datatype="http://www.w3.org/2001/XMLSchema#string"> </ResourceMatch> </Resource> </Resources> <Actions> <Action> <ActionMatch matchid="urn:oasis:names:tc:xacml:1.0:function:string-equal"> <AttributeValue datatype="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>

<ActionAttributeDesignator attributeid="urn:oasis:names:tc:xacml:1.0:action:action-id" datatype="http://www.w3.org/2001/XMLSchema#string"> </ActionMatch> </Action> </Actions> </Target> <Condition> <Apply functionid="urn:oasis:names:tc:xacml:1.0:function:and"> <Apply functionid="urn:oasis:names:tc:xacml:1.0:function:string-equal"> <Apply functionid="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"> <SubjectAttributeDesignator attributeid="urn:oasis:names:tc:xacml:2.0:example:attribute:parent-guardian-id" datatype="http://www.w3.org/2001/XMLSchema#string"> </Apply> <Apply functionid="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only"> <AttributeSelector requestcontextpath="//md:record/md:parentGuardian/md:parentGuardianId/text()" datatype="http://www.w3.org/2001/XMLSchema#string"> </Apply> </Apply> <VariableReference variableid="17590035"> </Apply> </Condition> </Rule> the match(ctx) method will check whether the Request (with XAMCL request schema) satisfies the <Target> tuple (which include <Subjects, Resources, Actions>)

#### 5.90.2.2 virtual Result ArcSec::Policy::eval (EvaluationCtx * *ctx*) [pure virtual]

Evaluete policy.

#### 5.90.2.3 virtual void ArcSec::Policy::addPolicy (Policy * *pl*) [inline, virtual]

Add a policy element to into "this" object.

#### 5.90.2.4 virtual std::string ArcSec::Policy::getEffect () [pure virtual]

Get the "Effect" attribute.

#### 5.90.2.5 virtual EvalResult& ArcSec::Policy::getEvalResult () [pure virtual]

Get eveluation result.

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

- Policy.h

# 5.91 Arc::RegularExpression Class Reference

A regular expression class.

`#include <ArcRegex.h>`

## Public Member Functions

- RegularExpression ()
- RegularExpression (std::string pattern)
- RegularExpression (const RegularExpression &regex)
- ~RegularExpression ()
- const 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 ()

### 5.91.1 Detailed Description

A regular expression class.

This class is a wrapper around the functions provided in regex.h.

### 5.91.2 Constructor & Destructor Documentation

#### 5.91.2.1 Arc::RegularExpression::RegularExpression () [inline]

default constructor

#### 5.91.2.2 Arc::RegularExpression::RegularExpression (std::string *pattern*)

Creates a reges from a pattern string.

#### 5.91.2.3 Arc::RegularExpression::RegularExpression (const RegularExpression & *regex*)

Copy constructor.

#### 5.91.2.4 Arc::RegularExpression::~RegularExpression ()

Destructor.

### 5.91.3 Member Function Documentation

#### 5.91.3.1 const RegularExpression& Arc::RegularExpression::operator= (const RegularExpression & *regex*)

Assignment operator.

#### 5.91.3.2 bool Arc::RegularExpression::isOk ()

Returns true if the pattern of this regex is ok.

#### 5.91.3.3 bool Arc::RegularExpression::hasPattern (std::string *str*)

Returns true if this regex has the pattern provided.

#### 5.91.3.4 bool Arc::RegularExpression::match (const std::string & *str*) const

Returns true if this regex matches whole string provided.

#### 5.91.3.5 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'.

#### 5.91.3.6 std::string Arc::RegularExpression::getPattern ()

Returns patter.

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

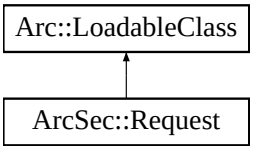
- ArcRegex.h

# 5.92 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 =0
- virtual void setRequestItems (ReqItemList sl)=0
- virtual void addRequestItem (Attrs &sub, Attrs &res, Attrs &act, Attrs &ctx)=0
- virtual void setAttributeFactory (AttributeFactory *attributefactory)=0
- virtual void make_request ()=0
- Request ()
- Request (const Arc::XMLNode *)

## Protected Member Functions

- Request (const char *)

## Protected Attributes

- ReqItemList **rlist**

### 5.92.1 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 informtation, see the example configuration below: <Service name="pdp.service" id="pdp_service"> <pdp:PDPConfig> <.....> <pdp:Request name="arc.request" /> <.....> </pdp:PDPConfig> </Service>

There can be different types of subclass which inherit Request, such like XACMLRequest, ArcRequest, GACLRequest

### 5.92.2 Constructor & Destructor Documentation

#### 5.92.2.1 ArcSec::Request::Request () `[inline]`

Default constructor

#### 5.92.2.2 ArcSec::Request::Request (const Arc::XMLNode *) [inline]

Constructor: Parse request information from a xml stucture in memory

#### 5.92.2.3 ArcSec::Request::Request (const char *) [inline, protected]

Constructor: Parse request information from a input file, internal used only

### 5.92.3 Member Function Documentation

#### 5.92.3.1 virtual ReqItemList ArcSec::Request::getRequestItems () const [pure virtual]

Get all the RequestItem inside RequestItem container

#### 5.92.3.2 virtual void ArcSec::Request::setRequestItems (ReqItemList *sl*) [pure virtual]

Set the content of the container

#### 5.92.3.3 virtual void ArcSec::Request::addRequestItem (Attrs & *sub*, Attrs & *res*, Attrs & *act*, Attrs & *ctx*) [pure virtual]

Add request tuple from non-XMLNode

#### 5.92.3.4 virtual void ArcSec::Request::setAttributeFactory (AttributeFactory * *attributefactory*) [pure virtual]

Set the attribute factory for the usage of Request

#### 5.92.3.5 virtual void ArcSec::Request::make_request () [pure virtual]

Create the objects included in Request according to the node attached to the Request object

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

- Request.h

# 5.93 ArcSec::RequestAttribute Class Reference

Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml.

`#include <RequestAttribute.h>`

## Public Member Functions

- RequestAttribute (Arc::XMLNode &node, AttributeFactory *attrfactory)
- Arc::XMLNode **getNode** ()
- std::string **getAttributeId** () const
- void **setAttributeId** (const std::string attributeId)
- std::string **getDataType** () const
- void **setDataType** (const std::string dataType)
- std::string **getIssuer** () const
- void **setIssuer** (const std::string issuer)
- virtual AttributeValue * **getAttributeValue** () const
- virtual AttributeFactory * **getAttributeFactory** () const
- RequestAttribute & duplicate (RequestAttribute &)

### 5.93.1 Detailed Description

Wrapper which includes AttributeValue object which is generated according to date type of one spefic node in Request.xml.

### 5.93.2 Constructor & Destructor Documentation

#### 5.93.2.1 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>

### 5.93.3 Member Function Documentation

#### 5.93.3.1 RequestAttribute& ArcSec::RequestAttribute::duplicate (RequestAttribute &)

Duplicate the parameter into "this"

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

- RequestAttribute.h

## 5.94 ArcSec::RequestItem Class Reference

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

`#include <RequestItem.h>`

### Public Member Functions

- RequestItem (Arc::XMLNode &, AttributeFactory *)
- virtual SubList **getSubjects** () const =0
- virtual void **setSubjects** (const SubList &sl)=0
- virtual ResList **getResources** () const =0
- virtual void **setResources** (const ResList &rl)=0
- virtual ActList **getActions** () const =0
- virtual void **setActions** (const ActList &al)=0
- virtual CtxList **getContexts** () const =0
- virtual void **setContexts** (const CtxList &ctx)=0

### Protected Attributes

- SubList **subjects**
- ResList **actions**
- ActList **resources**
- CtxList **contexts**

### 5.94.1 Detailed Description

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

### 5.94.2 Constructor & Destructor Documentation

#### 5.94.2.1 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

# 5.95 ArcSec::RequestTuple Class Reference

RequestTuple, container which includes the.

`#include <EvaluationCtx.h>`

## Public Member Functions

- RequestTuple & **duplicate** (const RequestTuple &)
- Arc::XMLNode & **getNode** ()
- void **erase** ()

## Public Attributes

- Subject **sub**
- Resource **res**
- Action **act**
- Context **ctx**

### 5.95.1 Detailed Description

RequestTuple, container which includes the.

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

- EvaluationCtx.h

## 5.96 ArcSec::Response Class Reference

Container for the evaluation results.

`#include <Response.h>`

### Public Member Functions

- virtual ResponseList & **getResponseItems** ()
- virtual void **setResponseItems** (const ResponseList &rl)
- virtual void **addResponseItem** (ResponseItem *respitem)

### Protected Attributes

- ResponseList **rlist**

### 5.96.1 Detailed Description

Container for the evaluation results.

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

- Response.h

# 5.97 ArcSec::ResponseItem Struct Reference

Evaluation result concerning one RequestTuple.

`#include <Response.h>`

## Public Attributes

- RequestTuple * **reqtp**
- Arc::XMLNode **reqxml**
- Policies **pls**
- std::list< Arc::XMLNode > **plsxml**

## 5.97.1 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 struct was generated from the following file:

- Response.h

## 5.98 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)
- 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 **AssignInitializer** (void(*initializer_func)(void *), void *initializer_arg)
- void **AssignKicker** (void(*kicker_func)(void *), void *kicker_arg)
- void AssignWorkingDirectory (std::string &wd)
- void Kill (int timeout)

### Protected Member Functions

- bool **stdout_handler** (Glib::IOCondition cond)
- bool **stderr_handler** (Glib::IOCondition cond)
- bool **stdin_handler** (Glib::IOCondition cond)
- void **child_handler** (Glib::Pid pid, int result)

### Protected Attributes

- std::string **working_directory**
- int **stdout_**
- int **stderr_**
- int **stdin_**
- std::string * **stdout_str_**
- std::string * **stderr_str_**
- std::string * **stdin_str_**

- bool **stdout_keep_**
- bool **stderr_keep_**
- bool **stdin_keep_**
- sigc::connection **stdout_conn_**
- sigc::connection **stderr_conn_**
- sigc::connection **stdin_conn_**
- sigc::connection **child_conn_**
- Glib::Pid **pid_**
- Glib::ArrayHandle< std::string > **argv_**
- void(* **initializer_func_** )(void *)
- void * **initializer_arg_**
- void(* **kicker_func_** )(void *)
- void * **kicker_arg_**
- bool **started_**
- bool **running_**
- int **result_**
- Glib::Mutex **lock_**
- Glib::Cond **cond_**

## Friends

- class **RunPump**

### 5.98.1 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.

### 5.98.2 Constructor & Destructor Documentation

#### 5.98.2.1 Arc::Run::Run (const std::string & *cmdline*)

Constructor preapres object to run cmdline

#### 5.98.2.2 Arc::Run::Run (const std::list< std::string > & *argv*)

Constructor preapres object to run executable and arguments specified in argv

#### 5.98.2.3 Arc::Run::~Run (void)

Destructor kill running executable and releases associated resources

### 5.98.3 Member Function Documentation

#### 5.98.3.1 Arc::Run::operator bool (void) [inline]

Returns true if object is valid

#### 5.98.3.2 bool Arc::Run::operator! (void) [inline]

Returns true if object is invalid

#### 5.98.3.3 bool Arc::Run::Start (void)

Starts running executable. This method may be called only once.

#### 5.98.3.4 bool Arc::Run::Wait (int *timeout*)

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

#### 5.98.3.5 bool Arc::Run::Wait (void)

Wait till execution finished

#### 5.98.3.6 int Arc::Run::Result (void) [inline]

Returns exit code of execution.

#### 5.98.3.7 bool Arc::Run::Running (void)

Return true if execution is going on.

#### 5.98.3.8 int Arc::Run::ReadStdout (int *timeout*, char * *buf*, int *size*)

Read from stdout handle of running executable. This method may be used while stdout is directed to string. But result is unpredictable.

#### 5.98.3.9 int Arc::Run::ReadStderr (int *timeout*, char * *buf*, int *size*)

Read from stderr handle of running executable. This method may be used while stderr is directed to string. But result is unpredictable.

#### 5.98.3.10 int Arc::Run::WriteStdin (int *timeout*, const char * *buf*, int *size*)

Write to stdin handle of running executable. This method may be used while stdin is directed to string. But result is unpredictable.

#### 5.98.3.11 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.

#### 5.98.3.12 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.

#### 5.98.3.13 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.

#### 5.98.3.14 void Arc::Run::KeepStdout (bool *keep* = `true`)

Keep stdout same as parent's if keep = true

#### 5.98.3.15 void Arc::Run::KeepStderr (bool *keep* = `true`)

Keep stderr same as parent's if keep = true

#### 5.98.3.16 void Arc::Run::KeepStdin (bool *keep* = `true`)

Keep stdin same as parent's if keep = true

#### 5.98.3.17 void Arc::Run::CloseStdout (void)

Closes pipe associated with stdout handle

#### 5.98.3.18 void Arc::Run::CloseStderr (void)

Closes pipe associated with stderr handle

#### 5.98.3.19 void Arc::Run::CloseStdin (void)

Closes pipe associated with stdin handle

#### 5.98.3.20 void Arc::Run::AssignWorkingDirectory (std::string & *wd*) `[inline]`

Assign working direcotry of the running process

#### 5.98.3.21 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. Curently this method does not work for Windows OS

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

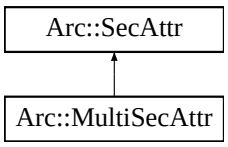
- Run.h

## 5.99 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 ()
- virtual bool Export (Format format, std::string &val) const
- virtual bool Export (Format format, XMLNode &val) const
- virtual bool Import (Format format, const std::string &val)
- virtual bool **Import** (Format format, const XMLNode &val)

### Static Public Attributes

- static Format **UNDEFINED**
- static Format ARCAuth
- static Format XACML
- static Format SAML

### Protected Member Functions

- virtual bool **equal** (const SecAttr &b) const

### Classes

- class Format

  *Export/import format.*

### 5.99.1 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.

## 5.99.2 Constructor & Destructor Documentation

### 5.99.2.1 Arc::SecAttr::SecAttr () [inline]

suitable for inclusion into SAML structures

## 5.99.3 Member Function Documentation

### 5.99.3.1 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.

### 5.99.3.2 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.

### 5.99.3.3 virtual Arc::SecAttr::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::MultiSecAttr.

### 5.99.3.4 virtual bool Arc::SecAttr::Export (Format *format*, std::string & *val*) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute.

### 5.99.3.5 virtual bool Arc::SecAttr::Export (Format *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.

### 5.99.3.6 virtual bool Arc::SecAttr::Import (Format *format*, const std::string & *val*) [virtual]

Fills internal structure from external object of specified format. Retrns 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.

### 5.99.4 Member Data Documentation

#### 5.99.4.1 Format Arc::SecAttr::ARCAuth [static]

own serialization/deserialization format

#### 5.99.4.2 Format Arc::SecAttr::XACML [static]

representation for ARC authorization policy

#### 5.99.4.3 Format Arc::SecAttr::SAML [static]

represenation for XACML policy

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

- SecAttr.h

# 5.100 Arc::SecAttr::Format Class Reference

Export/import format.

`#include <SecAttr.h>`

## Public Member Functions

- **Format** (const Format &format)
- **Format** (const char *format="")
- Format **operator=** (Format format)
- Format **operator=** (const char *format)
- bool **operator==** (Format format)
- bool **operator==** (const char *format)
- bool **operator!=** (Format format)
- bool **operator!=** (const char *format)

### 5.100.1 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:

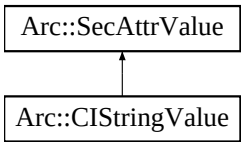
- SecAttr.h

## 5.101 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 ()

### Protected Member Functions

- virtual bool **equal** (SecAttrValue &b)

### 5.101.1 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.

### 5.101.2 Member Function Documentation

#### 5.101.2.1 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.

#### 5.101.2.2 bool Arc::SecAttrValue::operator!= (SecAttrValue & *b*)

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

#### 5.101.2.3 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::CIStringValue.

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

- SecAttrValue.h

## 5.102 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

`#include <SecHandler.h>`

### Public Member Functions

- **SecHandler** (Arc::Config *)
- virtual bool **Handle** (Arc::Message *msg)=0

### Static Protected Attributes

- static Arc::Logger **logger**

### 5.102.1 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 abd are called on incoming and outgoing messages in various MCC and Service plugins. Return value of Handle() defines either processing should continie (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

# 5.103 sechandler_descriptor Struct Reference

Identifier of SecHandler plugin.

`#include <SecHandlerLoader.h>`

## Public Attributes

- const char * **name**
- int **version**
- ArcSec::SecHandler *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

## 5.103.1 Detailed Description

Identifier of SecHandler plugin.

This structure describes one of the SecHandlers stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the SecHandler class.

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

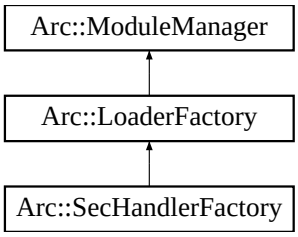
- SecHandlerLoader.h

## 5.104 Arc::SecHandlerFactory Class Reference

SecHandler Plugins handler.

`#include <SecHandlerFactory.h>`

Inheritance diagram for Arc::SecHandlerFactory::




### Public Member Functions

- SecHandlerFactory (Config *cfg)
- ArcSec::SecHandler * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- ArcSec::SecHandler * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- ArcSec::SecHandler * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

### 5.104.1 Detailed Description

SecHandler Plugins handler.

This class handles shared libraries containing SecHandlers

### 5.104.2 Constructor & Destructor Documentation

#### 5.104.2.1 Arc::SecHandlerFactory::SecHandlerFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

### 5.104.3 Member Function Documentation

#### 5.104.3.1 ArcSec::SecHandler* Arc::SecHandlerFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of SecHandler and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created SecHandler instance.

Reimplemented from Arc::LoaderFactory.

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

- SecHandlerFactory.h

# 5.105 ArcSec::Security Class Reference

Common stuff used by security related slasses.

`#include <Security.h>`

## Friends

- class **SecHandler**
- class **PDP**

### 5.105.1 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:

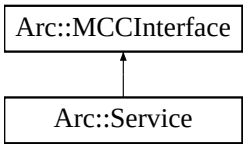
- Security.h

## 5.106 Arc::Service Class Reference

Service - last component in a Message Chain.

`#include <Service.h>`

Inheritance diagram for Arc::Service::




### Public Member Functions

- Service (Arc::Config *)
- virtual void AddSecHandler (Arc::Config *cfg, ArcSec::SecHandler *sechandler, const std::string &label="")

### Protected Member Functions

- bool ProcessSecHandlers (Arc::Message &message, const std::string &label="")

### Protected Attributes

- std::map< std::string, std::list< ArcSec::SecHandler * > > sechandlers_

### Static Protected Attributes

- static Logger **logger**

### 5.106.1 Detailed Description

Service - last component in a Message Chain.

This is virtual class which defines interface (in a future also 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 utilites. 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 by linked to SOAP MCC it must accept and generate messages with PayloadSOAP payload. Method process() of class derived from Service class may be called concurently 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 couterpart of corresponding service is undefined yet. For example see /src/tests/echo/test.cpp .

## 5.106.2 Constructor & Destructor Documentation

### 5.106.2.1 Arc::Service::Service (Arc::Config *) [inline]

Example contructor - Server takes at least it's configuration subtree

## 5.106.3 Member Function Documentation

### 5.106.3.1 bool Arc::Service::ProcessSecHandlers (Arc::Message & *message*, const std::string & *label* = "") [protected]

Executes security handlers of specified queue. For more information please see description of MCC::ProcessSecHandlers

### 5.106.3.2 virtual void Arc::Service::AddSecHandler (Arc::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

## 5.106.4 Member Data Documentation

### 5.106.4.1 std::map<std::string,std::list<ArcSec::SecHandler*> > Arc::Service::sechandlers_ [protected]

Set of labeled 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.

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

- Service.h

## 5.107 service_descriptor Struct Reference

Identifier of Service plugin.

`#include <ServiceLoader.h>`

### Public Attributes

- const char * **name**
- int **version**
- Arc::Service *(* **get_instance** )(Arc::Config *cfg, Arc::ChainContext *ctx)

### 5.107.1 Detailed Description

Identifier of Service plugin.

This structure describes one of the Services stored in a shared library. It contains name of plugin, version number and pointer to function which creates an instance of an object inherited from the Service class.

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

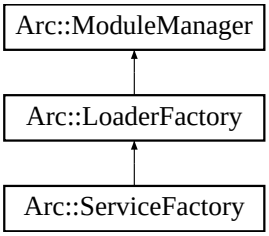
- ServiceLoader.h

# 5.108 Arc::ServiceFactory Class Reference

Service Plugins handler.

`#include <ServiceFactory.h>`

Inheritance diagram for Arc::ServiceFactory::




## Public Member Functions

- ServiceFactory (Config *cfg)
- Service * get_instance (const std::string &name, Config *cfg, ChainContext *ctx)
- Service * **get_instance** (const std::string &name, int version, Config *cfg, ChainContext *ctx)
- Service * **get_instance** (const std::string &name, int min_version, int max_version, Config *cfg, ChainContext *ctx)

### 5.108.1 Detailed Description

Service Plugins handler.

This class handles shared libraries containing Services

### 5.108.2 Constructor & Destructor Documentation

#### 5.108.2.1 Arc::ServiceFactory::ServiceFactory (Config * *cfg*)

Constructor - accepts configuration (not yet used) meant to tune loading of module.

### 5.108.3 Member Function Documentation

#### 5.108.3.1 Service* Arc::ServiceFactory::get_instance (const std::string & *name*, Config * *cfg*, ChainContext * *ctx*)

These methods load shared library named lib'name', locate symbol representing descriptor of Service and calls it's constructor function. Supplied configuration tree is passed to constructor. Returns created Service instance.

Reimplemented from Arc::LoaderFactory.

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

- ServiceFactory.h

## 5.109 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)

### 5.109.1 Detailed Description

Simple triggered condition.

Provides condition and semaphor objects in one element.

### 5.109.2 Member Function Documentation

#### 5.109.2.1 void Arc::SimpleCondition::lock (void) [inline]

Acquire semaphor

#### 5.109.2.2 void Arc::SimpleCondition::unlock (void) [inline]

Release semaphor

#### 5.109.2.3 void Arc::SimpleCondition::signal (void) [inline]

Signal about condition

#### 5.109.2.4 void Arc::SimpleCondition::signal_nonblock (void) [inline]

Signal about condition without using semaphor

#### 5.109.2.5 void Arc::SimpleCondition::broadcast (void) [inline]

Signal about condition to all waiting threads

#### 5.109.2.6 void Arc::SimpleCondition::wait (void) [inline]

Wait for condition

#### 5.109.2.7 void Arc::SimpleCondition::wait_nonblock (void) [inline]

Wait for condition without using semaphor

#### 5.109.2.8 bool Arc::SimpleCondition::wait (int *t*) [inline]

Wait for condition no longer than t milliseconds

#### 5.109.2.9 void Arc::SimpleCondition::reset (void) [inline]

Reset object to initial state

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

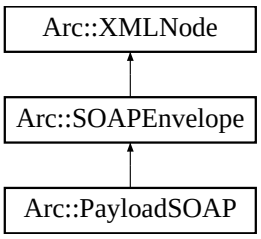
- Thread.h

## 5.110 Arc::SOAPEnvelope Class Reference

Extends XMLNode class to support structures of SOAP message.

`#include <SOAPEnvelope.h>`

Inheritance diagram for Arc::SOAPEnvelope::




### Public Types

- enum **SOAPVersion** { **Version_1_1**, **Version_1_2** }

### Public Member Functions

- SOAPEnvelope (const std::string &xml)
- SOAPEnvelope (const char *xml, int len=-1)
- SOAPEnvelope (const NS &ns, bool fault=false)
- SOAPEnvelope (XMLNode root)
- SOAPEnvelope (const SOAPEnvelope &soap)
- SOAPEnvelope * New (void)
- void Namespaces (const NS &namespaces)
- void GetXML (std::string &out_xml_str, bool user_friendly=false) const
- XMLNode Header (void)
- bool IsFault (void)
- SOAPFault * Fault (void)
- SOAPEnvelope & operator= (const SOAPEnvelope &soap)
- SOAPVersion **Version** (void)

### 5.110.1 Detailed Description

Extends XMLNode class to support structures of SOAP message.

All XMLNode methods are exposed by inheriting from XMLNode and node itself is translated into Envelope part of SOAP.

### 5.110.2 Constructor & Destructor Documentation

#### 5.110.2.1 Arc::SOAPEnvelope::SOAPEnvelope (const std::string & *xml*)

Create new SOAP message from textual representation of XML document. Created XML structure is owned by this instance. This constructor also sets default namespaces to default prefixes as specified below.

#### 5.110.2.2 Arc::SOAPEnvelope::SOAPEnvelope (const char * *xml*, int *len* = -1)

Same as previous

#### 5.110.2.3 Arc::SOAPEnvelope::SOAPEnvelope (const NS & *ns*, bool *fault* = false)

Create new SOAP message with specified namespaces. Created XML structure is owned by this instance. If argument fault is set to true created message is fault.

#### 5.110.2.4 Arc::SOAPEnvelope::SOAPEnvelope (XMLNode *root*)

Acquire XML document as SOAP message. Created XML structure is NOT owned by this instance.

#### 5.110.2.5 Arc::SOAPEnvelope::SOAPEnvelope (const SOAPEnvelope & *soap*)

Create a copy of another SOAPEnvelope object.

### 5.110.3 Member Function Documentation

#### 5.110.3.1 SOAPEnvelope* Arc::SOAPEnvelope::New (void)

Creates complete copy of SOAP. Do not use New() method of XMLNode - use this one.

#### 5.110.3.2 void Arc::SOAPEnvelope::Namespaces (const NS & *namespaces*)

Modify assigned namespaces. Default namespaces and prefixes are soap-enc http://schemas.xmlsoap.org/soap/encoding/ soap-env http://schemas.xmlsoap.org/soap/envelope/ xsi http://www.w3.org/2001/XMLSchema-instance xsd http://www.w3.org/2001/XMLSchema

Reimplemented from Arc::XMLNode.

#### 5.110.3.3 void Arc::SOAPEnvelope::GetXML (std::string & *out_xml_str*, bool *user_friendly* = false) const

Fills argument with this instance XML subtree textual representation

Reimplemented from Arc::XMLNode.

#### 5.110.3.4 XMLNode Arc::SOAPEnvelope::Header (void) [inline]

Get SOAP header as XML node

#### 5.110.3.5 bool Arc::SOAPEnvelope::IsFault (void) [inline]

Returns true if message is Fault

#### 5.110.3.6 SOAPFault* Arc::SOAPEnvelope::Fault (void) [inline]

Get Fault part of message. Returns NULL if message is not Fault.

#### 5.110.3.7 SOAPEnvelope& Arc::SOAPEnvelope::operator= (const SOAPEnvelope & *soap*)

Makes this object a copy of another SOAPEnvelope object.

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

- SOAPEnvelope.h

# 5.111 Arc::SOAPFault Class Reference

Interface to SOAP Fault message.

`#include <SOAPEnvelope.h>`

## Public Types

- enum SOAPFaultCode {
  **undefined**, **unknown**, **VersionMismatch**, **MustUnderstand**,
  **Sender**, **Receiver**, **DataEncodingUnknown** }

## Public Member Functions

- SOAPFault (XMLNode &body)
- operator bool (void)
- SOAPFaultCode Code (void)
- void Code (SOAPFaultCode code)
- std::string Subcode (int level)
- void Subcode (int level, const char *subcode)
- std::string Reason (int num=0)
- void Reason (int num, const char *reason)
- void Reason (const char *reason)
- std::string Node (void)
- void Node (const char *node)
- std::string Role (void)
- void Role (const char *role)
- XMLNode Detail (bool create=false)

## Friends

- class **SOAPEnvelope**

### 5.111.1 Detailed Description

Interface to SOAP Fault message.

SOAPFault class provides a convenience interface for accessing elements of SOAP faults. It also tries to expose single interface for both version 1.0 and 1.2 faults. This class is not intended to 'own' any information stored. It's purpose is to manipulate information which is kept under control of XMLNode or SOAPEnvelope classes. If instance does not refer to valid SOAP Fault structure all manipulation methods will have no effect.

### 5.111.2 Member Enumeration Documentation

#### 5.111.2.1 enum Arc::SOAPFault::SOAPFaultCode

Detail element of SOAP Fault Fault codes of SOAP specs

## 5.111.3 Constructor & Destructor Documentation

### 5.111.3.1 Arc::SOAPFault::SOAPFault (XMLNode & *body*)

Parse Fault elements of SOAP Body or any other XML tree with Fault element

## 5.111.4 Member Function Documentation

### 5.111.4.1 Arc::SOAPFault::operator bool (void) [inline]

Returns true if instance refers to SOAP Fault

### 5.111.4.2 SOAPFaultCode Arc::SOAPFault::Code (void)

Returns Fault Code element

### 5.111.4.3 void Arc::SOAPFault::Code (SOAPFaultCode *code*)

Set Fault Code element

### 5.111.4.4 std::string Arc::SOAPFault::Subcode (int *level*)

Returns Fault Subcode element at various levels (0 is for Code)

### 5.111.4.5 void Arc::SOAPFault::Subcode (int *level*, const char * *subcode*)

Set Fault Subcode element at various levels (0 is for Code) to 'subcode'

### 5.111.4.6 std::string Arc::SOAPFault::Reason (int *num* = 0)

Returns content of Fault Reason element at various levels

Referenced by Reason().

### 5.111.4.7 void Arc::SOAPFault::Reason (int *num*, const char * *reason*)

Set Fault Reason content at various levels to 'reason'

### 5.111.4.8 void Arc::SOAPFault::Reason (const char * *reason*) [inline]

Set Fault Reason element at top level

References Reason().

### 5.111.4.9 std::string Arc::SOAPFault::Node (void)

Returns content of Fault Node element

#### 5.111.4.10 void Arc::SOAPFault::Node (const char * *node*)

Set content of Fault Node element to 'node'

#### 5.111.4.11 std::string Arc::SOAPFault::Role (void)

Returns content of Fault Role element

#### 5.111.4.12 void Arc::SOAPFault::Role (const char * *role*)

Set content of Fault Role element to 'role'

#### 5.111.4.13 XMLNode Arc::SOAPFault::Detail (bool *create* = `false`)

Access Fault Detail element. If create is set to true this element is creted if not present.

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

- SOAPEnvelope.h

## 5.112 Arc::SOAPMessage Class Reference

Message restricted to SOAP payload.

`#include <SOAPMessage.h>`

### Public Member Functions

- SOAPMessage (void)
- SOAPMessage (long msg_ptr_addr)
- SOAPMessage (Arc::Message &msg)
- ~SOAPMessage (void)
- Arc::SOAPEnvelope * Payload (void)
- void Payload (Arc::SOAPEnvelope *new_payload)
- Arc::MessageAttributes * Attributes (void)
- void **Attributes** (Arc::MessageAttributes *attributes)
- Arc::MessageAuth * **Auth** (void)
- void **Auth** (Arc::MessageAuth *auth)
- Arc::MessageContext * **Context** (void)
- void **Context** (Arc::MessageContext *context)

### 5.112.1 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.

### 5.112.2 Constructor & Destructor Documentation

#### 5.112.2.1 Arc::SOAPMessage::SOAPMessage (void) `[inline]`

Dummy constructor

#### 5.112.2.2 Arc::SOAPMessage::SOAPMessage (long *msg_ptr_addr*)

Copy constructor. Used by language bindigs

#### 5.112.2.3 Arc::SOAPMessage::SOAPMessage (Arc::Message & *msg*)

Copy constructor. Ensures shallow copy.

#### 5.112.2.4 Arc::SOAPMessage::~SOAPMessage (void)

Destructor does not affect refered objects

## 5.112.3 Member Function Documentation

### 5.112.3.1 Arc::SOAPEnvelope* Arc::SOAPMessage::Payload (void)

Returns pointer to current payload or NULL if no payload assigned.

### 5.112.3.2 void Arc::SOAPMessage::Payload (Arc::SOAPEnvelope * *new_payload*)

Replace payload with a COPY of new one

### 5.112.3.3 Arc::MessageAttributes* Arc::SOAPMessage::Attributes (void) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

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

- SOAPMessage.h

## 5.113 Arc::Time Class Reference

A class for storing and manipulating times.

`#include <DateTime.h>`

### Public Member Functions

- Time ()
- Time (const time_t &)
- Time (const std::string &)
- Time & operator= (const time_t &)
- Time & operator= (const Time &)
- void SetTime (const time_t &)
- 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 ()

### 5.113.1 Detailed Description

A class for storing and manipulating times.

### 5.113.2 Constructor & Destructor Documentation

#### 5.113.2.1 Arc::Time::Time ()

Default constructor. The time is put equal the current time.

#### 5.113.2.2 Arc::Time::Time (const time_t &)

Constructor that takes a time_t variable and stores it.

#### 5.113.2.3 Arc::Time::Time (const std::string &)

Constructor that tries to convert a string into a time_t.

## 5.113.3 Member Function Documentation

### 5.113.3.1 Time& Arc::Time::operator= (const time_t &)

Assignment operator from a time_t.

### 5.113.3.2 Time& Arc::Time::operator= (const Time &)

Assignment operator from a Time.

### 5.113.3.3 void Arc::Time::SetTime (const time_t &)

sets the time

### 5.113.3.4 time_t Arc::Time::GetTime () const

gets the time

### 5.113.3.5 Arc::Time::operator std::string () const

Returns a string representation of the time, using the default format.

### 5.113.3.6 std::string Arc::Time::str (const TimeFormat & = time_format) const

Returns a string representation of the time, using the specified format.

### 5.113.3.7 static void Arc::Time::SetFormat (const TimeFormat &) [static]

Sets the default format for time strings.

### 5.113.3.8 static TimeFormat Arc::Time::GetFormat () [static]

Gets the default format for time strings.

### 5.113.3.9 bool Arc::Time::operator< (const Time &) const

Comparing two Time objects.

### 5.113.3.10 bool Arc::Time::operator> (const Time &) const

Comparing two Time objects.

### 5.113.3.11 bool Arc::Time::operator<= (const Time &) const

Comparing two Time objects.

#### 5.113.3.12 bool Arc::Time::operator>= (const Time &) const

Comparing two Time objects.

#### 5.113.3.13 bool Arc::Time::operator== (const Time &) const

Comparing two Time objects.

#### 5.113.3.14 bool Arc::Time::operator!= (const Time &) const

Comparing two Time objects.

#### 5.113.3.15 Time Arc::Time::operator+ (const Period &) const

Adding Time object with Period object.

#### 5.113.3.16 Time Arc::Time::operator- (const Period &) const

Subtracting Period object from Time object.

#### 5.113.3.17 Period Arc::Time::operator- (const Time &) const

Subtracting Time object from the other Time object.

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

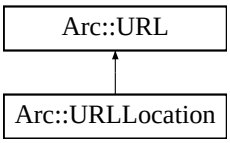
- DateTime.h

# 5.114 Arc::URL Class Reference

Class to hold general URL's.

`#include <URL.h>`

Inheritance diagram for Arc::URL::




## Public Types

- enum Scope { **base**, **onelevel**, **subtree** }

## Public Member Functions

- URL ()
- URL (const std::string &url)
- virtual ~URL ()
- const std::string & Protocol () const
- void ChangeProtocol (const std::string &newprot)
- 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
- void ChangePath (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
- const std::list< std::string > & LDAPAttributes () const
- Scope LDAPScope () const
- const std::string & LDAPFilter () const
- const std::map< std::string, std::string > & Options () const
- const std::string & Option (const std::string &option, const std::string &undefined="") const
- void AddOption (const std::string &option, const std::string &value, bool overwrite=true)
- 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
- virtual std::string str () 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 **operator!** () const

## 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
- int port
- std::string path
- std::map< std::string, std::string > httpoptions
- 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

## Friends

- std::ostream & operator<< (std::ostream &out, const URL &u)

### 5.114.1 Detailed Description

Class to hold general URL's.

The URL is split into protocol, hostname, port and path.

### 5.114.2 Member Enumeration Documentation

#### 5.114.2.1 enum Arc::URL::Scope

Scope for LDAP URLs

### 5.114.3 Constructor & Destructor Documentation

#### 5.114.3.1 Arc::URL::URL ()

Empty constructor. Necessary when the class is part of another class and the like.

#### 5.114.3.2 Arc::URL::URL (const std::string & *url*)

Constructs a new URL from a string representation.

#### 5.114.3.3 virtual Arc::URL::~URL () [virtual]

URL Destructor

### 5.114.4 Member Function Documentation

#### 5.114.4.1 const std::string& Arc::URL::Protocol () const

Returns the protocol of the URL.

#### 5.114.4.2 void Arc::URL::ChangeProtocol (const std::string & *newprot*)

Changes the protocol of the URL.

#### 5.114.4.3 const std::string& Arc::URL::Username () const

Returns the username of the URL.

#### 5.114.4.4 const std::string& Arc::URL::Passwd () const

Returns the password of the URL.

#### 5.114.4.5 const std::string& Arc::URL::Host () const

Returns the hostname of the URL.

#### 5.114.4.6 void Arc::URL::ChangeHost (const std::string & *newhost*)

Changes the hostname of the URL.

#### 5.114.4.7 int Arc::URL::Port () const

Returns the port of the URL.

#### 5.114.4.8 void Arc::URL::ChangePort (int *newport*)

Changes the port of the URL.

#### 5.114.4.9 const std::string& Arc::URL::Path () const

Returns the path of the URL.

#### 5.114.4.10 void Arc::URL::ChangePath (const std::string & *newpath*)

Changes the path of the URL.

#### 5.114.4.11 const std::map<std::string, std::string>& Arc::URL::HTTPOptions () const

Returns HTTP options if any.

#### 5.114.4.12 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.

#### 5.114.4.13 const std::list<std::string>& Arc::URL::LDAPAttributes () const

Returns the LDAP attributes if any.

#### 5.114.4.14 Scope Arc::URL::LDAPScope () const

Returns the LDAP scope.

#### 5.114.4.15 const std::string& Arc::URL::LDAPFilter () const

Returns the LDAP filter.

#### 5.114.4.16 const std::map<std::string, std::string>& Arc::URL::Options () const

Returns URL options if any.

#### 5.114.4.17 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.

#### 5.114.4.18 void Arc::URL::AddOption (const std::string & *option*, const std::string & *value*, bool *overwrite* = true)

Adds a URL option.

#### 5.114.4.19 const std::list<URLLocation>& Arc::URL::Locations () const

Returns the locations if any.

#### 5.114.4.20 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions () const

Returns the common location options if any.

#### 5.114.4.21 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.

#### 5.114.4.22 virtual std::string Arc::URL::str () const [virtual]

Returns a string representation of the URL.

Reimplemented in Arc::URLLocation.

#### 5.114.4.23 virtual std::string Arc::URL::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented in Arc::URLLocation.

#### 5.114.4.24 virtual std::string Arc::URL::ConnectionURL () const [virtual]

Returns a string representation with protocol, host and port only

#### 5.114.4.25 bool Arc::URL::operator< (const URL & *url*) const

Compares one URL to another

#### 5.114.4.26 bool Arc::URL::operator== (const URL & *url*) const

Is one URL equal to another?

#### 5.114.4.27 Arc::URL::operator bool () const

Check if instance holds valid URL

#### 5.114.4.28 static std::string Arc::URL::BaseDN2Path (const std::string &) [static, protected]

a private method that converts an ldap basedn to a path.

#### 5.114.4.29 static std::string Arc::URL::Path2BaseDN (const std::string &) [static, protected]

a private method that converts an ldap path to a basedn.

### 5.114.5 Friends And Related Function Documentation

#### 5.114.5.1 std::ostream& operator<< (std::ostream & *out*, const URL & *u*) [friend]

Overloaded operator << to print a URL.

### 5.114.6 Member Data Documentation

#### 5.114.6.1 std::string Arc::URL::protocol [protected]

the url protocol.

#### 5.114.6.2 std::string Arc::URL::username [protected]

username of the url.

#### 5.114.6.3 std::string Arc::URL::passwd [protected]

password of the url.

#### 5.114.6.4 std::string Arc::URL::host [protected]

hostname of the url.

#### 5.114.6.5 int Arc::URL::port [protected]

portnumber of the url.

#### 5.114.6.6 std::string Arc::URL::path [protected]

the url path.

#### 5.114.6.7 std::map<std::string, std::string> Arc::URL::httpoptions [protected]

HTTP options of the url.

#### 5.114.6.8 std::list<std::string> Arc::URL::ldapattributes [protected]

LDAP attributes of the url.

#### 5.114.6.9 Scope Arc::URL::ldapscope [protected]

LDAP scope of the url.

#### 5.114.6.10 std::string Arc::URL::ldapfilter [protected]

LDAP filter of the url.

#### 5.114.6.11 std::map<std::string, std::string> Arc::URL::urloptions [protected]

options of the url.

#### 5.114.6.12 std::list<URLLocation> Arc::URL::locations [protected]

locations for index server URLs.

#### 5.114.6.13 std::map<std::string, std::string> Arc::URL::commonlocoptions [protected]

common location options for index server URLs.

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

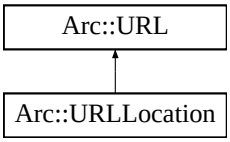
- URL.h

## 5.115 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

### 5.115.1 Detailed Description

Class to hold a resolved URL location.

It is specific to file indexing service registrations.

### 5.115.2 Constructor & Destructor Documentation

#### 5.115.2.1 Arc::URLLocation::URLLocation (const std::string & *url*)

Creates a URLLocation from a string representaion.

#### 5.115.2.2 Arc::URLLocation::URLLocation (const std::string & *url*, const std::string & *name*)

Creates a URLLocation from a string representaion and a name.

#### 5.115.2.3 Arc::URLLocation::URLLocation (const URL & *url*)

Creates a URLLocation from a URL.

#### 5.115.2.4 Arc::URLLocation::URLLocation (const URL & *url*, const std::string & *name*)

Creates a URLLocation from a URL and a name.

#### 5.115.2.5 Arc::URLLocation::URLLocation (const std::map< std::string, std::string > & *options*, const std::string & *name*)

Creates a URLLocation from options and a name.

#### 5.115.2.6 virtual Arc::URLLocation::~URLLocation () [virtual]

URLLocation destructor.

### 5.115.3 Member Function Documentation

#### 5.115.3.1 const std::string& Arc::URLLocation::Name () const

Returns the URLLocation name.

#### 5.115.3.2 virtual std::string Arc::URLLocation::str () const [virtual]

Returns a string representation of the URLLocation.

Reimplemented from Arc::URL.

#### 5.115.3.3 virtual std::string Arc::URLLocation::fullstr () const [virtual]

Returns a string representation including options and locations

Reimplemented from Arc::URL.

### 5.115.4 Member Data Documentation

#### 5.115.4.1 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

# 5.116 Arc::UsernameToken Class Reference

Interface for manipulation of WS-Security Username Token Profile.

`#include <UsernameToken.h>`

## Public Member Functions

- UsernameToken (SOAPEnvelope &soap)
- UsernameToken (SOAPEnvelope &soap, std::string &uid, bool pwdtype, bool milliseconds)
- UsernameToken (SOAPEnvelope &soap, std::string &username, bool mac, int iteration, std::string &id)

## Protected Attributes

- XMLNode **header_**

## 5.116.1 Detailed Description

Interface for manipulation of WS-Security Username Token Profile.

## 5.116.2 Constructor & Destructor Documentation

### 5.116.2.1 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*)

SOAP header element Link to existing SOAP header to parse username token information

### 5.116.2.2 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, std::string & *uid*, bool *pwdtype*, bool *milliseconds*)

Set username token information into the SOAP header

**Parameters:**

***soap*** the SOAP message

***username*** <wsse:Username>...</wsse:Username>

***password*** <wsse:Password Type="...">...</wsse:Password>

***uid*** <wsse:UsernameToken wsu:ID="...">

***pwdtype*** <wsse:Password Type="...">...</wsse:Password>

***milliseconds*** precision of created time — <wsu:Created>...</wsu/Created>

### 5.116.2.3 Arc::UsernameToken::UsernameToken (SOAPEnvelope & *soap*, std::string & *username*, bool *mac*, int *iteration*, std::string & *id*)

Set username token information into the SOAP header

**Parameters:**

***soap*** the SOAP message

***username*** <wsse:Username>...</wsse:Username>

***salt*** <wsse11:Salt>...</wsse11:Salt>

***iteration*** <wsse11:Iteration>...</wsse11:Iteration>

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

- UsernameToken.h

## 5.117 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Adressing Endpoint Reference.

`#include <WSA.h>`

### Public Member Functions

- WSAEndpointReference (XMLNode epr)
- WSAEndpointReference (const std::string &address)
- WSAEndpointReference (void)
- ~WSAEndpointReference (void)
- std::string Address (void) const
- void Address (const std::string &uri)
- WSAEndpointReference & operator= (const std::string &address)
- XMLNode ReferenceParameters (void)
- XMLNode MetaData (void)
- operator XMLNode (void)

### Protected Attributes

- XMLNode **epr_**

### 5.117.1 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.

### 5.117.2 Constructor & Destructor Documentation

#### 5.117.2.1 Arc::WSAEndpointReference::WSAEndpointReference (XMLNode *epr*)

Link to top level EPR XML node Linking to existing EPR in XML tree

#### 5.117.2.2 Arc::WSAEndpointReference::WSAEndpointReference (const std::string & *address*)

Creating independent EPR - not implemented

#### 5.117.2.3 Arc::WSAEndpointReference::WSAEndpointReference (void)

Dummy constructor - creates invalid instance

#### 5.117.2.4 Arc::WSAEndpointReference::~WSAEndpointReference (void)

Destructor. All empty elements of EPR XML are destroyed here too

## 5.117.3 Member Function Documentation

### 5.117.3.1 std::string Arc::WSAEndpointReference::Address (void) const

Returns Address (URL) encoded in EPR

### 5.117.3.2 void Arc::WSAEndpointReference::Address (const std::string & *uri*)

Assigns new Address value. If EPR had no Address element it is created.

### 5.117.3.3 WSAEndpointReference& Arc::WSAEndpointReference::operator= (const std::string & *address*)

Same as Address(uri)

### 5.117.3.4 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.

### 5.117.3.5 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.

### 5.117.3.6 Arc::WSAEndpointReference::operator XMLNode (void)

Returns reference to EPR top XML node

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

- WSA.h

## 5.118 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
- void To (const std::string &uri)
- WSAEndpointReference From (void)
- WSAEndpointReference ReplyTo (void)
- WSAEndpointReference FaultTo (void)
- std::string Action (void) const
- void Action (const std::string &uri)
- std::string MessageID (void) const
- void MessageID (const std::string &uri)
- std::string RelatesTo (void) const
- void RelatesTo (const std::string &uri)
- std::string RelationshipType (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

- XMLNode **header_**
- bool header_allocated_

### 5.118.1 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.

### 5.118.2 Constructor & Destructor Documentation

#### 5.118.2.1 Arc::WSAHeader::WSAHeader (SOAPEnvelope & *soap*)

Linking to a header of existing SOAP message

#### 5.118.2.2 Arc::WSAHeader::WSAHeader (const std::string & *action*)

Creating independent SOAP header - not implemented

### 5.118.3 Member Function Documentation

#### 5.118.3.1 std::string Arc::WSAHeader::To (void) const

Returns content of To element of SOAP Header.

#### 5.118.3.2 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.

#### 5.118.3.3 WSAEndpointReference Arc::WSAHeader::From (void)

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

#### 5.118.3.4 WSAEndpointReference Arc::WSAHeader::ReplyTo (void)

Returns ReplyTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

#### 5.118.3.5 WSAEndpointReference Arc::WSAHeader::FaultTo (void)

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulted.

#### 5.118.3.6 std::string Arc::WSAHeader::Action (void) const

Returns content of Action element of SOAP Header.

#### 5.118.3.7 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.

#### 5.118.3.8 std::string Arc::WSAHeader::MessageID (void) const

Returns content of MessageID element of SOAP Header.

#### 5.118.3.9 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.

#### 5.118.3.10 std::string Arc::WSAHeader::RelatesTo (void) const

Returns content of RelatesTo element of SOAP Header.

#### 5.118.3.11 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.

#### 5.118.3.12 std::string Arc::WSAHeader::RelationshipType (void) const

Returns content of RelationshipType element of SOAP Header.

#### 5.118.3.13 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.

#### 5.118.3.14 XMLNode Arc::WSAHeader::ReferenceParameter (int *n*)

Return n-th ReferenceParameter element

#### 5.118.3.15 XMLNode Arc::WSAHeader::ReferenceParameter (const std::string & *name*)

Returns first ReferenceParameter element with specified name

#### 5.118.3.16 XMLNode Arc::WSAHeader::NewReferenceParameter (const std::string & *name*)

Creates new ReferenceParameter element with specified name. Returns reference to created element.

#### 5.118.3.17 Arc::WSAHeader::operator XMLNode (void)

Returns reference to SOAP Header - not implemented

#### 5.118.3.18 static bool Arc::WSAHeader::Check (SOAPEnvelope & *soap*) [static]

Tells if specified SOAP message has WSA header

### 5.118.4 Member Data Documentation

#### 5.118.4.1 bool Arc::WSAHeader::header_allocated_ [protected]

SOAP header element

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

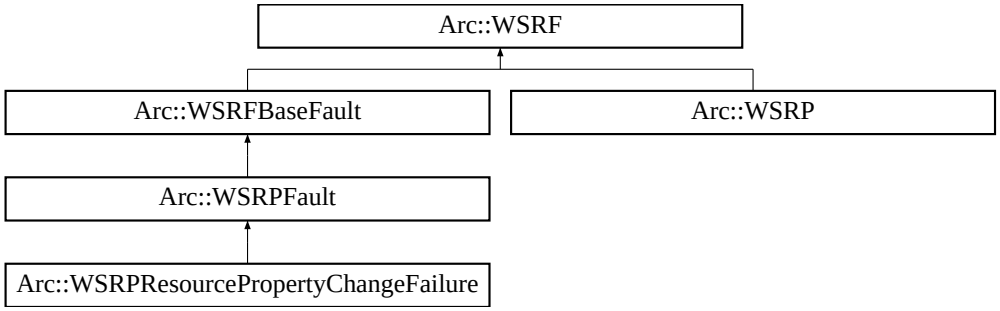
- WSA.h

# 5.119 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)
- virtual bool **operator!** (void)

## Protected Member Functions

- void set_namespaces (void)

## Protected Attributes

- SOAPEnvelope & **soap_**
- bool allocated_
- bool valid_

### 5.119.1 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.

### 5.119.2 Constructor & Destructor Documentation

#### 5.119.2.1 Arc::WSRF::WSRF (SOAPEnvelope & *soap*, const std::string & *action* = "")

Constructor - creates object out of supplied SOAP tree.

#### 5.119.2.2 Arc::WSRF::WSRF (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - creates new WSRF object

### 5.119.3 Member Function Documentation

#### 5.119.3.1 void Arc::WSRF::set_namespaces (void) [protected]

true if object represents valid WSRF message set WS Resource namespaces and default prefixes in SOAP message

Reimplemented in Arc::WSRP, and Arc::WSRFBaseFault.

#### 5.119.3.2 virtual SOAPEnvelope& Arc::WSRF::SOAP (void) [inline, virtual]

Direct access to underlying SOAP element

#### 5.119.3.3 virtual Arc::WSRF::operator bool (void) [inline, virtual]

Returns true if instance is valid

References valid_.

### 5.119.4 Member Data Documentation

#### 5.119.4.1 bool Arc::WSRF::allocated_ [protected]

Associated SOAP message - it's SOAP message after all

#### 5.119.4.2 bool Arc::WSRF::valid_ [protected]

true if soap_ needs to be deleted in destructor

Referenced by operator bool().

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

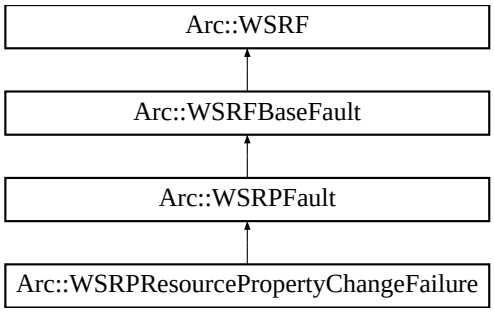
- WSRF.h

# 5.120 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)
- std::string **Type** (void)
- Time **Timestamp** (void)
- void **Timestamp** (Time)
- WSAEndpointReference **Originator** (void)
- void **ErrorCode** (const std::string &dialect, const XMLNode &error)
- XMLNode **ErrorCode** (void)
- std::string **ErrorCodeDialect** (void)
- void **Description** (int pos, const std::string &desc, const std::string &lang)
- std::string **Description** (int pos)
- std::string **DescriptionLang** (int pos)
- void **FaultCause** (int pos, const XMLNode &cause)
- XMLNode **FaultCause** (int pos)

## Protected Member Functions

- void set_namespaces (void)

### 5.120.1 Detailed Description

Base class for WSRF fault messages.

Use classes inherited from it for specific faults.

### 5.120.2 Constructor & Destructor Documentation

#### 5.120.2.1 Arc::WSRFBaseFault::WSRFBaseFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

#### 5.120.2.2 Arc::WSRFBaseFault::WSRFBaseFault (const std::string & *type*)

Constructor - creates new WSRF fault

### 5.120.3 Member Function Documentation

#### 5.120.3.1 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:

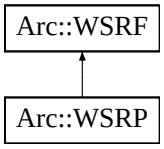
- WSRFBaseFault.h

# 5.121 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)

### 5.121.1 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.

### 5.121.2 Constructor & Destructor Documentation

#### 5.121.2.1 Arc::WSRP::WSRP (bool *fault* = `false`, const std::string & *action* = `""`)

Constructor - prepares object for creation of new WSRP request/response/fault

#### 5.121.2.2 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.

### 5.121.3 Member Function Documentation

#### 5.121.3.1 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:

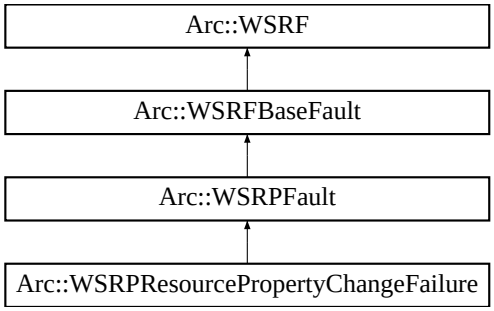
- WSResourceProperties.h

# 5.122 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)

## 5.122.1 Detailed Description

Base class for WS-ResourceProperties faults.

## 5.122.2 Constructor & Destructor Documentation

### 5.122.2.1 Arc::WSRPFault::WSRPFault (SOAPEnvelope & *soap*)

Constructor - creates object out of supplied SOAP tree.

### 5.122.2.2 Arc::WSRPFault::WSRPFault (const std::string & *type*)

Constructor - creates new WSRP fault

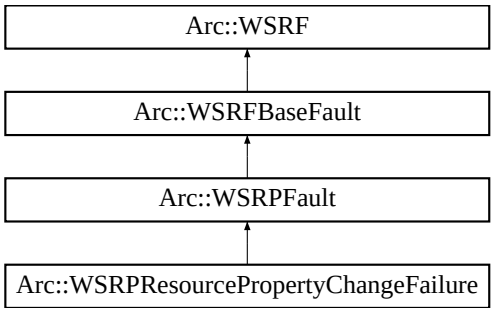
The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 5.123 Arc::WSRPResourcePropertyChangeFailure Class Reference

`#include <WSResourceProperties.h>`

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure::




### Public Member Functions

- WSRPResourcePropertyChangeFailure (SOAPEnvelope &soap)
- WSRPResourcePropertyChangeFailure (const std::string &type)
- XMLNode **CurrentProperties** (bool create=false)
- XMLNode **RequestedProperties** (bool create=false)

### 5.123.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

### 5.123.2 Constructor & Destructor Documentation

#### 5.123.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (SOAPEnvelope & *soap*) [inline]

Constructor - creates object out of supplied SOAP tree.

#### 5.123.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure (const std::string & *type*) [inline]

Constructor - creates new WSRP fault

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

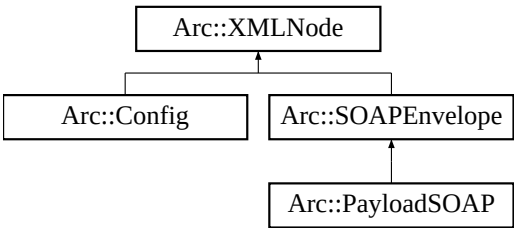
- WSResourceProperties.h

# 5.124 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 (const Arc::NS &ns, const char *name)
- ~XMLNode (void)
- void New (XMLNode &new_node) const
- operator bool (void) const
- bool operator! (void) const
- XMLNode Child (int n=0) const
- 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 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) const
- XMLNode Attribute (const char *name) const

- XMLNode Attribute (const std::string &name) const
- XMLNode NewAttribute (const char *name)
- XMLNode NewAttribute (const std::string &name)
- int AttributesSize (void) const
- void Namespaces (const NS &namespaces)
- 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 XPathLookup (const std::string &xpathExpr, const Arc::NS &nsList)
- XMLNode GetRoot (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)

## Protected Member Functions

- XMLNode (xmlNodePtr node)

## Protected Attributes

- xmlNodePtr **node_**
- bool is_owner_
- bool is_temporary_

## Friends

- class **XMLNodeContainer**
- 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)

### 5.124.1 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.

### 5.124.2 Constructor & Destructor Documentation

#### 5.124.2.1 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.

#### 5.124.2.2 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.

#### 5.124.2.3 Arc::XMLNode::XMLNode (const XMLNode & *node*) [inline]

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited.

#### 5.124.2.4 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

#### 5.124.2.5 Arc::XMLNode::XMLNode (const char * *xml*, int *len* = -1)

Same as previous

#### 5.124.2.6 Arc::XMLNode::XMLNode (const Arc::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

#### 5.124.2.7 Arc::XMLNode::~XMLNode (void)

Destructor Also destroys underlying XML document if owned by this instance

## 5.124.3 Member Function Documentation

### 5.124.3.1 void Arc::XMLNode::New (XMLNode & *new_node*) const

Creates a copy of XML (sub)tree. If object does not represent whole document - top level document is created. 'new_node' becomes a pointer owning new XML document.

### 5.124.3.2 Arc::XMLNode::operator bool (void) const [inline]

Returns true if instance points to XML element - valid instance

References is_temporary_.

### 5.124.3.3 bool Arc::XMLNode::operator! (void) const [inline]

Returns true if instance does not point to XML element - invalid instance

References is_temporary_.

### 5.124.3.4 XMLNode Arc::XMLNode::Child (int *n* = 0) const

Returns XMLNode instance representing n-th child of XML element. If such does not exist invalid XMLNode instance is returned

Referenced by ArcSec::ArcAttributeProxy< TheAttribute >::getAttribute().

### 5.124.3.5 XMLNode Arc::XMLNode::operator[] (const char * *name*) const

Returns XMLNode instance representing first child element with specified name. Name may be "namespace_prefix:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid XMLNode instance is returned

Referenced by Get(), and operator[]().

### 5.124.3.6 XMLNode Arc::XMLNode::operator[] (const std::string & *name*) const [inline]

Similar to previous method

References operator[]().

### 5.124.3.7 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]

### 5.124.3.8 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.

#### 5.124.3.9 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.

#### 5.124.3.10 int Arc::XMLNode::Size (void) const

Returns number of children nodes

#### 5.124.3.11 XMLNode Arc::XMLNode::Get (const std::string & *name*) const [inline]

Same as operator[]

References operator[ ]().

#### 5.124.3.12 std::string Arc::XMLNode::Name (void) const

Returns name of XML node

Referenced by FullName(), and Name().

#### 5.124.3.13 std::string Arc::XMLNode::Prefix (void) const

Returns namespace prefix of XML node

Referenced by FullName().

#### 5.124.3.14 std::string Arc::XMLNode::FullName (void) const [inline]

Returns prefix:name of XML node

References Name(), and Prefix().

#### 5.124.3.15 std::string Arc::XMLNode::Namespace (void) const

Returns namespace URI of XML node

#### 5.124.3.16 void Arc::XMLNode::Name (const char * *name*)

Assigns new name to XML node

#### 5.124.3.17 void Arc::XMLNode::Name (const std::string & *name*) [inline]

Assigns new name to XML node

References Name().

#### 5.124.3.18 void Arc::XMLNode::GetXML (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with this instance XML subtree textual representation

Reimplemented in Arc::SOAPEnvelope.

#### 5.124.3.19 void Arc::XMLNode::GetDoc (std::string & *out_xml_str*, bool *user_friendly* = `false`) const

Fills argument with whole XML document textual representation

#### 5.124.3.20 Arc::XMLNode::operator std::string (void) const

Returns textual content of node excluding content of children nodes

#### 5.124.3.21 XMLNode& Arc::XMLNode::operator= (const char * *content*)

Sets textual content of node. All existing children nodes are discarded.

Referenced by operator=(), and Set().

#### 5.124.3.22 XMLNode& Arc::XMLNode::operator= (const std::string & *content*) `[inline]`

Sets textual content of node. All existing children nodes are discarded.

References operator=().

#### 5.124.3.23 void Arc::XMLNode::Set (const std::string & *content*) `[inline]`

Same as operator=. Used for bindings.

References operator=().

#### 5.124.3.24 XMLNode& Arc::XMLNode::operator= (const XMLNode & *node*)

Make instance refer to another XML node. Ownership is not inherited.

#### 5.124.3.25 XMLNode Arc::XMLNode::Attribute (int *n* = `0`) const

Returns list of all attributes of node.

Returns XMLNode instance reresenting n-th attribute of node.

Referenced by Attribute(), and ArcSec::ArcAttributeProxy< TheAttribute >::getAttribute().

#### 5.124.3.26 XMLNode Arc::XMLNode::Attribute (const char * *name*) const

Returns XMLNode instance representing first attribute of node with specified by name

#### 5.124.3.27 XMLNode Arc::XMLNode::Attribute (const std::string & *name*) const [inline]

Returns XMLNode instance representing first attribute of node with specified by name

References Attribute().

#### 5.124.3.28 XMLNode Arc::XMLNode::NewAttribute (const char * *name*)

Creates new attribute with specified name.

Referenced by NewAttribute().

#### 5.124.3.29 XMLNode Arc::XMLNode::NewAttribute (const std::string & *name*) [inline]

Creates new attribute with specified name.

References NewAttribute().

#### 5.124.3.30 int Arc::XMLNode::AttributesSize (void) const

Returns number of attributes of node

#### 5.124.3.31 void Arc::XMLNode::Namespaces (const NS & *namespaces*)

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 usefull to apply this method to XML being processed in order to refer to it's elements by known prefix.

Reimplemented in Arc::SOAPEnvelope.

#### 5.124.3.32 NS Arc::XMLNode::Namespaces (void)

Returns namespaces known at this node

#### 5.124.3.33 std::string Arc::XMLNode::NamespacePrefix (const char * *urn*)

Returns prefix of specified namespace. Empty string if no such namespace.

#### 5.124.3.34 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

Referenced by NewChild().

#### 5.124.3.35 XMLNode Arc::XMLNode::NewChild (const std::string & *name*, int *n* = -1, bool *global_order* = false) [inline]

Same as NewChild(const char*,int,bool)

References NewChild().

#### 5.124.3.36 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)

#### 5.124.3.37 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)

References NewChild().

#### 5.124.3.38 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 refering to new child. XML element is a copy of supplied one but not owned by returned instance

#### 5.124.3.39 void Arc::XMLNode::Replace (const XMLNode & *node*)

Makes a copy of supplied XML node and makes this instance refere to it

#### 5.124.3.40 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

#### 5.124.3.41 XMLNodeList Arc::XMLNode::XPathLookup (const std::string & *xpathExpr*, const Arc::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.

#### 5.124.3.42 XMLNode Arc::XMLNode::GetRoot (void)

Get the root node from any child node of the tree

#### 5.124.3.43 bool Arc::XMLNode::SaveToFile (const std::string & *file_name*) const

Save string representation of node to file

#### 5.124.3.44 bool Arc::XMLNode::SaveToStream (std::ostream & *out*) const

Save string representation of node to stream

#### 5.124.3.45 bool Arc::XMLNode::ReadFromFile (const std::string & *file_name*)

Read XML document from file and associate it with this node

#### 5.124.3.46 bool Arc::XMLNode::ReadFromStream (std::istream & *in*)

Read XML document from stream and associate it with this node

### 5.124.4 Friends And Related Function Documentation

#### 5.124.4.1 bool MatchXMLName (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements have same names

#### 5.124.4.2 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

#### 5.124.4.3 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

#### 5.124.4.4 bool MatchXMLNamespace (const XMLNode & *node1*, const XMLNode & *node2*) [friend]

Returns true if underlying XML elements belong to same namespaces

#### 5.124.4.5 bool MatchXMLNamespace (const XMLNode & *node*, const char * *uri*) [friend]

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

#### 5.124.4.6 bool MatchXMLNamespace (const XMLNode & *node*, const std::string & *uri*) [friend]

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

### 5.124.5 Member Data Documentation

#### 5.124.5.1 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.

#### 5.124.5.2 bool Arc::XMLNode::is_temporary_ [protected]

This variable is for future

Referenced by operator bool(), and operator!().

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

- XMLNode.h

# 5.125 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)
- XMLNode operator[ ] (int)
- std::list< XMLNode > Nodes (void)

### 5.125.1 Detailed Description

Container for multiple XMLNode elements

### 5.125.2 Constructor & Destructor Documentation

#### 5.125.2.1 Arc::XMLNodeContainer::XMLNodeContainer (void)

Default constructor

#### 5.125.2.2 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.

### 5.125.3 Member Function Documentation

#### 5.125.3.1 XMLNodeContainer& Arc::XMLNodeContainer::operator= (const XMLNodeContainer &)

Same as copy constructor with current nodes being deleted first.

#### 5.125.3.2 void Arc::XMLNodeContainer::Add (const XMLNode &)

Link XML subtree refered by node to container. XML tree must be available as long as this object is used.

#### 5.125.3.3 void Arc::XMLNodeContainer::Add (const std::list< XMLNode > &)

Link multiple XML subtrees to container.

#### 5.125.3.4 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.

#### 5.125.3.5 void Arc::XMLNodeContainer::AddNew (const std::list< XMLNode > &)

Copy multiple XML subtrees to container.

#### 5.125.3.6 int Arc::XMLNodeContainer::Size (void)

Return number of refered/stored nodes.

#### 5.125.3.7 XMLNode Arc::XMLNodeContainer::operator[ ] (int)

Returns n-th node in a store.

#### 5.125.3.8 std::list<XMLNode> Arc::XMLNodeContainer::Nodes (void)

Returns all stored nodes.

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

- XMLNode.h

# Index


~Counter
    Arc::Counter, 61
~DataBufferPar
    Arc::DataBufferPar, 70
~DataCache
    Arc::DataCache, 77
~DataMover
    Arc::DataMover, 82
~DataPoint
    Arc::DataPoint, 88
~DataSpeed
    Arc::DataSpeed, 112
~IntraProcessCounter
    Arc::IntraProcessCounter, 146
~Loader
    Arc::Loader, 150
~Message
    Arc::Message, 175
~PayloadRaw
    Arc::PayloadRaw, 189
~PayloadStream
    Arc::PayloadStream, 195
~Plexer
    Arc::Plexer, 206
~RegularExpression
    Arc::RegularExpression, 211
~Run
    Arc::Run, 221
~SOAPMessage
    Arc::SOAPMessage, 246
~URL
    Arc::URL, 252
~URLLocation
    Arc::URLLocation, 259
~WSAEndpointReference
    Arc::WSAEndpointReference, 262
~XMLNode
    Arc::XMLNode, 277

acc_descriptor, 31
AcceptsMeta
    Arc::DataPoint, 94
    Arc::DataPointDirect, 101
    Arc::DataPointIndex, 106
ACCFactory
    Arc::ACCFactory, 32
Acquire
    Arc::DelegationConsumer, 116
    Arc::InformationContainer, 139
Action
    Arc::WSAHeader, 265
Add
    Arc::MessageContext, 183
    Arc::XMLNodeContainer, 285
add
    Arc::MessageAttributes, 178
AddCADir
    Arc::BaseConfig, 46
AddCAFile
    Arc::BaseConfig, 46
AddCertificate
    Arc::BaseConfig, 45
addDestination
    Arc::Logger, 157
AddLocation
    Arc::DataPoint, 95
    Arc::DataPointDirect, 102
    Arc::DataPointIndex, 106
AddNew
    Arc::XMLNodeContainer, 285, 286
AddOption
    Arc::URL, 254
AddOverlay
    Arc::BaseConfig, 46
AddPluginsPath
    Arc::BaseConfig, 45
addPolicy
    ArcSec::Policy, 210
AddPrivateKey
    Arc::BaseConfig, 45
AddProxy
    Arc::BaseConfig, 46
addRequestItem
    ArcSec::Request, 214
Address
    Arc::WSAEndpointReference, 263
AddSecHandler
    Arc::MCC, 165
    Arc::Service, 235
AddUrl

Arc::InfoRegister, 137
allocated_
Arc::WSRF, 268
Arc, 11
AttrConstIter, 20
AttrIter, 20
AttrMap, 20
BUSY_ERROR, 20
ContentFromPayload, 21
CreateThreadFunction, 21
ETERNAL, 23
GENERIC_ERROR, 20
GUID, 21
HISTORIC, 23
loader_descriptors, 20
LogLevel, 20
MatchXMLName, 21
MatchXMLNamespace, 21, 22
operator<<, 22
PARSING_ERROR, 20
PROTOCOL_RECOGNIZED_ERROR, 20
ReadURLList, 22
SESSION_CLOSE, 20
STATUS_OK, 20
StatusKind, 20
string, 22
stringto, 22
TimeFormat, 21
TimeStamp, 22, 23
tokenize, 23
tostring, 23
trim, 23
UNKNOWN_SERVICE_ERROR, 20
upper, 23
UUID, 23
WSAFault, 21
WSAFaultAssign, 23
WSAFaultExtract, 23
WSAFaultInvalidAddressingHeader, 21
WSAFaultUnknown, 21
Arc::ACCFactory, 32
ACCFactory, 32
get_instance, 32
Arc::ArcLocation, 35
Get, 35
GetPlugins, 35
Init, 35
Arc::AttributeIterator, 38
AttributeIterator, 38
current_, 40
end_, 40
hasMore, 40
key, 39
MessageAttributes, 40
operator*, 39
operator++, 39
operator->, 39
Arc::BaseConfig, 45
AddCADir, 46
AddCAFile, 46
AddCertificate, 45
AddOverlay, 46
AddPluginsPath, 45
AddPrivateKey, 45
AddProxy, 46
GetOverlay, 46
MakeConfig, 46
Arc::ChainContext, 47
operator MCCFactory *, 47
operator PDPFactory *, 47
operator SecHandlerFactory *, 47
operator ServiceFactory *, 47
Arc::CheckSum, 49
Arc::CheckSumAny, 50
Arc::CIStringValue, 52
CIStringValue, 52
equal, 53
operator bool, 53
Arc::ClientSOAP, 54
ClientSOAP, 54
process, 54
Arc::Config, 57
Config, 57, 58
parse, 58
print, 58
Arc::Counter, 59
~Counter, 61
cancel, 63
changeExcess, 62
changeLimit, 62
Counter, 61
CounterTicket, 65
ExpirationReminder, 65
extend, 63
getCounterTicket, 64
getCurrentTime, 64
getExcess, 62
getExpirationReminder, 65
getExpiryTime, 64
getLimit, 61
getValue, 63
IDType, 61
reserve, 63
setExcess, 62
setLimit, 61
Arc::CounterTicket, 66
cancel, 67
Counter, 67

CounterTicket, 66
extend, 67
isValid, 66
Arc::CRC32Sum, 68
Arc::DataBufferPar, 69
~DataBufferPar, 70
buffer_size, 74
checksum_object, 74
checksum_valid, 74
DataBufferPar, 70
eof_position, 74
eof_read, 72, 73
eof_write, 73
error, 73
error_read, 73
error_transfer, 73
error_write, 73
for_read, 71
for_write, 71, 72
is_notwritten, 72
is_read, 71
is_written, 72
operator bool, 70
set, 70
speed, 75
wait, 73
wait_eof, 74
wait_eof_read, 74
wait_eof_write, 74
wait_read, 74
wait_used, 74
wait_write, 74
Arc::DataCache, 76
~DataCache, 77
cb, 78
CheckCreated, 78
CheckValid, 79
clean, 78
copy, 78
DataCache, 77
file, 77
GetCreated, 79
GetValid, 79
link, 78
operator bool, 78
SetCreated, 78
SetValid, 79
start, 77
stop, 77
Arc::DataCallback, 80
Arc::DataHandle, 81
Arc::DataMover, 82
~DataMover, 82
checks, 84
DataMover, 82
force_to_meta, 84
passive, 84
retry, 84
secure, 84
set_default_max_inactivity_time, 85
set_default_min_average_speed, 84
set_default_min_speed, 84
Transfer, 83
verbose, 83
Arc::DataPoint, 86
~DataPoint, 88
AcceptsMeta, 94
AddLocation, 95
BufNum, 93
BufSize, 93
Cache, 93
Check, 89
CheckCheckSum, 92
CheckCreated, 93
CheckSize, 92
CheckValid, 93
CompareMeta, 94
CurrentLocation, 95
CurrentLocationMetadata, 95
DataPoint, 88
GetAdditionalChecks, 90
GetCheckSum, 93
GetCreated, 93
GetSecure, 90
GetSize, 92
GetTries, 94
GetURL, 88
GetValid, 93
HaveLocations, 95
IsIndex, 94
ListFiles, 89
Local, 94
LocationValid, 95
NextLocation, 95
operator bool, 88
operator!, 88
Passive, 90
PostRegister, 91
PreRegister, 91
PreUnregister, 92
ProvidesMeta, 94
Range, 91
ReadOutOfOrder, 89
Registered, 91
Remove, 89
RemoveLocation, 96
RemoveLocations, 96
Resolve, 91

SetAdditionalChecks, 90
SetCheckSum, 93
SetCreated, 93
SetMeta, 94
SetSecure, 90
SetSize, 92
SetTries, 94
SetValid, 93
StartReading, 88
StartWriting, 88
StopReading, 89
StopWriting, 89
str, 88
Unregister, 92
WriteOutOfOrder, 90
Arc::DataPointDirect, 97
AcceptsMeta, 101
AddLocation, 102
BufNum, 98
BufSize, 98
Cache, 98
CurrentLocation, 101
CurrentLocationMetadata, 102
GetAdditionalChecks, 99
GetSecure, 99
HaveLocations, 102
IsIndex, 98
Local, 98
LocationValid, 102
NextLocation, 102
Passive, 100
PostRegister, 101
PreRegister, 100
PreUnregister, 101
ProvidesMeta, 101
Range, 100
ReadOutOfOrder, 99
Registered, 100
RemoveLocation, 102
RemoveLocations, 102
Resolve, 100
SetAdditionalChecks, 99
SetSecure, 99
Unregister, 101
WriteOutOfOrder, 99
Arc::DataPointIndex, 104
AcceptsMeta, 106
AddLocation, 106
BufNum, 107
BufSize, 107
Cache, 107
Check, 108
CurrentLocation, 105
CurrentLocationMetadata, 105
GetAdditionalChecks, 109
GetSecure, 109
HaveLocations, 105
IsIndex, 106
Local, 107
locations, 110
LocationValid, 105
NextLocation, 105
Passive, 109
ProvidesMeta, 106
Range, 109
ReadOutOfOrder, 108
Registered, 106
Remove, 108
RemoveLocation, 105
RemoveLocations, 106
SetAdditionalChecks, 109
SetSecure, 109
SetTries, 106
StartReading, 107
StartWriting, 107
StopReading, 108
StopWriting, 108
WriteOutOfOrder, 108
Arc::DataSpeed, 111
~DataSpeed, 112
DataSpeed, 111
hold, 114
max_inactivity_time_failure, 114
min_average_speed_failure, 114
min_speed_failure, 114
reset, 113
set_base, 113
set_max_data, 113
set_max_inactivity_time, 113
set_min_average_speed, 112
set_min_speed, 112
set_progress_indicator, 113
transfer, 113
transfered_size, 114
verbose, 112
Arc::DelegationConsumer, 115
Acquire, 116
Backup, 116
DelegationConsumer, 115
Generate, 116
ID, 116
LogError, 116
Request, 116
Restore, 116
Arc::DelegationConsumerSOAP, 117
DelegateCredentialsInit, 117
DelegatedToken, 118
DelegationConsumerSOAP, 117

UpdateCredentials, 117
Arc::DelegationContainerSOAP, 119
context_lock_, 120
DelegateCredentialsInit, 119
DelegatedToken, 119
max_duration_, 119
max_size_, 119
max_usage_, 120
restricted_, 120
UpdateCredentials, 119
Arc::DelegationProvider, 121
Delegate, 121
DelegationProvider, 121
Arc::DelegationProviderSOAP, 123
DelegateCredentialsInit, 124
DelegatedToken, 124
DelegationProviderSOAP, 123
UpdateCredentials, 124
Arc::DMCFactory, 127
DMCFactory, 127
get_instance, 127
Arc::ExpirationReminder, 132
Counter, 133
getExpiryTime, 132
getReservationID, 132
operator<, 132
Arc::FileInfo, 134
Arc::InfoRegister, 137
AddUrl, 137
InfoRegister, 137
registration, 137
registration_forever, 137
Arc::InformationContainer, 138
Acquire, 139
Assign, 139
doc_, 139
Get, 138
InformationContainer, 138
Arc::InformationInterface, 140
Get, 140
InformationInterface, 140
lock_, 141
Arc::InformationRequest, 142
InformationRequest, 142
SOAP, 142
Arc::InformationResponse, 144
InformationResponse, 144
Result, 144
Arc::IntraProcessCounter, 145
~IntraProcessCounter, 146
cancel, 148
changeExcess, 147
changeLimit, 146
extend, 148
getExcess, 146
getLimit, 146
getValue, 147
IntraProcessCounter, 145
reserve, 147
setExcess, 147
setLimit, 146
Arc::Loader, 149
~Loader, 150
getACC, 150
Loader, 150
Arc::loader_descriptor, 151
Arc::LoaderFactory, 152
get_instance, 152
load_all_instances, 152
LoaderFactory, 152
Arc::LogDestination, 154
log, 155
LogDestination, 154
Arc::Logger, 156
addDestination, 157
getRootLogger, 157
getThreshold, 158
Logger, 157
msg, 158
removeDestinations, 157
setThreshold, 157
Arc::LogMessage, 159
getLevel, 160
Logger, 160
LogMessage, 159
operator<<, 160
setIdentifier, 160
Arc::LogStream, 161
log, 162
LogStream, 161
Arc::MCC, 164
AddSecHandler, 165
logger, 166
MCC, 165
Next, 165
next_, 166
process, 165
ProcessSecHandlers, 165
sechandlers_, 166
Unlink, 165
Arc::MCC_Status, 168
getExplanation, 169
getKind, 168
getOrigin, 169
isOk, 168
MCC_Status, 168
operator bool, 169
operator std::string, 169

operator!, 169
Arc::MCCFactory, 171
get_instance, 171
MCCFactory, 171
Arc::MCCInterface, 172
process, 172
Arc::MD5Sum, 173
Arc::Message, 174
~Message, 175
Attributes, 175
Auth, 175
AuthContext, 176
Context, 175, 176
Message, 175
operator=, 175
Payload, 175
Arc::MessageAttributes, 177
add, 178
attributes_, 179
count, 178
get, 179
getAll, 179
MessageAttributes, 177
remove, 178
removeAll, 178
set, 178
Arc::MessageAuth, 180
Export, 180
Filter, 181
get, 180
remove, 180
set, 180
Arc::MessageAuthContext, 182
Arc::MessageContext, 183
Add, 183
Arc::MessageContextElement, 184
Arc::MessagePayload, 185
Arc::ModuleManager, 186
load, 186
ModuleManager, 186
setCfg, 186
Arc::MultiSecAttr, 187
Export, 187
operator bool, 187
Arc::PayloadRaw, 189
~PayloadRaw, 189
Buffer, 190
BufferPos, 190
BufferSize, 190
Content, 190
Insert, 190
PayloadRaw, 189
Size, 190
Truncate, 191
Arc::PayloadRawInterface, 192
Buffer, 193
BufferPos, 193
BufferSize, 193
Content, 192
Insert, 193
Size, 192
Truncate, 193
Arc::PayloadSOAP, 194
PayloadSOAP, 194
Arc::PayloadStream, 195
~PayloadStream, 195
Get, 196
GetHandle, 197
handle_, 197
operator bool, 196
operator!, 197
PayloadStream, 195
Put, 196
seekable_, 197
Timeout, 197
Arc::PayloadStreamInterface, 198
Get, 198
operator bool, 199
operator!, 199
Put, 199
Timeout, 199
Arc::PayloadWSRF, 200
PayloadWSRF, 200
Arc::PDPFactory, 204
get_instance, 204
PDPFactory, 204
Arc::Plexer, 206
~Plexer, 206
logger, 207
Next, 207
Plexer, 206
process, 207
Arc::PlexerEntry, 208
Arc::RegularExpression, 211
~RegularExpression, 211
getPattern, 212
hasPattern, 212
isOk, 212
match, 212
operator=, 212
RegularExpression, 211
Arc::Run, 220
~Run, 221
AssignStderr, 222
AssignStdin, 223
AssignStdout, 222
AssignWorkingDirectory, 223
CloseStderr, 223

CloseStdin, 223
CloseStdout, 223
KeepStderr, 223
KeepStdin, 223
KeepStdout, 223
Kill, 223
operator bool, 221
operator!, 221
ReadStderr, 222
ReadStdout, 222
Result, 222
Run, 221
Running, 222
Start, 222
Wait, 222
WriteStdin, 222
Arc::SecAttr, 224
ARCAuth, 226
Export, 225
Import, 225
operator bool, 225
operator!=, 225
operator==, 225
SAML, 226
SecAttr, 225
XACML, 226
Arc::SecAttr::Format, 227
Arc::SecAttrValue, 228
operator bool, 228
operator!=, 228
operator==, 228
Arc::SecHandlerFactory, 232
get_instance, 232
SecHandlerFactory, 232
Arc::Service, 234
AddSecHandler, 235
ProcessSecHandlers, 235
sechandlers_, 235
Service, 235
Arc::ServiceFactory, 237
get_instance, 237
ServiceFactory, 237
Arc::SimpleCondition, 238
broadcast, 238
lock, 238
reset, 239
signal, 238
signal_nonblock, 238
unlock, 238
wait, 238, 239
wait_nonblock, 239
Arc::SOAPEnvelope, 240
Fault, 241
GetXML, 241
Header, 241
IsFault, 241
Namespaces, 241
New, 241
operator=, 242
SOAPEnvelope, 240, 241
Arc::SOAPFault, 243
Code, 244
Detail, 245
Node, 244
operator bool, 244
Reason, 244
Role, 245
SOAPFault, 244
SOAPFaultCode, 243
Subcode, 244
Arc::SOAPMessage, 246
~SOAPMessage, 246
Attributes, 247
Payload, 247
SOAPMessage, 246
Arc::Time, 248
GetFormat, 249
GetTime, 249
operator std::string, 249
operator!=, 250
operator<, 249
operator<=, 249
operator>, 249
operator>=, 249
operator+, 250
operator-, 250
operator=, 249
operator==, 250
SetFormat, 249
SetTime, 249
str, 249
Time, 248
Arc::URL, 251
~URL, 252
AddOption, 254
BaseDN2Path, 255
ChangeHost, 253
ChangePath, 253
ChangePort, 253
ChangeProtocol, 253
CommonLocOption, 255
CommonLocOptions, 254
commonlocoptions, 257
ConnectionURL, 255
fullstr, 255
Host, 253
host, 256
HTTPOption, 254

HTTPOptions, 253
httpoptions, 256
LDAPAttributes, 254
ldapattributes, 256
LDAPFilter, 254
ldapfilter, 257
LDAPScope, 254
ldapscope, 256
Locations, 254
locations, 257
operator bool, 255
operator<, 255
operator<<, 256
operator==, 255
Option, 254
Options, 254
Passwd, 253
passwd, 256
Path, 253
path, 256
Path2BaseDN, 255
Port, 253
port, 256
Protocol, 253
protocol, 256
Scope, 252
str, 255
URL, 252
urloptions, 257
Username, 253
username, 256
Arc::URLLocation, 258
~URLLocation, 259
fullstr, 259
Name, 259
name, 259
str, 259
URLLocation, 258, 259
Arc::UsernameToken, 260
UsernameToken, 260
Arc::WSAEndpointReference, 262
~WSAEndpointReference, 262
Address, 263
MetaData, 263
operator XMLNode, 263
operator=, 263
ReferenceParameters, 263
WSAEndpointReference, 262
Arc::WSAHeader, 264
Action, 265
Check, 266
FaultTo, 265
From, 265
header_allocated_, 266
MessageID, 265
NewReferenceParameter, 266
operator XMLNode, 266
ReferenceParameter, 266
RelatesTo, 265, 266
RelationshipType, 266
ReplyTo, 265
To, 265
WSAHeader, 264
Arc::WSRF, 267
allocated_, 268
operator bool, 268
set_namespaces, 268
SOAP, 268
valid_, 268
WSRF, 267
Arc::WSRFBaseFault, 269
set_namespaces, 270
WSRFBaseFault, 269
Arc::WSRP, 271
set_namespaces, 271
WSRP, 271
Arc::WSRPFault, 273
WSRPFault, 273
Arc::WSRPResourcePropertyChangeFailure, 274
WSRPResourcePropertyChangeFailure, 274
Arc::XMLNode, 275
~XMLNode, 277
Attribute, 280
AttributesSize, 281
Child, 278
Destroy, 282
FullName, 279
Get, 279
GetDoc, 280
GetRoot, 282
GetXML, 279
is_owner_, 283
is_temporary_, 283
MatchXMLName, 283
MatchXMLNamespace, 283
Name, 279
Namespace, 279
NamespacePrefix, 281
Namespaces, 281
New, 278
NewAttribute, 281
NewChild, 281, 282
operator bool, 278
operator std::string, 280
operator!, 278
operator++, 278
operator–, 278
operator=, 280

Prefix, 279
ReadFromFile, 283
ReadFromStream, 283
Replace, 282
SaveToFile, 282
SaveToStream, 282
Set, 280
Size, 279
XMLNode, 277
XPathLookup, 282
Arc::XMLNodeContainer, 285
Add, 285
AddNew, 285, 286
Nodes, 286
operator=, 285
Size, 286
XMLNodeContainer, 285
ARCAuth
Arc::SecAttr, 226
ArcSec, 25
DECISION_DENY, 29
DECISION_INDETERMINATE, 29
DECISION_NOT_APPLICABLE, 29
DECISION_PERMIT, 29
INDETERMINATE, 29
MATCH, 29
MatchResult, 29
NO_MATCH, 29
ReqItemList, 27
Result, 29
Subject, 27
SubList, 28
ArcSec::AlgFactory, 33
ArcSec::ArcAttributeProxy, 34
getAttribute, 34
ArcSec::Attr, 36
ArcSec::AttributeFactory, 37
ArcSec::AttributeProxy, 41
ArcSec::AttributeValue, 42
encode, 42
equal, 42
getId, 42
getType, 42
ArcSec::Attrs, 43
ArcSec::AuthzRequestSection, 44
ArcSec::CombiningAlg, 56
combine, 56
ArcSec::DenyOverridesCombiningAlg, 125
combine, 125
ArcSec::EqualFunction, 128
getFunctionName, 128
ArcSec::EvalResult, 129
ArcSec::EvaluationCtx, 130
EvaluationCtx, 130
split, 130
ArcSec::EvaluatorContext, 131
operator AlgFactory ∗, 131
operator AttributeFactory ∗, 131
operator FnFactory ∗, 131
ArcSec::FnFactory, 135
ArcSec::Function, 136
ArcSec::MatchFunction, 163
getFunctionName, 163
ArcSec::PDP, 202
ArcSec::PermitOverridesCombiningAlg, 205
combine, 205
ArcSec::Policy, 209
addPolicy, 210
eval, 210
getEffect, 210
getEvalResult, 210
match, 209
ArcSec::Request, 213
addRequestItem, 214
getRequestItems, 214
make_request, 214
Request, 213, 214
setAttributeFactory, 214
setRequestItems, 214
ArcSec::RequestAttribute, 215
duplicate, 215
RequestAttribute, 215
ArcSec::RequestItem, 216
RequestItem, 216
ArcSec::RequestTuple, 217
ArcSec::Response, 218
ArcSec::ResponseItem, 219
ArcSec::SecHandler, 230
ArcSec::Security, 233
Assign
Arc::InformationContainer, 139
AssignStderr
Arc::Run, 222
AssignStdin
Arc::Run, 223
AssignStdout
Arc::Run, 222
AssignWorkingDirectory
Arc::Run, 223
AttrConstIter
Arc, 20
Attribute
Arc::XMLNode, 280
AttributeIterator
Arc::AttributeIterator, 38
Attributes
Arc::Message, 175
Arc::SOAPMessage, 247

attributes_
Arc::MessageAttributes, 179
AttributesSize
Arc::XMLNode, 281
AttrIter
Arc, 20
AttrMap
Arc, 20
Auth
Arc::Message, 175
AuthContext
Arc::Message, 176

Backup
Arc::DelegationConsumer, 116
BaseDN2Path
Arc::URL, 255
broadcast
Arc::SimpleCondition, 238
Buffer
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 193
buffer_size
Arc::DataBufferPar, 74
BufferPos
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 193
BufferSize
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 193
BufNum
Arc::DataPoint, 93
Arc::DataPointDirect, 98
Arc::DataPointIndex, 107
BufSize
Arc::DataPoint, 93
Arc::DataPointDirect, 98
Arc::DataPointIndex, 107
BUSY_ERROR
Arc, 20

Cache
Arc::DataPoint, 93
Arc::DataPointDirect, 98
Arc::DataPointIndex, 107
cancel
Arc::Counter, 63
Arc::CounterTicket, 67
Arc::IntraProcessCounter, 148
cb
Arc::DataCache, 78
changeExcess
Arc::Counter, 62
Arc::IntraProcessCounter, 147
ChangeHost
Arc::URL, 253
changeLimit
Arc::Counter, 62
Arc::IntraProcessCounter, 146
ChangePath
Arc::URL, 253
ChangePort
Arc::URL, 253
ChangeProtocol
Arc::URL, 253
Check
Arc::DataPoint, 89
Arc::DataPointIndex, 108
Arc::WSAHeader, 266
CheckCheckSum
Arc::DataPoint, 92
CheckCreated
Arc::DataCache, 78
Arc::DataPoint, 93
checks
Arc::DataMover, 84
CheckSize
Arc::DataPoint, 92
checksum_object
Arc::DataBufferPar, 74
checksum_valid
Arc::DataBufferPar, 74
CheckValid
Arc::DataCache, 79
Arc::DataPoint, 93
Child
Arc::XMLNode, 278
CIStringValue
Arc::CIStringValue, 52
clean
Arc::DataCache, 78
ClientSOAP
Arc::ClientSOAP, 54
CloseStderr
Arc::Run, 223
CloseStdin
Arc::Run, 223
CloseStdout
Arc::Run, 223
Code
Arc::SOAPFault, 244
combine
ArcSec::CombiningAlg, 56
ArcSec::DenyOverridesCombiningAlg, 125
ArcSec::PermitOverridesCombiningAlg, 205
CommonLocOption
Arc::URL, 255
CommonLocOptions

Arc::URL, 254
commonlocoptions
Arc::URL, 257
CompareMeta
Arc::DataPoint, 94
Config
Arc::Config, 57, 58
ConnectionURL
Arc::URL, 255
Content
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 192
ContentFromPayload
Arc, 21
Context
Arc::Message, 175, 176
context_lock_
Arc::DelegationContainerSOAP, 120
copy
Arc::DataCache, 78
count
Arc::MessageAttributes, 178
Counter
Arc::Counter, 61
Arc::CounterTicket, 67
Arc::ExpirationReminder, 133
CounterTicket
Arc::Counter, 65
Arc::CounterTicket, 66
CreateThreadFunction
Arc, 21
current_
Arc::AttributeIterator, 40
CurrentLocation
Arc::DataPoint, 95
Arc::DataPointDirect, 101
Arc::DataPointIndex, 105
CurrentLocationMetadata
Arc::DataPoint, 95
Arc::DataPointDirect, 102
Arc::DataPointIndex, 105

DataBufferPar
Arc::DataBufferPar, 70
DataCache
Arc::DataCache, 77
DataMover
Arc::DataMover, 82
DataPoint
Arc::DataPoint, 88
DataSpeed
Arc::DataSpeed, 111
DECISION_DENY
ArcSec, 29
DECISION_INDETERMINATE
ArcSec, 29
DECISION_NOT_APPLICABLE
ArcSec, 29
DECISION_PERMIT
ArcSec, 29
Delegate
Arc::DelegationProvider, 121
DelegateCredentialsInit
Arc::DelegationConsumerSOAP, 117
Arc::DelegationContainerSOAP, 119
Arc::DelegationProviderSOAP, 124
DelegatedToken
Arc::DelegationConsumerSOAP, 118
Arc::DelegationContainerSOAP, 119
Arc::DelegationProviderSOAP, 124
DelegationConsumer
Arc::DelegationConsumer, 115
DelegationConsumerSOAP
Arc::DelegationConsumerSOAP, 117
DelegationProvider
Arc::DelegationProvider, 121
DelegationProviderSOAP
Arc::DelegationProviderSOAP, 123
Destroy
Arc::XMLNode, 282
Detail
Arc::SOAPFault, 245
dmc_descriptor, 126
DMCFactory
Arc::DMCFactory, 127
doc_
Arc::InformationContainer, 139
duplicate
ArcSec::RequestAttribute, 215

encode
ArcSec::AttributeValue, 42
end_
Arc::AttributeIterator, 40
eof_position
Arc::DataBufferPar, 74
eof_read
Arc::DataBufferPar, 72, 73
eof_write
Arc::DataBufferPar, 73
equal
Arc::CIStringValue, 53
ArcSec::AttributeValue, 42
error
Arc::DataBufferPar, 73
error_read
Arc::DataBufferPar, 73
error_transfer

Arc::DataBufferPar, 73
error_write
Arc::DataBufferPar, 73
ETERNAL
Arc, 23
eval
ArcSec::Policy, 210
EvaluationCtx
ArcSec::EvaluationCtx, 130
ExpirationReminder
Arc::Counter, 65
Export
Arc::MessageAuth, 180
Arc::MultiSecAttr, 187
Arc::SecAttr, 225
extend
Arc::Counter, 63
Arc::CounterTicket, 67
Arc::IntraProcessCounter, 148

Fault
Arc::SOAPEnvelope, 241
FaultTo
Arc::WSAHeader, 265
file
Arc::DataCache, 77
Filter
Arc::MessageAuth, 181
for_read
Arc::DataBufferPar, 71
for_write
Arc::DataBufferPar, 71, 72
force_to_meta
Arc::DataMover, 84
From
Arc::WSAHeader, 265
FullName
Arc::XMLNode, 279
fullstr
Arc::URL, 255
Arc::URLLocation, 259

Generate
Arc::DelegationConsumer, 116
GENERIC_ERROR
Arc, 20
Get
Arc::ArcLocation, 35
Arc::InformationContainer, 138
Arc::InformationInterface, 140
Arc::PayloadStream, 196
Arc::PayloadStreamInterface, 198
Arc::XMLNode, 279
get
Arc::MessageAttributes, 179
Arc::MessageAuth, 180
get_instance
Arc::ACCFactory, 32
Arc::DMCFactory, 127
Arc::LoaderFactory, 152
Arc::MCCFactory, 171
Arc::PDPFactory, 204
Arc::SecHandlerFactory, 232
Arc::ServiceFactory, 237
getACC
Arc::Loader, 150
GetAdditionalChecks
Arc::DataPoint, 90
Arc::DataPointDirect, 99
Arc::DataPointIndex, 109
getAll
Arc::MessageAttributes, 179
getAttribute
ArcSec::ArcAttributeProxy, 34
GetCheckSum
Arc::DataPoint, 93
getCounterTicket
Arc::Counter, 64
GetCreated
Arc::DataCache, 79
Arc::DataPoint, 93
getCurrentTime
Arc::Counter, 64
GetDoc
Arc::XMLNode, 280
getEffect
ArcSec::Policy, 210
getEvalResult
ArcSec::Policy, 210
getExcess
Arc::Counter, 62
Arc::IntraProcessCounter, 146
getExpirationReminder
Arc::Counter, 65
getExpiryTime
Arc::Counter, 64
Arc::ExpirationReminder, 132
getExplanation
Arc::MCC_Status, 169
GetFormat
Arc::Time, 249
getFunctionName
ArcSec::EqualFunction, 128
ArcSec::MatchFunction, 163
GetHandle
Arc::PayloadStream, 197
getId
ArcSec::AttributeValue, 42

getKind
Arc::MCC_Status, 168
getLevel
Arc::LogMessage, 160
getLimit
Arc::Counter, 61
Arc::IntraProcessCounter, 146
getOrigin
Arc::MCC_Status, 169
GetOverlay
Arc::BaseConfig, 46
getPattern
Arc::RegularExpression, 212
GetPlugins
Arc::ArcLocation, 35
getRequestItems
ArcSec::Request, 214
getReservationID
Arc::ExpirationReminder, 132
GetRoot
Arc::XMLNode, 282
getRootLogger
Arc::Logger, 157
GetSecure
Arc::DataPoint, 90
Arc::DataPointDirect, 99
Arc::DataPointIndex, 109
GetSize
Arc::DataPoint, 92
getThreshold
Arc::Logger, 158
GetTime
Arc::Time, 249
GetTries
Arc::DataPoint, 94
getType
ArcSec::AttributeValue, 42
GetURL
Arc::DataPoint, 88
GetValid
Arc::DataCache, 79
Arc::DataPoint, 93
getValue
Arc::Counter, 63
Arc::IntraProcessCounter, 147
GetXML
Arc::SOAPEnvelope, 241
Arc::XMLNode, 279
GUID
Arc, 21

handle_
Arc::PayloadStream, 197
hasMore
Arc::AttributeIterator, 40
hasPattern
Arc::RegularExpression, 212
HaveLocations
Arc::DataPoint, 95
Arc::DataPointDirect, 102
Arc::DataPointIndex, 105
Header
Arc::SOAPEnvelope, 241
header_allocated_
Arc::WSAHeader, 266
HISTORIC
Arc, 23
hold
Arc::DataSpeed, 114
Host
Arc::URL, 253
host
Arc::URL, 256
HTTPOption
Arc::URL, 254
HTTPOptions
Arc::URL, 253
httpoptions
Arc::URL, 256

ID
Arc::DelegationConsumer, 116
IDType
Arc::Counter, 61
Import
Arc::SecAttr, 225
INDETERMINATE
ArcSec, 29
InfoRegister
Arc::InfoRegister, 137
InformationContainer
Arc::InformationContainer, 138
InformationInterface
Arc::InformationInterface, 140
InformationRequest
Arc::InformationRequest, 142
InformationResponse
Arc::InformationResponse, 144
Init
Arc::ArcLocation, 35
Insert
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 193
IntraProcessCounter
Arc::IntraProcessCounter, 145
is_notwritten
Arc::DataBufferPar, 72
is_owner_

Arc::XMLNode, 283
is_read
Arc::DataBufferPar, 71
is_temporary_
Arc::XMLNode, 283
is_written
Arc::DataBufferPar, 72
IsFault
Arc::SOAPEnvelope, 241
IsIndex
Arc::DataPoint, 94
Arc::DataPointDirect, 98
Arc::DataPointIndex, 106
isOk
Arc::MCC_Status, 168
Arc::RegularExpression, 212
isValid
Arc::CounterTicket, 66

KeepStderr
Arc::Run, 223
KeepStdin
Arc::Run, 223
KeepStdout
Arc::Run, 223
key
Arc::AttributeIterator, 39
Kill
Arc::Run, 223

LDAPAttributes
Arc::URL, 254
ldapattributes
Arc::URL, 256
LDAPFilter
Arc::URL, 254
ldapfilter
Arc::URL, 257
LDAPScope
Arc::URL, 254
ldapscope
Arc::URL, 256
link
Arc::DataCache, 78
ListFiles
Arc::DataPoint, 89
load
Arc::ModuleManager, 186
load_all_instances
Arc::LoaderFactory, 152
Loader
Arc::Loader, 150
loader_descriptors
Arc, 20
LoaderFactory
Arc::LoaderFactory, 152
Local
Arc::DataPoint, 94
Arc::DataPointDirect, 98
Arc::DataPointIndex, 107
Locations
Arc::URL, 254
locations
Arc::DataPointIndex, 110
Arc::URL, 257
LocationValid
Arc::DataPoint, 95
Arc::DataPointDirect, 102
Arc::DataPointIndex, 105
lock
Arc::SimpleCondition, 238
lock_
Arc::InformationInterface, 141
log
Arc::LogDestination, 155
Arc::LogStream, 162
LogDestination
Arc::LogDestination, 154
LogError
Arc::DelegationConsumer, 116
Logger
Arc::Logger, 157
Arc::LogMessage, 160
logger
Arc::MCC, 166
Arc::Plexer, 207
LogLevel
Arc, 20
LogMessage
Arc::LogMessage, 159
LogStream
Arc::LogStream, 161

make_request
ArcSec::Request, 214
MakeConfig
Arc::BaseConfig, 46
MATCH
ArcSec, 29
match
Arc::RegularExpression, 212
ArcSec::Policy, 209
MatchResult
ArcSec, 29
MatchXMLName
Arc, 21
Arc::XMLNode, 283
MatchXMLNamespace

Arc, 21, 22
Arc::XMLNode, 283
max_duration_
Arc::DelegationContainerSOAP, 119
max_inactivity_time_failure
Arc::DataSpeed, 114
max_size_
Arc::DelegationContainerSOAP, 119
max_usage_
Arc::DelegationContainerSOAP, 120
MCC
Arc::MCC, 165
mcc_descriptor, 167
MCC_Status
Arc::MCC_Status, 168
MCCFactory
Arc::MCCFactory, 171
Message
Arc::Message, 175
MessageAttributes
Arc::AttributeIterator, 40
Arc::MessageAttributes, 177
MessageID
Arc::WSAHeader, 265
MetaData
Arc::WSAEndpointReference, 263
min_average_speed_failure
Arc::DataSpeed, 114
min_speed_failure
Arc::DataSpeed, 114
ModuleManager
Arc::ModuleManager, 186
msg
Arc::Logger, 158

Name
Arc::URLLocation, 259
Arc::XMLNode, 279
name
Arc::URLLocation, 259
Namespace
Arc::XMLNode, 279
NamespacePrefix
Arc::XMLNode, 281
Namespaces
Arc::SOAPEnvelope, 241
Arc::XMLNode, 281
New
Arc::SOAPEnvelope, 241
Arc::XMLNode, 278
NewAttribute
Arc::XMLNode, 281
NewChild
Arc::XMLNode, 281, 282
NewReferenceParameter
Arc::WSAHeader, 266
Next
Arc::MCC, 165
Arc::Plexer, 207
next_
Arc::MCC, 166
NextLocation
Arc::DataPoint, 95
Arc::DataPointDirect, 102
Arc::DataPointIndex, 105
NO_MATCH
ArcSec, 29
Node
Arc::SOAPFault, 244
Nodes
Arc::XMLNodeContainer, 286

operator AlgFactory *
ArcSec::EvaluatorContext, 131
operator AttributeFactory *
ArcSec::EvaluatorContext, 131
operator bool
Arc::CIStringValue, 53
Arc::DataBufferPar, 70
Arc::DataCache, 78
Arc::DataPoint, 88
Arc::MCC_Status, 169
Arc::MultiSecAttr, 187
Arc::PayloadStream, 196
Arc::PayloadStreamInterface, 199
Arc::Run, 221
Arc::SecAttr, 225
Arc::SecAttrValue, 228
Arc::SOAPFault, 244
Arc::URL, 255
Arc::WSRF, 268
Arc::XMLNode, 278
operator FnFactory *
ArcSec::EvaluatorContext, 131
operator MCCFactory *
Arc::ChainContext, 47
operator PDPFactory *
Arc::ChainContext, 47
operator SecHandlerFactory *
Arc::ChainContext, 47
operator ServiceFactory *
Arc::ChainContext, 47
operator std::string
Arc::MCC_Status, 169
Arc::Time, 249
Arc::XMLNode, 280
operator XMLNode
Arc::WSAEndpointReference, 263

Arc::WSAHeader, 266
operator!
Arc::DataPoint, 88
Arc::MCC_Status, 169
Arc::PayloadStream, 197
Arc::PayloadStreamInterface, 199
Arc::Run, 221
Arc::XMLNode, 278
operator!=
Arc::SecAttr, 225
Arc::SecAttrValue, 228
Arc::Time, 250
operator<
Arc::ExpirationReminder, 132
Arc::Time, 249
Arc::URL, 255
operator<<
Arc, 22
Arc::LogMessage, 160
Arc::URL, 256
operator<=
Arc::Time, 249
operator>
Arc::Time, 249
operator>=
Arc::Time, 249
operator∗
Arc::AttributeIterator, 39
operator+
Arc::Time, 250
operator++
Arc::AttributeIterator, 39
Arc::XMLNode, 278
operator-
Arc::Time, 250
operator->
Arc::AttributeIterator, 39
operator–
Arc::XMLNode, 278
operator=
Arc::Message, 175
Arc::RegularExpression, 212
Arc::SOAPEnvelope, 242
Arc::Time, 249
Arc::WSAEndpointReference, 263
Arc::XMLNode, 280
Arc::XMLNodeContainer, 285
operator==
Arc::SecAttr, 225
Arc::SecAttrValue, 228
Arc::Time, 250
Arc::URL, 255
Option
Arc::URL, 254
Options
Arc::URL, 254

parse
Arc::Config, 58
PARSING_ERROR
Arc, 20
Passive
Arc::DataPoint, 90
Arc::DataPointDirect, 100
Arc::DataPointIndex, 109
passive
Arc::DataMover, 84
Passwd
Arc::URL, 253
passwd
Arc::URL, 256
Path
Arc::URL, 253
path
Arc::URL, 256
Path2BaseDN
Arc::URL, 255
Payload
Arc::Message, 175
Arc::SOAPMessage, 247
PayloadRaw
Arc::PayloadRaw, 189
PayloadSOAP
Arc::PayloadSOAP, 194
PayloadStream
Arc::PayloadStream, 195
PayloadWSRF
Arc::PayloadWSRF, 200
pdp_descriptor, 203
PDPFactory
Arc::PDPFactory, 204
Plexer
Arc::Plexer, 206
Port
Arc::URL, 253
port
Arc::URL, 256
PostRegister
Arc::DataPoint, 91
Arc::DataPointDirect, 101
Prefix
Arc::XMLNode, 279
PreRegister
Arc::DataPoint, 91
Arc::DataPointDirect, 100
PreUnregister
Arc::DataPoint, 92
Arc::DataPointDirect, 101

print
Arc::Config, 58
process
Arc::ClientSOAP, 54
Arc::MCC, 165
Arc::MCCInterface, 172
Arc::Plexer, 207
ProcessSecHandlers
Arc::MCC, 165
Arc::Service, 235
Protocol
Arc::URL, 253
protocol
Arc::URL, 256
PROTOCOL_RECOGNIZED_ERROR
Arc, 20
ProvidesMeta
Arc::DataPoint, 94
Arc::DataPointDirect, 101
Arc::DataPointIndex, 106
Put
Arc::PayloadStream, 196
Arc::PayloadStreamInterface, 199

Range
Arc::DataPoint, 91
Arc::DataPointDirect, 100
Arc::DataPointIndex, 109
ReadFromFile
Arc::XMLNode, 283
ReadFromStream
Arc::XMLNode, 283
ReadOutOfOrder
Arc::DataPoint, 89
Arc::DataPointDirect, 99
Arc::DataPointIndex, 108
ReadStderr
Arc::Run, 222
ReadStdout
Arc::Run, 222
ReadURLList
Arc, 22
Reason
Arc::SOAPFault, 244
ReferenceParameter
Arc::WSAHeader, 266
ReferenceParameters
Arc::WSAEndpointReference, 263
Registered
Arc::DataPoint, 91
Arc::DataPointDirect, 100
Arc::DataPointIndex, 106
registration
Arc::InfoRegister, 137
registration_forever
Arc::InfoRegister, 137
RegularExpression
Arc::RegularExpression, 211
RelatesTo
Arc::WSAHeader, 265, 266
RelationshipType
Arc::WSAHeader, 266
Remove
Arc::DataPoint, 89
Arc::DataPointIndex, 108
remove
Arc::MessageAttributes, 178
Arc::MessageAuth, 180
removeAll
Arc::MessageAttributes, 178
removeDestinations
Arc::Logger, 157
RemoveLocation
Arc::DataPoint, 96
Arc::DataPointDirect, 102
Arc::DataPointIndex, 105
RemoveLocations
Arc::DataPoint, 96
Arc::DataPointDirect, 102
Arc::DataPointIndex, 106
Replace
Arc::XMLNode, 282
ReplyTo
Arc::WSAHeader, 265
ReqItemList
ArcSec, 27
Request
Arc::DelegationConsumer, 116
ArcSec::Request, 213, 214
RequestAttribute
ArcSec::RequestAttribute, 215
RequestItem
ArcSec::RequestItem, 216
reserve
Arc::Counter, 63
Arc::IntraProcessCounter, 147
reset
Arc::DataSpeed, 113
Arc::SimpleCondition, 239
Resolve
Arc::DataPoint, 91
Arc::DataPointDirect, 100
Restore
Arc::DelegationConsumer, 116
restricted_
Arc::DelegationContainerSOAP, 120
Result
Arc::InformationResponse, 144

Arc::Run, 222
ArcSec, 29
retry
Arc::DataMover, 84
Role
Arc::SOAPFault, 245
Run
Arc::Run, 221
Running
Arc::Run, 222

SAML
Arc::SecAttr, 226
SaveToFile
Arc::XMLNode, 282
SaveToStream
Arc::XMLNode, 282
Scope
Arc::URL, 252
SecAttr
Arc::SecAttr, 225
sechandler_descriptor, 231
SecHandlerFactory
Arc::SecHandlerFactory, 232
sechandlers_
Arc::MCC, 166
Arc::Service, 235
secure
Arc::DataMover, 84
seekable_
Arc::PayloadStream, 197
Service
Arc::Service, 235
service_descriptor, 236
ServiceFactory
Arc::ServiceFactory, 237
SESSION_CLOSE
Arc, 20
Set
Arc::XMLNode, 280
set
Arc::DataBufferPar, 70
Arc::MessageAttributes, 178
Arc::MessageAuth, 180
set_base
Arc::DataSpeed, 113
set_default_max_inactivity_time
Arc::DataMover, 85
set_default_min_average_speed
Arc::DataMover, 84
set_default_min_speed
Arc::DataMover, 84
set_max_data
Arc::DataSpeed, 113
set_max_inactivity_time
Arc::DataSpeed, 113
set_min_average_speed
Arc::DataSpeed, 112
set_min_speed
Arc::DataSpeed, 112
set_namespaces
Arc::WSRF, 268
Arc::WSRFBaseFault, 270
Arc::WSRP, 271
set_progress_indicator
Arc::DataSpeed, 113
SetAdditionalChecks
Arc::DataPoint, 90
Arc::DataPointDirect, 99
Arc::DataPointIndex, 109
setAttributeFactory
ArcSec::Request, 214
setCfg
Arc::ModuleManager, 186
SetCheckSum
Arc::DataPoint, 93
SetCreated
Arc::DataCache, 78
Arc::DataPoint, 93
setExcess
Arc::Counter, 62
Arc::IntraProcessCounter, 147
SetFormat
Arc::Time, 249
setIdentifier
Arc::LogMessage, 160
setLimit
Arc::Counter, 61
Arc::IntraProcessCounter, 146
SetMeta
Arc::DataPoint, 94
setRequestItems
ArcSec::Request, 214
SetSecure
Arc::DataPoint, 90
Arc::DataPointDirect, 99
Arc::DataPointIndex, 109
SetSize
Arc::DataPoint, 92
setThreshold
Arc::Logger, 157
SetTime
Arc::Time, 249
SetTries
Arc::DataPoint, 94
Arc::DataPointIndex, 106
SetValid
Arc::DataCache, 79

Arc::DataPoint, 93
signal
Arc::SimpleCondition, 238
signal_nonblock
Arc::SimpleCondition, 238
Size
Arc::PayloadRaw, 190
Arc::PayloadRawInterface, 192
Arc::XMLNode, 279
Arc::XMLNodeContainer, 286
SOAP
Arc::InformationRequest, 142
Arc::WSRF, 268
SOAPEnvelope
Arc::SOAPEnvelope, 240, 241
SOAPFault
Arc::SOAPFault, 244
SOAPFaultCode
Arc::SOAPFault, 243
SOAPMessage
Arc::SOAPMessage, 246
speed
Arc::DataBufferPar, 75
split
ArcSec::EvaluationCtx, 130
Start
Arc::Run, 222
start
Arc::DataCache, 77
StartReading
Arc::DataPoint, 88
Arc::DataPointIndex, 107
StartWriting
Arc::DataPoint, 88
Arc::DataPointIndex, 107
STATUS_OK
Arc, 20
StatusKind
Arc, 20
stop
Arc::DataCache, 77
StopReading
Arc::DataPoint, 89
Arc::DataPointIndex, 108
StopWriting
Arc::DataPoint, 89
Arc::DataPointIndex, 108
str
Arc::DataPoint, 88
Arc::Time, 249
Arc::URL, 255
Arc::URLLocation, 259
string
Arc, 22
stringto
Arc, 22
Subcode
Arc::SOAPFault, 244
Subject
ArcSec, 27
SubList
ArcSec, 28

Time
Arc::Time, 248
TimeFormat
Arc, 21
Timeout
Arc::PayloadStream, 197
Arc::PayloadStreamInterface, 199
TimeStamp
Arc, 22, 23
To
Arc::WSAHeader, 265
tokenize
Arc, 23
tostring
Arc, 23
Transfer
Arc::DataMover, 83
transfer
Arc::DataSpeed, 113
transfered_size
Arc::DataSpeed, 114
trim
Arc, 23
Truncate
Arc::PayloadRaw, 191
Arc::PayloadRawInterface, 193

UNKNOWN_SERVICE_ERROR
Arc, 20
Unlink
Arc::MCC, 165
unlock
Arc::SimpleCondition, 238
Unregister
Arc::DataPoint, 92
Arc::DataPointDirect, 101
UpdateCredentials
Arc::DelegationConsumerSOAP, 117
Arc::DelegationContainerSOAP, 119
Arc::DelegationProviderSOAP, 124
upper
Arc, 23
URL
Arc::URL, 252
URLLocation

Arc::URLLocation, 258, 259
urloptions
Arc::URL, 257
Username
Arc::URL, 253
username
Arc::URL, 256
UsernameToken
Arc::UsernameToken, 260
UUID
Arc, 23

valid_
Arc::WSRF, 268
verbose
Arc::DataMover, 83
Arc::DataSpeed, 112

Wait
Arc::Run, 222
wait
Arc::DataBufferPar, 73
Arc::SimpleCondition, 238, 239
wait_eof
Arc::DataBufferPar, 74
wait_eof_read
Arc::DataBufferPar, 74
wait_eof_write
Arc::DataBufferPar, 74
wait_nonblock
Arc::SimpleCondition, 239
wait_read
Arc::DataBufferPar, 74
wait_used
Arc::DataBufferPar, 74
wait_write
Arc::DataBufferPar, 74
WriteOutOfOrder
Arc::DataPoint, 90
Arc::DataPointDirect, 99
Arc::DataPointIndex, 108
WriteStdin
Arc::Run, 222
WSAEndpointReference
Arc::WSAEndpointReference, 262
WSAFault
Arc, 21
WSAFaultAssign
Arc, 23
WSAFaultExtract
Arc, 23
WSAFaultInvalidAddressingHeader
Arc, 21
WSAFaultUnknown
Arc, 21
WSAHeader
Arc::WSAHeader, 264
WSRF
Arc::WSRF, 267
WSRFBaseFault
Arc::WSRFBaseFault, 269
WSRP
Arc::WSRP, 271
WSRPFault
Arc::WSRPFault, 273
WSRPResourcePropertyChangeFailure
Arc::WSRPResourcePropertyChangeFailure, 274

XACML
Arc::SecAttr, 226
XMLNode
Arc::XMLNode, 277
XMLNodeContainer
Arc::XMLNodeContainer, 285
XPathLookup
Arc::XMLNode, 282